symmetry

MDPI

Article

Lightweight Computational Complexity Stepping Up the NTRU Post-Quantum Algorithm Using Parallel Computing

Ghada Farouk Elkabbany [1], Hassan I. Sayed Ahmed [1], Heba K. Aslan [1,2], Young-Im Cho [3,*] and Mohamed S. Abdallah [1,4,*]

[1] Informatics Department, Electronics Research Institute (ERI), Cairo 11843, Egypt; gelkabbany@eri.sci.eg (G.F.E.); hassanibrsayed@eri.sci.eg (H.I.S.A.); haslan@nu.edu.eg (H.K.A.)
[2] Center of Informatics Science, Faculty of Information Technology and Computer Science, Nile University, Giza 12588, Egypt
[3] Department of Computer Engineering, Gachon University, Seongnam 13415, Republic of Korea
[4] DeltaX Co., Ltd., AI Laboratory, 3F, 24 Namdaemun-ro 9-gil, Jung-gu, Seoul 04522, Republic of Korea
* Correspondence: yicho@gachon.ac.kr (Y.-I.C.); sameer@gachon.ac.kr (M.S.A.)

Abstract: The Nth-degree Truncated polynomial Ring Unit (NTRU) is one of the famous post-quantum cryptographic algorithms. Researchers consider NTRU to be the most important parameterized family of lattice-based public key cryptosystems that has been established to the IEEE P1363 standards. Lattice-based protocols necessitate operations on large vectors, which makes parallel computing one of the appropriate solutions to speed it up. NTRUEncrypt operations contain a large amount of data that requires many repetitive arithmetic operations. These operations make it a strong candidate to take advantage of the high degree of parallelism. The main costly operation that is repeated in all NTRU algorithm steps is polynomial multiplication. In this work, a Parallel Post-Quantum NTRUEncrypt algorithm called PPQNTRUEncrypt is proposed. This algorithm exploits the capabilities of parallel computing to accelerate the NTRUEncrypt algorithm. Both analytical and Apache Spark simulation models are used. The proposed algorithm enhanced the NTRUEncrypt algorithm by approximately 49.5%, 74.5%, 87.6%, 92.5%, 93.4%, and 94.5%, assuming that the number of processing elements is 2, 4, 8, 12, 16, and 20 respectively.

Keywords: post-quantum cryptography; NTRU; parallel computing; Apache Spark

Citation: Elkabbany, G.F.; Ahmed, H.I.S.; Aslan, H.K.; Cho, Y.-I.; Abdallah, M.S. Lightweight Computational Complexity Stepping Up the NTRU Post-Quantum Algorithm Using Parallel Computing. *Symmetry* **2024**, *16*, 12. https://doi.org/10.3390/sym16010012

Academic Editors: Marcio Basilio, Valdecy Pereira and Theodore E. Simos

Received: 1 November 2023
Revised: 12 December 2023
Accepted: 19 December 2023
Published: 21 December 2023

1. Introduction

Cryptography is a major tool for providing the confidentiality of data/information. Cryptography has evolved from the Caesar cipher, the well-known classical cipher, to the modern cryptosystems, which are based on modular computations, to quantum computing. Modern cryptosystems, which are based on modular arithmetic, are subject to a major threat if quantum computers are invented. This is because quantum computers will be able to solve computationally hard problems, which represent the strength of the modular arithmetic ciphers in deterministic time. For example, the Rivest Shamir Adleman (RSA) algorithm works by generating pairs of public and private keys that are mathematically related. Therefore, any attempt to guess the private key using the public key by executing a brute-force attack would require thousands of years using ordinary computers. However, quantum computers could easily guess the private key from the public key. Digital signature and encryption algorithms based on public key cryptography will no longer be strong enough to preserve information secrecy in the quantum computer era [1].

On the other hand, both the symmetric algorithms and hash functions are partially threatened by quantum computers. The throughput of these algorithms is doubled on quantum algorithms compared to classical computers. Therefore, doubling the key lengths and hash sizes will result in safe algorithms [2]. This raises the need to invent new algorithms (post-quantum algorithms) that can resist both classical and quantum computing attacks.

In quantum algorithmic development, there are two ground-breaking algorithms that enable the breaking of public key algorithms. In 1994, Shor proposed an efficient algorithm based on quantum computation [3,4] for solving discrete logarithm and integer factorization problems, which is the basis of the security of Elliptic Curve Cryptography (ECC) and RSA. In addition, in 1996, Grover proposed an $O(\sqrt{N})$-time quantum algorithm for functions with N-bit domains [5]. This algorithm, after being implemented on quantum computers can easily break symmetric key algorithms. To overcome this attack, key sizes must be doubled to have a similar level of security to the classical computers.

Post-quantum cryptography (PQC) is the science of introducing new encryption algorithms that are secure against both classical and quantum computers. These algorithms can be divided into five approaches [6–8]: code-based [9], hash-based [10], multivariate [11], lattice-based [12,13], and isogeny-based [14] solutions. As will be discussed later, the lattice-based has the most contributions among the standardized post-quantum algorithms. An example of lattice-based post-quantum algorithms is NTRU, which was developed in 1996. Contrary to ECC and RSA, NTRU can withstand attacks against quantum computers. In addition, it has higher performance compared to both RSA and ECC [15]. The inventors of NTRU declared that it is two hundred faster than other public key encryption algorithms. It requires only $O(N^2)$ time to encrypt/decrypt a message of length N, while RSA takes $O(N^3)$ time. In addition, it has the following characteristics: it can be easily implemented in software or hardware, and the storage requirement is small. This makes it suitable for use in cellular phones, smart cards, and Radio-Frequency Identification Devices (RFIDs).

The main difficulty of NTRU is the computational complexity of polynomial multiplication and vector multiplication. Most of the researchers used traditional algorithms such as NTT, Montgomery, CRT, and Karatsuba to decrease the complexity of multiplication [16]. With parallel implementation, the system can generate many candidates that cooperate to execute different polynomial multiplications simultaneously and hence mitigate the performance slowdown. Investigating the parallelization of polynomial multiplication on multi-core processors is considered one of the most important research trends at the present time [17].

In this work, a Parallel Post-Quantum NTRUEncrypt algorithm called PPQNTRUEncrypt is proposed. PPQNTRUEncrypt algorithm takes advantage of parallel computation to expedite the NTRUEncrypt algorithm. In comparison, the analytic model is derived as a scientific theoretical study to provide understanding and approach to perform results. A simulation test must be performed to validate the analytical results and to understand the reality of the method in the way the system behaves. To measure the achievement of the PPQNTRUEncrypt algorithm, two platforms are used. Whereas the first experiment computes the system performance analytically, the second one calculates it by implementing the proposed algorithm on the Apache Spark framework. The effect of both system size and problem size on the conductance of the PPQNTRUEncrypt algorithm is considered.

The rest of the paper is organized as follows: In Section 2, an introduction to the foremost concepts of the famous post-quantum cryptographic (PQC) schemes and the important details of the NTRU lattice-based public encryption algorithm are found. In Section 3, The Parallel Post-Quantum NTRUEncrypt algorithm (PPQNTRUEncrypt) is discussed in detail. Moreover, Section 4 presents the performance evaluation of the PPQNTRUEncrypt algorithm and summarizes its important results, showing the advantages of using parallel computing. Finally, in Section 5, the conclusions are discussed.

2. Literature Review

In the following subsections, we give a brief description of different post-quantum cryptographic schemes, NTRU Lattice-based encryption schemes, and a comparison between NTRU, RSA, and ECC. Then, we mention different parallel computing architectures used in cryptography.

2.1. Post-Quantum Cryptographic (PQC) Schemes

In the following paragraphs, a brief description of different post-post-quantum cryptographic schemes is illustrated.

- **Hash-based cryptography**

Hash-based cryptography aims to develop digital signature schemes that are inherent in their security from the security of cryptographic hash functions, e.g., SHA-3. These schemes require fewer security assumptions than schemes based on number theory, such as RSA and DSA. For more information about hash-based cryptography, the reader could refer to [18]. One algorithm of this category is the Sphimcs+ algorithm [19], which was chosen as an alternative solution in the result of the third round of the National Institute of Standards and Technology (NIST) standardization process.

- **Code-based cryptography**

The security of code-based cryptography [20] depends on the toughness of problems from coding theory such as syndrome decoding (SD) and learning parity with noise (LPN). They depend on error-correcting codes to build a one-way function. Their security depends on the difficulty of decoding a message that includes random errors and recovering the code structure. An example of this category is the classic McEliece code-based encryption scheme [10], which was chosen as a qualifier scheme as the result of the third round of the NIST standardization process.

- **Multivariate cryptography**

The security of multivariate-based cryptography is based on the difficulty of solving multivariate systems of equations over a finite field (F). The following algorithms are examples of multivariate cryptography: Rainbow, TTS, or MPKC schemes. More information about multivariate cryptography can be found in [21]. Two multivariate-based signature schemes are selected in the result of the third round of NIST: Rainbow [22] is listed as a qualifier, and GeMSS [23] is listed as an alternative scheme.

- **Lattice-based cryptography**

Lattice-based cryptography is considered one of the most promising post-quantum encryption algorithms for the following reasons. The security is guaranteed since it relies on well-known lattice problems such as the ring learning with errors (RLWE) problem and shortest vector problem (SVP) [24]. In addition, it allows powerful cryptographic primitives, such as fully homomorphic encryption and functional encryption [25]. Furthermore, new lattice-based cryptographic schemes have become efficient (key exchange protocol NewHope) [26], and a signature scheme BLISS [27]. Lattice-based cryptography has the most contributions in the third-round outcome of the NIST standardization process. Kyber [28], NTRU [29], and SABER [30] lattice-based encryption schemes are selected as the qualifiers' schemes. NTRUprime [31] is in the alternative finalist lattice-based encryption scheme. Dilithium [32] and Falcon [33] lattice-based signature schemes are also finalist schemes.

- **Isogeny-based cryptography**

Isogeny-based cryptography uses particular relations between abelian varieties over finite fields as its main component. It is characterized by relatively small keys. These schemes rely on supersingular elliptic curve isogenies [14], which could resist quantum attacks. They are used for digital signatures or key exchange, such as the supersingular Isogeny Diffie–Hellman (SIDH) scheme [34]. SIKE [35], which is an isogeny-based encryption scheme, is in the alternative list of NIST third-round outcomes.

NIST is one of the main institutions that are interested in the standardization of post-quantum algorithms [36]. In 2016, the standardization process began to choose candidates for post-quantum cryptographic algorithms. The process now is in the third round after two rounds of evaluations. By 2024, it is expected to announce the winners. In 2016, NIST

published a report on PQC [37], expecting that a quantum computer would be built by 2030. It can break 2000-bit RSA in a few hours. The third-round qualifiers contain four public-key encryption algorithms and three digital signature algorithms. In addition, it contains five and four alternative candidates for public key encryption and digital signature algorithms, respectively. These algorithms are listed in Table 1. From the table, lattice-based post-quantum technology has the most contributions among the list of standardized post-quantum algorithms.

Table 1. NIST Standardization Efforts [38].

