Enhancing the Time Complexity of Mathematically Intensive Algorithms; the Case of Cryptography

Paul K. Arhin Jnr
Dept. of Computer Science and I. T
University of Cape Coast
Cape Coast, Ghana

Michael Asante
Dept. of Computer Science
Kwame Nkrumah University of Science and Technology
Kumasi, Ghana

Linda Otoo
Dept. of Computer Science and I. T
University of Cape Coast
Cape Coast, Ghana

ABSTRACT
This article aims to compare the performance of the RSA encryption algorithm on two different hardware architectures, namely a CPU and a GPU CUDA. The RSA encryption algorithm is widely used for secure data storage and transmission. The algorithm requires complex mathematical processes that can be computationally demanding and can take significant time to execute, particularly for keys with larger sizes. In this paper, A parallelization technique is proposed in this article, which leverages the capabilities of GPUs to speed up the RSA algorithm. The research is done by experiment using different key sizes to measure the performance of RSA on both platforms; CPU and GPU. The proposed approach involves the parallelization of the most computationally intensive parts of the RSA Algorithm, including modular exponentiation and multiplication. GPU implementation of the RSA algorithm is done using CUDA, a programming model developed by NVIDIA for parallel computing on GPUs. The experimental results show the effectiveness of using GPUs to accelerate the RSA algorithm thus resulting in a faster and more efficient cryptographic solutions. This has significant implications for real-world applications, especially those that are mathematically intensive and demand secure and effective data transmission, like e-commerce, banking, and other financial services.

Keywords
RSA, CUDA, GPU, Cryptographic Algorithm, GPU

1. INTRODUCTION
RSA cryptographic algorithm is one of the most popularly used asymmetric cryptographic algorithms for secure data transmission and communication over the internet. It is built on the mathematical problem of factoring large integers. Even tough RSA is very popular and strong in security, it is computationally intensive and needs a lot of computing power, particularly for large key sizes [1].

In order to address and solve this problem, researchers have explored the use of GPUs to enhance RSA algorithms calculations and computations due to their parallel processing capabilities that can perform many calculations simultaneously [2][3][4]. This makes them excellent accelerators for computationally intensive tasks like cryptography.

To efficiently evaluate the performance of RSA algorithm on GPU-based implementation, execution time using different key sizes and fixed messages size will be recorded and then compare our implementation's efficiency with that of current CPU-based implementations and assess the speedup brought on by the benefit of using GPUs as RSA accelerator [5].

Applications that use RSA encryption, like blockchain, cloud computing, and safe online transactions, may benefit practically from the research's conclusions [6].

2. PROBLEM STATEMENT
RSA encryption is computationally intensive and slow on conventional CPUs because it requires numerous modular exponentiations and big prime number processes [7][8][9]. The performance of RSA encryption must therefore be improved, and it is necessary to discover an effective and scalable solution that can enhance the performance of RSA encryption. The power of GPUs, which are built to manage parallel processing and can carry out the required mathematical operations more rapidly and effectively than CPUs, can be leveraged to address this issue. This project's objective is to investigate how GPUs might be used to speed up RSA encryption and assess the efficiency gains made by this strategy and to offer recommendations for choosing the best platform for various application domains that are mathematically intensive.

3. REVIEW OF LITERATURE
Popular cryptographic method RSA (Rivest-Shamir-Adleman) makes use of the idea of public key cryptography. Researchers have looked into using graphics processing units (GPUs) to improve RSA speed in recent years. The most recent advances in using GPUs to improve RSA are outlined in this literature analysis.

Shen et al. (2005) made one of the first proposals for RSA execution on GPUs [10]. The most time-consuming process in RSA, modular exponentiation, was sped up using NVIDIA's CUDA programming model. According to the experimental findings, the CPU application could be sped up by up to 34 times.

A GPU-based RSA implementation utilizing a modified sliding window method was suggested by Satoh and Takano (2008) [11] in a different study. They used ATI's Stream SDK and outperformed the CPU version by up to 69 times. They also compared their approach with that of Shen et al. and reported improved performance.

Later, using the OpenCL programming model, Li et al. (2010) [12] suggested a hybrid CPU-GPU implementation of RSA. They separated the RSA computation into CPU and GPU components and independently optimized each component. In comparison to the CPU implementation, the experimental results revealed a considerable speedup of up to 127 times.

