IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 14 December 2023, accepted 7 January 2024, date of publication 10 January 2024,
date of current version 19 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3352157

RESEARCH ARTICLE

High- and Half-Degree Quantum Multiplication for Post-Quantum Security Evaluation

RINI WISNU WARDHANI[1], (Graduate Student Member, IEEE),
DEDY SEPTONO CATUR PUTRANTO[2,3], (Member, IEEE),
AND HOWON KIM[1], (Member, IEEE)
[1] School of Computer Science and Engineering, Pusan National University, Busan 609735, South Korea
[2] IoT Research Center, Pusan National University, Busan 609735, South Korea
[3] Blockchain Platform Research Center, Pusan National University, Busan 609735, South Korea

Corresponding author: Howon Kim (howonkim@pusan.ac.kr)

This work was supported in part by the Institute for Information and Communications Technology Planning and Evaluation (IITP) funded by the Korea Government Ministry of Science and Information and Communication Technology/ICT (MSIT), South Korea (Study on Quantum Security Evaluation of Cryptography Based on Computational Quantum Complexity, 50%) under Grant 2019-0-00033; and in part by MSIT under the Convergence Security Core Talent Training Business (Pusan National University) Support Program Supervised by IITP (50%) under Grant IITP-2023-2022-0-01201.

ABSTRACT This paper provides a thorough examination of strategies related to the design of the polynomial multiplication approach, which is essential to resolving large number operations, including multiplication in the post-quantum cryptography algorithm. Our research specifically centers on the implementation of Toom-Cook-based multiplication algorithms for high- and half-degree quantum multiplication, up to 20.5-way degrees, considering that existing Toom-Cook lower degrees are combined with a lattice-based approach like the Saber algorithm. In particular, we propose the quantum multiplication design, conduct computation step experiments, and implement the division-free method. In addition to examining the Toom-Cook 20.5-way algorithm, this study also provides an overview of the results obtained from the Toom-Cook 8.5-way and Toom-Cook 10.5-way algorithms. The Toom-Cook 20.5-way design exhibits significant performance improvement over its predecessor, as evidenced by its lower value of asymptotic complexity and lower cost of quantum implementation, with a qubit count of $n^{1.186}$, approximately $522n^{\log_{21} 41} - 540n$ Toffoli count, and $n^{1.033}$ Toffoli depth. Concerning the asymptotic performance analysis and quantum cost compared to alternative Toom-Cook-based multiplication, the results indicate the development of a more efficient multiplication polynomial that may be utilized in the evaluation of post-quantum security.

INDEX TERMS Toom-cook, high- and half-degree quantum multiplication, asymptotic performance analysis, quantum cost.

I. INTRODUCTION

The Toom-Cook algorithm, which is based on work from [1], [2], is well recognized as a highly efficient strategy for resolving complex multiplication problems involving large numbers. The concept can often be referred to as a universal multiplication [3]. In general, the approach entails decomposing the multiplication procedure into smaller multiplications and additions, resulting in a reduction of the overall computational complexity. In numerous fields, including computer algebra systems and cryptography, this method is utilized extensively to enhance the performance of polynomial multiplication operations, including the multiplication portion of the post-quantum cryptography (PQC) algorithm.

There has been a significant surge in the adoption of Toom-Cook and number theoretic transform (NTT)-based polynomial multiplication as a result of their incorporation as essential elements in the post-quantum standardization initiative [3], [4]. In addition to classical and quantum multiplication designs, i.e., [5], [6], [7], [8], several studies [3], [4], [9] have put forth an interesting investigation concerning

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

Toom-Cook multiplication. The current investigation, which reveals the vulnerabilities of the Toom-Cook multiplication algorithm, was carried out by [4] and [10]. Hence, the research described in [11] gives an insight into increasing the number of iterations to mitigate the vulnerability to peak analysis attacks in the correlation power analysis (CPA) technique to lower degree Toom-Cook, as noted by [4].

In their preliminary investigation, Bodrato et al. [6] proposed a multiplication method known as Toom'n'half, which integrates high- and half-degree multiplication to enhance the efficacy strategy and outperforms Toom-Cook in classical computing. In this work, we design the Toom-Cook-based multiplication algorithms for high- and half-degree quantum multiplication, up to 20.5-way degrees, and investigate asymptotic performance analysis and quantum resources for Toom-Cook-based multiplication in terms of qubit count, Toffoli count, and Toffoli depth.

This work presents a novel design that significantly improves the efficiency of the multiplication algorithm based on the substantial strategy for Toom-Cook high-degree algorithm from several prior works (e.g., [4], [5], [6], [7], [8], [11], [12], and [13]). To the best of our understanding, this is the first work to integrate high- and half-degree quantum multiplication in classical and quantum design while maintaining an architecture degree of up to 20.5-way for Toom-Cook-based multiplication. Achieving the highest possible asymptotic performance result while utilizing fewer quantum resources is the principal aim.

The contributions of the paper can be summed up as follows:

1) We conduct a thorough examination and synthesis of literature pertaining to multiplication techniques and works based on the Toom-Cook algorithm, focusing on strategies for polynomial multiplication in classical and quantum designs and also attacks based on Toom-Cook multiplication. In addition, we examine the application of the multiplication operation in the construction of a post-quantum algorithm, such as the lattice-based methodology, i.e., Saber and Kyber. In addition, we capture relevant research on multiplication-based attack techniques that can potentially be performed with a lower degree of Toom-Cook-based multiplication, specifically CPA.
2) We elaborate on several approaches and strategies from prior related works ([5], [6], [7], [8], [11]) to reduce computational resources, i.e., opposite points, inverses, and cost-exploiting symmetries. Our focus lies on the implementation of high- and half-degree quantum multiplication techniques, specifically addressing the design strategies employed in the Toom-Cook 8.5-way, 10.5-way, and 20.5-way methodologies.
3) We design the proposed multiplication and experiment computation steps such as splitting, evaluation, recursive multiplication, interpolation, and recomposition in a sequential manner and derive the quantum circuit for performing high- and half-degree quantum multiplication. As a result, we present the Toom-Cook 8.5-way, 10.5-way, and 20.5-way quantum design architectures, wherein the highest degree of Toom-Cook 20.5-way exhibits the lowest asymptotic performance and minimal utilization of quantum resources.
4) We analyze the asymptotic performance and quantum resource utilization of various multiplication algorithms, with a particular focus on the Toom-Cook-based multiplication algorithm for degrees up to 20.5-way degrees. Our investigation leads to the conclusion that the Toom-Cook 20.5-way architecture demonstrates the lowest quantum resource utilization, requiring qubit count of $n(\frac{41}{21})^{\frac{\log 41}{(2\log 41 - \log 21)}\log_{21} n} \approx n^{1.186}$, $522n^{\log_{21} 41} - 540n$ Toffoli count, and $n(\frac{41}{21})^{1-\frac{\log 41}{(2\log 41 - \log 21)}\log_{21} n} \approx n^{1.033}$ Toffoli depth.

The paper is structured in the following manner: Section I gives an overview related to the research. Section II presents a thorough examination of the existing literature, specifically focusing on the Toom-Cook algorithm and its associated works. In Section III, we present a succinct summary of the utilization and multiplication-based attacks. In Sections IV and V, the strategies and detailed procedures for designing the proposed high- and half-degree quantum multiplication are outlined, specifically the Toom-Cook 20.5-way designs. Section VI presents a comprehensive analysis of the space and time complexity, along with a comparative evaluation of multiplication techniques utilizing the Toom-Cook algorithm. Section VII encompasses the discussion and future work directions, while Section VIII presents the conclusions of the current study.

II. RELATED WORKS

The Toom-Cook algorithm, particularly the Toom-Cook k-way approach for multiplication, has superior time complexity when compared to the naive Schoolbook multiplication algorithm, which has a time complexity of $\mathcal{O}(n^2)$. Furthermore, this approach also achieves a lower asymptotic performance analysis compared to the Karatsuba algorithm, which is regarded as equivalent to the Toom 2-way algorithm with a complexity of $(n^{log(3)/log(2)}) \equiv \mathcal{O}(n^{1.58})$. In Table 1, a comprehensive analysis and synthesis of works related to multiplication methodologies and investigations centered around the Toom-Cook algorithm are presented. The discussion primarily revolves around approaches employed for polynomial multiplication, both in classical and quantum environments, as well as potential vulnerabilities stemming from Toom-Cook multiplication.

Alberto Zanoni [5] presents a computational implementation of a balanced Toom-Cook 8-way algorithm in a classical environment, specifically designed for integer multiplication and squaring. The Toom-8 classical method employs a technique that divides components into eight distinct portions. Previously, Bodrato and Zanoni have made advancements in the original Toom approach family, as discussed in

TABLE 1. Toom-Cook-based multiplication related research. We extracted information from [4], [5], [6], [7], [8], [11], [12], and [13] to elucidate on their particular Toom-Cook-based-related task in a classical-quantum environment.

No	References Paper	Year	Arithmetic Algorithm	Research Field Focus	Specific Step	Strategies	Results
1	Marco Bodrato [6], [12]	2007-2011	High-degree Toom'n'half (in general)	High degree Toom'n'half for balanced and unbalance multiplication (Classical)	Splitting, Evaluation, Recursive Multiplication, Interpolation, (Early)Recomposition	Estimation cost Toom'n'half, First comparison to Karatsuba	Memory optimization and and arithmetic optimization in classical environment
2	Zanoni et al. [5]	2009	Toom-Cook 8-way - degree 7	Toom-Cook 8-way implementation for Long Integers Multiplication (Classical)	Splitting, Evaluation, Recursive Multiplication, Interpolation, Recomposition	Toom-Cook 8-way implementation for Long Integers (multiplication and squaring)	Balance Toom-8 and Unbalanced Toom-8 Multiplication and comparing code with GMP 4.3.0 library
3	Dutta et al. [7]	2018	Toom-Cook 2.5 Multiplication	Toom-Cook Multiplication (Quantum Design)	Splitting, Evaluation, Recursive Multiplication, Interpolation, Recomposition	Implementation in k=2.5, tree structured method, and gate count analysis	Quantum design on Toom 2.5 and Comparison on asymptotic multiplication method (naive, naive improves, Karatsuba, Toom 2.5 and Const.Mult)
4	Larasati et al. [8]	2021	Toom-Cook 3 Multiplication	Toom-Cook Multiplication (Quantum Design)	Splitting, Evaluation, Recursive Multiplication, Interpolation, Recomposition	Implementation in k=3 and gate count analysis	Quantum design on Toom 3 and Comparison on asymptotic multiplication method (naive, naive improves, Karatsuba, Toom 2.5, Toom 3, and Const.Mult)
5	Mera et al. [3]	2020	Toom-Cook-based multiplication	Time-memory trade off in Toom-Cook multiplication (Classical)	Pre- and post-processing steps (Evaluation step and Interpolation)	Evaluation step is reduced by a factor of 2 (precomputation) and lazy-interpolation	Toom-Cook design that increase efficiency
6	Ghosh et al. [9]	2020	Toom-Cook	Design efficient polynomial multiplication with Toom-Cook (Classical)	Memory efficient striding Toom-Cook with lazy interpolation	Low-latency striding Toom-Cook Multiplication	Optimized Toom-Cook Multiplication implementation in Saber post-quantum accelerator
7	Gu et al. [13] [3]	2021	DT3 (Division-free Toom-3)	Proposed Montgomery Modular Multiplication Based on DT3 (Classical)	Division-Free Toom-Cook Multiplication	Employing Montgomery Modular Multiplication with a Division-Free Toom-Cook Multiplication	Achieve ASIC high-performance hardware implementation
8	Li et al. [4]	2022	Toom Cook Multiplication	Single Trace Side Channel attack on the Toom-Cook (Classical)	Side Channel Attack on the Toom Cook with case study Saber post-quantum Algorithm	Elaborate on Toom-Cook based multiplication on Saber PQC Algorithm and its correlation to power usage (correlation power analysis)	Side Channel Attack result on the Toom Cook with case study Saber post-quantum Algorithm (for lower degree ; Toom 4)
9	Putranto et al. [14]	2023	Toom-Cook 2, 4 and 8 Multiplication	Toom-Cook Multiplication (Quantum Design)	Splitting, Evaluation, Recursive Multiplication, Interpolation, Recomposition	Quantum division free implementation in Toom-Cook k=2,4,8	Comparison on asymptotic multiplication method (naive, naive improves, Karatsuba, Toom 2, Toom 2.5, Toom 3, Toom 4, Toom 8, and Const.Mult)
10	Ours	2023	Toom-Cook 8.5, 10.5 and 20.5 Multiplication	High- and half-Degree Quantum Multiplication (Quantum Design)	Splitting, Evaluation, Recursive Multiplication, Interpolation, Recomposition	Quantum design division free Implementation in high-degree Toom-Cook k=8.5,10.5,20.5 with division free and arithmetic optimization (i.e., opposite points, inverses, symmetries) and gate count analysis	First method in quantum for high- and half-degree quantum multiplication with division free and arithmetic optimization (i.e., , tree structured method, opposite points, inverses, symmetries)

their work [15], by incorporating unbalanced operands, which refer to polynomials with varying degrees. This has been achieved by the utilization of the Toom$-(k + 1/2)$ approaches. Later, in Marco Bodrato's advanced research in [6], several strategies and techniques for achieving a high degree of precision in the implementation of Toom-Cook algorithms, specifically for both balanced and unbalanced scenarios, are clarified in classical hardware implementation.

Dutta et al. [7] conducted a thorough investigation of the utilization of the Toom-Cook 2.5-way technique within the quantum circuit for multiplication, offering a detailed explanation of its workings. A bound on the count of Toffoli gates and qubits was found by analysis of the recursive tree structure of the method. The proposed quantum circuit exhibits superior asymptotic bounds for several performance metrics compared to previous implementations of multiplier circuits utilizing schoolbook and Karatsuba methods. Later, the research continued by Larasati et al. [8] shows findings that demonstrate the possibility of the $k-$way Toom-Cook method, which employs higher-degree polynomial interpolation, to exhibit lower asymptotic complexity in comparison to alternative approaches such as Toom-Cook 2.5-way.

In their study, Larasati et al. [8] expound upon the Toom-Cook 3-way algorithm by incorporating the division gate. This research specifically devised the quantum circuit that aligns with Bodrato's suggested sequence, aiming to minimize the number of operations required, particularly

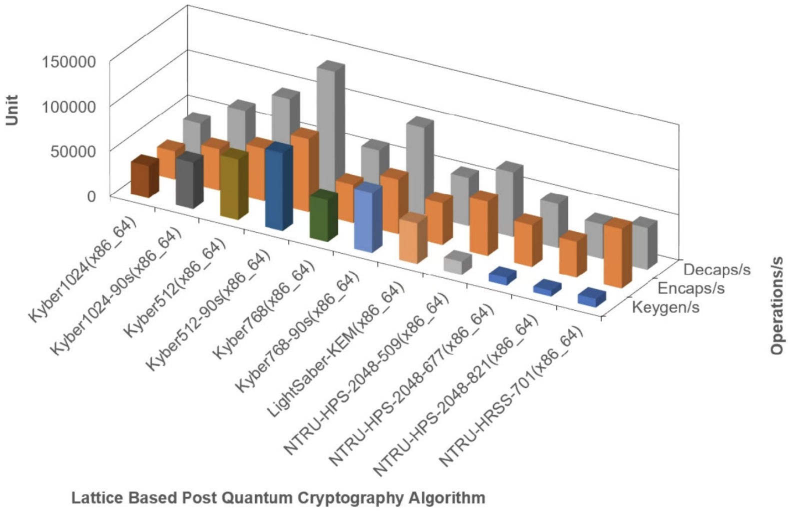

FIGURE 1. **Runtime analysis of Open Quantum Safe Lattice-based Cryptographic algorithms (Key Encapsulation Mechanisms).**

in relation to nontrivial division. This approach effectively reduces the need for nontrivial divisions, as only one exact division by 3 circuit is utilized per iteration. Furthermore, in order to further reduce the expenses associated with the remaining division, the researchers employ the distinctive characteristics of the specific division circuit. Based on the result of the numerical analysis, it can be observed that the circuit exhibits a reduced asymptotic complexity in terms of Toffoli depth and qubit count when compared to Toom-Cook 2.5-way [7]. However, it is worth noting that a significant number of Toffoli gates are required primarily for the implementation of the division operation.

The research on [6], [8], and [11] highlights the existence of a challenge in designing high degree Toom-Cook-based multiplication. However, it also presents an opportunity to achieve superior performance in asymptotic performance analysis, both in classical or quantum environments.

III. LOWER-DEGREE USAGE AND ATTACK VULNERABILITY OF TOOM-COOK IN THE PQC ERA

Numerous investigations have been conducted on the enhancement of public-key cryptosystems, aiming to protect against potential attacks deriving from both classical and quantum computing paradigms. The period characterized by the need for quantum-resistant encryption is commonly denoted as the PQC era, as elucidated in [16]. In this subsection, we highlight a brief example of the usage and implementation of Toom-Cook-based multiplication in the Saber and Kyber PQC algorithm, as well as the potential vulnerability that arises from the utilization of lower-degree multiplication.

According to the NIST PQC standardization process, the two main algorithms that are suggested for a range of applications, including digital signatures, are Crystals-Kyber [17] for public-key setup and Crystals-Dilithium [18] Lattice-based encryption is expected to exhibit optimal efficiency and resilience against quantum attacks, rendering it a feasible solution within the domain of PQC and appears to be the most rapid implementation as in [19], [20], [21], and [22]. Dilithium, Falcon, FrodoKEM, Kyber, NTRU, NTRU Prime, and Saber are seven of the fifteen candidates in the NIST third round that use lattice-based cryptography [16].