Post-Quantum Algorithm Type	Third Round Finalist	Technology	Alternate Candidates	Technology
Public Key Encryption/Key Encapsulation Mechanisms	Classic McEliece	Code	BIKE	Code
	CRYSTALS KYBER	Lattice	FrodoKEM	Lattice
	NTRU	Lattice	HQC	Code
	SABER	Lattice	SIKE	Isogeny
			NTRU Prime	Lattice
Digital Signature Algorithms	CRYSTALS-DILITHIUM	Lattice	GeMSS	Multivariate Polynomial
	FALCON	Lattice	PICNIC	Other
	RAINBOW	Multivariate Polynomial	SPHIMCS+	RAINBOW

In the literature, numerous papers have undertaken comparisons among various types of post-quantum cryptography [39–41]. These studies elucidate the main characteristics, advantages, and disadvantages of these cryptographic approaches. In this present paper, our focus is on key encapsulation mechanism algorithms. Accordingly, our interest lies in comparing the following mechanisms: code-based, lattice-based, and isogeny-based cryptographic algorithms. Table 2 presents the principal characteristics of these algorithms. However, isogeny algorithms are distinguished by their small key size, rendering them suitable for low-constrained devices, yet they exhibit a slower operation speed. Conversely, both code-based and lattice-based algorithms demonstrate a rapid operation speed at the expense of larger key sizes. Lattice algorithms have an advantage over code-based algorithms due to their shorter key size, making them more appropriate for low-constrained devices. Furthermore, the algorithms employed in lattice-based cryptosystems are distinguished by their simplicity, efficiency, and high parallelizability [42]. The cryptographic protocols grounded in lattice structures have demonstrated robust security, relying on established lattice problems such as the shortest vector problem (SVP) and the learning with errors problem (LWE) [42]. These factors collectively position lattice-based cryptographic schemes as a dynamic and leading area of exploration in post-quantum cryptography considered the most prominent and promising.

In the literature, numerous papers have conducted comparisons among lattice-based post-quantum cryptographic algorithms [42–44]. From these papers, we extract information for Table 3, which presents a comparative analysis of SABER, CRYSTALS-Kyber, NTRU, and FrodoKEM post-quantum algorithms. The comparison encompasses key size, CPU usage, operation speed, and ciphertext length. The table reveals that FrodoKEM exhibits the least favorable metric values among all other algorithms. Conversely, the SABER algorithm demonstrates moderate metric values. When comparing NTRU with CRYSTALS-KYBER, they exhibit similar operation speeds and ciphertext lengths. However, CRYSTALS-KYBER outperforms NTRU in terms of CPU usage, while NRTU excels in small key sizes compared to CRYSTAL-KYBER. Notably, a key advantage of NTRU is its commercial

implementation [42], indicating a well-established understanding of its security since its initial publication in 1996.

Table 2. Comparison among Lattice-based, Code-based, and Isogeny-based Post-Quantum Key Encapsulation Mechanisms.

Post-Quantum Algorithm Type	Key Size	Operation Speed	Algorithms
Lattice	Medium	Fast	CRYSTALS-KYBER NTRU SABER FrodoKEM NTRU Prime
Code	Large	Fast	Classic McEliece BIKE HQC
Isogeny	Small	Slow	SIKE

Table 3. Comparison among SABER, CRYSTALS-KYBER, NTRU, and FrodoKEM.

Post-Quantum Algorithm	Key Size	CPU Usage	Operation Speed	Ciphertext Length
SABER	Medium	Medium	Fast	Medium
CRYSTALS-KYBER	Medium	Light	Fast	Small
NTRU	Small	High	Fast	Small
FrodoKEM	Large	High	Slow	Large

In the next subsection, a description of the NTRU public encryption algorithm is detailed.

2.2. NTRU Lattice-Based Public Encryption Algorithm

NTRU uses three parameters (N, p, and q). N is used for modulo operation, the polynomial degree to be less than N. To enhance security, it is preferred that N be a prime number. On the other hand, p and q are required to be co-prime, and they are used to minimize the polynomial coefficients. It should be noted that p must be less than q; in addition, q must be less than N. The recommended parameters of NTRU for different security levels are shown in Table 4.

The NTRU encryption algorithm has three modules, namely key generation, message encryption, and message decryption. These modules are described below.

Table 4. Recommended NTRU Parameters [45].

(a) NTRU Parameters		
Parameter	**Description**	**Knowledge**
p	Small Modulus to reduce coefficients	Public
q	Large Modulus to reduce coefficients	Public
N	Number. of coefficients in a polynomial	Public
f	Polynomial private key	Private
g	Used in public key generation	Private
h	Polynomial public key	Public
r	Random blinding polynomial	Private
d_f	Number of coefficients with value 1 in polynomial f	Public

Table 4. *Cont.*

(b) Security Levels for p, q, and N				
	Security levels			
Parameters	**Highest**	**High**	**Standard**	**Moderate**
P	3	3	3	3
q	256	128	128	128
N	503	347	251	167

2.2.1. Key Generation

The first step is to choose two random polynomials, which are used to calculate the public key (f(y) and g(y)). In f(y), the number of '1 s' coefficients must be greater than the other coefficients. In g(y), the number of both coefficients '1' and '−1' must be equal. The next step is to compute the inverse of f(y) modulo q and p with the given properties:

$$f(y) * fq = 1 \bmod q \quad (1)$$

and

$$f(y) * fp = 1 \bmod p \quad (2)$$

In some cases, f(y) does not have an inverse modulo q and p. Thus, another polynomial f(y) must be randomly chosen to calculate f_q and f_p. The private key is represented by f(y) and f_p, while the public key P_k is calculated as follows:

$$Pk = p * (fq * g(y)) \bmod q \quad (3)$$

2.2.2. Encryption

To encrypt a message Msg, it is, first, converted into polynomial form with coefficients in the range of $[(p-1)/2, (p-1)/2]$, and the degree must be less than N. A polynomial r(y) is also selected with equal positive and negative coefficients. The encrypted message (Msge) is calculated as:

$$Msge = (r(y) * Pk + Msg)) \bmod q \quad (4)$$

2.2.3. Decryption

The following steps are performed to restore Msg. First, Msge is multiplied by the private key f(y) as follows:

$$M_{sgd1} = f(y) * Msge \bmod q \quad (5)$$

Then, modulo q is applied on M_{sgd1} to reduce the coefficients of the polynomial to be in the range of $-p/2$ and $p/2$ as follows:

$$Msgd2 = M_{sgd1} \bmod q \quad (6)$$

The final step is to compute the original message M_{sg} using f_p as follows:

$$Msg = fp * M_{sgd2} \bmod p \quad (7)$$

It should be noted that all modules used to implement the NTRU algorithm utilize costly polynomial multiplication. In the next subsection, the performance comparison of NTRU, RSA, and ECC will be illustrated.

2.2.4. NTRU Security Analysis

NTRU is grounded in the shortest vector problem within a lattice, employing polynomial multiplication. Considered resilient to quantum computer attacks, particularly Shor's attack [43], NTRU stands out as a prominent post-quantum cryptosystem. It is designed to resist various types of attacks, including lattice basis reduction, meet-in-the-middle attacks, and chosen ciphertext attacks [42]. Ongoing cryptanalysis efforts continually examine

NTRU, leading to the development of various attack strategies [42]. Some detailed attacks, as per [42], include:

Brute-Force Attack: This approach entails testing all possible values of the private key until the correct one is identified. While generally impractical for NTRU due to its large key space.

Key Recovery Attack: This attack exploits vulnerabilities in the key-generation process of NTRU. For instance, if the arbitrary number generator used for creating the confidential key is found to be fragile, a malicious user may recover the private key.

Side-Channel Attack: All lattice-based algorithms suffer from side-channel attacks. Side-channel attacks encompass timing attacks, power analysis attacks, and fault attacks. Physical accessibility to the device running the implementation is required for such attacks. It has to be noted that the parallel implementation of NTRU inherits the susceptibility of side-channel attacks. Researchers employ various techniques, including parameter selection, randomization, and error-correcting codes, to safeguard NTRU against these threats and prevent the leakage of secret data and information.

2.3. Comparison between NTRU, RSA, and ECC

In this subsection, a comparison between NTRU, RSA, and ECC public encryption algorithms is given. Table 5 shows the security level bits for these algorithms. The table shows that the security level of NTRU-251 is comparable with RSA, which has a key length of 1024 bits, and ECC, which has a key length of 160. Table 6 shows that NTRU is considerably faster than both RSA and ECC. In addition to being faster than RSA and ECC, the NTRU has the advantage of being a post-quantum encryption algorithm. Finally, Table 7 shows that both the encryption and decryption modules of NTRU consume more time than the key generation module. Therefore, decreasing the time consumed in these modules will enhance the algorithm's performance. As noticed from the algorithm's description, the costliest operation is polynomial multiplication. In the present paper, we present a parallel design to achieve a higher encryption/decryption throughput.

Table 5. PKC Standard Comparison.

Security Levels (Bits)	RSA	ECC	NTRU-N
80	RSA-1024	ECC 160-223	NTRU-251
112	RSA-2048	ECC 224-255	NTRU-347
128	RSA-3072	ECC 256-383	NTRU-397
192	RSA-7680	ECC 384-511	NTRU-587
256	RSA-15360	ECC 512	NTRU-787

Table 6. A Comparison of NTRUEncrypt, RSA, and the Elliptic Curves Cryptosystem on an 800 MHz Pentium III Computer [46].

	NTRU 251	RSA 1024	ECC 163
Public key (bits)	2008	1024	164
Private key (bits)	251	1024	163
Plaintext block (bits)	160	702	163
Ciphertext block (bits)	2008	1024	163
Encryption speed (blocks/sec) Encryption speed (Mbits/sec)	22,727 3.6	1280 0.9	458 0.075
Encryption Speed (blocks/sec) Encryption speed (Mbits/sec)	10,869 1.7	110 0.077	702 0.11

Table 7. A Comparison of Key Generation, Message Encryption, and Message Decryption for both NTRUEncrypt and RSA [47].

	Key Generations	Encryption	Decryption
NTRU167	90 msecs	21 s	20 s
RSA512	5 s	3.5 s	34 s
NTRU263	150 msecs	31 s	30 s
RSA1024	10 s	5.8 s	100 s
NTRU503	420 msecs	56 s	55 s
RSA2048	50 s	10.8 s	345 s

2.4. Parallel Computing in Cryptography

Parallelism is a valuable technique that enhances the efficiency and performance of cryptographic systems by enabling more effective execution of cryptographic operations. It is well recognized that parallel computing plays a pivotal role in cryptography, contributing to improved efficiency, speed, and scalability. Recent applications of parallel computing in cryptography include parallelized key generation, parallel encryption and decryption, parallel password cracking, parallel hash function computation, and more. Furthermore, parallel processing can be implemented in various post-quantum cryptography algorithms, such as parallelized lattice-based cryptography, parallel implementations of hash-based signatures, parallel strategies for multivariate cryptography, parallel execution of code-based cryptography, and others. It is crucial to acknowledge that the choice of specific parallelization techniques may vary depending on the unique characteristics and requirements of each cryptographic algorithm. Within this subsection, several instances of recent cryptographic applications that leverage parallel implementation are provided as examples [48–50]. To elaborate further on specific examples:

- Parallelizing Encryption and Decryption Protocols: Parallel computing has been utilized to accelerate the encryption and decryption processes in various cryptographic algorithms. In [48], Aldahdooh presents a parallel implementation and analysis of several encryption algorithms (such as Advanced Encryption Standard (AES), Blowfish, Twofish, Data Encryption Standard (DES), Triple DES, and Serpent). Parallel processing is employed to enhance efficiency, and the experiments demonstrate significant improvement compared to sequential implementations. The algorithms are evaluated based on encryption time and speedup.
- Parallelizing Quantum Key Distribution (QKD): QKD is a fundamental application of quantum cryptography. Wu et al., in their work [49], propose a parallel simulation with an optimized scheme for QKD networks, specifically addressing network partitioning.
- Parallelizing Lattice-based Cryptography: Wan et al., in their work [50], presented an NTT box based on the NVIDIA AI accelerator, Tensor Core. After that, they presented a high-performance implementation of CRYSTALS-Kyber with the suggested NTT box and achieved considerable performance improvement. This work illustrated the tremendous potential of Tensor Core in LBC acceleration.