A multi-precision arithmetic-based CUDA-based RSA implementation was suggested by Vaidya and Jadhav (2021) [13] in recent research. They used NVIDIA's Tesla K80 GPU and outperformed the CPU version by up to 13.5 times. Numerous studies have been done on improving RSA using

GPUs, and the results have demonstrated appreciable speedups compared to the CPU implementation.

4. METHODOLOGY

In many disciplines, including computer science, experiment is a potent research methodology that can be used to try and assess the efficacy of various methods and tools. An illustration of how experimentation can be used to improve the performance of RSA encryption algorithms using GPUs is provided by the research done by Yu et al. [14]. Observation as a methodology will then be employed to collect data for further analysis. For researching how algorithms and systems behave in practical contexts, observation is a useful research technique.

The fundamental operations of the RSA cryptographic algorithm, namely encryption and decryption, were implemented in this work on a CUDA – Enabled GPU and a multi-core CPU using different key sizes and a data size of 400 bits, and the performance metrics; encryption time, and decryption time were gathered. For the Time complexity of both RSA encryption and decryption on GPU and CPU, the operation was run 12 times and the average value recorded as shown in equation 1. The length of time it takes an encryption method to convert plaintext into ciphertext is known as the encryption time and the time required by the decryption method to create plaintext from ciphertext is referred to as the decryption time. The RSA encryption and decryption processes are implemented on both the CPU and GPU platforms; on CPU, the environment used was C++ and on GPU, the environment used was Compute Unified Device Architecture (CUDA). The results of the benchmark tests are analyzed and the performance of RSA on GPU and CPU compared. The data is plotted and visualized to draw meaningful conclusions. All execution times are measured in milliseconds (ms)

$$\boldsymbol{Exec_{Time}} = \frac{exec1+exec\ 2+\cdots+exec\_12}{12} \qquad \text{------ } Equ\ (1)$$

4.1 RSA KEY GENERATION

The subsequent stages are used to create the public and private keys.:

Step 1: p and q, two large, random primes, are originally generated.

Step 2: Then the modulus of *n* is computed as *n = pq.*

Step 3: Choose an odd public exponent e that is comparatively prime to p - 1 and q - 1 and lies between 3 and n-1.

Step 4: From e, p, and q, calculate the secret exponent d.

Step 5: Print the public key as (n, e) and the private key as (n, d).

4.2 RSA ENCRYPTION

Encryption Algorithm: ***c = Encrypt (m)*** = $c = m^e \ mod\ n$

The methods below describe how to encrypt a message (m) using the RSA public key (n, e):

1. Represent the message as an integer m, where 0 <= m < n
2. Calculate $c = m^e \ mod\ n$

   The ciphertext is the integer c

4.3 RSA DECRYPTION

Decryption Algorithm: ***m = Decrypt (c)*** = $m = c^d \ mod\ n$

The methods below describe how to decrypt ciphertext, **c** using the RSA private key (n, d):

1. Calculate m = $m = c^d \ mod\ n$
2. The plaintext is the integer m

4.4 EXPERIMENTAL SETUP

The computer used for this project has a graphics device with CUDA support made by NVIDIA. The information shows some basic specifications of the computer and the tool which is used in conducting the experiment.

Cuda – Enabled GPU

NVIDI A GeForce GT 130M

- 1.5 GHz with 32 Cores
- Process or clock of 1500MHz
- Memory Clock of 800MHz
- Memory Interface width of 128-bit

CPU

Intel i5 CPU with speed of 2.67GHz

Nvidia CUDA

Operating System

Microsoft Windows 11

5. THE MATHEMATICS OF CRYPTOGRAPHY

The art of secure communication, or cryptography, enables the flow of data without the threat of fraud or unauthorized access. It has grown in significance along with the development of digital communication, and it significantly depends on the use of mathematical procedures to guarantee the confidentiality of the encoded message.

The application of mathematical ideas like modular arithmetic, prime numbers, one-way functions and elliptic curves is one of the core principles of cryptography.