Polynomial multiplications, such as Toom-Cook and NTT, play a crucial role in lattice-based post-quantum encryption by serving as the essential constituents. Lattice-based cryptographic systems commonly employ either the NTT with time complexity of $(\mathcal{O}(n \log n))$ [23] or the Toom-Cook/Karatsuba algorithm with time complexity of $(\mathcal{O}(n^{1+\epsilon})$, where $0 < \epsilon < 1)$, [1], [2], [24], to achieve efficient polynomial multiplication involving n coefficients [10]. These multiplications facilitate the division of the resultant sub-polynomial, as highlighted in [10]. The results of the runtime analysis for a post-quantum lattice-based cryptographic algorithm, specifically a key encapsulation mechanism, are displayed in Figure 1. In this figure, our focus is solely on the Kyber algorithm. The analysis is conducted by comparing the algorithm's runtime behavior and memory consumption statistics, as documented in the work by Mujdei et al. [10].

While the majority of other research concentrates on optimizing NTT-based multiplications, research [9] optimizes a Toom-Cook-based multiplier to an exceptional degree. A memory-efficient striding Toom-Cook with delayed interpolation yields a highly compact, low-power implementation that allows for a very regular memory access scheme. Besides the usage of Toom-Cook multiplication, they demonstrate the multiplier's effectiveness and integrate it into one of the four NIST finalists, the Saber post-quantum accelerator. The Saber algorithm employs an additional division of the resultant sub-polynomials into two Karatsuba layers, followed by the execution of a 16-coefficient schoolbook operation [10]. Figure 2 displays an image that portrays an occurrence of Toom-Cook-based multiplication executed

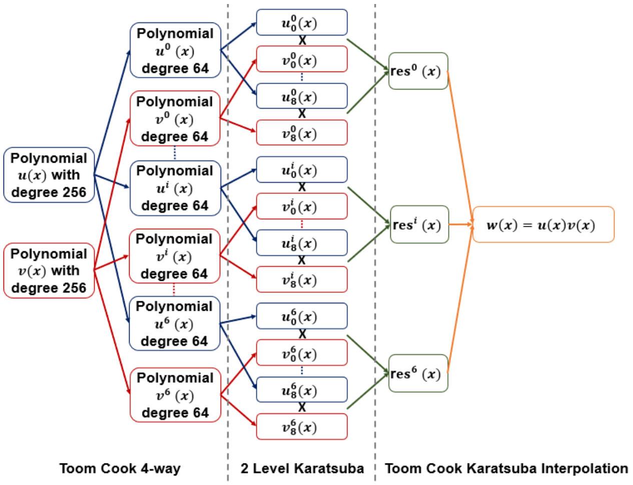
Polynomial $u(x)$ with degree 256
Polynomial $v(x)$ with degree 256
Polynomial $u^0(x)$ degree 64
Polynomial $v^0(x)$ degree 64
Polynomial $u^i(x)$ degree 64
Polynomial $v^i(x)$ degree 64
Polynomial $u^6(x)$ degree 64
Polynomial $v^6(x)$ degree 64
$u_0^0(x)$ X $v_0^0(x)$
$u_8^0(x)$ X $v_8^0(x)$
$u_0^i(x)$ X $v_0^i(x)$
$u_8^i(x)$ X $v_8^i(x)$
$u_0^6(x)$ X $v_0^6(x)$
$u_8^6(x)$ X $v_8^6(x)$
$res^0(x)$
$res^i(x)$
$res^6(x)$
$w(x) = u(x)v(x)$
Toom Cook 4-way
2 Level Karatsuba
Toom Cook Karatsuba Interpolation

FIGURE 2. The Toom-Cook 4-way and Karatsuba Multiplication used in Saber Post-Quantum Cryptography Algorithm.

within the Saber structures. We redraw to demonstrate the application of the Toom-Cook 4-way method in the implementation of the Saber post-quantum cryptography algorithm from the work of Mera et al. [3].

Regarding attacks based on multiplication, the utilization of side-channel information can be targeted towards the multiplication process, as demonstrated in recent studies [4], [10]. The exploitation of side-channel information, such as power consumption, electromagnetic radiation, and execution time, has been shown to be a method for gaining unauthorized access to sensitive data [25]. CPA is widely recognized as a very effective technique that leverages the correlation between a device's power consumption and the data it is processing. This approach exploits power fluctuations that are caused by mathematical processes such as multiplication. Hence, the evaluation of potential risks associated with multiplication exploitation in side-channel analysis attacks, particularly when utilizing the CPA approach, is crucial during the construction of cryptographic algorithms. This concern arises due to the frequent use of arithmetic multiplication as a sub-operation multiplier in real implementations.

Mujdei et al. conducted an experimental analysis to investigate the potential occurrence of CPA peaks when employing the schoolbook sub-operation in the processing of 3-way and 4-way Toom-Cook within the lattice-based PQC algorithm. The post-quantum algorithm NTRU−HPS−4096821 elaborated in [10], can be subjected to a multiplication-based attack utilizing side-channel measurements. Mujdei et al. study encompasses an examination of the variance plot of 500 instances of schoolbook multiplication, wherein a comprehensive analysis reveals the identification of a total of 72 apparent peaks. Proficiency in mathematical approaches is essential for the development of PQC algorithms that can effectively withstand SCA. Furthermore, the utilization of effective mathematical techniques is imperative in the construction of quantum circuits, which can be employed for the creation of cryptanalysis circuits. The primary function of these cryptanalysis circuits is to evaluate the resilience of a method.

Efficient arithmetic operations, particularly multiplication, play a vital role in conducting comprehensive investigations within the domain of quantum-based cryptanalysis. According to Roche [26], Parent et al. [27], Gidney [28], Banegas et al. [29], and Putranto et al. [11], [14], the development of a fundamental arithmetic constructor that demonstrates efficiency in terms of space use and time consumption is crucial for expediting the cryptanalysis process. The primary objective of these investigations is to reduce the complexity that is typically encountered during the execution of quantum cryptanalysis. The efficacy of basic mathematical operations, particularly multiplication, can significantly impact the predictive analysis of the utilization of multiplication inside the lattice-based PQC algorithm, as well as the quantum computer's ability to solve conventional public key cryptography through cryptanalysis, which further leads to post-quantum security evaluation.

IV. TOOM-COOK MULTIPLICATION COMPUTATIONAL STEPS AND STRATEGIES

A. TOOM-COOK COMPUTATION STEPS

In the present subsection, we employ the identical overarching parameter as Marco Bodrato's prior study on classical computation [6] to establish a direct correlation. Given two polynomials, u and v, belonging to the ring of polynomials over an integral domain $\mathbb{R}[x]$, we aim to calculate the product $\mathbb{R}[x]$. The algorithm can be delineated into five distinct steps, denoted as splitting, evaluation, recursive multiplication, interpolation, and recomposition.

1) SPLITTING

Let $Y = x^b$ be selected as the fundamental value. Subsequently, u and v can be represented by two homogeneous polynomials, namely $\mathfrak{u}(y, z) = \Sigma_{i=0}^{n-1} u_i z^{n-1-i} y^i$ and $\mathfrak{v}(y, z) = \Sigma_{i=0}^{m-1} v_i z^{m-1-i} y^i$. These polynomials consist of n and m coefficients, respectively, and their degrees, denoted as $\deg(\mathfrak{u}) = n - 1$ and $\deg(\mathfrak{v}) = m - 1$, align with the aforementioned coefficients and degrees. Such that $\mathfrak{u}(x^b, 1)$ equals the variable u and $\mathfrak{v}(x^b, 1)$ equals the variable v under the specified circumstances. The coefficients of $u_i, v_i \in \mathbb{R}[x]$ can be represented as polynomial functions and can be chosen to have a specified degree, denoted as $\forall i$, $\deg(u_i) < b$, $\deg(v_i) < b$.

The typical way of implementing the Toom k-way algorithm requires the operands to be evenly sized, with one operand m being the same size as the other operand n. In contrast to the same-size operand value, we expand our approach to elaborate on unbalanced or not-the-same-size operand value circumstances. This approach refers to [5] and [6] that labels the methodology as Toom-$\frac{n+m}{2}$ for Toom'n'half terminology. In this research, we specifically refer to it as Toom-Cook high- and half-degree quantum multiplication.

2) EVALUATION

In order to calculate the product $\mathfrak{w} = \mathfrak{u} \cdot \mathfrak{v}$, where the degree of the polynomial is denoted as $d = n + m - 2$, it is necessary to have $d + 1 = n + m - 1$ evaluation points $P_d = (\alpha_0, \beta_0), \ldots, (\alpha_d, \beta_d)$, where $(\alpha_i, \beta_i) \in \mathbb{R}[x]$. The process of computing the evaluation of a single polynomial, for example $\mathfrak{u}$, at specific positions (α_i, β_i), can be achieved by the utilization of matrix by vector multiplication.

3) RECURSIVE MULTIPLICATION

During this stage, the computation of $\forall i, \mathfrak{w}(\alpha_i, \beta_i) = \mathfrak{u}(\alpha_i, \beta_i)\mathfrak{v}(\alpha_i, \beta_i)$ will be performed, which will involve the multiplication of $d + 1$ polynomials with degrees that are either less than or equal to the degree of $Y = x^b$.

4) INTERPOLATION

The outcome of this phase is solely determined by the expected degree of the result d as well as the selection of $d + 1$ points (α_i, β_i). It is not influenced by n and m individually. The coefficients of the polynomial $\mathfrak{w}(y, z) = \Sigma_{i=0}^{d} w_i z^{d-i} y^i$ are currently required. Given that we possess the values of w assessed at $d + 1$ distinct points, we are met with a classical interpolation problem. We apply the inverse of the A_d, as also described in the [6] paper, often referred to as a $(d+1)x(d+1)$ Vandermonde-like matrix.

5) RECOMPOSITION

Calculating the required output in an effective manner is made possible by performing an additional evaluation using $w = \mathfrak{w}(x^b, 1)$. This particular stage requires at most d shifts and additions to be implemented.

B. DESIGN STRATEGIES

Within the classical context strategies for Toom-Cook multiplication, prior work [6] notes the presence of two critical stages, notably evaluation and interpolation. The rationale provided is that both evaluation and interpolation necessitate the utilization of matrix by vector multiplication. Therefore, both of these stages necessitate numerous operations such as additions, subtractions, shifts, and even little multiplications or precise divisions (that is, interpolation) involving small elements in $\mathbb{R}[x]$. Furthermore, within the [6] works, the process of splitting apart is executed by a mix of all phases and interlaced operations, with the objective of reducing both memory consumption and the overall count of operations in $\mathbb{R}$. This section provides a concise explanation of the methods employed in classical Toom'n'half by Bodrato [6]. These strategies include the utilization of opposite points, the implementation of inverse matrix strategies during the interpolation step, and the incorporation of cost-exploiting symmetries.

1) OPPOSITE POINTS

The interpolation can be achieved by performing a series of operations on the rows of the matrix during the interpolation step. By eliminating more entries in each step, it is possible to shorten the sequence [15]. The lines produced by two contrasting evaluations exhibit identical absolute values, with all values being positive on one line and alternating on the other. By performing the operations of addition and subtraction on the two lines and subsequently dividing the result by two, we are able to derive the following lines:

$$\begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \alpha^2\beta^{d-2}\ldots & \alpha^d \\ \beta^d & -\alpha\beta^{d-1} & \alpha^2\beta^{d-2}\ldots & (-\alpha)^d \end{pmatrix} \tag{1}$$

In this procedure, half of the entries are adjusted to zero.

$$\begin{pmatrix} 0 & \alpha\beta^{d-1} & 0 & \alpha^3\beta^{d-3} & \ldots \\ \beta^d & 0 & \alpha^2\beta^{d-2} & 0 & \ldots \end{pmatrix} \tag{2}$$

In this work, we employ opposite points, which implies the necessity of consistently employing pairs of points. This implies that in the presence of the two unusual points (1,0) and (0,1), commonly referred to as zero and infinity, it is desirable to have an even number of points. In the case of the conventional balanced Toom-Cook k-way algorithm, the number of points needed is determined by the equation $d + 1 = 2n - 1$, which always results in an odd value. On the other hand, we have advantages that serve as the primary motivation for investigating the unbalanced Toom'N'half algorithm which necessitates the use of $2n$ points.

2) INVERSES

In the interpolation matrix, if we evaluate (α, β) and its inverse (β, α), it reveals the presence of two symmetric lines in the matrix. These lines exhibit identical values and signs but are arranged in opposite orders, as described in Equation 3. By performing the operations of addition and subtraction on the two lines, we are able to derive two distinct lines. One of these lines exhibits symmetry, while the other does not possess this characteristic (asymmetric), as shown in Equation 4.

$$\begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \ldots & \alpha\beta^{d-1} & \alpha^d \\ \alpha^d & \alpha^{d-1}\beta & \ldots & \alpha^{d-1}\beta & \beta^d \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} \alpha^d + \beta^d & \alpha^{d-1}\beta + \alpha\beta^{d-1} & \ldots & \beta^d + \alpha^d \\ \alpha^d - \beta^d & \alpha^{d-1}\beta - \alpha\beta^{d-1} & \ldots & \beta^d - \alpha^d \end{pmatrix} \tag{4}$$

If each line in the interpolation matrix possesses its own inverse, it is possible to derive a new matrix that can be represented in a block-wise manner [6], as presented in Equation 5.

$$\left(\begin{array}{cc} c|cA & A' \\ \hline -B & B' \end{array}\right) \tag{5}$$

The matrices A' and B' represent the mirrored versions of matrices A and B, respectively. The two exceptional points, $(1, 0)$ and $(0, 1)$, were disregarded, as they can be handled independently. The two regular patterns, $(1, 1)$ and $(-1, 1)$, exhibit symmetrical lines and should be included in A, A' without the requirement of precomputations. In the following

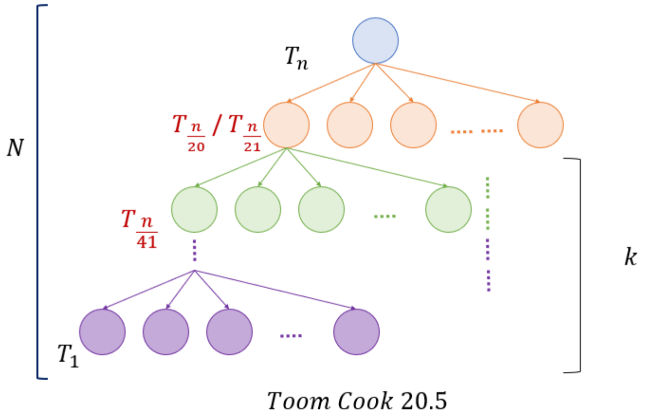

FIGURE 3. The Toom-Cook 20.5-way Multiplication Recursion Tree Structure, where T represents the Toom-Cook k-way Multiplication and n and N represent the bit length for each level and the overall depth of the tree, respectively.

steps, when operating lines in A, any entry that is zeroed in A will have a corresponding entry zeroed through the same action in A'. A similar phenomenon occurs with both B and B'. In this manner, the nullifying impact of the following operations is doubled.

3) COST EXPLOITING SYMMETRIES

It is observed from the studies of [6] that, in order to encompass all symmetries, it is necessary to consider quartets of points denoted as $\pm\alpha$, $\pm\alpha^{-1}$, along with the four "standard" points 0, ± 1, and ∞. To fully exploit this advantage, we aim to minimize computational costs in our quantum design. It is crucial to carefully determine the appropriate selection of the $4k$ evaluation points. The extremal points 0 and ∞ do not necessitate any operations for evaluation. To satisfy the condition of $n + m - 1 = 4k$, the two operands must be denoted by two polynomials, each possessing a degree of n and m, respectively. This implies that the evaluation of each pair of $\pm\alpha$ incurs a cost $(n+1)+(m+1) = (4k + 1)$. There are a total of $2k - 1$ couples. Hence, evaluation necessitates the consideration of $(4k + 1)(2k - 1)$ combinations. To address a wide range of unbalancements, Toom balanced addresses the issue of operand ratios that are unbalanced and cannot be handled well [6]. One such approach is to disregard the evaluation at ∞ and substitute it with a value of zero. A potentially more efficient approach might involve disregarding the assessment in 0, as the desired product from the evaluation in ∞ could potentially be shorter.

V. HIGH- AND HALF-DEGREE QUANTUM MULTIPLICATION DESIGN

Figure 3 depicts the visual interpretation of the recursive tree structures associated with Toom-Cook 20.5-way. Further, Figure 4 draws quantum circuits pertaining to the Toom-Cook 20.5-way multiplications design. The function block boxes are utilized as visual representations of the discrete stages encompassed in the construction process of the Toom-Cook quantum circuit. The multiplication algorithm employs a

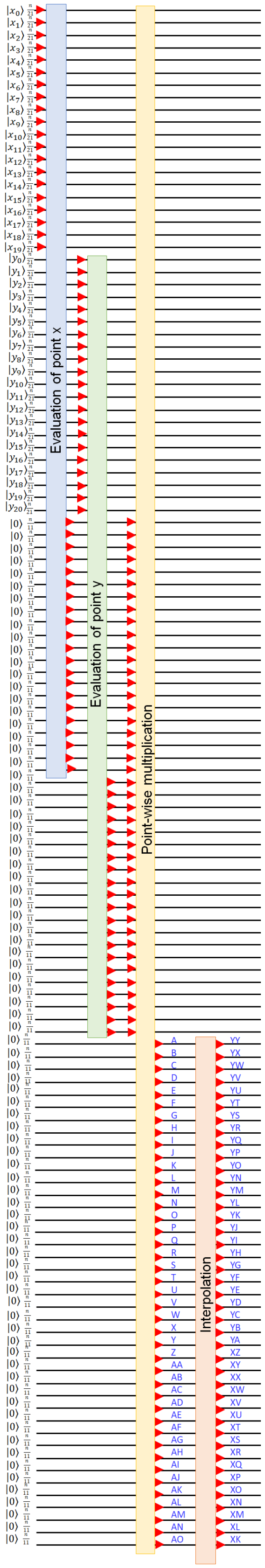

FIGURE 4. Quantum Design for the Toom-Cook 20.5-way Multiplication Algorithms.

quantum circuit wherein red triangles are employed to represent the input and output of each operation within the function blocks. A notation symbol is utilized to represent the quantum state of the input, where each line corresponds to a necessary register in the quantum circuit. The inclusion of triangles situated on the left side of a block serves to emphasize the positioning of its input entry point. The symbolization of the output location on the right side is represented by triangles. In order to preserve a sense of simplicity, the ancilla registers have been excluded from the visual representation.