The above examples serve to emphasize the potential advantages of parallel implementations in enhancing the speed and efficiency of cryptographic systems. As previously mentioned in the previous sections, this research is concerned with accelerating the polynomial multiplication operation of the NTRU Lattice-based cryptography. The nature of the NTRUEncrypt operations makes it one of the appropriate applications for the use of data parallelism [51]. Several researchers have discussed the great potential of using the ability of parallel computing to speed up the NTRUEncrypt algorithm. However, most of the researchers used GPUs in their work due to their high speed compared to CPUs [52,53]. The GPU's high cost has forced others to use CPUs (with the concept of high-speed computing), especially in the case of small devices such as smartphones, which need high speed in security and privacy operations without having to raise their prices. In this work, parallel

computing is used to uplift the performance of the NTRUEncrypt algorithm using multi-processing architecture. The proposed Parallel Post-Quantum NTRUEncrypt algorithm called PPQNTRUEncrypt helps the system to generate many nominees that can cooperate in executing different polynomial multiplications simultaneously and hence amelioration the NTRUEncrypt algorithm behavior. The next section discusses the PPQNTRUEncrypt in detail.

3. The Proposed Parallel Post-Quantum NTRU Encrypt Algorithm (PPQNTRUEncrypt)

With the development of high-performance technologies, different cryptographic schemes have been presented to protect information. Nowadays, Post-Quantum Cryptography (PQC) protocols have been recommended to confront quantum algorithms that threaten public key cryptography. Post-Quantum NTRUEncrypt is one of the well-known PQC platforms. By using the characteristics of parallel computing, a Parallel Post-Quantum NTRUEncrypt algorithm called PPQNTRUEncrypt is proposed. PPQNTRUEncrypt is an amended NTRUEncrypt algorithm that performs parallelization of the large computational NTRUEncrypt operations (such as matrix generation, matrix arithmetic, and polynomial-based operations) to accelerate the NTRUEncrypt algorithm.

Since the computational complexity of polynomial multiplication is the bottleneck of NTRU, by using parallel computation, the polynomial multiplication operations must be accelerated to improve the performance of the NTRU. To speed up this protocol, a bi-level parallel design is proposed, which is based on two levels of parallelism: the first level is based on computing different polynomial multiplication operations in parallel, while the second level is used to enhance the execution time of each polynomial multiplication operation.

3.1. Parallel NTRUEncrypt Model

Lattice-based protocols need different operations that must be executed on large vectors; these operations are suitable for parallel computing. The PPQNTRUEncrypt algorithm is designed to implement both encryption and decryption steps in parallel computer architecture. In addition, it is working on balancing the load between different processing elements (PEs).

Assumptions

In general, for parallel models, both the machine structure and the description of the application/operation/job are the two components that must be considered.

- **The Structure of the Parallel Machine:**

A distributed memory parallel structure with "M" nodes/processing elements (PEs) {PE_0, PE_1, . . ., PE_i, . . ., PE_{M-1}} is considered. The inter-processor communication between different PEs is done via message passing). PE_0 is suggested as the master PE and is accountable for the system control. All other PEs, together with the master PE_0, contribute to the task execution. An Intel® Core™ i3-2328M CPU@ 2.20 GHz 2.20 GHz (Intel, Santa Clara, CA, USA) machine is used to execute the NTRU sequentially. For both analytical and Apache Spark simulation models and to ensure fairness in the comparison, each node/PE of this simulated parallel machine has the same specifications as the aforementioned machine.

- **Operation description: Polynomial multiplication:**

In NTRUEncrypt, each encryption and decryption operation has more than one polynomial multiplication operation. Polynomial multiplication is a process for multiplying two or more polynomials. In this operation, the terms of the 1st polynomial are multiplied by the 2nd polynomial to get the resultant polynomial. To multiply polynomials, the coefficient is multiplied with a coefficient, and the variable is multiplied with a variable using exponent rules. Each polynomial multiplication operation can be divided into small processes that can be executed in parallel, as shown in Figure 1, (Appendix A has extra information and details about the figures).

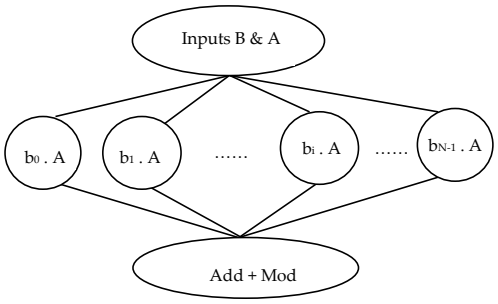


Figure 1. Directed Acyclic Graph (DAG) of the polynomial multiplication operation.

The function that is needed to be parallelized is the polynomial multiplication (Z.Y) where:

$$Y = y_0 + y_1x + ... + y_{N-2}x^{N-2} + y_{N-1}x^{N-1} \quad (8)$$

$$Z = z_0 + z_1x + ... + z_{N-2}x^{N-2} + z_{N-1}x^{N-1} \quad (9)$$

To multiply Z.Y

$$Y.Z = (z_0.Y) + (z_1x.Y) + ... + (z_{N-2}x^{N-2}.Y) + (z_{N-1}x^{N-1}.Y). \quad (10)$$

3.2. Parallelizing the NTRU Encryption and Decryption Models

In the current paper, the authors suppose that "*N*" is the number of tasks that will be executed in parallel (*N*: the coefficients of the polynomial multiplication operation as shown in Equations (8)–(10)). Moreover, it is assumed that all the "*M*" PEs are homogenous. There are two levels of parallelism:

3.2.1. Level One: Coarse-Grained Level of Parallelization ($N \geq M$)

In case the number of tasks "N" is greater than or equal to the number of PEs "M", each PE will execute "N/M" tasks. The total sequential time "T_s" that is needed for one polynomial multiplication is calculated as follows:

$$T_s = N * T_i + t_{add} + t_{mod} \quad (11)$$

where t_{add} is the time needed for the addition operation, and t_{mod} is the time needed for modular operation.

Moreover, the overall parallel (execution) time of the problem when using the parallel system "T_{par}" is calculated using the following equation:

$$T_{par} = T_{comp} + T_{ov} \quad (12)$$

where the total computational time T_{comp} is calculated as:

$$T_{comp} = \max\{T_{comp}(PE_i)\}\ 0 \leq i \leq M - 1 \quad (13)$$

$0 \leq i \leq M - 1$

In addition, the overhead time "T_{ov}" can be computed using the following equation:

$$T_{ov} = T_c + T_g + T_{app} \quad (14)$$

where:

T_c: The duration consumed in the memory operations.

(accessing, contention, and synchronization).

T_g: The time spent in data exchange between different PEs.

T_{app}: The time wasted due to application dependency.

In the case of "*M*" divides "*N*" (N/M = integer), each PE executes the same number of tasks, and the total execution time T_{par} is calculated as follows:

$$T_{par} = [(N/M) * T_i + t_{add} + t_{mod}] + T_{ov} \quad (15)$$

On the other hand, using Equation (13), when "M" does not divide "N" (N/M $\neq$ integer). For the first $(N - \lfloor N/M \rfloor \times M)$ PEs, each PE calculates $\lceil N/M \rceil$ tasks, and for the remaining PEs, each PE executes $\lfloor N/M \rfloor$ tasks:

$$T_{comp} = [((Quotient\ (N/M))+1) * T_i + t_{add} + t_{mod}] \quad (16)$$

Using Equation (15), T_{par} is given by:

$$T_{par} = [((Quotient\ (N/M))+1) * T_i + t_{add} + t_{mod}] + T_{ov} \quad (17)$$

3.2.2. Level Two: Medium-Grained Level of Parallelism (N < M)

This level is also called load balancing. In case "N" is lesser than "M", some PEs will be inactive, which leads to load imbalance. To solve this problem, the load is reallocated between PEs, and then more than one "PE" cooperates to execute one task. This solution is called load balancing (LB). LB techniques are categorized as centralized or decentralized/distributed techniques [54,55]. In centralized techniques, the global load information is collected at a master PE, which controls the load redistribution based on the information that is exchanged with other PEs. Furthermore, in distributed techniques, the balancing protocol is imitation in every "PE," and the immigration process could be started by any "PE". In NTRUEncryption, every multiplication operation is split into small tasks and then executed by more than one PE. Unfortunately, this may lead to communication overhead due to the need for data exchange. When the number of iterations increases, resorting to load balancing becomes necessary. In this case, the decentralized load-balancing technique is appropriate to be used. Besides, using a cluster of PEs for each operation is the best choice to reduce this overhead and achieve load balancing.

A pseudo-code that illustrates the internal mechanism of the PPQNTRUEncrypt algorithm and the specific implementation of parallel processing of two polynomial multiplications is given in Algorithm 1:

Algorithm 1. Internal mechanism of the PPQNTRUEncrypt and Parallel processing of two polynomial multiplications

Assume that:
There are "M" processing elements (PEs) {PE_0, PE_1, …, PE_i, …, PE_{M-1}}

- PE_0 is the master PE and is accountable for the system control
- From PE_1 to PE_{M-1}, together with the master, contributes to the task execution

Input: NTRU parameters (Polynomial-Y, Polynomial-Z)

1. Divided each Polynomial Multiplication operation into "N" homogenous tasks
2. if N $\geq$ M

```
if N/M = integer
        For loop (1 to N)
            For loop (1 to M)
                Each PE assigned a task
if N/M ≠ integer
        For loop (1 to (N − ⌊N/M⌋ × M))
                Each PE assigned ⌈N/M⌉ tasks
        For loop ((N − ⌊N/M⌋ × M) to N)
                Each PE assigned ⌊N/M⌋ tasks
 if N < M
            Load balancing must be taken into account
```

3. Each PE sends its results to the master PE_0
4. PE_0 aggregates the data from other PEs multiplication operation
5. PE_0 calculates mod value of the aggregated values
6. Finally, PE_0 calculates the total value of the multiplication operation

Output: Multiplication (Y.Z)
These steps repeated for each Polynomial Multiplication operation

4. Methods and Experiments

Parallel computing stands out as an environment for computation-intensive applications. Performance is an important key factor in determining the achievement of any parallel system. Therefore, analyzing and evaluating the performance of the system is an imperative feature of parallel computing research. Experimental, analytical modeling, and simulation are the most widely used methods to evaluate the performance of parallel systems. Experimental models implement real or synthetic workloads (benchmarks) and measure their performance on a real machine. Analytical models are mathematical representations of the system; they are modeling modelling approaches that aim to provide an idea of system performance. Simulation models exploit computer resources to represent and imitate the behavior of the application in a controlled manner. While analytic analysis supports the proposed design theoretically, simulation analysis shows whether this design can be physically implementable or not. Therefore, both simulation and analytical results are needed [56,57]. This work justifies the need for an integrated model combining the advantages of both analytical and simulation models to evaluate the performance of parallel systems. Both analytical and simulation approaches are used to evaluate system performance.

To evaluate the PPQNTRUEncrypt model, various performance metrics such as parallel time, speedup, efficiency, and the degree of improvement, the improvement in the execution time, system scalability, and model validation are presented [58,59]:

- Parallel/execution time "T_{par}": The total time to complete executing the whole problem when using the parallel system, as shown in Equation (12).
- Speedup "S_P": The unit measuring the ability of the parallel system to speed up problem-solving compared with the serial/sequential one ($S_p = T_S/T_{par}$).

  From Equation (11), the total sequential time "T_s" is calculated as:

$$T_s = Z.Y = (N * T_i) + €ˆ \quad (18)$$

where $€ = t_{add} + t_{mod}$.