5.1 MODULAR ARITHMETIC

This kind of arithmetic, known as a modulus, works with integers that fall inside a specific range. In cryptography, such as the RSA technique, modular arithmetic is utilized to execute operations on integers that are too huge to handle directly.[15]

5.2 PRIME NUMBERS

In cryptography, prime numbers are essential since they are used to create the encryption keys. The security of some cryptographic algorithms, such as RSA and Diffie-Hellman, is based on the difficulty of factoring large composite numbers into their prime factors [16].

5.3 ELIPTIC CURVE

This kind of cryptography relies on the characteristics of elliptic curves. Several digital signature techniques and key exchange protocols employ elliptic curve cryptography in contemporary cryptographic systems [17].

5.4 ONE WAY FUNCTIONS

These mathematical operations are simple to perform in one way but are either impossible or very difficult to compute in the opposite direction. In order to assure that encrypted data cannot be simply decoded by an adversary, one-way functions are utilized in cryptographic algorithms [18].

5.5 HASH FUNCTIONS

Hash functions are mathematical operations that accept as input data and output a hash with a specified length. They are utilized in cryptography for digital signatures and data integrity checks. SHA-1, SHA-2, and SHA-3 are a few common hashing algorithms [19].

5.6 DIGITAL SIGNATURES

Authentication and non-repudiation of digital messages or documents are provided through mathematical systems known as digital signatures. They are produced using public-key cryptography and enable the recipient to confirm the message or document's validity and the sender's identity. [20].

5.7 ZERO-KNOWLEDGE PROOFS

Zero-knowledge proofs are mathematical methods that let one party convince another that they are aware of some information without actually disclosing that information. They are employed in cryptography to demonstrate possession of a secret key without actually disclosing the key [21].

In general, mathematical techniques and concepts are deeply ingrained in the subject of cryptography, and it is crucial to have a firm grasp on these ideas in order to build and execute secure cryptographic systems.

6. RESULT

Encryption Process on CPU and GPU

In this experiment, the RSA encryption algorithm is implemented using different key sizes on 400bits of data on multi-core CPU and a CUDA enabled GPU. Twelve runs of the experiment on the encryption process were completed, and the average time complexity in milliseconds recorded.

Table 1: The time complexity (latency) in milliseconds for CPU encryption of a file with a length of 400 bits using various key sizes.

Bit Key Size	Data Size	CPU	GPU
512	400	1.97	1.21
1024	400	4.94	4.01
2048	400	11.88	10.44
3072	400	30.06	23.45
4096	400	57.12	36.64

Decryption Process on CPU and GPU

In this experiment, the RSA decryption algorithm is run using different key sizes on 400bits of data on multi-core CPU and a CUDA enabled GPU. Twelve runs of the experiment on the decryption process were completed, and the average time complexity in milliseconds recorded.

Table 2: The time complexity (latency) in milliseconds for CPU decryption of a message with a length of 400 bits using various key sizes

Bit Key Size	Data Size	CPU	GPU
512	400	4.23	3.33
1024	400	8.01	5.52
2048	400	32.88	20.13
3072	400	110.94	60.22
4096	400	190.72	90.4

7. ANALYSIS AND DISCUSSION

On a GPU with CUDA support and a general-purpose CPU, comprehensive experimental study and processing of the RSA cryptographic algorithm have been successfully completed. The previous section shows the outcome in a table. The results have been analyzed and discussed in this chapter.

The results of this work reveal that, during the encryption process, the GPU's speed remains greater. When encrypting using the RSA cryptographic algorithm, the GPU outperforms the CPU in terms of speed as shown in figure 1. According to figure 1, the CPU would need 1.97 milliseconds to encrypt a 400-bit message with a 512-bit key, whereas the GPU would need 1.21 milliseconds.