The computational steps encompass the following steps: splitting, evaluation, recursive multiplication, interpolation, and recomposition, as expounded upon in prior literary investigations [5], [6], [8], [11]. The input operands, denoted as variables x and y, represent the quantities to be multiplied. The variable x is employed as a symbol to denote the entirety of the numerical input. The subscripts $x_0, x_1, x_{-1}, x_{-2}, \ldots$ are employed to denote the individual constituents of the input. Conversely, the symbols $x(0), x(1), x(-1), x(-2), \ldots$ are utilized to denote the outcomes derived from assessing the variable x at certain places.

In the splitting computation stage of Toom-Cook 20.5-way, the two given homogeneous polynomial inputs, represented by Equations 6 and 7 as x and y correspondingly, are divided into 20 and 21 smaller pieces of length $\frac{n}{21}$ each. These polynomials consist of n and m operands, respectively, 20 and 21, which align with the aforementioned coefficients and degrees.

$$\begin{aligned} x = {} & x_{19}s^{19j} + x_{18}s^{18j} + x_{17}s^{17j} + x_{16}s^{16j} + x_{15}s^{15j} + x_{14}s^{14j} \\ & + x_{13}s^{13j} + x_{12}s^{12j} + x_{11}s^{11j} + x_{10}s^{10j} + x_9s^{9j} + x_8s^{8j} \\ & + x_7s^{7j} + x_6s^{6j} + x_5s^{5j} + x_4s^{4j} + x_3s^{3j} + x_2s^{2j} \\ & + x_1s^{j} + x_0 \end{aligned} \tag{6}$$

$$\begin{aligned} y = {} & y_{20}s^{20j} + y_{19}s^{19j} + y_{18}s^{18j} + y_{17}s^{17j} + y_{16}s^{16j} + y_{15}s^{15j} \\ & + y_{14}s^{14j} + y_{13}s^{13j} + y_{12}s^{12j} + y_{11}s^{11j} + y_{10}s^{10j} + y_9s^{9j} \\ & + y_8s^{8j} + y_7s^{7j} + y_6s^{6j} + y_5s^{5j} + y_4s^{4j} + y_3s^{3j} + y_2s^{2j} \\ & + y_1s^{j} + y_0 \end{aligned} \tag{7}$$

In the beginning stages of Toom-Cook's splitting steps, when employing Toom-Cook's $k-$way algorithm to partition a given quantity into k segments, it is imperative to select a base $Y = x^b$ and compute the radix j value in the equations, which can be predetermined by utilizing Equation 8.

$$j = max\left\{ \left\lfloor \frac{\lceil \log_2 x \rceil}{21} \right\rfloor, \left\lfloor \frac{\lceil \log_2 y \rceil}{20} \right\rfloor \right\} + 1 \tag{8}$$

In the evaluation computation step, Values in Equation 9 are utilized to evaluate the points x and y, specifically, all 41 predefined evaluation points. In order to achieve evaluation as described in equation 10, 41 predefined evaluation points in Equation 9 are utilized. The utilization of the infinity value is intentionally avoided, as it aligns with the strategies outlined in the research conducted by [6]. One possible strategy involves disregarding the evaluation at infinity and replacing it with a value of zero.

$$\begin{aligned} & x_1 = 0, x_2 = 1,\ x_3 = -1,\ x_4 = 2,\ x_5 = -2,\ x_6 = 4, \\ & x_7 = -4, \\ & x_8 = 0.5,\ x_9 = -0.5,\ x_{10} = 0.25,\ x_{11} = -0.25, \\ & x_{12} = 0.125,\ x_{13} = -0.125,\ x_{14} = 0.0625, \\ & x_{15} = -0.0625,\ x_{16} = 0.03125,\ x_{17} = -0.03125, \\ & x_{18} = 0.015625,\ x_{19} = -0.015625,\ x_{20} = 0.0078125, \\ & x_{21} = -0.0078125,\ x_{22} = 0.00390625, \\ & x_{23} = -0.00390625,\ x_{24} = 0.001953125, \\ & x_{25} = -0.001953125,\ x_{26} = 0.000976565, \\ & x_{27} = -0.000976565,\ x_{28} = 0.0004882825, \\ & x_{29} = -0.0004882825,\ x_{30} = 0.0002441415, \\ & x_{31} = -0.0002441415,\ x_{32} = 0.00012206, \\ & x_{33} = -0.00012206,\ x_{34} = 0.00006103, \\ & x_{35} = -0.00006103,\ x_{36} = 0.000030515, \\ & x_{37} = -0.000030515,\ x_{38} = 0.0000152575, \\ & x_{39} = -0.0000152575,\ x_{40} = 0.00000762875, \\ & x_{41} = -0.00000762875 \end{aligned} \tag{9}$$

$$\begin{aligned} & x(0),\ x(1), x(-1),\ x(2),\ x(-2),\ x(4),\ x(-4),\ x(0.5), \\ & x(-0.5),\ x(0.25), x(-0.25),\ x(0.125),\ x(-0.125), \\ & x(0.0625),\ x(-0.0625),\ x(0.03125),\ x(-0.03125), \\ & x(0.015625),\ x(-0.015625),\ x(0.0078125), \\ & x(-0.0078125),\ x(0.00390625),\ x(-0.00390625), \\ & x(0.001953125),\ x(-0.001953125),\ x(0.000976565), \\ & x(-0.000976565),\ x(0.0004882825),\ x(-0.0004882825), \\ & x(0.0002441415),\ x(-0.0002441415),\ x(0.00012206), \\ & x(-0.00012206),\ x(0.00006103),\ x(-0.00006103), \\ & x(0.000030515),\ x(-0.000030515),\ x(0.0000152575), \\ & x(-0.0000152575),\ x(0.00000762875), \\ & x(-0.00000762875) \end{aligned} \tag{10}$$

Note that, in the interest of clarity, the exact equation used to calculate the evaluation points has been excluded. However, it can be inferred from the evaluation multiplication equation, Tables 3 and 4 in the Appendix. The evaluation points, represented as x and y in the evaluation step of the Toom-Cook 20.5-way multiplications design, are illustrated in the Appendix in Figure 5 and Figure 6, respectively. It is important to note that we propose the utilization of the free-division design as a means to mitigate the complexity associated with multiplication operations, particularly when dealing with high-degree multiplications, such as polynomial multiplication.

After the completion of the splitting and evaluation computation steps, the subsequent computational step for Toom-Cook 20.5-way involves recursive multiplication as the third step of computation in Toom-Cook. A single iteration of non-recursive point-wise multiplication for Toom-Cook

20.5-way multiplication utilizes a total of 41 multiplications, each with $\frac{n}{21}$ lengths. To multiply each value of Equation 10, the result is expressed in Tables 3 and 4 (in the Appendix), denoted as $A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, X, Y, Z, AA, AB, AC, AD, AE, AF, AG, AH, AI, AJ, AK, AL, AM, AN$ and AO respectively.

$$\begin{pmatrix} YY \\ YX \\ YW \\ YV \\ YU \\ YT \\ YS \\ YR \\ YQ \\ YP \\ YO \\ YN \\ YM \\ YL \\ YK \\ YJ \\ YI \\ YH \\ YG \\ YF \\ YE \\ YD \\ YC \\ YB \\ YA \\ XZ \\ XY \\ XX \\ XW \\ XV \\ XU \\ XT \\ XS \\ XR \\ XQ \\ XP \\ XO \\ XN \\ XM \\ XL \\ XK \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \\ L \\ M \\ N \\ O \\ P \\ Q \\ R \\ S \\ T \\ U \\ V \\ W \\ X \\ Y \\ Z \\ AA \\ AB \\ AC \\ AD \\ AE \\ AF \\ AG \\ AH \\ AI \\ AJ \\ AK \\ AL \\ AM \\ AN \\ AO \end{pmatrix} \tag{11}$$

For the fourth computation step, the interpolation, we present the mathematical formulation for the interpolation computation step that employs matrix operations. This procedure is the inverse operation of point multiplication, as depicted in Equation 11, as shown at the bottom of the previous page. The aforementioned procedure employs an inverse matrix derived from the coefficient sub-multiplication ($k_0 \ldots k_{40}$) in Tables 3 and 4, further clarified in Equation 11.

In this work, we utilize opposite points and inverse strategies, as previously discussed in the relevant literature [6] and subsection. This highlights the importance of consistently employing specific pairs of points to reduce the multiplication complexity. Conversely, there exist certain benefits that act as the principal impetus for examining the unbalanced Toom'n'half algorithm, which requires the utilization of $2n$ points.

In the context of the Toom-Cook 20.5-way algorithm, the recomposition computational step involves the recomposition originating from the interpolation result. The recomposition is denoted by a series of variables, namely YY, YX, YW, YV, YU, YT, YS, YR, YQ, YP, YO, YN, YM, YL, YK, YJ, YI, YH, YG, YF, YE, YD, YC, YB, YA, XZ, XY, XX, XW, XV, XU, XT, XS, XR, XQ, XP, XO, XN, XM, XL, and XK.

The mathematical expression xy in Equation 12 represents the recomposition outcome of the interpolation process, or in other words, the final result obtained from the Toom-Cook 20.5-way multiplication algorithm.

$$\begin{aligned} xy = {} & XK2^{40j} + XL2^{39j} + XM2^{38j} + XN2^{37j} + XO2^{36j} \\ & + XP2^{35j} + XQ2^{34j} + XR2^{33j} + XS2^{32j} + XT2^{31j} \\ & + XU2^{30j} + XV2^{29j} + XW2^{28j} + XX2^{27j} + XY2^{26j} \\ & + XZ2^{25j} + YA2^{24j} + YB2^{23j} + YC2^{22j} + YD2^{21j} \\ & + YE2^{20j} + YF2^{19j} + YG2^{18j} + YH2^{17j} + YI2^{16j} \\ & + YJ2^{15j} + YK2^{14j} + YL2^{13j} + YM2^{12j} + YN2^{11j} \\ & + YO2^{10j} + YP2^{9j} + YQ2^{8j} + YR2^{7j} + YS2^{6j} + YT2^{5j} \\ & + YU2^{4j} + YV2^{3j} + YW2^{2j} + YX2^{j} + YZ \end{aligned} \tag{12}$$

VI. RESULTS AND ANALYSIS

A. TOFFOLI GATE COUNT

The variable T_n is employed as a symbol to denote the cost associated with the execution of multiplication on two larger n-bit quantities using the Toom-Cook multiplier. Therefore, the term A_n represents costs linked to the addition or subtraction of n-bit. In order to implement an n-bit Toom-Cook 20.5-way multiplication, we carried out a total of 41 operations, which encompass $\frac{n}{21}$ sub-multiplications as well as three distinct types of adders that possess varying lengths. The number of operations required for $\frac{n}{21}$-bit adders is 130, while $\frac{2n}{21}$-bit adders require 1640 operations.

$$T_n = 41T_{\frac{n}{21}} + 130A_{\frac{n}{21}} + 1640A_{\frac{2n}{21}} \tag{13}$$

The Toffoli cost associated with a Toom-Cook 20.5 multiplication operation on n-bit can be calculated using Equation 13. The cost of recursive implementations increases with Equation 14, and Equation 15 becomes equivalent when the Toffoli cost of $A_n = 2n$ is substituted.

$$\begin{aligned} T_n = {} & 41^{\log_{21} n}T_1 + 130(A\frac{n}{21} + 65A\frac{n}{441} + \cdots + 65^{\log_{21}(n)-1}A_1) \\ & + 1640(A\frac{2n}{21} + 820A\frac{2n}{81} + \cdots + 640^{\log_{21}(n)-1}A_2) \end{aligned} \tag{14}$$

TABLE 2. Asymptotic Performance and Quantum Implementation Cost Multipliers Comparison.

No	Reference	Arithmetic Algorithm	Asymptotic Performance Analysis			Cost of Quantum Implementation of Multiplication			
			Qubit Count	Toffoli Count	Toffoli Depth	Qubit Count	Toffoli Count	Toffoli Depth	CNOT
1	Kepley and Steinwandt (2015, [33])	Karatsuba	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{\log_2^3})$	-	-	-	-	$\mathcal{O}(n^{\log_2 3})$
2	Parent et al. (2017, [27])	Karatsuba	$\mathcal{O}(n^{1.427})$	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{1.158})$	$n(\frac{3}{2})^{\frac{\log 2}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.427}$	$42n^{\log_2 3}$	$n(\frac{3}{2})^{1-\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.158}$	-
3	Dutta et al. (2018, [7])	Toom-Cook-2.5	$\mathcal{O}(n^{1.404})$	$\mathcal{O}(n^{\log_6^{16}})$	$\mathcal{O}(n^{1.143})$	$n(\frac{8}{3})^{\frac{\log 16}{(6\log 16-\log 6)}\log_6 n} \approx n^{1.404}$	$49n^{log_6 16}$	$n(\frac{8}{3})^{1-\frac{\log 16}{(2\log 16-\log 6)}\log_6 n} \approx n^{1.143}$	-
4	Larasati et al.(2021, [8])	Toom-Cook 3	$\mathcal{O}(n^{1.35})$	$O(n^2)$	$\mathcal{O}(n^{1.112})$	$n(\frac{5}{3})^{\frac{\log 5}{(2\log 5-\log 3)}\log_3 n} \approx n^{1.353}$	$8n^2+66n^{\log_3 5}-72$	$n(\frac{5}{3})^{1-\frac{\log 5}{(2\log 5-\log 3)}\log_3 n} \approx n^{1.112}$	-
5	Van Hoof (2020, [34])	Karatsuba	$3n$	$\mathcal{O}(n^{log_2 3})$	-	-	-	-	$\mathcal{O}(n^2)$
6	Putranto et al. (2023, [14])	Karatsuba	$3n$	$\mathcal{O}(n^{log_2 3})$	-	-	-	-	$\mathcal{O}(n^{log_2 3})$
7	Putranto et al. (2023, [11]	Toom-Cook 2-way	$\mathcal{O}(n^{1.589})$	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{1.217})$	$n(\frac{3}{2})^{\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.589}$	$34n^{\log_2 3}-32n$	$n(\frac{3}{2})^{1-\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.217}$	-
8	Putranto et al. (2023, [11]	Toom-Cook 4-way	$\mathcal{O}(n^{1.313})$	$\mathcal{O}(n^{\log_4 7})$	$\mathcal{O}(n^{1.09})$	$n(\frac{7}{4})^{\frac{\log 7}{(2\log 7-\log 4)}\log_4 n} \approx n^{1.313}$	$122n^{\log_4 7}-160n$	$n(\frac{7}{4})^{1-\frac{\log 7}{(2\log 7-\log 4)}\log_4 n} \approx n^{1.09}$	-
9	Putranto et al. (2023, [11]	Toom-Cook 8-way	$\mathcal{O}(n^{1.245})$	$\mathcal{O}(n^{\log_8 15})$	$\mathcal{O}(n^{1.0569})$	$n(\frac{15}{8})^{\frac{\log 15}{(2\log 15-\log 8)}\log_8 n} \approx n^{1.245}$	$112n^{\log_8 15}-128n$	$n(\frac{15}{8})^{1-\frac{\log 15}{(2\log 15-\log 8)}\log_8 n} \approx n^{1.0569}$	-
10	our	Toom-Cook 8.5-way	$\mathcal{O}(n^{1.236})$	$\mathcal{O}(n^{\log_9 17})$	$\mathcal{O}(n^{1.053})$	$n(\frac{17}{9})^{\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.236}$	$186n^{\log_9 17}-202n$	$n(\frac{17}{9})^{1-\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.053}$	-
11	our	Toom-Cook 10.5-way	$\mathcal{O}(n^{1.222})$	$\mathcal{O}(n^{\log_{11} 21})$	$\mathcal{O}(n^{1.047})$	$n(\frac{21}{11})^{\frac{\log 21}{(2\log 21-\log 11)}\log_{11} n} \approx n^{1.222}$	$206n^{\log_{11} 21}-132n$	$n(\frac{21}{11})^{1-\frac{\log 21}{(2\log 21-\log 11)}\log_1 1n} \approx n^{1.047}$	-
12	our	Toom-Cook 20.5-way	$\mathcal{O}(n^{1.186})$	$\mathcal{O}(n^{\log_{21} 41})$	$\mathcal{O}(n^{1.033})$	$n(\frac{41}{21})^{\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.186}$	$522n^{\log_{21} 41}-540n$	$n(\frac{41}{21})^{1-\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.033}$	-

$$T_n = 41^{\log_{21} n} + \sum_{i=0}^{\log_{21}(n)-1} \left[260n(\frac{41}{21})^i \right] \tag{15}$$

The Toffoli cost of a recursive implementation can be determined by utilizing the geometric series calculation $\sum_{i=0}^{m-1} r^i = \frac{1-r^m}{1-r}$, as denoted by Equation 16. However, the result obtained from Equation 16 does not incorporate the conventional practice of uncomputation carried out in a quantum environment. In Equation 17, we incorporated the concept of an uncomputed operation as a strategy to reduce a significant increase in the previously determined cost. It is of utmost importance to acknowledge that the term "clean cost" utilized in the following equation aligns with the definitions put forth by Larasati et al. [8] and Putranto et al. [11].

$$\begin{aligned} T_n &= 41^{\log_{21} n} + 2604n\left(\frac{1-(\frac{41}{21})^{\log_{21} n}}{1-(\frac{41}{21})}\right) \\ &= n^{\log_{21} 41} + 260n\left(\frac{1-n^{\log_{21}(\frac{41}{21})}}{1-(\frac{41}{21})}\right) \qquad (16) \\ m &= 261n^{\log_{21} 41} - 270n \end{aligned}$$

$$T_{n(clean)} = 522n^{\log_{41} 21} - 540n \tag{17}$$

B. SPACE-TIME COMPLEXITY ANALYSIS

A prior work from Bennett in [30] introduced the technique of reversible pebble games as a means to measure asymptotic improvements in performance, specifically in relation to space consumption within the framework of space-time complexity analysis. The utilization of this technique is widespread in the field of reversible computing, allowing for the analysis of time and space complexity analysis possible and enabling time-efficient finite-space computing [31]. This approach will enable us to assess the difference in the cost of the efficiently optimized multiplication and compare it with the findings of prior investigations. The optimal cost of multiplication was determined by implementing the methodologies described in the works of Parent et al. [27], Dutta et al. [7], Larasati et al. [8], and Putranto et al. [11].