- Efficiency "E_P": The average participation of each PE ($E = Sp/M$)
- The improvement in the execution time, which is achieved through using "M" PEs, is given by the ratio. ratio = $\frac{T_s - T_{par}}{T_s}$

The authors assumed that the parameter set of the NTRU (N, q, p) = (503, 256, 3), (347, 128, 3), (251, 128, 3) and (167, 128, 3) are used. Table 8 shows the time needed to execute one polynomial multiplication sequentially with different values of N = 167, 251, 347, and 503 on the selected machine.

Table 8. Time to execute one polynomial multiplication sequentially (N = 503, 347, 251, and 167).

N	The Time to Multiply One Element (b_i.A)	The Time to Execute 100 × 100 Polynomial Multiplication
503	38.25	19,239.8 s
347	29.21	10,135.4 s
251	22.42	5627.4 s
167	16.53	2760.5 s

4.1. Parallel Computing Using the Analytical Model

An analytic model serves as an effective approach for the study and analysis of parallel systems. In Section 3, a layered parallel architecture was put forward, which effectively leverages the parallelism of the NTRU polynomial operations and provides robust support for them. Moreover, the layered architecture segregates the functionality of each layer, resulting in improved execution speed of the NTRU protocol. A detailed representation

of the proposed analytical model can be found in Equations (11)–(17). By applying these equations to varying values of "N and M", the NTRUEncrypt algorithm is enhanced. The outcomes of these evaluations will be thoroughly discussed in Section 4.3.

4.2. Parallel Computing Using Apache Spark Framework

Apache Spark is a widely utilized, quick, and efficient system for processing massive amounts of data in both business and academics. When it comes to high-performance big data analytics, it is regarded as the finest option. Spark Core uses YARN and Hadoop to distribute resources [60,61]. Additionally, it can retrieve data from the Hadoop Distributed File System (HDFS), which is used to read and store data and partition it into Resilient Distributed Datasets (RDDs). Due to its in-memory data processing, Spark performs substantially faster than other platforms like MapReduce (Google's parallel programming methodology that uses Hadoop). Additionally, Spark can enable batch, interactive, iterative, and streaming computations inside the same runtime as a general system [62]. Furthermore, Apache Spark is an open-source framework that processes data in real time. Regarding the aforementioned characteristics, Spark is advantageous for complicated applications that make use of many computing techniques, such as cryptography. Consequently, Spark is one of the data analytics frameworks that may be utilized to speed up the NTRU cryptography algorithm. To facilitate in-memory processing, Apache Spark uses RDD rather than the conventional read from and write to the disc. To make Spark a fault-tolerant framework without the need to write to the disc after each operation, RDD is a read-only data structure kept in memory. It also offers outstanding batch and stream processing abilities. There are numerous distinguishing qualities in Apache Spark:

Speed: Spark makes use of RDD, which provides memory-based data storage.

- Usability: Spark supports programming in any language, including Python, Java, Scala, and R.
- Advanced analytics are possible, including data streaming and more difficult analytics like machine learning and graph algorithms.
- Real-time stream processing takes advantage of the micro batching [63] method, in which data streams are processed as a collection of extremely small batches, which the Spark batch engine then treats as a regular task.

Apache Spark's architecture, which consists of a master node with a driver program in charge of calling an application's main program, is another characteristic. As seen in Figure 2, the cluster manager first handles resource distribution. The job is then divided up into several jobs and sent to the worker nodes. An RDD can be allocated among many workers and cached as soon as it is formed in the Spark context, as shown in Figure 2 [64]. These tasks are then returned to the Spark context. Tasks are carried out by the executor, and its lifespan is identical to Spark. The number of worker nodes must be increased in order for the jobs to be further separated into more logical chunks, which will improve system performance.

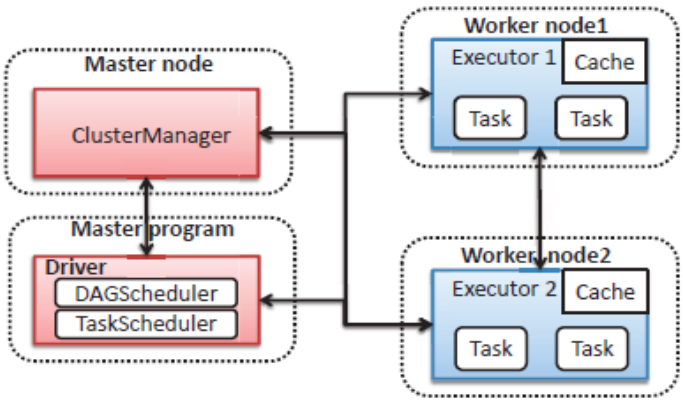


Figure 2. Spark framework architecture [65].

4.2.1. Programming Model and Core Techniques of Spark

The RDD abstraction model, which is an immutable collection of records divided among several computers, is the foundation upon which Spark is built, as was discussed in the previous subsection [66]. Through coarse-grained transformations, each RDD is created from data in external, reliable storage systems (like HDFS) or from other RDDs (including map, filter, and group by key). Each RDD's transformation information is logged in order to create a lineage dataset, which provides fault tolerance. When a node failure causes a data partition of an RDD to be lost, the RDD can recompute that partition using all of the information that was available when it was first computed from other RDDs. It's important to note that the transformation is a lazy operation that just defines a new RDD rather than instantly calculating it. Spark provides additional action operations, including count, collect, save, and reduce, which either return a value or export the RDD's data to an external storage system in order to start the computation of the RDD. Additionally, users can decide whether to persist an RDD's data in memory or on a disc. One driver master process is created for each Spark application, and this driver master process is in charge of scheduling tasks. Jobs, stages, and tasks are used in a hierarchical scheduling process, where stages are smaller groups of tasks separated from interdependent jobs, which mimic the map and reduce phases of a MapReduce job. It contains two schedulers, DAGScheduler and TaskScheduler. The DAGScheduler calculates a DAG (Directed Acyclic Graph, which is used to represent any task, job, or stage) of stages for a job and keeps track of both the materialized RDDs and stage outputs. The low-level scheduler TaskScheduler, on the other hand, is in charge of gathering and sending tasks from each stage to the cluster for execution.

Benefits of using the Apache Spark framework.

- Easy to use: Spark offers more than 80 high-level, simple operators (such as map, reduce, reduceByKey, and filter) that simplify the design of parallel computing frameworks without requiring users to consider the underlying complex parallel computing problems like data partitioning, task scheduling, and other issues. Additionally, by providing comparable APIs, Spark enables users to create user-defined functions in a variety of computer languages, including Java, Scala, and Python.
- Quicker than other frameworks like MapReduce: Spark has proven to be quicker than MapReduce in batch processing thanks to its in-memory computing.
- Support for computation: Spark is an integrated system that offers batch, interactive, iterative, and streaming processing options. Additionally, it offers a stack of high-level APIs and a sophisticated DAG execution engine for complex DAG applications. Additionally, it provides specialized tools for a variety of applications, such as Shark, Spark SQL, MLlib, and Graphx.
- Flexible running assistance. When running on YARN or Mesos, Spark can operate independently or share the cluster with other computing systems like MapReduce. Additionally, it offers APIs, for consumers to set up and use cloud computing (e.g., Amazon EC2). Additionally, it may allow access to a variety of data sources, such as HDFS, Tachyon, HBase, Cassandra, and Amazon S3.

4.2.2. Model Validation

Model validation is the crucial process of ensuring that the model fulfills its intended purpose. Validating a parallel model involves scrutinizing its performance, accuracy, and efficiency through various ways such as speedup, efficiency, scalability, comparative analysis, accuracy, sensitivity analysis, and more. Furthermore, analytical solutions can be leveraged for validation of the simulator model, especially when the theoretical foundation behind the problem is well-established, facilitating a direct comparison of the simulation results with the analytical solution.

To validate a computation problem in Apache Spark, various techniques can be applied depending on specific requirements. A common approach to validate results, such as polynomial multiplication in Spark, is to compare them with a known result. In cases

where the correct outcome is already known for a small-scale computation problem, cross-referencing the Spark-computed result with the predetermined one is standard practice. Manually executing the computation for a modest input serves as a benchmark to assess the accuracy of Spark's output [67].

4.3. Discussion of Results

Figures 3–6 show the system performance when using the PPQNTRUEncrypt with respect to the performance when using one machine. The above metrics are used to evaluate the system's performance. In addition, a comparison of the system's performance when using both the analytical model and Apache Spark is done to verify the applicability of the PPQNTRUEncrypt algorithm, as shown in Figures 3–6).

The experiments were conducted using Apache Spark, an open-source framework for big data processing, within the Databricks cloud platform. The configuration of Apache Spark involved optimizing various settings and parameters, with the specified spark_version being '10.4.x-scala2.12'. Key parameters, such as the number of worker nodes and executors, were configured. The Spark cluster was specifically configured with 'cluster_id': '0626-104331-u8vxnmc6'. Throughout the experiments, the number of nodes in the cluster was systematically varied, reaching up to 20 worker nodes, as observed in the results. The experiments assumed that encryption or decryption operations, based on the NTRU algorithm, were performed more than 10,000 times per iteration. This signifies an increase in the count of connected devices within the network. The findings underscore the necessity for a distributed processing framework like Apache Spark, demonstrating its scalability to accommodate a growing number of connected elements in the network.

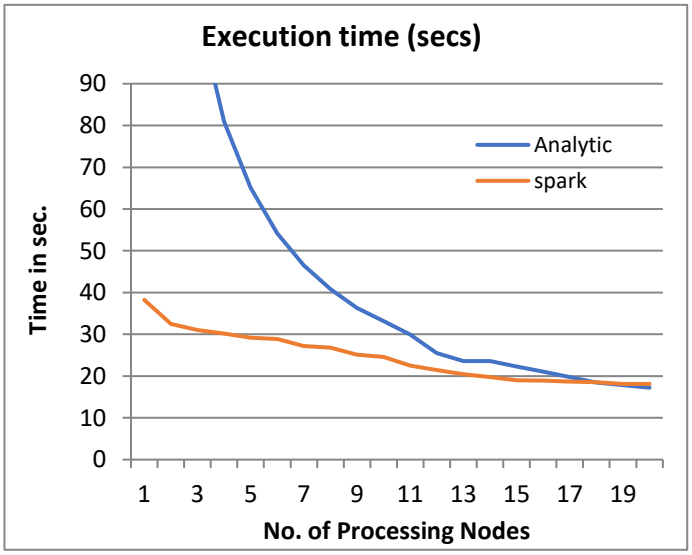
A pseudo-code of two polynomial multiplications is given in Algorithm 2:

Algorithm 2. Two polynomial multiplications

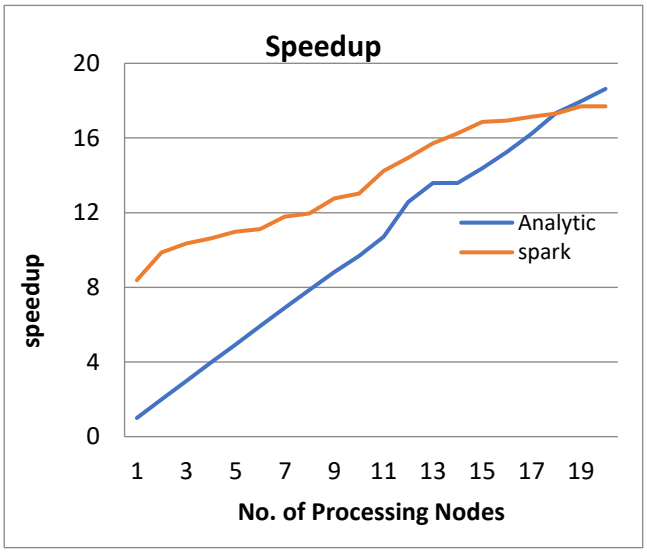
Input:
NTRU parameters (Polynomial1, Polynomial2)

```
1.  Spark Configurationconf = SparkConf().setAppName("PolynomialMultiplication")
2.  for numNodes in range(1, 21)
3.  Set the number of worker nodes in the configuration
4.  Conf.setMaster(f"local[{numNodes}]")
5.  Create Spark Context sc = SparkContext(conf)
6.  for numIterations in range(1, 10,001):
7.  Convert Arrays to RDDs
        poly1RDD = sc.parallelize([(exp, coeff) for coeff, exp in enumerate(polynomial1)])
        poly2RDD = sc.parallelize([(exp, coeff) for coeff, exp in enumerate(polynomial2)])
8.  productRDD = poly1RDD.cartesian(poly2RDD) \
        .map(lambda ((exp1, coeff1), (exp2, coeff2)): (exp1 + exp2, coeff1 * coeff2)) \
        .reduceByKey(lambda x, y: x + y) \
        .sortByKey()
9.  Collect and Format Results
10. product = " + ".join([f"{coeff}*x^{exp}" for exp, coeff in productRDD.collect()])
11. Print the Result for each set of worker nodes and iteration
12. Stop the Spark Context
```