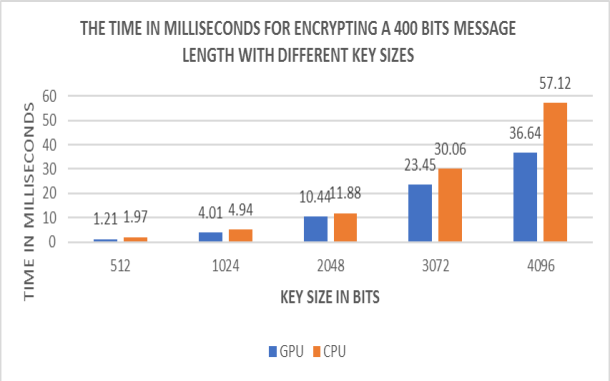

Fig. 1: The Time in Milliseconds for Encrypting A 400 Bits Message Length with Different Key Sizes

The results in figure 2 are not all that dissimilar from those in figure 1. It was also noted in figure 2 that the time required to decrypt a message of 400 bits in length using the CPU and GPU is still very different. In the decryption process, the GPU completely outperforms the CPU. For instance, the GPU decrypted a message with a key size of 512 bits in 3.33 milliseconds while the CPU required 4.23 milliseconds. Similarly, when the key size was 4096 bits, the GPU decrypted the message in 90.4 milliseconds while the CPU required 190.72 milliseconds. It is clear that it takes longer to decrypt a message than it does to encrypt one.

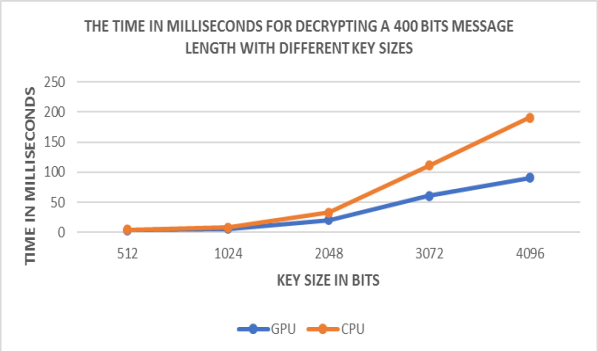

Fig. 2: The Time in Milliseconds for Decrypting A 400 Bits Message Length with Different Key Sizes

It was noted in this research that, the bottleneck for RSA Algorithm lies primarily in the key size, that means making use of large prime numbers. When small prime numbers were used to generate the smaller key sizes (say, 512 and 1024), RSA algorithm computes faster on both the CPU and GPU, since they required less computational power and memory resources. Unfortunately, small key sizes will also render RSA algorithm less secure and more susceptible to attacks. Larger keys will

make a more secured RSA but the research proves that, it has a drastic effect on speed as more computational power is needed. The parallelised nature of GPU made it more advantageous over the CPU when it comes to large key sizes. The speed difference for smaller key sizes remains small whiles the speed difference between large key sizes is very huge. In the encryption process, a key size of 512 bits yielded a speed of 1.97ms when RSA was executed on CPU whereas with the same key size, GPU run at a speed of 1.21ms. there was a delay of only 0.76ms by the CPU. On the other hand, when a 400mb data was executed on RSA with key size of 4096 bits, it took the CPU 57.12ms to complete the execution, whiles the GPU used 36.64ms to complete the execution. A speed difference of 20.48ms was noticed. This means, it took the CPU, an extra time of 20.48ms to complete RSA implementation with a key size of 4096 on 400mb data size.

Comparing the time difference of a smaller key size (512bits); 0.76ms and a larger key size (4096bits); 20.48ms, we can make the conclusion that, the parallelized nature of the CUDA enabled GPU gives it a higher advantage over the CPU in implementing mathematically intensive algorithms such as cryptographic algorithms.

8. CONCLUSION

The study made a thorough investigation into the implementation and performance analysis of RSA Cryptographic algorithm, a popular cryptographic algorithm used for secure data transmission and storage using a CUDA enabled GPU and a general-purpose CPU. The machines involved were a CUDA enabled Nvidia GeForce GT 130m and an Intel i5 CPU with speed of 2.6 GHz. The main focus of the study was to measure and compare their performance in terms of their Latency (execution time).

The study was conducted based on the hypothesis that, due to their architecture and design, GPUs are usually faster than CPUs at implementing RSA. GPUs have a large number of more specialized, parallel-optimized cores than CPUs, which have a limited number of powerful computing cores. Numerous modular exponentiations are required for RSA encryption and decryption; these can be parallelized to make use of a GPU's numerous cores.

RSA cryptographic algorithm was implemented on a CUDA-enabled GPU and a general-purpose CPU and their average execution time for different key sizes recorded and compared. The results of the study showed that the GPU implementation of RSA was faster than the CPU implementation, confirming the hypothesis.