The Toom-Cook 20.5-way algorithm involved the execution of 41 concurrent multiplications through a recursive process, resulting in the formation of a 21 structure. There are 41^l nodes of size $21^{-l}n$ for an input of size n at level l, and this input has a total circuit cost of $n(\frac{41}{21})^l$. The equation presented as Equation 18 represents the aggregate cost associated with the quinary tree. To ascertain the most suitable height k of a tree for achieving optimal performance, Equation 20 should be employed.

$$n\sum_{i=0}^{N}\left(\frac{41}{21}\right)^i, \quad N = \log_{21} n \tag{18}$$

$$n\sum_{i=0}^{N-k-1}\left(\frac{41}{21}\right)^i = \frac{1}{21^{N-k}}\sum_{i=0}^{k-1}\left(\frac{41}{21}\right)^i \tag{19}$$

In a manner identical to Equation 17, the identity of the geometric series enables us to locate the boundaries specified by Equation 20. Therefore, it is possible to decrease the amount of space required, as demonstrated by the qubit count equation presented in Equation 21. The result obtained from Equation 21, which is approximately equal to $\mathcal{O}(n^{1.245})$, is found to be smaller than the initially estimated space requirement determined by Equation 22, which is limited to the value $\mathcal{O}(n^{\log_8 15}) \approx \mathcal{O}(n^{n^{1.30229}})$.

$$k \leq \frac{N}{2-\frac{\log 21}{\log 41}} \approx 0.847 \tag{20}$$

$$QC = \mathcal{O}\left(n\left(\frac{41}{21}^{\left(\frac{\log 41}{2\log 41 - 2\log 21}\right)\log_{21} n}\right)\right) \approx \mathcal{O}(n^{1.186}) \tag{21}$$

$$n\sum_{k=0}^{\log_{41} n-1}\left(\frac{41}{21}\right)^k$$

TABLE 3. The Evaluation Multiplication.