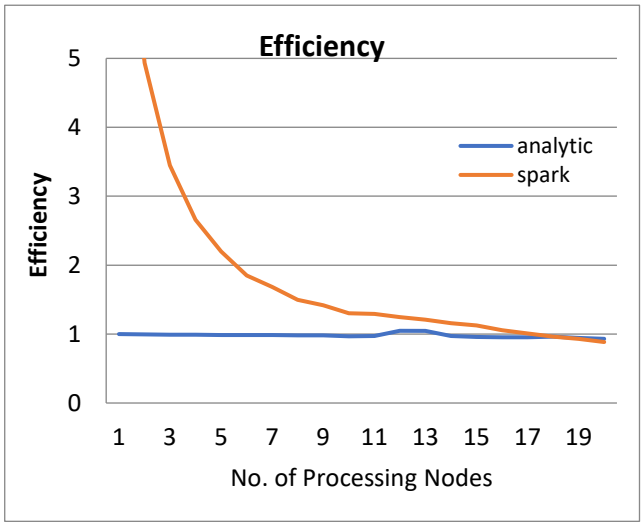
Output:
Multiplication



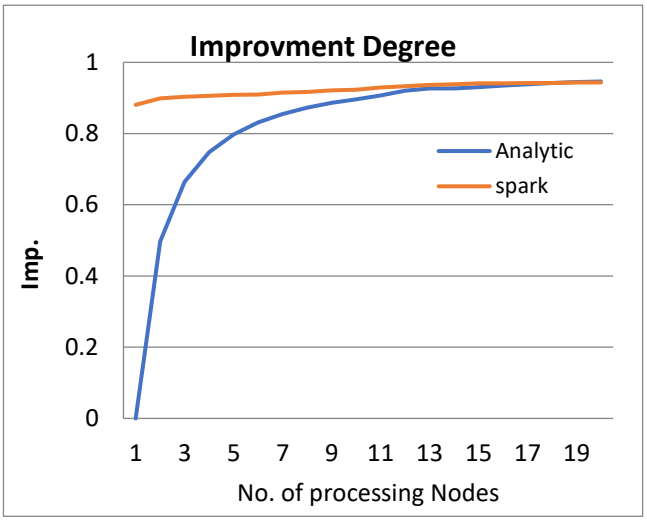
(**a**)



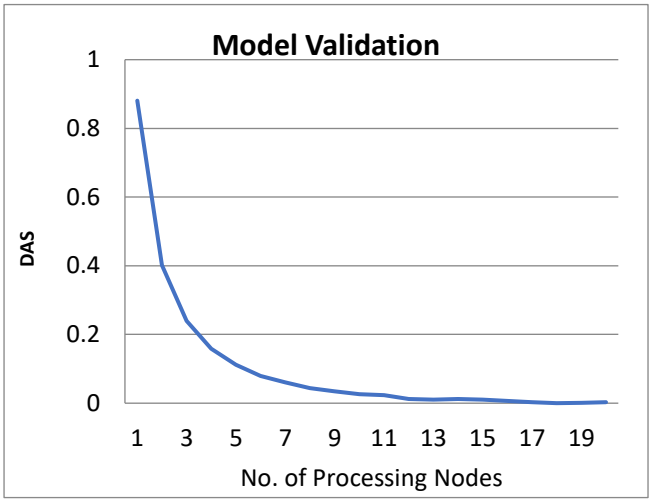
(**b**)



(**c**)

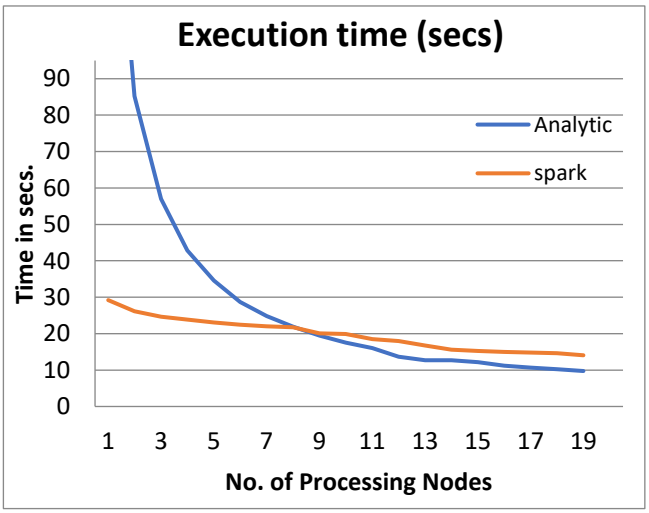


(**d**)

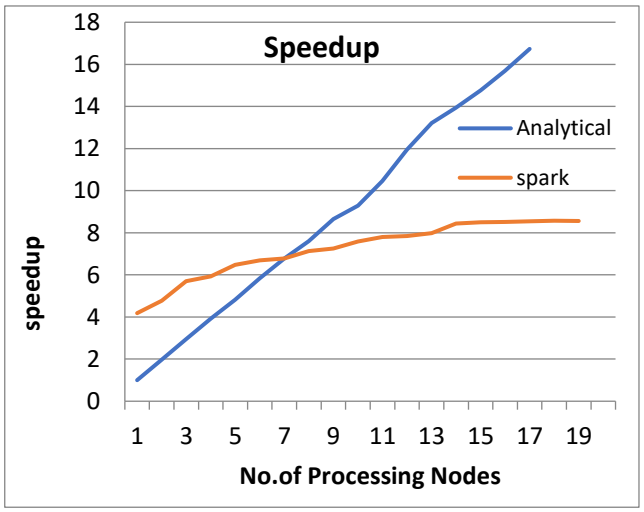


(**e**)

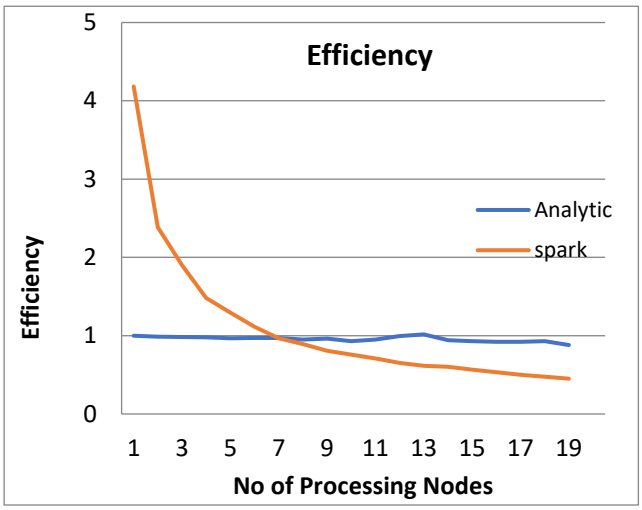
Figure 3. The Performance of the PPQNTRUEncrypt for p = 3, q = 256, and N = 503.



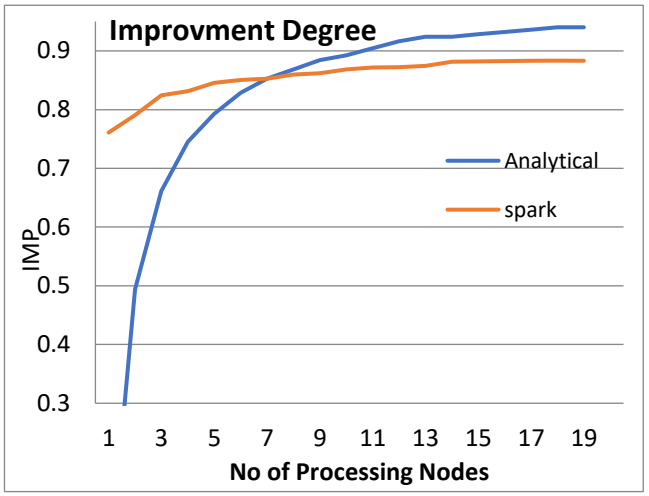
(**a**)



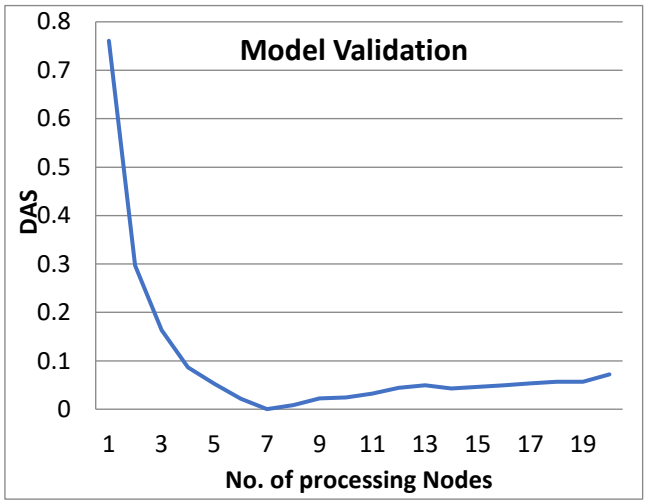
(**b**)



(**c**)

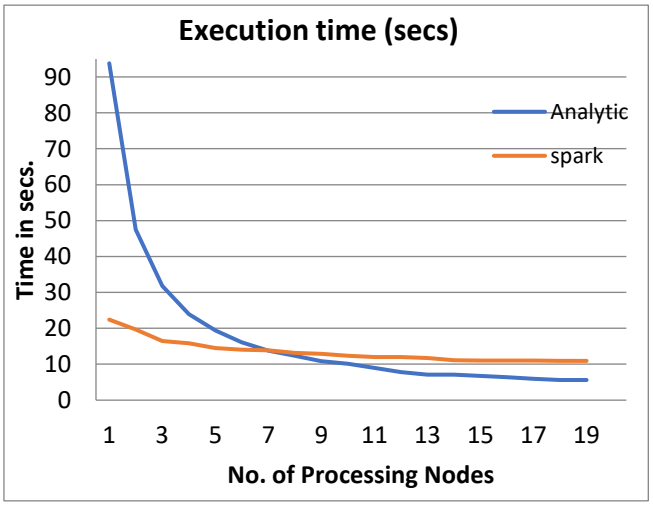


(**d**)

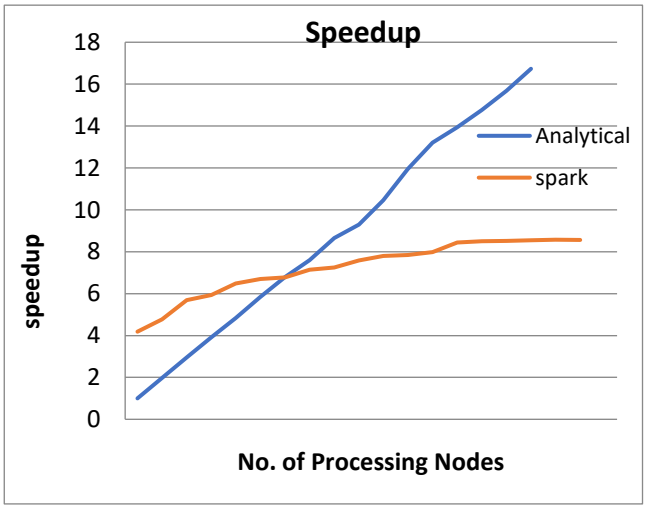


(**e**)

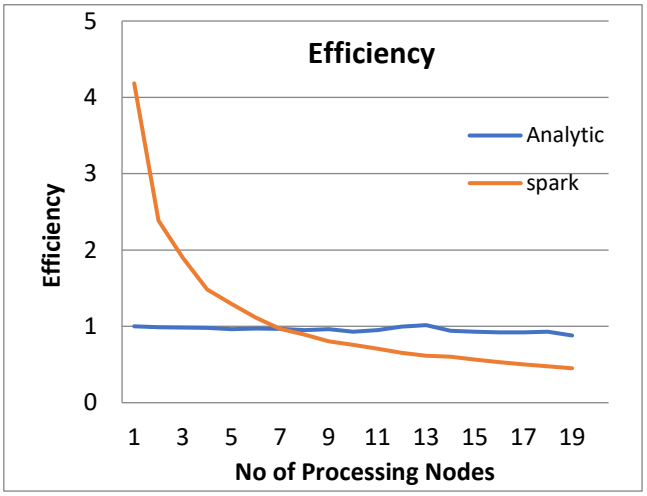
Figure 4. The Performance of the PPQNTRUEncrypt for p = 3, q = 128, and N = 347.