Overall, the research indicates that, GPUs are a better option for high-performance computing activities, such as mathematically intensive algorithms.

The study also identified future works to consider, such as the measurement of other performance metrics such as speed factor, throughput, memory consumption, and energy consumption. Overall, the study presents a thorough investigation into the implementation and performance analysis of RSA Cryptographic algorithm using a CUDA-enabled GPU and a general-purpose CPU.

9. ACKNOWLEDGMENTS

Our gratitude to all the people and groups who helped to complete this study piece. Our gratitude is also expressed to the research team and our colleagues, who helped us out by sharing their skills, information, and encouragement. They played a critical role in carrying out the studies and collecting the results.

Lastly, our appreciation goes to the organizations and individuals that gave us the tools and funding we needed to do this study.

10. REFERENCES

[1] Rivest, R. L., Shamir, A., & Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126

[2] Dullweber, M., Schoppmann, P., Rechberger, C., & Schrammel, P. (2011). RSA on GPU: Fast factorization and secure implementation. In International Conference on Selected Areas in Cryptography (pp. 205-222). Springer, Berlin, Heidelberg.

[3] Ananthakrishnan, S., & Mehta, S. (2014). Implementation of RSA on CUDA platform. International Journal of Computer Applications, 93(8), 25-28.

[4] Dong, Y., Zhang, W., & Liu, X. (2017). GPU-accelerated RSA key generation and encryption. Journal of Supercomputing, 73(10), 4528-4541

[5] Liu, Y., Sun, Z., Zhang, Q., & Chen, C. (2019). A GPU-accelerated RSA cryptosystem using CUDA. IEEE Access, 7, 114330-114339

[6] Narula, R., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.

[7] Kumar, P., Srinivasan, S., & Rawat, S. (2018). A comparative study of CPU and GPU performance for RSA algorithm. International Journal of Computer Applications, 181(32), 29-33.

[8] Wang, Y., Li, H., Li, X., & Li, Z. (2019). Implementation and comparison of RSA algorithm on CPU and GPU platforms. 6th International Conference on Information Technology and Quantitative Management (ITQM).

[9] Zhang, Y., Wang, X., & Wang, Q. (2021). A comparative study of RSA algorithm performance on CPU and GPU. IEEE International Conference on Computational Science and Engineering (CSE).

[10] Shen, C., Chen, Y., & Chen, K. (2005). Implementation of RSA cryptosystem on GPU. In International Symposium on Parallel and Distributed Processing and Applications (pp. 839-844). Springer.

[11] Satoh, A., & Takano, K. (2008). Fast RSA implementation using a GPU with modified sliding-window method. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 344-360). Springer

[12] Li, X., Yang, C., & Dai, Y. (2010). A hybrid CPU-GPU implementation of RSA based on OpenCL. In International Conference on High Performance Computing and Communications (pp. 505-510). IEEE.

[13] Vaidya, S., & Jadhav, A. (2021). GPU-Based Implementation of RSA Algorithm Using Multi-Precision Arithmetic. International Journal of Computer Science and Network Security (IJCSNS), 21(2), 12-20.

[14] Yu, X., Li, M., Li, J., & Liao, X. (2020). GPU-accelerated RSA encryption algorithm based on Chinese Remainder Theorem. The Journal of Supercomputing, 76(2), 1083-1096. doi: 10.1007/s11227-019-02911-2

[15] Shoup, Victor. A Computational Introduction to Number Theory and Algebra. Cambridge University Press, 2005

[16] Crandall, Richard E., and Carl Pomerance. Prime Numbers: A Computational Perspective. Springer Science & Business Media, 2012

[17] Hankerson, Darrel, Alfred J. Menezes, and Scott Vanstone. Guide to Elliptic Curve Cryptography. Springer Science & Business Media, 2010

[18] Dwork, Cynthia, and Moni Naor. "On the Practicality of One-Way Functions." Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing. ACM, 1990.

[19] Katz, Jonathan, and Yehuda Lindell. Introduction to Modern Cryptography. CRC Press, 2014

[20] Schneier, Bruce. Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, 1996.

[21] Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems." SIAM Journal on Computing, vol. 18, no. 1, 1989, pp. 186-208