A	=	x0	*	y0
B	=	(((x0 * 1) + (x1 * 1) + (x2 * 1) + (x3 * 1) + (x4 * 1) + (x5 * 1) + (x6 * 1) + (x7 * 1) + (x8 * 1) + (x9 * 1) + (x10 * 1) + (x11 * 1) + (x12 * 1) + (x13 * 1) + (x14 * 1) + (x15 * 1) + (x16 * 1) + (x17 * 1) + (x18 * 1) + (x19 * 1))	*	((y0 * 1) + (y1 * 1) + (y2 * 1) + (y3 * 1) + (y4 * 1) + (y5 * 1) + (y6 * 1) + (y7 * 1) + (y8 * 1) + (y9 * 1) + (y10 * 1) + (y11 * 1) + (y12 * 1) + (y13 * 1) + (y14 * 1) + (y15 * 1) + (y16 * 1) + (y17 * 1) + (y18 * 1) + (y19 * 1) + (y20 * 1)))
C	=	(((x0 * 1) + (x1 * −1) + (x2 * 1) + (x3 * −1) + (x4 * 1) + (x5 * −1) + (x6 * 1) + (x7 * −1) + (x8 * 1) + (x9 * −1) + (x10 * 1) + (x11 * −1) + (x12 * 1) + (x13 * −1) + (x14 * 1) + (x15 * −1) + (x16 * 1) + (x17 * −1) + (x18 * 1) + (x19 * −1))	*	((y0 * 1) + (y1 * −1) + (y2 * 1) + (y3 * −1) + (y4 * 1) + (y5 * −1) + (y6 * 1) + (y7 * −1) + (y8 * 1) + (y9 * −1) + (y10 * 1) + (y11 * −1) + (y12 * 1) + (y13 * −1) + (y14 * 1) + (y15 * −1) + (y16 * 1) + (y17 * −1) + (y18 * 1) + (y19 * −1) + (y20 * 1)))
D	=	(((x0 * 1) + (x1 * 2) + (x2 * 4) + (x3 * 8) + (x4 * 16) + (x5 * 32) + (x6 * 64) + (x7 * 128) + (x8 * 256) + (x9 * 512) + (x10 * 1024) + (x11 * 2048) + (x12 * 4096) + (x13 * 8192) + (x14 * 16384) + (x15 * 32768) + (x16 * 65536) + (x17 * 131072) + (x18 * 262144) + (x19 * 524288))	*	((y0 * 1) + (y1 * 2) + (y2 * 4) + (y3 * 8) + (y4 * 16) + (y5 * 32) + (y6 * 64) + (y7 * 128) + (y8 * 256) + (y9 * 512) + (y10 * 1024) + (y11 * 2048) + (y12 * 4096) + (y13 * 8192) + (y14 * 16384) + (y15 * 32768) + (y16 * 65536) + (y17 * 131072) + (y18 * 262144) + (y19 * 524288) + (y20 * 1048576)))
E	=	(((x0 * 1) + (x1 * −2) + (x2 * 4) + (x3 * −8) + (x4 * 16) + (x5 * −32) + (x6 * 64) + (x7 * −128) + (x8 * 256) + (x9 * −512) + (x10 * 1024) + (x11 * −2048) + (x12 * 4096) + (x13 * −8192) + (x14 * 16384) + (x15 * −32768) + (x16 * 65536) + (x17 * −131072) + (x18 * 262144) + (x19 * −524288))	*	((y0 * 1) + (y1 * −2) + (y2 * 4) + (y3 * −8) + (y4 * 16) + (y5 * −32) + (y6 * 64) + (y7 * −128) + (y8 * 256) + (y9 * −512) + (y10 * 1024) + (y11 * −2048) + (y12 * 4096) + (y13 * −8192) + (y14 * 16384) + (y15 * −32768) + (y16 * 65536) + (y17 * −131072) + (y18 * 262144) + (y19 * −524288) + (y20 * 1048576)))
F	=	(((x0 * 1) + (x1 * 4) + (x2 * 16) + (x3 * 64) + (x4 * 256) + (x5 * 1024) + (x6 * 4096) + (x7 * 16384) + (x8 * 65536) + (x9 * 262144) + (x10 * 1048576) + (x11 * 4194304) + (x12 * 16777216) + (x13 * 67108864) + (x14 * 268435456) + (x15 * 1073741824) + (x16 * 4294967296) + (x17 * 17179869184) + (x18 * 68719476736) + (x19 * 274877906944))	*	((y0 * 1) + (y1 * 4) + (y2 * 16) + (y3 * 64) + (y4 * 256) + (y5 * 1024) + (y6 * 4096) + (y7 * 16384) + (y8 * 65536) + (y9 * 262144) + (y10 * 1048576) + (y11 * 4194304) + (y12 * 16777216) + (y13 * 67108864) + (y14 * 268435456) + (y15 * 1073741824) + (y16 * 4294967296) + (y17 * 17179869184) + (y18 * 68719476736) + (y19 * 274877906944) + (y20 * 1099511627776)))
G	=	(((x0 * 1) + (x1 * −4) + (x2 * 16) + (x3 * −64) + (x4 * 256) + (x5 * −1024) + (x6 * 4096) + (x7 * −16384) + (x8 * 65536) + (x9 * −262144) + (x10 * 1048576) + (x11 * −4194304) + (x12 * 16777216) + (x13 * −67108864) + (x14 * 268435456) + (x15 * −1073741824) + (x16 * 4294967296) + (x17 * −17179869184) + (x18 * 68719476736) + (x19 * −274877906944))	*	((y0 * 1) + (y1 * −4) + (y2 * 16) + (y3 * −64) + (y4 * 256) + (y5 * −1024) + (y6 * 4096) + (y7 * −16384) + (y8 * 65536) + (y9 * −262144) + (y10 * 1048576) + (y11 * −4194304) + (y12 * 16777216) + (y13 * −67108864) + (y14 * 268435456) + (y15 * −1073741824) + (y16 * 4294967296) + (y17 * −17179869184) + (y18 * 68719476736) + (y19 * −274877906944) + (y20 * 1099511627776)))
H	=	(((x0 * 1) + (x1 * 0.5) + (x2 * 0.25) + (x3 * 0.125) + (x4 * 0.0625) + (x5 * 0.03125) + (x6 * 0.015625) + (x7 * 0.0078125) + (x8 * 0.00390625) + (x9 * 0.001953125) + (x10 * 0.0009765625) + (x11 * 0.00048828125) + (x12 * 0.000244140625) + (x13 * 0.0001220703125) + (x14 * 0.00006103515625) + (x15 * 0.000030517578125) + (x16 * 0.0000152587890625) + (x17 * 0.00000762939453125) + (x18 * 0.000003814697265625) + (x19 * 1.9073486328125E − 06))	*	((y0 * 1) + (y1 * 0.5) + (y2 * 0.25) + (y3 * 0.125) + (y4 * 0.0625) + (y5 * 0.03125) + (y6 * 0.015625) + (y7 * 0.0078125) + (y8 * 0.00390625) + (y9 * 0.001953125) + (y10 * 0.0009765625) + (y11 * 0.00048828125) + (y12 * 0.000244140625) + (y13 * 0.0001220703125) + (y14 * 0.00006103515625) + (y15 * 0.000030517578125) + (y16 * 0.0000152587890625) + (y17 * 0.00000762939453125) + (y18 * 0.000003814697265625) + (y19 * 1.9073486328125E − 06) + (y20 * 9.5367431640625E − 07)))
I	=	(((x0 * 1) + (x1 * −0.5) + (x2 * 0.25) + (x3 * −0.125) + (x4 * 0.0625) + (x5 * −0.03125) + (x6 * 0.015625) + (x7 * −0.0078125) + (x8 * 0.00390625) + (x9 * −0.001953125) + (x10 * 0.0009765625) + (x11 * −0.00048828125) + (x12 * 0.000244140625) + (x13 * −0.0001220703125) + (x14 * 0.00006103515625) + (x15 * −0.000030517578125) + (x16 * 0.0000152587890625) + (x17 * −0.00000762939453125) + (x18 * 0.000003814697265625) + (x19 * −1.9073486328125E − 06))	*	((y0 * 1) + (y1 * −0.5) + (y2 * 0.25) + (y3 * −0.125) + (y4 * 0.0625) + (y5 * −0.03125) + (y6 * 0.015625) + (y7 * −0.0078125) + (y8 * 0.00390625) + (y9 * −0.001953125) + (y10 * 0.0009765625) + (y11 * −0.00048828125) + (y12 * 0.000244140625) + (y13 * −0.0001220703125) + (y14 * 0.00006103515625) + (y15 * −0.000030517578125) + (y16 * 0.0000152587890625) + (y17 * −0.00000762939453125) + (y18 * 0.000003814697265625) + (y19 * −1.9073486328125E − 06) + (y20 * 9.5367431640625E − 07)))
J	=	(((x0 * 1) + (x1 * 0.25) + (x2 * 0.0625) + (x3 * 0.015625) + (x4 * 0.00390625) + (x5 * 0.0009765625) + (x6 * 0.000244140625) + (x7 * 0.00006103515625) + (x8 * 0.0000152587890625) + (x9 * 0.000003814697265625) + (x10 * 9.5367431640625E − 07) + (x11 * 2.38418579101562E − 07) + (x12 * 5.96046447753906E − 08) + (x13 * 1.49011611938477E − 08) + (x14 * 3.72529029846191E − 09) + (x15 * 9.31322574615479E − 10) + (x16 * 2.3283064365387E − 10) + (x17 * 5.82076609134674E − 11) + (x18 * 1.45519152283669E − 11) + (x19 * 3.63797880709171E − 12))	*	((y0 * 1) + (y1 * 0.25) + (y2 * 0.0625) + (y3 * 0.015625) + (y4 * 0.00390625) + (y5 * 0.0009765625) + (y6 * 0.000244140625) + (y7 * 0.00006103515625) + (y8 * 0.0000152587890625) + (y9 * 0.000003814697265625) + (y10 * 9.5367431640625E − 07) + (y11 * 2.38418579101562E − 07) + (y12 * 5.96046447753906E − 08) + (y13 * 1.49011611938477E − 08) + (y14 * 3.72529029846191E − 09) + (y15 * 9.31322574615479E − 10) + (y16 * 2.3283064365387E − 10) + (y17 * 5.82076609134674E − 11) + (y18 * 1.45519152283669E − 11) + (y19 * 3.63797880709171E − 12) + (y20 * 9.09494701772928E − 13)))
K	=	(((x0 * 1) + (x1 * −0.25) + (x2 * 0.0625) + (x3 * −0.015625) + (x4 * 0.00390625) + (x5 * −0.0009765625) + (x6 * 0.000244140625) + (x7 * −0.00006103515625) + (x8 * 0.0000152587890625) + (x9 * −0.000003814697265625) + (x10 * 9.5367431640625E − 07) + (x11 * −2.38418579101562E − 07) + (x12 * 5.96046447753906E − 08) + (x13 * −1.49011611938477E − 08) + (x14 * 3.72529029846191E − 09) + (x15 * −9.31322574615479E − 10) + (x16 * 2.3283064365387E − 10) + (x17 * −5.82076609134674E − 11) + (x18 * 1.45519152283669E − 11) + (x19 * −3.63797880709171E − 12))	*	((y0 * 1) + (y1 * −0.25) + (y2 * 0.0625) + (y3 * −0.015625) + (y4 * 0.00390625) + (y5 * −0.0009765625) + (y6 * 0.000244140625) + (y7 * −0.00006103515625) + (y8 * 0.0000152587890625) + (y9 * −0.000003814697265625) + (y10 * 9.5367431640625E − 07) + (y11 * −2.38418579101562E − 07) + (y12 * 5.96046447753906E − 08) + (y13 * −1.49011611938477E − 08) + (y14 * 3.72529029846191E − 09) + (y15 * −9.31322574615479E − 10) + (y16 * 2.3283064365387E − 10) + (y17 * −5.82076609134674E − 11) + (y18 * 1.45519152283669E − 11) + (y19 * −3.63797880709171E − 12) + (y20 * 9.09494701772928E − 13)))
L	=	(((x0 * 1) + (x1 * 0.125) + (x2 * 0.015625) + (x3 * 0.001953125) + (x4 * 0.000244140625) + (x5 * 0.000030517578125) + (x6 * 0.000003814697265625) + (x7 * 4.76837158203125E − 07) + (x8 * 5.96046447753906E − 08) + (x9 * 7.45058059692383E − 09) + (x10 * 9.31322574615479E − 10) + (x11 * 1.16415321826935E − 10) + (x12 * 1.45519152283669E − 11) + (x13 * 1.81898940354586E − 12) + (x14 * 2.27373675443232E − 13) + (x15 * 2.8421709430404E − 14) + (x16 * 3.5527136788005E − 15) + (x17 * 4.44089209850063E − 16) + (x18 * 5.55111512312578E − 17) + (x19 * 6.93889390390723E − 18))	*	((y0 * 1) + (y1 * 0.125) + (y2 * 0.015625) + (y3 * 0.001953125) + (y4 * 0.000244140625) + (y5 * 0.000030517578125) + (y6 * 0.000003814697265625) + (y7 * 4.76837158203125E − 07) + (y8 * 5.96046447753906E − 08) + (y9 * 7.45058059692383E − 09) + (y10 * 9.31322574615479E − 10) + (y11 * 1.16415321826935E − 10) + (y12 * 1.45519152283669E − 11) + (y13 * 1.81898940354586E − 12) + (y14 * 2.27373675443232E − 13) + (y15 * 2.8421709430404E − 14) + (y16 * 3.5527136788005E − 15) + (y17 * 4.44089209850063E − 16) + (y18 * 5.55111512312578E − 17) + (y19 * 6.93889390390723E − 18) + (y20 * 8.67361737988404E − 19)))
M	=	(((x0 * 1) + (x1 * −0.125) + (x2 * 0.015625) + (x3 * −0.001953125) + (x4 * 0.000244140625) + (x5 * −0.000030517578125) + (x6 * 0.000003814697265625) + (x7 * −4.76837158203125E − 07) + (x8 * 5.96046447753906E − 08) + (x9 * −7.45058059692383E − 09) + (x10 * 9.31322574615479E − 10) + (x11 * −1.16415321826935E − 10) + (x12 * 1.45519152283669E − 11) + (x13 * −1.81898940354586E − 12) + (x14 * 2.27373675443232E − 13) + (x15 * −2.8421709430404E − 14) + (x16 * 3.5527136788005E − 15) + (x17 * −4.44089209850063E − 16) + (x18 * 5.55111512312578E − 17) + (x19 * −6.93889390390723E − 18))	*	((y0 * 1) + (y1 * −0.125) + (y2 * 0.015625) + (y3 * −0.001953125) + (y4 * 0.000244140625) + (y5 * −0.000030517578125) + (y6 * 0.000003814697265625) + (y7 * −4.76837158203125E − 07) + (y8 * 5.96046447753906E − 08) + (y9 * −7.45058059692383E − 09) + (y10 * 9.31322574615479E − 10) + (y11 * −1.16415321826935E − 10) + (y12 * 1.45519152283669E − 11) + (y13 * −1.81898940354586E − 12) + (y14 * 2.27373675443232E − 13) + (y15 * −2.8421709430404E − 14) + (y16 * 3.5527136788005E − 15) + (y17 * −4.44089209850063E − 16) + (y18 * 5.55111512312578E − 17) + (y19 * −6.93889390390723E − 18) + (y20 * 8.67361737988404E − 19)))
N	=	(((x0 * 1) + (x1 * 0.0625) + (x2 * 0.00390625) + (x3 * 0.000244140625) + (x4 * 0.0000152587890625) + (x5 * 9.5367431640625E − 07) + (x6 * 5.96046447753906E − 08) + (x7 * 3.72529029846191E − 09) + (x8 * 2.3283064365387E − 10) + (x9 * 1.45519152283669E − 11) + (x10 * 9.09494701772928E − 13) + (x11 * 5.6843418860808E − 14) + (x12 * 3.5527136788005E − 15) + (x13 * 2.22044604925031E − 16) + (x14 * 1.38777878078145E − 17) + (x15 * 8.67361737988404E − 19) + (x16 * 5.42101086242752E − 20) + (x17 * 3.3881317890172E − 21) + (x18 * 2.11758236813575E − 22) + (x19 * 1.32348898008484E − 23))	*	((y0 * 1) + (y1 * 0.0625) + (y2 * 0.00390625) + (y3 * 0.000244140625) + (y4 * 0.0000152587890625) + (y5 * 9.5367431640625E − 07) + (y6 * 5.96046447753906E − 08) + (y7 * 3.72529029846191E − 09) + (y8 * 2.3283064365387E − 10) + (y9 * 1.45519152283669E − 11) + (y10 * 9.09494701772928E − 13) + (y11 * 5.6843418860808E − 14) + (y12 * 3.5527136788005E − 15) + (y13 * 2.22044604925031E − 16) + (y14 * 1.38777878078145E − 17) + (y15 * 8.67361737988404E − 19) + (y16 * 5.42101086242752E − 20) + (y17 * 3.3881317890172E − 21) + (y18 * 2.11758236813575E − 22) + (y19 * 1.32348898008484E − 23) + (y20 * 8.27180612553028E − 25)))
O	=	(((x0 * 1) + (x1 * −0.0625) + (x2 * 0.00390625) + (x3 * −0.000244140625) + (x4 * 0.0000152587890625) + (x5 * −9.5367431640625E − 07) + (x6 * 5.96046447753906E − 08) + (x7 * −3.72529029846191E − 09) + (x8 * 2.3283064365387E − 10) + (x9 * −1.45519152283669E − 11) + (x10 * 9.09494701772928E − 13) + (x11 * −5.6843418860808E − 14) + (x12 * 3.5527136788005E − 15) + (x13 * −2.22044604925031E − 16) + (x14 * 1.38777878078145E − 17) + (x15 * −8.67361737988404E − 19) + (x16 * 5.42101086242752E − 20) + (x17 * −3.3881317890172E − 21) + (x18 * 2.11758236813575E − 22) + (x19 * −1.32348898008484E − 23))	*	((y0 * 1) + (y1 * −0.0625) + (y2 * 0.00390625) + (y3 * −0.000244140625) + (y4 * 0.0000152587890625) + (y5 * −9.5367431640625E − 07) + (y6 * 5.96046447753906E − 08) + (y7 * −3.72529029846191E − 09) + (y8 * 2.3283064365387E − 10) + (y9 * −1.45519152283669E − 11) + (y10 * 9.09494701772928E − 13) + (y11 * −5.6843418860808E − 14) + (y12 * 3.5527136788005E − 15) + (y13 * −2.22044604925031E − 16) + (y14 * 1.38777878078145E − 17) + (y15 * −8.67361737988404E − 19) + (y16 * 5.42101086242752E − 20) + (y17 * −3.3881317890172E − 21) + (y18 * 2.11758236813575E − 22) + (y19 * −1.32348898008484E − 23) + (y20 * 8.27180612553028E − 25)))
P	=	(((x0 * 1) + (x1 * 0.03125) + (x2 * 0.0009765625) + (x3 * 0.000030517578125) + (x4 * 9.5367431640625E − 07) + (x5 * 2.98023223876953E − 08) + (x6 * 9.31322574615479E − 10) + (x7 * 2.91038304567337E − 11) + (x8 * 9.09494701772928E − 13) + (x9 * 2.8421709430404E − 14) + (x10 * 8.88178419700125E − 16) + (x11 * 2.77555756156289E − 17) + (x12 * 8.67361737988404E − 19) + (x13 * 2.71050543121376E − 20) + (x14 * 8.470329472543E − 22) + (x15 * 2.64697796016969E − 23) + (x16 * 8.27180612553028E − 25) + (x17 * 2.58493941422821E − 26) + (x18 * 8.07793566946316E − 28) + (x19 * 2.52435489670724E − 29))	*	((y0 * 1) + (y1 * 0.03125) + (y2 * 0.0009765625) + (y3 * 0.000030517578125) + (y4 * 9.5367431640625E − 07) + (y5 * 2.98023223876953E − 08) + (y6 * 9.31322574615479E − 10) + (y7 * 2.91038304567337E − 11) + (y8 * 9.09494701772928E − 13) + (y9 * 2.8421709430404E − 14) + (y10 * 8.88178419700125E − 16) + (y11 * 2.77555756156289E − 17) + (y12 * 8.67361737988404E − 19) + (y13 * 2.71050543121376E − 20) + (y14 * 8.470329472543E − 22) + (y15 * 2.64697796016969E − 23) + (y16 * 8.27180612553028E − 25) + (y17 * 2.58493941422821E − 26) + (y18 * 8.07793566946316E − 28) + (y19 * 2.52435489670724E − 29) + (y20 * 7.88860905221012E − 31)))
Q	=	(((x0 * 1) + (x1 * −0.03125) + (x2 * 0.0009765625) + (x3 * −0.000030517578125) + (x4 * 9.5367431640625E − 07) + (x5 * −2.98023223876953E − 08) + (x6 * 9.31322574615479E − 10) + (x7 * −2.91038304567337E − 11) + (x8 * 9.09494701772928E − 13) + (x9 * −2.8421709430404E − 14) + (x10 * 8.88178419700125E − 16) + (x11 * −2.77555756156289E − 17) + (x12 * 8.67361737988404E − 19) + (x13 * −2.71050543121376E − 20) + (x14 * 8.470329472543E − 22) + (x15 * −2.64697796016969E − 23) + (x16 * 8.27180612553028E − 25) + (x17 * −2.58493941422821E − 26) + (x18 * 8.07793566946316E − 28) + (x19 * −2.52435489670724E − 29))	*	((y0 * 1) + (y1 * −0.03125) + (y2 * 0.0009765625) + (y3 * −0.000030517578125) + (y4 * 9.5367431640625E − 07) + (y5 * −2.98023223876953E − 08) + (y6 * 9.31322574615479E − 10) + (y7 * −2.91038304567337E − 11) + (y8 * 9.09494701772928E − 13) + (y9 * −2.8421709430404E − 14) + (y10 * 8.88178419700125E − 16) + (y11 * −2.77555756156289E − 17) + (y12 * 8.67361737988404E − 19) + (y13 * −2.71050543121376E − 20) + (y14 * 8.470329472543E − 22) + (y15 * −2.64697796016969E − 23) + (y16 * 8.27180612553028E − 25) + (y17 * −2.58493941422821E − 26) + (y18 * 8.07793566946316E − 28) + (y19 * −2.52435489670724E − 29) + (y20 * 7.88860905221012E − 31)))
R	=	(((x0 * 1) + (x1 * 0.015625) + (x2 * 0.000244140625) + (x3 * 0.000003814697265625) + (x4 * 5.96046447753906E − 08) + (x5 * 9.31322574615479E − 10) + (x6 * 1.45519152283669E − 11) + (x7 * 2.27373675443232E − 13) + (x8 * 3.5527136788005E − 15) + (x9 * 5.55111512312578E − 17) + (x10 * 8.67361737988404E − 19) + (x11 * 1.35525271560688E − 20) + (x12 * 2.11758236813575E − 22) + (x13 * 3.30872245021211E − 24) + (x14 * 5.16987882845642E − 26) + (x15 * 8.07793566946316E − 28) + (x16 * 1.26217744835362E − 29) + (x17 * 1.97215226305253E − 31) + (x18 * 3.08148791101958E − 33) + (x19 * 4.81482486096809E − 35))	*	((y0 * 1) + (y1 * 0.015625) + (y2 * 0.000244140625) + (y3 * 0.000003814697265625) + (y4 * 5.96046447753906E − 08) + (y5 * 9.31322574615479E − 10) + (y6 * 1.45519152283669E − 11) + (y7 * 2.27373675443232E − 13) + (y8 * 3.5527136788005E − 15) + (y9 * 5.55111512312578E − 17) + (y10 * 8.67361737988404E − 19) + (y11 * 1.35525271560688E − 20) + (y12 * 2.11758236813575E − 22) + (y13 * 3.30872245021211E − 24) + (y14 * 5.16987882845642E − 26) + (y15 * 8.07793566946316E − 28) + (y16 * 1.26217744835362E − 29) + (y17 * 1.97215226305253E − 31) + (y18 * 3.08148791101958E − 33) + (y19 * 4.81482486096809E − 35) + (y20 * 7.52316384526264E − 37)))
S	=	(((x0 * 1) + (x1 * −0.015625) + (x2 * 0.000244140625) + (x3 * −0.000003814697265625) + (x4 * 5.96046447753906E − 08) + (x5 * −9.31322574615479E − 10) + (x6 * 1.45519152283669E − 11) + (x7 * −2.27373675443232E − 13) + (x8 * 3.5527136788005E − 15) + (x9 * −5.55111512312578E − 17) + (x10 * 8.67361737988404E − 19) + (x11 * −1.35525271560688E − 20) + (x12 * 2.11758236813575E − 22) + (x13 * −3.30872245021211E − 24) + (x14 * 5.16987882845642E − 26) + (x15 * −8.07793566946316E − 28) + (x16 * 1.26217744835362E − 29) + (x17 * −1.97215226305253E − 31) + (x18 * 3.08148791101958E − 33) + (x19 * −4.81482486096809E − 35))	*	((y0 * 1) + (y1 * −0.015625) + (y2 * 0.000244140625) + (y3 * −0.000003814697265625) + (y4 * 5.96046447753906E − 08) + (y5 * −9.31322574615479E − 10) + (y6 * 1.45519152283669E − 11) + (y7 * −2.27373675443232E − 13) + (y8 * 3.5527136788005E − 15) + (y9 * −5.55111512312578E − 17) + (y10 * 8.67361737988404E − 19) + (y11 * −1.35525271560688E − 20) + (y12 * 2.11758236813575E − 22) + (y13 * −3.30872245021211E − 24) + (y14 * 5.16987882845642E − 26) + (y15 * −8.07793566946316E − 28) + (y16 * 1.26217744835362E − 29) + (y17 * −1.97215226305253E − 31) + (y18 * 3.08148791101958E − 33) + (y19 * −4.81482486096809E − 35) + (y20 * 7.52316384526264E − 37)))