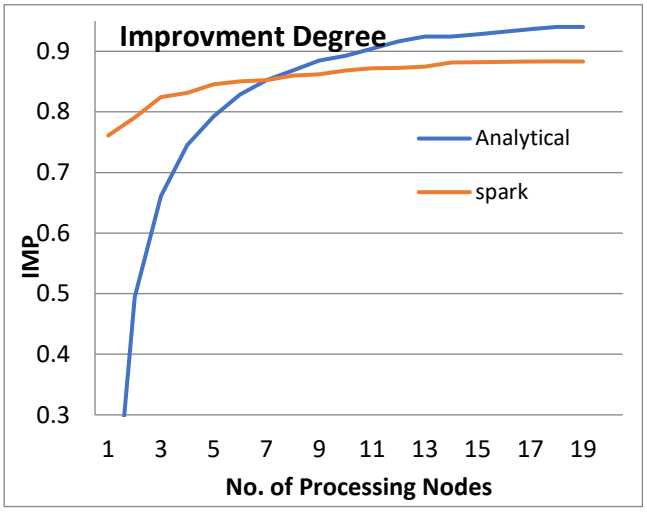
(**a**)



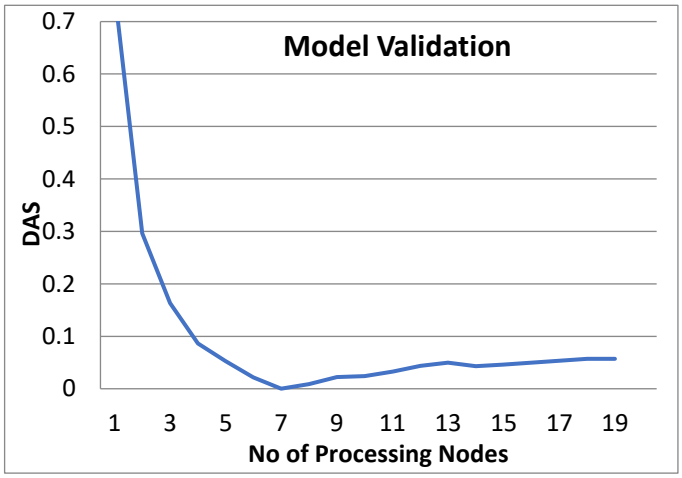
(**b**)



(**c**)

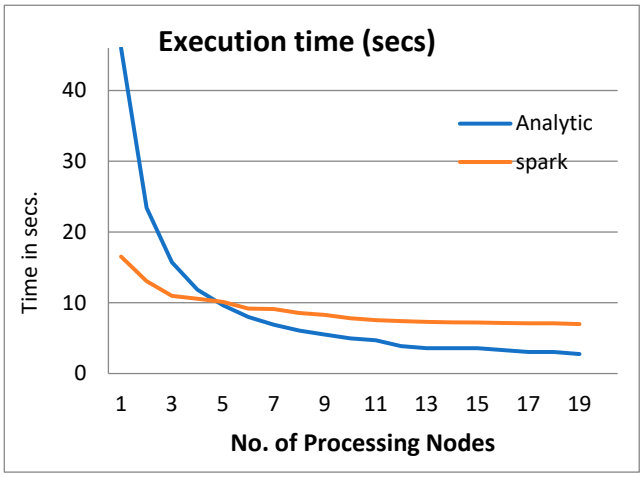


(**d**)

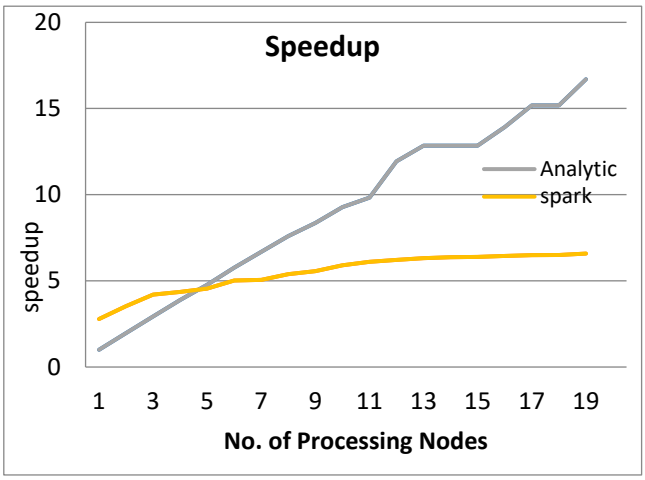


(**e**)

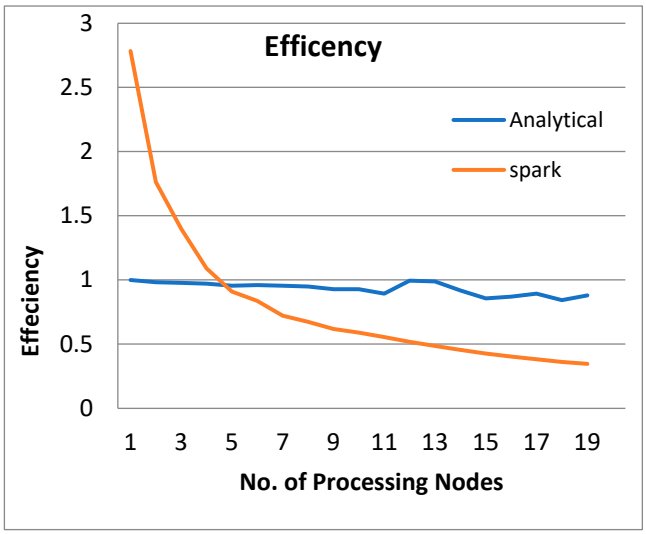
Figure 5. The Performance of the PPQNTRUEncrypt for p = 3, q = 128, and N = 251.



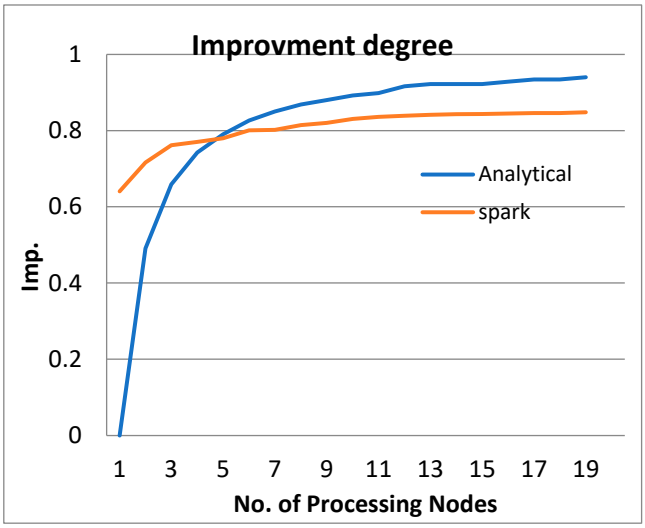
(**a**)



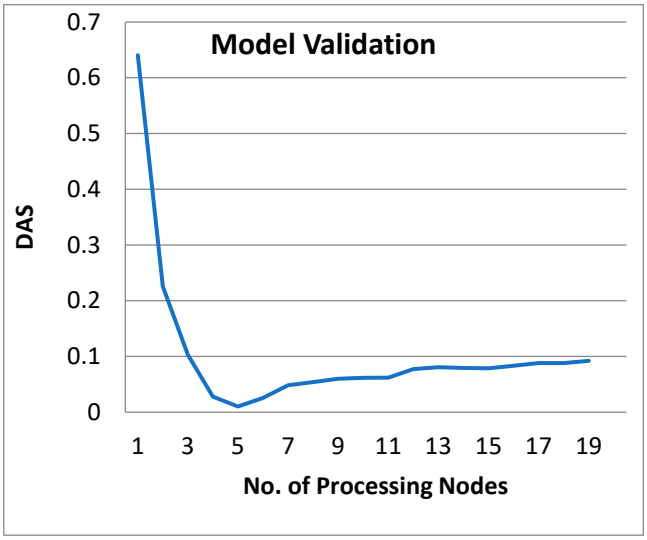
(**b**)



(**c**)



(**d**)



(**e**)

Figure 6. The Performance of the PPQNTRUEncrypt for p = 3, q = 128, and N = 167.

By observing the above figures, the next interpretations can be recognized:

- Figures 3–6 show that the PPQNTRUEncrypt considerably reduces the total time for completing the NTRUEncrypt cryptography algorithm, which makes it suitable for cryptography algorithms (by parallelizing polynomial multiplication).
- From Figures 3a, 4a, 5a and 6a, it is noted that when the number of PEs increases, the total parallel/execution time decreases regardless of the parameter set. This is clear when using both analytical and Spark implementation. That achieves the scalability characteristics of the PPQNTRUEncrypt algorithm.
- Figures Figures 3b, 4b, 5b and 6b show that the speedup increases when the number of PEs increases for both analytical and Spark implementation. Moreover, the system efficiency decreases unrelated to the parameter set, as shown in Figures 3c, 4c, 5c and 6c. These figures show that efficiency is affected by the escalation of the PEs number. That is to say, when the number of PEs increases, the system efficiency decreases, which leads to unstable system efficiency. To solve this problem, the load must be reallocated between different PEs. This ascends the need to split each polynomial multiplication operation into parts (sub-tasks). Therefore, more than one PE works together to execute one polynomial multiplication operation. In this case, the overhead time due to inter-processor communication increases (because of the need to exchange large amounts of data). Therefore, the total parallel/execution time increases. For that reason, it is not preferable to resort to a load-balancing solution except in the case of a large value of "N".
- Figures 3d, 4d, 5d and 6d show that for both analytical and Spark models, the degree of parallel time improvement increases when the number of PEs increases regardless of the parameter set.
- In order to validate the Apache Spark model, a comparison between the performance of the system using the analytical model and its performance using Apache Spark is performed. This compassion is shown in Figures 3e, 4e, 5e and 6e. The results indicated that the difference in the percentage of system performance improvement when using the analytical model and Apache Spark (Difference between analytical and Spark (DAS)) was very small and did not exceed 2%. (it ranges from 2% to 0.2%). It is observed that this difference decreases when the number of PEs (M) increases. Furthermore, this percentage decreases as the number of tasks (N) increases.
- Experimental results demonstrate that achieving significant system performance improvement requires a maximum of twenty processing elements (PEs). Beyond this threshold, the performance gains become economically and hardware-wise insignificant. In other words, the rate of improvement becomes economically and hardware-wise insignificant. Therefore, to ensure practicality, the proposed model suggests limiting the number of PEs to twenty, making it suitable for deployment in small devices like smartphones. Consequently, the authors establish a boundary on the number of PEs, specifically when applying the proposed model to compact devices such as smartphones.
- Adding extra nodes to a Spark cluster can initially enhance performance by increasing parallelism. However, there is a saturation point where the returns start diminishing. Once the number of worker nodes exceeds a certain threshold (specifically, twenty worker nodes), adding extra nodes may not yield significant performance improvements. In fact, it could introduce coordination overhead among the nodes.
- In summary, the findings affirm the effectiveness of employing distributed processing through big data analytics platforms for cryptographic techniques. Specifically, the PPQNTRUEncrypt model in this study utilizes the Apache Spark platform. This is particularly relevant due to the increasing number of connected nodes and the simultaneous encryption and decryption tasks. It can be inferred that the volume of required computational workload, such as polynomial multiplication commonly found in the cryptographic techniques (specifically NTRUEncrypt in this study), significantly increases, potentially involving hundreds of concurrent instances. Furthermore, the

ability of the algorithm to operate in real-time becomes essential to accommodate the expanding number of nodes in the environment.