T	=	(((x0 * 1) + (x1 * 0.0078125) + (x2 * 0.00006103515625) + (x3 * 4.76837158203125E − 07) + (x4 * 3.72529029846191E − 09) + (x5 * 2.91038304567337E − 11) + (x6 * 2.27373675443232E − 13) + (x7 * 1.77635683940025E − 15) + (x8 * 1.38777878078145E − 17) + (x9 * 1.0842021724855E − 19) + (x10 * 8.470329472543E − 22) + (x11 * 6.61744490042422E − 24) + (x12 * 5.16987882845642E − 26) + (x13 * 4.03896783473158E − 28) + (x14 * 3.15544362088405E − 30) + (x15 * 2.46519032881566E − 32) + (x16 * 1.92592994438724E − 34) + (x17 * 1.50463276905253E − 36) + (x18 * 1.17549435082229E − 38) + (x19 * 9.18354961579912E − 41))	*	((y0 * 1) + (y1 * 0.0078125) + (y2 * 0.00006103515625) + (y3 * 4.76837158203125E − 07) + (y4 * 3.72529029846191E − 09) + (y5 * 2.91038304567337E − 11) + (y6 * 2.27373675443232E − 13) + (y7 * 1.77635683940025E − 15) + (y8 * 1.38777878078145E − 17) + (y9 * 1.0842021724855E − 19) + (y10 * 8.470329472543E − 22) + (y11 * 6.61744490042422E − 24) + (y12 * 5.16987882845642E − 26) + (y13 * 4.03896783473158E − 28) + (y14 * 3.15544362088405E − 30) + (y15 * 2.46519032881566E − 32) + (y16 * 1.92592994438724E − 34) + (y17 * 1.50463276905253E − 36) + (y18 * 1.17549435082229E − 38) + (y19 * 9.18354961579912E − 41) + (y20 * 7.17464813734306E − 43)))
U	=	(((x0 * 1) + (x1 * −0.0078125) + (x2 * 0.00006103515625) + (x3 * −4.76837158203125E − 07) + (x4 * 3.72529029846191E − 09) + (x5 * −2.91038304567337E − 11) + (x6 * 2.27373675443232E − 13) + (x7 * −1.77635683940025E − 15) + (x8 * 1.38777878078145E − 17) + (x9 * −1.0842021724855E − 19) + (x10 * 8.470329472543E − 22) + (x11 * −6.61744490042422E − 24) + (x12 * 5.16987882845642E − 26) + (x13 * −4.03896783473158E − 28) + (x14 * 3.15544362088405E − 30) + (x15 * −2.46519032881566E − 32) + (x16 * 1.92592994438724E − 34) + (x17 * −1.50463276905253E − 36) + (x18 * 1.17549435082229E − 38) + (x19 * −9.18354961579912E − 41))	*	((y0 * 1) + (y1 * −0.0078125) + (y2 * 0.00006103515625) + (y3 * −4.76837158203125E − 07) + (y4 * 3.72529029846191E − 09) + (y5 * −2.91038304567337E − 11) + (y6 * 2.27373675443232E − 13) + (y7 * −1.77635683940025E − 15) + (y8 * 1.38777878078145E − 17) + (y9 * −1.0842021724855E − 19) + (y10 * 8.470329472543E − 22) + (y11 * −6.61744490042422E − 24) + (y12 * 5.16987882845642E − 26) + (y13 * −4.03896783473158E − 28) + (y14 * 3.15544362088405E − 30) + (y15 * −2.46519032881566E − 32) + (y16 * 1.92592994438724E − 34) + (y17 * −1.50463276905253E − 36) + (y18 * 1.17549435082229E − 38) + (y19 * −9.18354961579912E − 41) + (y20 * 7.17464813734306E − 43)))
V	=	(((x0 * 1) + (x1 * 0.00390625) + (x2 * 0.0000152587890625) + (x3 * 5.96046447753906E − 08) + (x4 * 2.3283064365387E − 10) + (x5 * 9.09494701772928E − 13) + (x6 * 3.5527136788005E − 15) + (x7 * 1.38777878078145E − 17) + (x8 * 5.42101086242752E − 20) + (x9 * 2.11758236813575E − 22) + (x10 * 8.27180612553028E − 25) + (x11 * 3.23117426778526E − 27) + (x12 * 1.26217744835362E − 29) + (x13 * 4.93038065763132E − 32) + (x14 * 1.92592994438724E − 34) + (x15 * 7.52316384526264E − 37) + (x16 * 2.93873587705572E − 39) + (x17 * 1.14794370197489E − 41) + (x18 * 4.48415508583941E − 44) + (x19 * 1.75162308040602E − 46))	*	((y0 * 1) + (y1 * 0.00390625) + (y2 * 0.0000152587890625) + (y3 * 5.96046447753906E − 08) + (y4 * 2.3283064365387E − 10) + (y5 * 9.09494701772928E − 13) + (y6 * 3.5527136788005E − 15) + (y7 * 1.38777878078145E − 17) + (y8 * 5.42101086242752E − 20) + (y9 * 2.11758236813575E − 22) + (y10 * 8.27180612553028E − 25) + (y11 * 3.23117426778526E − 27) + (y12 * 1.26217744835362E − 29) + (y13 * 4.93038065763132E − 32) + (y14 * 1.92592994438724E − 34) + (y15 * 7.52316384526264E − 37) + (y16 * 2.93873587705572E − 39) + (y17 * 1.14794370197489E − 41) + (y18 * 4.48415508583941E − 44) + (y19 * 1.75162308040602E − 46) + (y20 * 6.84227765783602E − 49)))
W	=	(((x0 * 1) + (x1 * −0.00390625) + (x2 * 0.0000152587890625) + (x3 * −5.96046447753906E − 08) + (x4 * 2.3283064365387E − 10) + (x5 * −9.09494701772928E − 13) + (x6 * 3.5527136788005E − 15) + (x7 * −1.38777878078145E − 17) + (x8 * 5.42101086242752E − 20) + (x9 * −2.11758236813575E − 22) + (x10 * 8.27180612553028E − 25) + (x11 * −3.23117426778526E − 27) + (x12 * 1.26217744835362E − 29) + (x13 * −4.93038065763132E − 32) + (x14 * 1.92592994438724E − 34) + (x15 * −7.52316384526264E − 37) + (x16 * 2.93873587705572E − 39) + (x17 * −1.14794370197489E − 41) + (x18 * 4.48415508583941E − 44) + (x19 * −1.75162308040602E − 46))	*	((y0 * 1) + (y1 * −0.00390625) + (y2 * 0.0000152587890625) + (y3 * −5.96046447753906E − 08) + (y4 * 2.3283064365387E − 10) + (y5 * −9.09494701772928E − 13) + (y6 * 3.5527136788005E − 15) + (y7 * −1.38777878078145E − 17) + (y8 * 5.42101086242752E − 20) + (y9 * −2.11758236813575E − 22) + (y10 * 8.27180612553028E − 25) + (y11 * −3.23117426778526E − 27) + (y12 * 1.26217744835362E − 29) + (y13 * −4.93038065763132E − 32) + (y14 * 1.92592994438724E − 34) + (y15 * −7.52316384526264E − 37) + (y16 * 2.93873587705572E − 39) + (y17 * −1.14794370197489E − 41) + (y18 * 4.48415508583941E − 44) + (y19 * −1.75162308040602E − 46) + (y20 * 6.84227765783602E − 49)))
X	=	(((x0 * 1) + (x1 * 0.001953125) + (x2 * 0.000003814697265625) + (x3 * 7.45058059692383E − 09) + (x4 * 1.45519152283669E − 11) + (x5 * 2.8421709430404E − 14) + (x6 * 5.55111512312578E − 17) + (x7 * 1.0842021724855E − 19) + (x8 * 2.11758236813575E − 22) + (x9 * 4.13590306276514E − 25) + (x10 * 8.07793566946316E − 28) + (x11 * 1.57772181044202E − 30) + (x12 * 3.08148791101958E − 33) + (x13 * 6.01853107621011E − 36) + (x14 * 1.17549435082229E − 38) + (x15 * 2.29588740394978E − 41) + (x16 * 4.48415508583941E − 44) + (x17 * 8.75811540203011E − 47) + (x18 * 1.71056941445901E − 49) + (x19 * 3.34095588761524E − 52))	*	((y0 * 1) + (y1 * 0.001953125) + (y2 * 0.000003814697265625) + (y3 * 7.45058059692383E − 09) + (y4 * 1.45519152283669E − 11) + (y5 * 2.8421709430404E − 14) + (y6 * 5.55111512312578E − 17) + (y7 * 1.0842021724855E − 19) + (y8 * 2.11758236813575E − 22) + (y9 * 4.13590306276514E − 25) + (y10 * 8.07793566946316E − 28) + (y11 * 1.57772181044202E − 30) + (y12 * 3.08148791101958E − 33) + (y13 * 6.01853107621011E − 36) + (y14 * 1.17549435082229E − 38) + (y15 * 2.29588740394978E − 41) + (y16 * 4.48415508583941E − 44) + (y17 * 8.75811540203011E − 47) + (y18 * 1.71056941445901E − 49) + (y19 * 3.34095588761524E − 52) + (y20 * 6.52530446799852E − 55)))
Y	=	(((x0 * 1) + (x1 * −0.00195313) + (x2 * 0.0000038147167969) + (x3 * −7.4506378175293E − 09) + (x4 * 1.4552064240551E − 11) + (x5 * −2.84220732301474E − 14) + (x6 * 5.55120038879977E − 17) + (x7 * −1.08422160153765E − 19) + (x8 * 2.11762573661123E − 22) + (x9 * −4.13599835494749E − 25) + (x10 * 8.07814246699859E − 28) + (x11 * −1.5777662396569E − 30) + (x12 * 3.08158257566107E − 33) + (x13 * −6.01873137600091E − 36) + (x14 * 1.17553648124087E − 38) + (x15 * −2.29597556760597E − 41) + (x16 * 4.48433876035825E − 44) + (x17 * −8.75849656301852E − 47) + (x18 * 1.71064823921284E − 49) + (x19 * −3.34111839545377E − 52))	*	((y0 * 1) + (y1 * −0.00195313) + (y2 * 0.0000038147167969) + (y3 * −7.4506378175293E − 09) + (y4 * 1.4552064240551E − 11) + (y5 * −2.84220732301474E − 14) + (y6 * 5.55120038879977E − 17) + (y7 * −1.08422160153765E − 19) + (y8 * 2.11762573661123E − 22) + (y9 * −4.13599835494749E − 25) + (y10 * 8.07814246699859E − 28) + (y11 * −1.5777662396569E − 30) + (y12 * 3.08158257566107E − 33) + (y13 * −6.01873137600091E − 36) + (y14 * 1.17553648124087E − 38) + (y15 * −2.29597556760597E − 41) + (y16 * 4.48433876035825E − 44) + (y17 * −8.75849656301852E − 47) + (y18 * 1.71064823921284E − 49) + (y19 * −3.34111839545377E − 52) + (y20 * 6.52563857171261E − 55)))
Z	=	(((x0 * 1) + (x1 * 0.000976565) + (x2 * 0.000000953679199225) + (x3 * 9.31329727191162E − 10) + (x4 * 9.09504015034437E − 13) + (x5 * 8.88189788442105E − 16) + (x6 * 8.67375060749965E − 19) + (x7 * 8.47048126201289E − 22) + (x8 * 8.27197553363762E − 25) + (x9 * 8.07812178700682E − 28) + (x10 * 7.88881100292831E − 31) + (x11 * 7.70393671707469E − 34) + (x12 * 7.52339496011004E − 37) + (x13 * 7.34708419921986E − 40) + (x14 * 7.17490528101115E − 43) + (x15 * 7.00676137575065E − 46) + (x16 * 6.84255792290993E − 49) + (x17 * 6.68220257798654E − 52) + (x18 * 6.52560516057142E − 55) + (x19 * 6.37267760363343E − 58))	*	((y0 * 1) + (y1 * 0.000976565) + (y2 * 0.000000953679199225) + (y3 * 9.31329727191162E − 10) + (y4 * 9.09504015034437E − 13) + (y5 * 8.88189788442105E − 16) + (y6 * 8.67375060749965E − 19) + (y7 * 8.47048126201289E − 22) + (y8 * 8.27197553363762E − 25) + (y9 * 8.07812178700682E − 28) + (y10 * 7.88881100292831E − 31) + (y11 * 7.70393671707469E − 34) + (y12 * 7.52339496011004E − 37) + (y13 * 7.34708419921986E − 40) + (y14 * 7.17490528101115E − 43) + (y15 * 7.00676137575065E − 46) + (y16 * 6.84255792290993E − 49) + (y17 * 6.68220257798654E − 52) + (y18 * 6.52560516057142E − 55) + (y19 * 6.37267760363343E − 58) + (y20 * 6.22333390399228E − 61)))
AA	=	(((x0 * 1) + (x1 * −0.000976565) + (x2 * 0.000000953679199225) + (x3 * −9.31329727191162E − 10) + (x4 * 9.09504015034437E − 13) + (x5 * −8.88189788442105E − 16) + (x6 * 8.67375060749965E − 19) + (x7 * −8.47048126201289E − 22) + (x8 * 8.27197553363762E − 25) + (x9 * −8.07812178700682E − 28) + (x10 * 7.88881100292831E − 31) + (x11 * −7.70393671707469E − 34) + (x12 * 7.52339496011004E − 37) + (x13 * −7.34708419921986E − 40) + (x14 * 7.17490528101115E − 43) + (x15 * −7.00676137575065E − 46) + (x16 * 6.84255792290993E − 49) + (x17 * −6.68220257798654E − 52) + (x18 * 6.52560516057142E − 55) + (x19 * −6.37267760363343E − 58))	*	((y0 * 1) + (y1 * −0.000976565) + (y2 * 0.000000953679199225) + (y3 * −9.31329727191162E − 10) + (y4 * 9.09504015034437E − 13) + (y5 * −8.88189788442105E − 16) + (y6 * 8.67375060749965E − 19) + (y7 * −8.47048126201289E − 22) + (y8 * 8.27197553363762E − 25) + (y9 * −8.07812178700682E − 28) + (y10 * 7.88881100292831E − 31) + (y11 * −7.70393671707469E − 34) + (y12 * 7.52339496011004E − 37) + (y13 * −7.34708419921986E − 40) + (y14 * 7.17490528101115E − 43) + (y15 * −7.00676137575065E − 46) + (y16 * 6.84255792290993E − 49) + (y17 * −6.68220257798654E − 52) + (y18 * 6.52560516057142E − 55) + (y19 * −6.37267760363343E − 58) + (y20 * 6.22333390399228E − 61)))
AB	=	(((x0 * 1) + (x1 * 0.0004882825) + (x2 * 2.3841979980625E − 07) + (x3 * 1.16416215898895E − 10) + (x4 * 5.68440009396523E − 14) + (x5 * 2.77559308888158E − 17) + (x6 * 1.35527353242182E − 20) + (x7 * 6.61756348594757E − 24) + (x8 * 3.23124044282719E − 27) + (x9 * 1.57775816152477E − 30) + (x10 * 7.70391699504718E − 34) + (x11 * 3.76168785013413E − 37) + (x12 * 1.83676634768312E − 40) + (x13 * 8.96860864162581E − 44) + (x14 * 4.37921464905466E − 47) + (x15 * 2.13829387687703E − 50) + (x16 * 1.04409147993621E − 53) + (x17 * 5.09811598051952E − 57) + (x18 * 2.48932081625802E − 60) + (x19 * 1.21549179146451E − 63))	*	((y0 * 1) + (y1 * 0.0004882825) + (y2 * 2.3841979980625E − 07) + (y3 * 1.16416215898895E − 10) + (y4 * 5.68440009396523E − 14) + (y5 * 2.77559308888158E − 17) + (y6 * 1.35527353242182E − 20) + (y7 * 6.61756348594757E − 24) + (y8 * 3.23124044282719E − 27) + (y9 * 1.57775816152477E − 30) + (y10 * 7.70391699504718E − 34) + (y11 * 3.76168785013413E − 37) + (y12 * 1.83676634768312E − 40) + (y13 * 8.96860864162581E − 44) + (y14 * 4.37921464905466E − 47) + (y15 * 2.13829387687703E − 50) + (y16 * 1.04409147993621E − 53) + (y17 * 5.09811598051952E − 57) + (y18 * 2.48932081625802E − 60) + (y19 * 1.21549179146451E − 63) + (y20 * 5.93503370665768E − 67)))
AC	=	(((x0 * 1) + (x1 * −0.000488283) + (x2 * 0.000000238420288089) + (x3 * −1.16416573528961E − 10) + (x4 * 5.68442337724417E − 14) + (x5 * −2.77560729991092E − 17) + (x6 * 1.3552818592224E − 20) + (x7 * −6.61761092066692E − 24) + (x8 * 3.23126691317601E − 27) + (x9 * −1.57777270216632E − 30) + (x10 * 7.70399588331877E − 34) + (x11 * −3.76173022189454E − 37) + (x12 * 1.83678891793733E − 40) + (x13 * −8.96872803217193E − 44) + (x14 * 4.37927742973301E − 47) + (x15 * −2.13832672122232E − 50) + (x16 * 1.0441085864186E − 53) + (x17 * −5.09820472902233E − 57) + (x18 * 2.48936669970121E − 60) + (x19 * −1.21551544023021E − 63))	*	((y0 * 1) + (y1 * −0.000488283) + (y2 * 0.000000238420288089) + (y3 * −1.16416573528961E − 10) + (y4 * 5.68442337724417E − 14) + (y5 * −2.77560729991092E − 17) + (y6 * 1.3552818592224E − 20) + (y7 * −6.61761092066692E − 24) + (y8 * 3.23126691317601E − 27) + (y9 * −1.57777270216632E − 30) + (y10 * 7.70399588331877E − 34) + (y11 * −3.76173022189454E − 37) + (y12 * 1.83678891793733E − 40) + (y13 * −8.96872803217193E − 44) + (y14 * 4.37927742973301E − 47) + (y15 * −2.13832672122232E − 50) + (y16 * 1.0441085864186E − 53) + (y17 * −5.09820472902233E − 57) + (y18 * 2.48936669970121E − 60) + (y19 * −1.21551544023021E − 63) + (y20 * 5.93515525701925E − 67)))
AD	=	(((x0 * 1) + (x1 * 0.0002441415) + (x2 * 5.960507202225E − 08) + (x3 * 1.45520716911201E − 11) + (x4 * 3.55276461077761E − 15) + (x5 * 8.67377281222161E − 19) + (x6 * 2.117627905035E − 22) + (x7 * 5.17000853177103E − 26) + (x8 * 1.26221363795938E − 29) + (x9 * 3.08158730891859E − 33) + (x10 * 7.52343347980348E − 37) + (x11 * 1.83678233490944E − 40) + (x12 * 4.48434794418294E − 44) + (x13 * 1.09481543361474E − 47) + (x14 * 2.67289882185853E − 51) + (x15 * 6.52565527716773E − 55) + (x16 * 1.59318326785065E − 58) + (x17 * 3.88962152787958E − 62) + (x18 * 9.49618034248813E − 66) + (x19 * 2.31841171308557E − 69))	*	((y0 * 1) + (y1 * 0.0002441415) + (y2 * 5.960507202225E − 08) + (y3 * 1.45520716911201E − 11) + (y4 * 3.55276461077761E − 15) + (y5 * 8.67377281222161E − 19) + (y6 * 2.117627905035E − 22) + (y7 * 5.17000853177103E − 26) + (y8 * 1.26221363795938E − 29) + (y9 * 3.08158730891859E − 33) + (y10 * 7.52343347980348E − 37) + (y11 * 1.83678233490944E − 40) + (y12 * 4.48434794418294E − 44) + (y13 * 1.09481543361474E − 47) + (y14 * 2.67289882185853E − 51) + (y15 * 6.52565527716773E − 55) + (y16 * 1.59318326785065E − 58) + (y17 * 3.88962152787958E − 62) + (y18 * 9.49618034248813E − 66) + (y19 * 2.31841171308557E − 69) + (y20 * 5.6602051325028E − 73)))
AE	=	(((x0 * 1) + (x1 * −0.00024412) + (x2 * 0.0000000595945744) + (x3 * −1.4548227502528E − 11) + (x4 * 3.55151329791713E − 15) + (x5 * −8.66995426287531E − 19) + (x6 * 2.11650923465312E − 22) + (x7 * −5.1668223436352E − 26) + (x8 * 1.26132467052822E − 29) + (x9 * −3.0791457856935E − 33) + (x10 * 7.51681069203497E − 37) + (x11 * −1.83500382613958E − 40) + (x12 * 4.47961134037194E − 44) + (x13 * −1.0935627204116E − 47) + (x14 * 2.66960531306879E − 51) + (x15 * −6.51704049026353E − 55) + (x16 * 1.59093992448313E − 58) + (x17 * −3.88380254364823E − 62) + (x18 * 9.48113876955405E − 66) + (x19 * −2.31453559642353E − 69))	*	((y0 * 1) + (y1 * −0.00024412) + (y2 * 0.0000000595945744) + (y3 * −1.4548227502528E − 11) + (y4 * 3.55151329791713E − 15) + (y5 * −8.66995426287531E − 19) + (y6 * 2.11650923465312E − 22) + (y7 * −5.1668223436352E − 26) + (y8 * 1.26132467052822E − 29) + (y9 * −3.0791457856935E − 33) + (y10 * 7.51681069203497E − 37) + (y11 * −1.83500382613958E − 40) + (y12 * 4.47961134037194E − 44) + (y13 * −1.0935627204116E − 47) + (y14 * 2.66960531306879E − 51) + (y15 * −6.51704049026353E − 55) + (y16 * 1.59093992448313E − 58) + (y17 * −3.88380254364823E − 62) + (y18 * 9.48113876955405E − 66) + (y19 * −2.31453559642353E − 69) + (y20 * 5.65024429798913E − 73)))
AF	=	(((x0 * 1) + (x1 * 0.00012206) + (x2 * 0.0000000148986436) + (x3 * 1.818528437816E − 12) + (x4 * 2.21969581119821E − 16) + (x5 * 2.70936070714853E − 20) + (x6 * 3.3070456791455E − 24) + (x7 * 4.036579955965E − 28) + (x8 * 4.92704949425088E − 32) + (x9 * 6.01395661268262E − 36) + (x10 * 7.3406354414404E − 40) + (x11 * 8.95997961982216E − 44) + (x12 * 1.09365511239549E − 47) + (x13 * 1.33491543018994E − 51) + (x14 * 1.62939777408984E − 55) + (x15 * 1.98884292305406E − 59) + (x16 * 2.42758167187978E − 63) + (x17 * 2.96310618869646E − 67) + (x18 * 3.6167674139229E − 71) + (x19 * 4.41462630543429E − 75))	*	((y0 * 1) + (y1 * 0.00012206) + (y2 * 0.0000000148986436) + (y3 * 1.818528437816E − 12) + (y4 * 2.21969581119821E − 16) + (y5 * 2.70936070714853E − 20) + (y6 * 3.3070456791455E − 24) + (y7 * 4.036579955965E − 28) + (y8 * 4.92704949425088E − 32) + (y9 * 6.01395661268262E − 36) + (y10 * 7.3406354414404E − 40) + (y11 * 8.95997961982216E − 44) + (y12 * 1.09365511239549E − 47) + (y13 * 1.33491543018994E − 51) + (y14 * 1.62939777408984E − 55) + (y15 * 1.98884292305406E − 59) + (y16 * 2.42758167187978E − 63) + (y17 * 2.96310618869646E − 67) + (y18 * 3.6167674139229E − 71) + (y19 * 4.41462630543429E − 75) + (y20 * 5.3884928684131E − 79)))
AG	=	(((x0 * 1) + (x1 * −0.00012206) + (x2 * 0.0000000148986436) + (x3 * −1.818528437816E − 12) + (x4 * 2.21969581119821E − 16) + (x5 * −2.70936070714853E − 20) + (x6 * 3.3070456791455E − 24) + (x7 * −4.036579955965E − 28) + (x8 * 4.92704949425088E − 32) + (x9 * −6.01395661268262E − 36) + (x10 * 7.3406354414404E − 40) + (x11 * −8.95997961982216E − 44) + (x12 * 1.09365511239549E − 47) + (x13 * −1.33491543018994E − 51) + (x14 * 1.62939777408984E − 55) + (x15 * −1.98884292305406E − 59) + (x16 * 2.42758167187978E − 63) + (x17 * −2.96310618869646E − 67) + (x18 * 3.6167674139229E − 71) + (x19 * −4.41462630543429E − 75))	*	((y0 * 1) + (y1 * −0.00012206) + (y2 * 0.0000000148986436) + (y3 * −1.818528437816E − 12) + (y4 * 2.21969581119821E − 16) + (y5 * −2.70936070714853E − 20) + (y6 * 3.3070456791455E − 24) + (y7 * −4.036579955965E − 28) + (y8 * 4.92704949425088E − 32) + (y9 * −6.01395661268262E − 36) + (y10 * 7.3406354414404E − 40) + (y11 * −8.95997961982216E − 44) + (y12 * 1.09365511239549E − 47) + (y13 * −1.33491543018994E − 51) + (y14 * 1.62939777408984E − 55) + (y15 * −1.98884292305406E − 59) + (y16 * 2.42758167187978E − 63) + (y17 * −2.96310618869646E − 67) + (y18 * 3.6167674139229E − 71) + (y19 * −4.41462630543429E − 75) + (y20 * 5.3884928684131E − 79)))