4.4. Limitations

Cryptographic algorithms often involve computationally intensive operations, such as encryption and decryption, which can benefit from parallel processing to enhance performance and efficiency. To achieve optimal efficiency, concurrency must be integrated into software development. This integration enables the comprehensive utilization of the architecture found in distributed processing systems. Emphasizing the significance of conducting a thorough algorithm analysis beforehand is crucial when employing multiprocessor technologies to prevent runtime problems such as synchronization conditions, communication overhead, or attempting to parallelize an inherently impossible algorithm. Parallelism in software is a powerful tool that must be utilized appropriately to yield optimal results and facilitate the creation of more efficient software [68].

Apache Spark distributed processing framework offers significant advantages in terms of scalability and parallelism. In this work, when implementing the NTRU cryptography algorithm using Apache Spark, specific challenges arise in the encryption and decryption operations. One significant challenge is the communication overhead that arises when data is distributed, and processing is coordinated across multiple nodes. This overhead can impact system performance due to the frequent data exchanges involved in encryption and decryption operations. Experimental results indicate that the optimal number of PEs for achieving significant system performance improvement is twenty. Beyond this value, the gains in performance become negligible, considering the associated cost and size implications. Furthermore, distributing sensitive information introduces another challenge, namely security risks.

To address these challenges in the future, efforts should be directed toward refining communication protocols, enhancing security mechanisms, and developing encryption algorithms optimized for distributed processing. These advancements aim to mitigate overhead and achieve more efficient and secure distributed cryptography algorithms.

5. Conclusions

Public key cryptography can be considered one of the essential protocols in protecting the key exchange between two parties due to its ability to provide secure communication between these two parties. NTRU is a public key cryptographic algorithm that has become popular recently due to its ability to resist attacks from quantum computers. Polynomial multiplication is the core of many NTRU algorithms, such as NTRUEncrypt cryptography, since it contains a huge number of computational operations, which makes it preferable to be parallelized. In this work, a Parallel Post-Quantum NTRUEncrypt algorithm called PPQNTRUEncrypt is proposed. This algorithm uses the skills of parallel computing to speed up the NTRUEncrypt algorithm, by executing polynomial multiplication operations in parallel. Different parameter sets (N, q, p) are explored on a multiprocessor architecture using both analytical and Apache Spark platforms. The implementation results show that utilizing the PPQNTRUEncrypt algorithm enhanced the NTRUEncrypt algorithm by approximately 49.5%, 74.5%, 87.6%, 92.5%, 93.4%, and 94.5%, assuming that the number of processing elements is 2, 4, 8, 12, 16 and 20 respectively.

For future research, we can explore the application of our model to other lattice-based cryptographic protocols. Since the primary focus of our paper is to expedite polynomial multiplication—a core operation in all lattice-based cryptosystems, including LWE and Ring LWE. Another aspect of improvement is to investigate the security risks associated with both serial and parallel implementations of lattice-based cryptographic protocols.

Author Contributions: Conceptualization, G.F.E., H.I.S.A. and H.K.A.; Methodology, G.F.E., H.I.S.A. and H.K.A.; Software, G.F.E., H.I.S.A. and H.K.A.; Validation, G.F.E., H.I.S.A., H.K.A. and M.S.A.; Formal analysis, G.F.E., H.I.S.A., H.K.A. and M.S.A.; Writing—original draft, G.F.E., H.I.S.A. and H.K.A.; Writing—review & editing, G.F.E., H.I.S.A., H.K.A. and M.S.A.; Visualization, G.F.E., H.I.S.A.,

H.K.A. and M.S.A.; Supervision, G.F.E., H.I.S.A., H.K.A., Y.-I.C. and M.S.A.; Project administration, Y.-I.C. All authors have read and agreed to the published version of the manuscript.

Funding: This paper is supported by Korean Agency for Technology and Standard under Ministry of Trade, Industry and Energy in 2023, project numbers are K_G012002236201, K_G012002073401 and K_G012002234001.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

This appendix is an extra information section that contains details about the figures' Titles and Legend:

- Figure 1: Directed Acyclic Graph (DAG) of the polynomial multiplication operation.

A directed acyclic graph (DAG) represents the polynomial multiplication operation in a graph structure without any cycles. The DAG represents the step-by-step multiplication process. Each node in the graph represents a multiplication operation between two terms, and the edges represent the flow of the computation. The DAG structure allows for efficient computation of polynomial multiplication by eliminating redundant operations and utilizing intermediate results.

- Figure 2: Spark framework architecture.

The Spark framework architecture follows a distributed computing model in which there is a master node and multiple slave nodes. The master node is responsible for managing the distributed computing tasks and coordinating the execution of jobs on the slave nodes.

Here is an overview of the master and slave architecture in Spark:

- Master Node: The master node is the central coordinator of the Spark application. It is responsible for dividing the application into tasks and distributing them to the different slave nodes for execution. The master node also tracks the progress of tasks and collects the results from each slave node. The master node generally runs on a dedicated machine.
- Slave Nodes: The slave nodes are worker machines that execute the tasks assigned by the master node. Each slave node typically runs multiple executor processes that perform the actual computations. These executor processes are responsible for processing the data in parallel and communicating with the master node.

The tasks are divided into smaller units called partitions, and these partitions are distributed across the available slave nodes. Each partition is processed by a single executor process on a slave node, and the results are merged at the master node to form the final output.

- Figure 3: The Performance of the PPQNTRUEncrypt for p = 3, q = 256 and N = 503.

Figure 3 shows the performance analysis results of the analytical method and distributed processing (Spark) method for the PPQNTRUEncrypt for p = 3, q = 256, and N = 503. Figure 3a shows the results of the execution time analysis for the two methods. Figure 3b shows the speedup of the two methods against the number of processing nodes. Figure 3c shows the efficiency of the model analysis based on the two methods. Figure 3d shows the degree of improvement in using distributed processing analytically and Spark against the number of processes. Figure 3e shows the error decreasing against the number of analytical processing using distributed processing and Spark.

- Figure 4: The Performance of the PPQNTRUEncrypt for p = 3, q = 128 and N = 347.

Figure 4 shows the performance analysis results of the analytical method and distributed processing (Spark) method for the PPQNTRUEncrypt for p = 3, q = 128, and N = 347. Figure 4a shows the results of the execution time analysis for the two methods.

Figure 4b shows the speedup of the two methods against the number of processing nodes. Figure 4c shows the efficiency of the model analysis based on the two methods. Figure 4d shows the degree of improvement in using distributed processing analytically and Spark against the number of processes. Figure 4e shows the error decreasing against the number of analytical processing using distributed processing and Spark.

- Figure 5: The Performance of the PPQNTRUEncrypt for p = 3, q = 128 and N = 251.

Figure 5 shows the performance analysis results of the analytical method and distributed processing (Spark) method for the PPQNTRUEncrypt for p = 3, q = 128, and N = 251. Figure 5a shows the results of the execution time analysis for the two methods. Figure 5b shows the speedup of the two methods against the number of processing nodes. Figure 5c shows the efficiency of the model analysis based on the two methods. Figure 5d shows the degree of improvement in using distributed processing analytically and Spark against the number of processes. Figure 5e shows the error decreasing against the number of analytical processing using distributed processing and Spark.

- Figure 6: The Performance of the PPQNTRUEncrypt for p = 3, q = 128 and N = 167.

Figure 6 shows the performance analysis results of the analytical method and distributed processing (Spark) method for the PPQNTRUEncrypt for p = 3, q = 128, and N = 167. Figure 6a shows the results of the execution time analysis for the two methods. Figure 6b shows the speedup of the two methods against the number of processing nodes. Figure 6c shows the efficiency of the model analysis based on the two methods. Figure 6d shows the degree of improvement in using distributed processing analytically and Spark against the number of processes. Figure 6e shows the error decreasing against the number of analytical processing using distributed processing and Spark.

References

1. Balamurugan, C.; Singh, K.; Ganesan, G.; Rajarajan, M. Code-based Post-Quantum Cryptography. *Preprints* **2021**, 2021040734. [CrossRef]
2. Wang, L.; Zhang, K.; Wang, J.; Cheng, J.; Yang, Y.; Tang, S.; Yan, D.; Tang, Y.; Liu, Z.; Yu, Y.; et al. Experimental Authentication of Quantum Key Distribution with Post-Quantum Cryptography. *Npj Quantum Inf.* **2021**, *7*, 67. [CrossRef]
3. Nielsen, M.; Chuang, I. Quantum Computation and Quantum Information. *Phys. Today* **2002**, *54*, 60. [CrossRef]
4. Shor, P. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; IEEE Computer Society: Washington, DC, USA, 1994; pp. 124–134.
5. Grover, L. A Fast Quantum Mechanical Algorithm for Database Search. In Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
6. Buchmann, J.; Lauter, K.; Mosca, M. Postquantum Cryptography-State of the Art. *IEEE Secur. Priv.* **2017**, *15*, 12–13. [CrossRef]
7. Umana, V. Post Quantum Cryptography. Ph.D. Thesis, Technical University of Denmark, Kongens Lyngby, Denmark, 2011.
8. Wikipedia. Post-Quantum Cryptography. Available online: https://en.wikipedia.org/w/index.php?title=Post-quantumcryptography&oldid=999863701 (accessed on 12 December 2023).
9. McEliece, R. A Public-Key Cryptosystem Based on Algebraic. *Coding Thv* **1978**, *4244*, 114–116.
10. Merkle, R. *Secrecy, Authentication, and Public Key Systems*; Computer Science Series; UMI Research Press: Ann Arbor, MI, USA, 1982.
11. Patarin, J. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Saragossa, Spain, 12–16 May 1996; pp. 33–48.
12. Hoffstein, J.; Pipher, J.; Silverman, J. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 267–288.
13. Regev, O. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM (JACM)* **2009**, *56*, 34. [CrossRef]
14. Jao, D.; Feo, L. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. *PQCrypto* **2011**, *7071*, 19–34.
15. Kamal, A.; Ahmad, K.; Hassan, R.; Khalim, K. NTRU Algorithm: Nth Degree Truncated Polynomial Ring Units. In *Functional Encryption, EAI/Springer Innovations in Communication and Computing*; Springer: Cham, Switzerland, 2021. [CrossRef]
16. El-Hassane, L.; Azize, A. Boosted Performances of NTRUencrypt Post-Quantum Cryptosystem. *J. Cyber Secur. Mobil.* **2021**, *10*, 725–744. [CrossRef]
17. Mansouri, F. On the Parallelization of Integer Polynomial Multiplication. Master's Theses, The School of Graduate and Postdoctoral Studies, The University of Western Ontario, London, ON, Canada, 2014.

18. Butin, D. Hash-based signatures: State of play. *IEEE Secur. Priv.* **2007**, *15*, 37–43. [CrossRef]
19. Bernstein, D.; Hülsing, A.; Kölbl, S.; Niederhagen, R.; Rijneveld, J.; Schwabe, P. *The SPHINCS+ Signature Framework*; Report 2019/1086; Cryptology ePrint Archive: 2019, University of California: San Diego, CA, USA.
20. Joachim, R. An Overview to Code Based Cryptography. 2016. Available online: https://hkumath.hku.hk/~ghan/WAM/Joachim.pdf (accessed on 12 December 2023).
21. Ding, J.; Petzoldt, A. Current state of Multivariate Cryptography. *IEEE Secur. Priv.* **2017**, *15*, 28–36. [CrossRef]
22. Chen, M.; Ding, J.; Kannwischer, M.; Patarin, J.; Petzoldt, A.; Schmidt, D.; Yang, B. Rainbow Signature. Available online: https://www.pqcrainbow.org/ (accessed on 12 December 2023).
23. Casanova, A.; Faueère, J.; Macario-Rat, G.; Patarin, J.; Perret, L.; Ryckeghem, J. GeMSS: A Great Multivariate Short Signature. Available online: https://www-polsys.lip6.fr/Links/NIST/GeMSS.html (accessed on 12 December 2023).
24. Chi, D.; Choi, J.; Kim, J.; Kim, T. *Lattice Based Cryptography for Beginners*; Report 2015/938; Cryptology ePrint Archive: 2015; University of California: San Diego, CA, USA.
25. Lepoint, T. Design and Implementation of Lattice-Based Cryptography. Ph.D. Thesis, Ecole Normale Euérieure de Paris—ENS, Paris, France, 2014.
26. Alkim, D.; Ducas, L.; Pöppelmann, T.; Schwabe, P. *Post-Quantum Key Exchange—A New Hope*; Report 2015/1092; Cryptology ePrint Archive: University of California, San Diego, CA, USA, 2015.
27. Ducas, L.; Durmus, A.; Lepoint, T.; Lyubashevsky, V. *Lattice Signatures and Bimodal Gaussians*; Report 2013/383; Cryptology ePrint Archive: University of California, San Diego, CA, USA, 2013.
28. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.; Schwabe, P.; Seiler, G.; Stehlé, D. *CRYSTALS—Kyber: A CCA-Secure Module-Lattice-Based KEM*; Report 2017/634; Cryptology ePrint Archive: University of California, San Diego, Ca, USA, 2017.
29. Chen, C.; Danba, O.; Hoffstein, J.; Hülsing, A.; Rijneveld, J.; Saito, T.; Schanck, J.; Schwabe, P.; Whyte, W.; Xagawa, K.; et al. NTRU: A Submission to the NIST Post-Quantum Standardization Effort. Available online: https://ntru.org/ (accessed on 12 December 2023).
30. D'Anvers, J.; Karmakar, A.; Roy, S.; Vercauteren, F. *Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM*; Report 2018/230; Cryptology ePrint Archive: University of California, San Diego, CA, USA, 2018.
31. Bernstein, D.; Chuengsatiansup, C.; Lange, T.; Vredendaal, C. *NTRU Prime: Reducing Attack Surface at Low Cost*; Report 2016/461; Cryptology ePrint Archive: University of California, San Diego, CA, USA, 2016.
32. Ducas, L.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehle, D. *CRYSTALS—Dilithium: Digital Signatures from Module Lattices*; Report 2017/633; Cryptology ePrint Archive: University of California, San Diego, CA, USA, 2017.
33. Fouque, P.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU. Available online: https://www.di.ens.fr/~prest/Publications/falcon.pdf (accessed on 12 December 2023).
34. Supersingular Isogeny Diffie–Hellman Key Exchange (SIDH). Available online: https://en.wikipedia.org/wiki/Supersingular_isogeny_key_exchange (accessed on 12 December 2023).
35. Costello, C.; Longa, P.; Naehrig, M. Efficient Algorithms for Supersingular Isogeny Diffie-Hellman. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2016.
36. Post-Quantum Cryptography | CSRC. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization (accessed on 12 December 2023).
37. *Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone, Report on Post-Quantum Cryptography*; Technical Report NISTIR 8105; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016.
38. *Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone, Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*; Technical Report NISTIR 8309; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020.
39. Ahn, J.; Kwon, H.-Y.; Ahn, B.; Park, K.; Kim, T.; Lee, M.-K.; Kim, J.; Chung, J. Toward Quantum Secured Distributed Energy Resources: Adoption of Post-Quantum Cryptography (PQC) and Quantum Key Distribution (QKD). *Energies* **2022**, *15*, 714. [CrossRef]
40. Kumar, M. Post-quantum cryptography Algorithm's standardization and performance analysis. *Array* **2022**, *15*, 100242. [CrossRef]
41. Dam, D.-T.; Tran, T.-H.; Hoang, V.-P.; Pham, C.-K.; Hoang, T.-T. A Survey of Post-Quantum Cryptography: Start of a New Race. *Cryptography* **2023**, *7*, 40. [CrossRef]
42. Sabani, M.E.; Savvas, I.K.; Poulakis, D.; Garani, G.; Makris, G.C. Evaluation and Comparison of Lattice-Based Cryptosystems for a Secure Quantum Computing Era. *Electronics* **2023**, *12*, 2643. [CrossRef]
43. Kumar, A.; Ottaviani, C.; Gill, S.S.; Buyya, R. Securing the future internet of things with post-quantum cryptography. *Secur. Priv.* **2022**, *5*, e200. [CrossRef]
44. Septien-Hernandez, J.-A.; Arellano-Vazquez, M.; Contreras-Cruz, M.A.; Ramirez-Paredes, J.-P. A Comparative Study of Post-Quantum Cryptosystems for Internet-of-Things Applications. *Sensors* **2022**, *22*, 489. [CrossRef] [PubMed]

45. Tata, P.; Narumanchi, H.; Emmadi, N. Analytical study of implementation issues of NTRU. In Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 24–27 September 2014; pp. 700–707. [CrossRef]
46. Karbasi, A.; Atani, S.; Atani, R. PairTRU: Pairwise Non-commutative Extension of The NTRU Public key Cryptosystem. *Int. J. Inf. Secur. Sci.* **2018**, *7*, 11–19.
47. D'Souza, R. *The NTRU Cryptosystem: Implementation and Comparative Analysis*; George Mason University: Fairfax, VA, USA, 2001.
48. Aldahdooh, R. Parallel Implementation and Analysis of Encryption Algorithms. Master's Thesis, Al-Azhar University-Gaza, Faculty of Engineering & Information Technology, Gaza, Gaza Strip. April 2018.
49. Tallapally, S.; Manjula, B. Suitable encrypting algorithms in Parallel Processing for improved efficiency. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *981*, 022017. [CrossRef]
50. Wan, L.; Zheng, F.; Fan, G.; Wei, R.; Gao, L.; Wang, Y.; Lin, J.; Dong, J. A Novel High-Performance Implementation of CRYSTALS-Kyber with AI Accelerator. In *European Symposium on Research in Computer Security, (ESORICS 2022): Computer Security—ESORICS*; Springer: Cham, Switzerland, 2022; pp. 514–534.
51. Kamal, A.A.; Youssef, A.M. Enhanced Implementation of the NTRUEncrypt Algorithm Using Graphics Cards. In Proceedings of the 1st International Conference on Parallel, Distributed and Grid Computing (PDGC-2010), Solan, India. 28–30 October 2010; pp. 168–174.
52. Dai, W.; Schanck, J.; Sunar, B.; Whyte, W.; Zhang, Z. NTRU modular lattice signature scheme on CUDA GPUs. In Proceedings of the 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, Austria, 18–22 July 2016. [CrossRef]
53. Wong, X.-F.; Goi, B.-M.; Lee, W.-K.; Phan, R.C.-W. Performance Evaluation of RSA and NTRU over GPU with Maxwell and Pascal Architecture. *J. Softw. Netw.* **2017**, 201–220. [CrossRef]
54. Law, M.; Monagan, M. A parallel implementation for polynomial multiplication modulo a prime. In *PASCO '15: Proceedings of the 2015 International Workshop on Parallel Symbolic Computation, Bath, UK, 10–12 July 2015*; ACM Digital Library: New York, NY, USA, 2015; pp. 78–86. [CrossRef]
55. Amit, C.; Gurvinder, S. *Analysis & Integrated Modeling of the Performance Evaluation Techniques for Evaluating Parallel Systems*; CSC Journals: Tulsa, OK, USA, 2008.
56. Jain, R. *The Art of Computer Systems Performance Analysis*; Wiley: Hoboken, NJ, USA, 1991.
57. Tang, S.; He, B.; Yu, C.; Li, Y.; Li, K. A Survey on Spark Ecosystem for Big Data Processing. *arXiv* **2018**. [CrossRef]
58. Hennessy, J.; Patterson, D. *Computer Architecture: A Quantitative Approach*; Morgan Kaufmann: Cambridge, MA, USA, 2003.
59. Rasslan, M.; Elkabbany, G.; Aslan, H. New Generic Design to Expedite Asymmetric Cryptosystems using Three-level Parallelism. *Int. J. Netw. Secur. (IJNS)* **2018**, *20*, 371–380.
60. Foldi, T.; von Csefalvay, C.; Perez, N. JAMPI: Efficient Matrix Multiplication in Spark Using Barrier Execution Mode. *Big Data Cogn. Comput.* **2020**, *4*, 32. [CrossRef]
61. Park, T.; Seo, H.; Kim, J.; Park, H.; Kim, H. Efficient Parallel Implementation of Matrix Multiplication for Lattice-Based Cryptography on Modern ARM Processor. *Secur. Commun. Netw.* **2018**, *2018*, 7012056. [CrossRef]
62. Jangla, G.; Amne, D. Development of an Intrusion Detection System based on Big Data for Detecting Unknown Attacks. *Int. J. Adv. Res. Comput. Commun. Eng.* **2015**, *4*, 229–232.
63. Ellingwood, J. Hadoop, Storm, Samza, Spark, and Flink: Big Data Frameworks Compared. 2016. Available online: https://www.digitalocean.com/community/tutorials/hadoopstorm-samza-spark-and-flink-big-data-frameworks-compared (accessed on 12 December 2023).
64. Deshai, N.; Sekhar, B.V.D.S.; Venkataramana, S. Mllib: Machine learning in apache spark. *Int. J. Recent Technol. Eng.* **2019**, *8*, 45–49.
65. Kumar, G. Evaluation Metrics for Intrusion Detection Systems—A Study. *Int. J. Comput. Sci. Mob. Appl.* **2014**, *2*, 11–17.
66. Kattemolle, J. Short introduction to Quantum Computing. 2017. Available online: https://www.kattemolle.com/KattemolleShortIntroToQC.pdf (accessed on 12 December 2023).
67. Apache Software Foundation. Apache Spark Documentation. Available online: https://spark.apache.org/docs/latest/ (accessed on 12 December 2023).
68. Mochurad, L.; Shchur, G. Parallelization of Cryptographic Algorithm Based on Different Parallel Computing Technologies. In Proceedings of the Symposium on Information Technologies & Applied Sciences (IT&AS'2021), Bratislava, Slovakia, 5 March 2021.