TABLE 4. The Evaluation Multiplication(continued).

AH =	(((x0 * 1) + (x1 * 0.00006103) + (x2 * 0.0000000037246609) + (x3 * 2.27316054727E − 13) + (x4 * 1.38730988199888E − 17) + (x5 * 8.46675220983917E − 22) + (x6 * 5.16725887366484E − 26) + (x7 * 3.15357809059765E − 30) + (x8 * 1.92462870869175E − 34) + (x9 * 1.17460090091457E − 38) + (x10 * 7.16858929828165E − 43) + (x11 * 4.37499004874129E − 47) + (x12 * 2.67005642674681E − 51) + (x13 * 1.62953543724358E − 55) + (x14 * 9.94505477349755E − 60) + (x15 * 6.06946692826555E − 64) + (x16 * 3.70419566632047E − 68) + (x17 * 2.26067061515538E − 72) + (x18 * 1.37968727642933E − 76) + (x19 * 8.42023144804819E − 81))	*	((y0 * 1) + (y1 * 0.00006103) + (y2 * 0.0000000037246609) + (y3 * 2.27316054727E − 13) + (y4 * 1.38730988199888E − 17) + (y5 * 8.46675220983917E − 22) + (y6 * 5.16725887366484E − 26) + (y7 * 3.15357809059765E − 30) + (y8 * 1.92462870869175E − 34) + (y9 * 1.17460090091457E − 38) + (y10 * 7.16858929828165E − 43) + (y11 * 4.37499004874129E − 47) + (y12 * 2.67005642674681E − 51) + (y13 * 1.62953543724358E − 55) + (y14 * 9.94505477349755E − 60) + (y15 * 6.06946692826555E − 64) + (y16 * 3.70419566632047E − 68) + (y17 * 2.26067061515538E − 72) + (y18 * 1.37968727642933E − 76) + (y19 * 8.42023144804819E − 81) + (y20 * 5.13886725274381E − 85)))
AI =	(((x0 * 1) + (x1 * −0.00006103) + (x2 * 0.0000000037246609) + (x3 * −2.27316054727E − 13) + (x4 * 1.38730988199888E − 17) + (x5 * −8.46675220983917E − 22) + (x6 * 5.16725887366484E − 26) + (x7 * −3.15357809059765E − 30) + (x8 * 1.92462870869175E − 34) + (x9 * −1.17460090091457E − 38) + (x10 * 7.16858929828165E − 43) + (x11 * −4.37499004874129E − 47) + (x12 * 2.67005642674681E − 51) + (x13 * −1.62953543724358E − 55) + (x14 * 9.94505477349755E − 60) + (x15 * −6.06946692826555E − 64) + (x16 * 3.70419566632047E − 68) + (x17 * −2.26067061515538E − 72) + (x18 * 1.37968727642933E − 76) + (x19 * −8.42023144804819E − 81))	*	((y0 * 1) + (y1 * −0.00006103) + (y2 * 0.0000000037246609) + (y3 * −2.27316054727E − 13) + (y4 * 1.38730988199888E − 17) + (y5 * −8.46675220983917E − 22) + (y6 * 5.16725887366484E − 26) + (y7 * −3.15357809059765E − 30) + (y8 * 1.92462870869175E − 34) + (y9 * −1.17460090091457E − 38) + (y10 * 7.16858929828165E − 43) + (y11 * −4.37499004874129E − 47) + (y12 * 2.67005642674681E − 51) + (y13 * −1.62953543724358E − 55) + (y14 * 9.94505477349755E − 60) + (y15 * −6.06946692826555E − 64) + (y16 * 3.70419566632047E − 68) + (y17 * −2.26067061515538E − 72) + (y18 * 1.37968727642933E − 76) + (y19 * −8.42023144804819E − 81) + (y20 * 5.13886725274381E − 85)))
AJ =	(((x0 * 1) + (x1 * 0.000030515) + (x2 * 0.000000000931165225) + (x3 * 2.8414506840875E − 14) + (x4 * 8.67068676249301E − 19) + (x5 * 2.64586006557474E − 23) + (x6 * 8.07384199010132E − 28) + (x7 * 2.46373288327942E − 32) + (x8 * 7.51808089332714E − 37) + (x9 * 2.29414238459878E − 41) + (x10 * 7.00057548660317E − 46) + (x11 * 2.13622560973696E − 50) + (x12 * 6.51869244811232E − 55) + (x13 * 1.98917900054148E − 59) + (x14 * 6.06997972015231E − 64) + (x15 * 1.85225431160448E − 68) + (x16 * 5.65215403186106E − 73) + (x17 * 1.7247548028224E − 77) + (x18 * 5.26308928081256E − 82) + (x19 * 1.60603169403995E − 86))	*	((y0 * 1) + (y1 * 0.000030515) + (y2 * 0.000000000931165225) + (y3 * 2.8414506840875E − 14) + (y4 * 8.67068676249301E − 19) + (y5 * 2.64586006557474E − 23) + (y6 * 8.07384199010132E − 28) + (y7 * 2.46373288327942E − 32) + (y8 * 7.51808089332714E − 37) + (y9 * 2.29414238459878E − 41) + (y10 * 7.00057548660317E − 46) + (y11 * 2.13622560973696E − 50) + (y12 * 6.51869244811232E − 55) + (y13 * 1.98917900054148E − 59) + (y14 * 6.06997972015231E − 64) + (y15 * 1.85225431160448E − 68) + (y16 * 5.65215403186106E − 73) + (y17 * 1.7247548028224E − 77) + (y18 * 5.26308928081256E − 82) + (y19 * 1.60603169403995E − 86) + (y20 * 4.90080571436292E − 91)))
AK =	(((x0 * 1) + (x1 * −0.000030515) + (x2 * 0.000000000931165225) + (x3 * −2.8414506840875E − 14) + (x4 * 8.67068676249301E − 19) + (x5 * −2.64586006557474E − 23) + (x6 * 8.07384199010132E − 28) + (x7 * −2.46373288327942E − 32) + (x8 * 7.51808089332714E − 37) + (x9 * −2.29414238459878E − 41) + (x10 * 7.00057548660317E − 46) + (x11 * −2.13622560973696E − 50) + (x12 * 6.51869244811232E − 55) + (x13 * −1.98917900054148E − 59) + (x14 * 6.06997972015231E − 64) + (x15 * −1.85225431160448E − 68) + (x16 * 5.65215403186106E − 73) + (x17 * −1.7247548028224E − 77) + (x18 * 5.26308928081256E − 82) + (x19 * −1.60603169403995E − 86))	*	((y0 * 1) + (y1 * 0.000030515) + (y2 * 0.000000000931165225) + (y3 * 2.8414506840875E − 14) + (y4 * 8.67068676249301E − 19) + (y5 * 2.64586006557474E − 23) + (y6 * 8.07384199010132E − 28) + (y7 * 2.46373288327942E − 32) + (y8 * 7.51808089332714E − 37) + (y9 * 2.29414238459878E − 41) + (y10 * 7.00057548660317E − 46) + (y11 * 2.13622560973696E − 50) + (y12 * 6.51869244811232E − 55) + (y13 * 1.98917900054148E − 59) + (y14 * 6.06997972015231E − 64) + (y15 * 1.85225431160448E − 68) + (y16 * 5.65215403186106E − 73) + (y17 * 1.7247548028224E − 77) + (y18 * 5.26308928081256E − 82) + (y19 * 1.60603169403995E − 86) + (y20 * 4.90080571436292E − 91)))
AL =	(((x0 * 1) + (x1 * 0.0000152575) + (x2 * 2.3279130625E − 10) + (x3 * 3.55181335510937E − 15) + (x4 * 5.41917922655813E − 20) + (x5 * 8.26831270492106E − 25) + (x6 * 1.26153781095333E − 29) + (x7 * 1.92479131506205E − 34) + (x8 * 2.93675034895592E − 39) + (x9 * 4.48074684491949E − 44) + (x10 * 6.83649949863591E − 49) + (x11 * 1.04307891100437E − 53) + (x12 * 1.59147764846492E − 58) + (x13 * 2.42819702214536E − 63) + (x14 * 3.70482160653828E − 68) + (x15 * 5.65263156617578E − 73) + (x16 * 8.62450261209269E − 78) + (x17 * 1.31588348604004E − 82) + (x18 * 2.00770922882559E − 87) + (x19 * 3.06326235588065E − 92))	*	((y0 * 1) + (y1 * −0.000030515) + (y2 * 0.000000000931165225) + (y3 * −2.8414506840875E − 14) + (y4 * 8.67068676249301E − 19) + (y5 * −2.64586006557474E − 23) + (y6 * 8.07384199010132E − 28) + (y7 * −2.46373288327942E − 32) + (y8 * 7.51808089332714E − 37) + (y9 * −2.29414238459878E − 41) + (y10 * 7.00057548660317E − 46) + (y11 * −2.13622560973696E − 50) + (y12 * 6.51869244811232E − 55) + (y13 * −1.98917900054148E − 59) + (y14 * 6.06997972015231E − 64) + (y15 * −1.85225431160448E − 68) + (y16 * 5.65215403186106E − 73) + (y17 * −1.7247548028224E − 77) + (y18 * 5.26308928081256E − 82) + (y19 * −1.60603169403995E − 86) + (y20 * 4.90080571436292E − 91)))
AM =	(((x0 * 1) + (x1 * −0.0000152575) + (x2 * 2.3279130625E − 10) + (x3 * −3.55181335510937E − 15) + (x4 * 5.41917922655813E − 20) + (x5 * −8.26831270492106E − 25) + (x6 * 1.26153781095333E − 29) + (x7 * −1.92479131506205E − 34) + (x8 * 2.93675034895592E − 39) + (x9 * −4.48074684491949E − 44) + (x10 * 6.83649949863591E − 49) + (x11 * −1.04307891100437E − 53) + (x12 * 1.59147764846492E − 58) + (x13 * −2.42819702214536E − 63) + (x14 * 3.70482160653828E − 68) + (x15 * −5.65263156617578E − 73) + (x16 * 8.62450261209269E − 78) + (x17 * −1.31588348604004E − 82) + (x18 * 2.00770922882559E − 87) + (x19 * −3.06326235588065E − 92))	*	((y0 * 1) + (y1 * 0.0000152575) + (y2 * 2.3279130625E − 10) + (y3 * 3.55181335510937E − 15) + (y4 * 5.41917922655813E − 20) + (y5 * 8.26831270492106E − 25) + (y6 * 1.26153781095333E − 29) + (y7 * 1.92479131506205E − 34) + (y8 * 2.93675034895592E − 39) + (y9 * 4.48074684491949E − 44) + (y10 * 6.83649949863591E − 49) + (y11 * 1.04307891100437E − 53) + (y12 * 1.59147764846492E − 58) + (y13 * 2.42819702214536E − 63) + (y14 * 3.70482160653828E − 68) + (y15 * 5.65263156617578E − 73) + (y16 * 8.62450261209269E − 78) + (y17 * 1.31588348604004E − 82) + (y18 * 2.00770922882559E − 87) + (y19 * 3.06326235588065E − 92) + (y20 * 4.6737725394849E − 97)))
AN =	(((x0 * 1) + (x1 * 0.00000762875) + (x2 * 5.81978265625E − 11) + (x3 * 4.43976669388672E − 16) + (x4 * 3.38698701659883E − 21) + (x5 * 2.58384772028783E − 26) + (x6 * 1.97115282961458E − 31) + (x7 * 1.50374321489222E − 36) + (x8 * 1.1471681050609E − 41) + (x9 * 8.75145868148337E − 47) + (x10 * 6.67626904163663E − 52) + (x11 * 5.09315874513854E − 57) + (x12 * 3.88544347769757E − 62) + (x13 * 2.96410769304853E − 67) + (x14 * 2.2612436563344E − 72) + (x15 * 1.7250462543261E − 77) + (x16 * 1.31599466126903E − 82) + (x17 * 1.00393942721561E − 87) + (x18 * 7.65880290537107E − 93) + (x19 * 5.84270926643496E − 98))	*	((y0 * 1) + (y1 * −0.0000152575) + (y2 * 2.3279130625E − 10) + (y3 * −3.55181335510937E − 15) + (y4 * 5.41917922655813E − 20) + (y5 * −8.26831270492106E − 25) + (y6 * 1.26153781095333E − 29) + (y7 * −1.92479131506205E − 34) + (y8 * 2.93675034895592E − 39) + (y9 * −4.48074684491949E − 44) + (y10 * 6.83649949863591E − 49) + (y11 * −1.04307891100437E − 53) + (y12 * 1.59147764846492E − 58) + (y13 * −2.42819702214536E − 63) + (y14 * 3.70482160653828E − 68) + (y15 * −5.65263156617578E − 73) + (y16 * 8.62450261209269E − 78) + (y17 * −1.31588348604004E − 82) + (y18 * 2.00770922882559E − 87) + (y19 * −3.06326235588065E − 92) + (y20 * 4.6737725394849E − 97)))
AO =	(((x0 * 1) + (x1 * −0.00000762875) + (x2 * 5.81978265625E − 11) + (x3 * −4.43976669388672E − 16) + (x4 * 3.38698701659883E − 21) + (x5 * −2.58384772028783E − 26) + (x6 * 1.97115282961458E − 31) + (x7 * −1.50374321489222E − 36) + (x8 * 1.1471681050609E − 41) + (x9 * −8.75145868148337E − 47) + (x10 * 6.67626904163663E − 52) + (x11 * −5.09315874513854E − 57) + (x12 * 3.88544347769757E − 62) + (x13 * −2.96410769304853E − 67) + (x14 * 2.2612436563344E − 72) + (x15 * −1.7250462543261E − 77) + (x16 * 1.31599466126903E − 82) + (x17 * −1.00393942721561E − 87) + (x18 * 7.65880290537107E − 93) + (x19 * −5.84270926643496E − 98))	*	((y0 * 1) + (y1 * 0.00000762875) + (y2 * 5.81978265625E − 11) + (y3 * 4.43976669388672E − 16) + (y4 * 3.38698701659883E − 21) + (y5 * 2.58384772028783E − 26) + (y6 * 1.97115282961458E − 31) + (y7 * 1.50374321489222E − 36) + (y8 * 1.1471681050609E − 41) + (y9 * 8.75145868148337E − 47) + (y10 * 6.67626904163663E − 52) + (y11 * 5.09315874513854E − 57) + (y12 * 3.88544347769757E − 62) + (y13 * 2.96410769304853E − 67) + (y14 * 2.2612436563344E − 72) + (y15 * 1.7250462543261E − 77) + (y16 * 1.31599466126903E − 82) + (y17 * 1.00393942721561E − 87) + (y18 * 7.65880290537107E − 93) + (y19 * 5.84270926643496E − 98) + (y20 * 4.4572568316316E − 103)))

$$= n\left(\frac{1-\left(\frac{41}{21}\right)^{\log_{21} n}}{1-\frac{41}{21}}\right) \tag{22}$$

The Toffoli depth of a circuit is a commonly used metric for quantifying its time complexity [7], [32]. The calculation can be performed by multiplying the quantity of subtrees S_k at the k-th level by the respective depth D_k. Therefore, the Toffoli depth T_d can be represented as shown in Equation 23.

$$\begin{aligned} S_k &= 41^{\left(1-\frac{\log 41}{2\log 41 - log21}\right)\log_{21} n} \\ D_k &= \frac{n}{21^{\left(1-\frac{\log 41}{2\log 41-\log 21}\right)\log_{21} n}} \\ T_d &= S_k D_k = n\left(\frac{41}{21}\right)^{\left(1-\frac{\log 41}{2\log 41-\log 21}\right)\log_{21} n} \approx n^{1.0335} \end{aligned} \tag{23}$$

C. COMPLEXITY ANALYSIS COMPARISON

To provide a comprehensive overview of advancements in complexity multiplication research, particularly in relation to methodologies based on Toom-Cook algorithms, we present the results of our cost analysis in Table 2. In this evaluation, the assessment of space-time complexity is conducted by employing metrics such as the Toffoli count, qubit count, and Toffoli depth.

The present study examined briefly different degrees of Toom-Cook multiplication complexity, including Toom-Cook 2-way [11], Toom-Cook 2.5-way [7], Toom-Cook 3-way [8], Toom-Cook 4-way [11], and Toom-Cook 8-way [5], [11], which have been extensively discussed in prior prominent research studies. In order to gain a comprehensive understanding of the distinctions, we additionally reported and examined the asymptotic performance and quantum resource allocation of the Toom-Cook 8.5-way, 10.5-way, and 20.5-way quantum design architectures. This analysis is presented in a consolidated table, Table 2. As a result, the designed 20.5-degree demonstrates a high degree of polynomial multiplication with superior asymptotic performance outcomes and the lowest utilization of quantum resources.

Based on the results from the analysis, it can be observed that Equation 17, denoting the Toffoli count, and Equation 23, representing the Toffoli Depth, indicate that the Toom-Cook 20.5-way method demonstrates more favorable multiplication costs in comparison to earlier Toom-Cook investigations, as well as commonly employed multiplication techniques such as the naive schoolbook and karatsuba methods. The naive algorithm exhibits a time complexity of $\mathcal{O}(n^2)$, where n denotes the size of the input. On the other hand, the Karatsuba algorithm has a time complexity of $\mathcal{O}(n^{\log_2(3)})$. According to Dutta et al. [7], the utilization of the Toom-Cook 2.5-way algorithm leads to reductions in the number of qubits ($n^{1.404}$), Toffoli gates ($49n^{\log_6 16}$), and Toffoli gate depth ($n^{1.143}$) compared to naive Schoolbook or Karatsuba. Based on Dutta et al.'s study, Larasati et al. [8] conducted a comprehensive examination of the asymptotic performance measures related to qubit count, Toffoli count, and Toffoli depth. They found that the estimated Toom-Cook 3-way value for the qubit count is $n^{1.353}$, the Toffoli count has a complexity of $\mathcal{O}(n^2)$, and the Toffoli depth follows a power law of $n^{1.112}$.

In a recent study by Putranto et al. [11], they presented an improved analysis of the asymptotic performance in terms of qubit count for the Toom-Cook 8-way and 8.5-way approach. The Toom-Cook 8-way estimation of the specific quantity is determined by the qubit count, which can be expressed as $n(\frac{15}{8})^{\frac{\log 15}{(2\log 15-\log 8)}\log_8 n}$, the approximation can be classified as $\mathcal{O}(n^{1.245})$. In the realm of efficient computation, the concept of Toffoli depth holds significance. In this regard, it is worth noting that the Toom-Cook 8-way design yields a lower limit on logical depth amounting to $\mathcal{O}(n^{1.0569})$, accompanied by a Toffoli count of $\mathcal{O}(n^{\log_8 15})$.

Our current work presents the findings displayed in Table 2. The Toom-Cook 8.5 method asymptotic performance analysis is observed to yield a qubit count of $\mathcal{O}(n^{1.236})$,

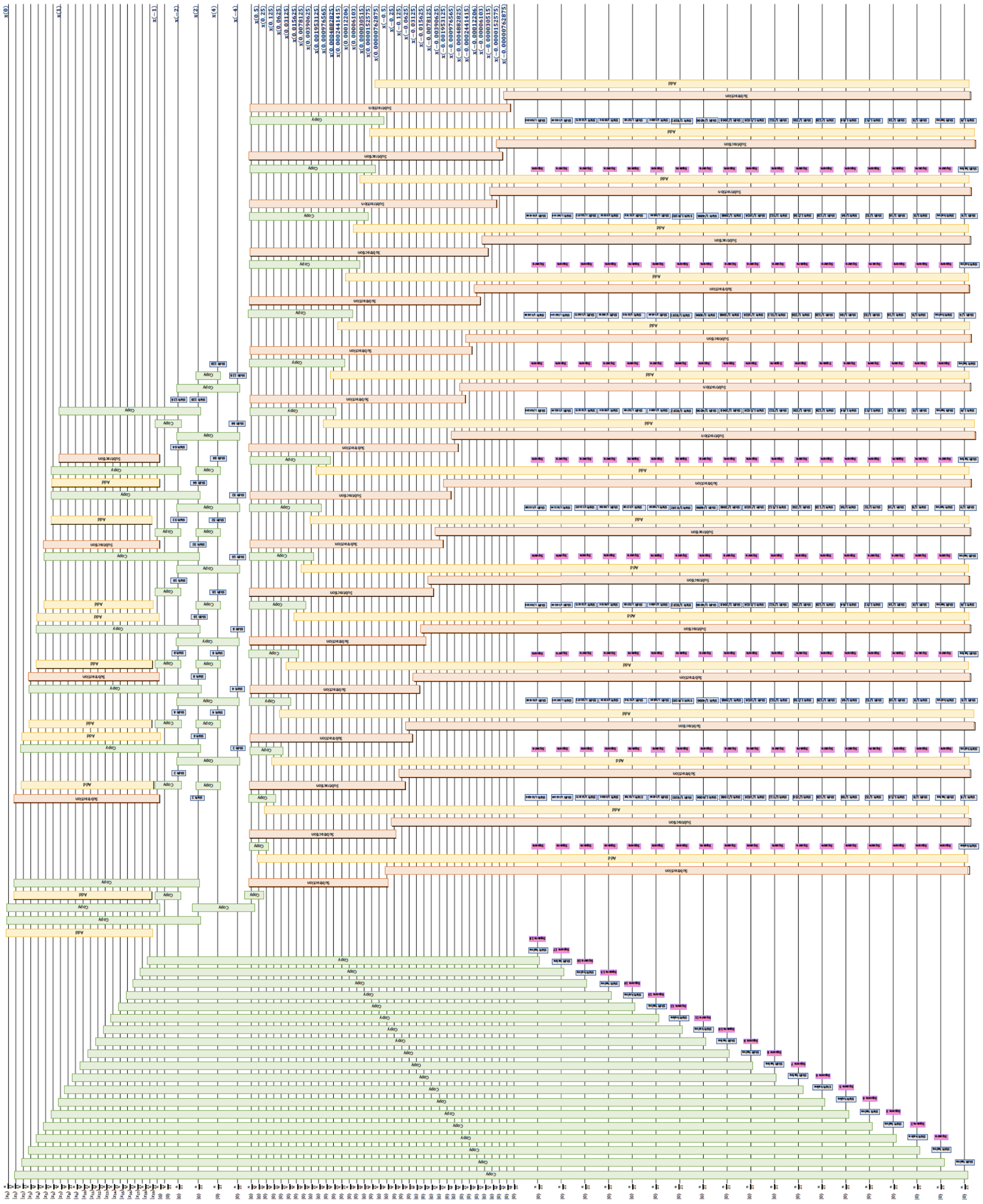

FIGURE 5. Evaluation point x.

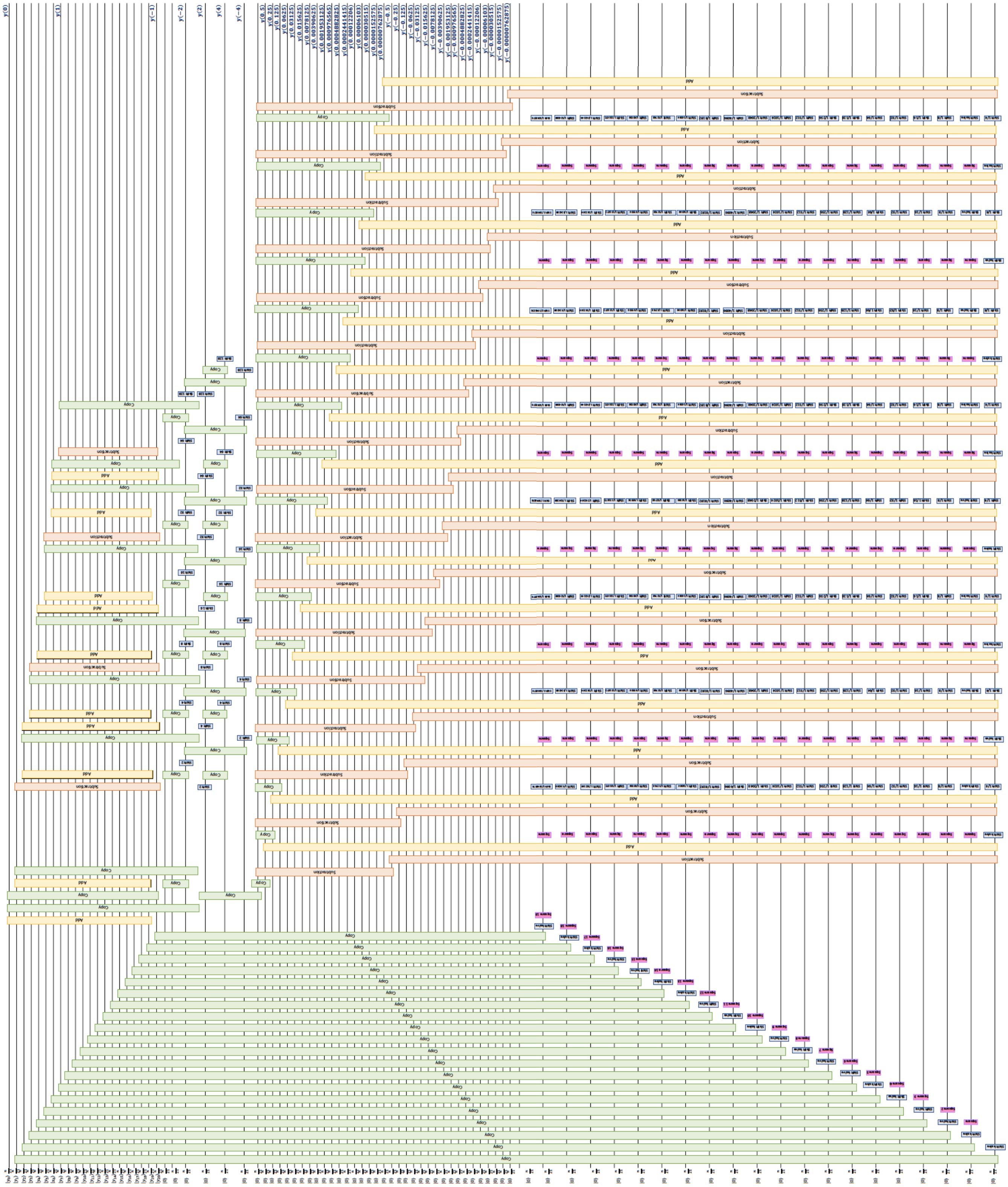

FIGURE 6. Evaluation point y.

a Toffoli count of $\mathcal{O}(n^{\log_9 17})$, and a Toffoli depth count of $\mathcal{O}(n^{1.053})$. The quantum implementation costs an identifiable cost, resulting in a certain number of $n(\frac{17}{9})^{\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.236}$ qubits, $186n^{\log_9 17}-202n$ Toffoli count operations, and Toffoli depth with $n(\frac{17}{9})^{1-\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.053}$.

Additionally, the analysis of the performance of the Toom-Cook 10.5 method can be reported that it demonstrates an asymptotic qubit count of $\mathcal{O}(n^{1.222})$, a Toffoli count of $\mathcal{O}(n^{\log_{11} 21})$, and a Toffoli depth count of $\mathcal{O}(n^{1.047})$. The cost of implementation in quantum is resulting in a quantity of $n(\frac{21}{11})^{\frac{\log 21}{(2\log 21-\log 11)}\log_{11} n} \approx n^{1.222}$ qubits. Additionally, the computational Toffoli count overhead is given by $206n^{\log_{11} 21} - 132n$. The Toffoli depth can be approximated by the expression $n(\frac{21}{11})^{1-\frac{\log 21}{(2\log 21-\log 11)}\log_1 1n} \approx n^{1.047}$.

The optimal outcome of this endeavor is the computation result of high- and half-degree quantum multiplication, particularly utilizing the Toom-Cook 20.5-way method, necessitates a qubit count with a complexity of $\mathcal{O}(n^{1.186})$, a toffoli count of $\mathcal{O}(n^{\log_{21} 41})$, and a toffoli depth count with a complexity of $\mathcal{O}(n^{1.033})$. The cost of quantum implementation for this multiplication is $n(\frac{41}{21})^{\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.186}$ qubits, $522n^{\log_{21} 41} - 540n$ Toffoli count operations, and Toffoli depth with $n(\frac{41}{21})^{1-\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.033}$.

VII. DISCUSSION

An extensive analysis from different multiplication techniques revealed that, despite a high degree, the Toom-Cook 20.5-way multiplier achieves reduced resource utilization. It is crucial to recognize that the effectiveness of the recently developed Toom-Cook 20.5-way algorithm exceeds that of the currently employed multiplication methods, particularly the Toom-Cook 4-way technique utilized in algorithms based on lattice algorithms during the post-quantum cryptography era.

The role of quantum cryptanalysis is significant in identifying potential weaknesses in classical cryptographic systems when confronted with quantum attacks. Additionally, it plays a crucial role in evaluating the security of post-quantum cryptographic algorithms to determine their capacity to provide a robust defense against quantum adversaries. For future work, the design multiplication needs to be incorporated into the real implementation of the PQC algorithm or notable cryptanalysis circuit, for example, utilizing the Shor algorithm technique. Moreover, it is crucial to improve the implementation by offering a more thorough explanation of multiplication-based attacks using SCA or CPA techniques, as well as measuring the potential for errors in quantum design for high-degree multiplication.

VIII. CONCLUSION

This work extensively analyzed the Toom-Cook algorithm and employed advanced techniques, including division-free approaches, tree-structured methods, opposite points, inverses, and cost-exploiting symmetries, to devise efficient strategies for implementing high- and half-degree quantum multiplication. The specific focus was on the Toom-Cook 8.5-way, 10.5-way, and 20.5-way degree multiplication approaches. The implementation outcome demonstrates exceptional performance in comparison to the existing cutting-edge outcomes, as determined by analyzing the asymptotic performance and the cost associated with quantum implementation, particularly in terms of qubit counts, Toffoli counts, and Toffoli depth. Our analysis indicates that the Toom-Cook 20.5-way architecture demonstrates the highest level of efficiency in utilizing quantum resources. This is observable from the qubit counts of the $n^{1.186}$, $522n^{\log_{21} 41} - 540n$ Toffoli counts, and the $n^{1.033}$ Toffoli depth required. This outcome represents a significant advancement when compared to the previous approach of employing classical schoolbook multiplication with a complexity of $\mathcal{O}(n^2)$ or utilizing quantum design for Karatsuba multiplication with a complexity of $\mathcal{O}(n^{\log_2(3)})$. Moreover, the result demonstrates a significant improvement in comparison to the presently employed Toom-k-way, Toom-Cook 2.5-way, Toom-Cook 8.5-way, and 10.5-way approaches.

APPENDIXES

See Tables 3 and 4 and Figures 5 and 6.

REFERENCES

[1] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Math. Doklady*, vol. 3, no. 4, pp. 714–716, 1963.

[2] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Trans. Amer. Math. Soc.*, vol. 142, pp. 291–314, Aug. 1969.

[3] J. M. Bermudo Mera, A. Karmakar, and I. Verbauwhede, "Time-memory trade-off in Toom–Cook multiplication: An application to module-lattice based cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 2, pp. 222–244, 2020.

[4] Y. Li, J. Zhu, Y. Huang, Z. Liu, and M. Tang, "Single-trace side-channel attacks on the Toom–Cook: The case study of saber," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, no. 4, pp. 285–310, 2022. [Online]. Available: https://doi.org/10.46586/tches.v2022.i4.285-310

[5] A. Zanoni, "Toom–Cook 8-way for long integers multiplication," in *Proc. 11th Int. Symp. Symbolic Numeric Algorithms Scientific Comput.*, Sep. 2009, pp. 54–57.

[6] M. Bodrato, "High degree Toom'n'half for balanced and unbalanced multiplication," in *Proc. IEEE 20th Symp. Comput. Arithmetic*, Jul. 2011, pp. 15–22.

[7] S. Dutta, D. Bhattacharjee, and A. Chattopadhyay, "Quantum circuits for Toom–Cook multiplication," *Phys. Rev. A, Gen. Phys.*, vol. 98, no. 1, Jul. 2018, Art. no. 012311.

[8] H. T. Larasati, A. M. Awaludin, J. Ji, and H. Kim, "Quantum circuit design of Toom 3-way multiplication," *Appl. Sci.*, vol. 11, no. 9, p. 3752, Apr. 2021.

[9] A. Ghosh, J. M. B. Mera, A. Karmakar, D. Das, S. Ghosh, I. Verbauwhede, and S. Sen, "A 334 μW 0.158 mm^2 ASIC for post-quantum key-encapsulation mechanism saber with low-latency striding Toom–Cook multiplication authors version," 2023, *arXiv:2305.10368*.

[10] C. Mujdei, L. Wouters, A. Karmakar, A. Beckers, J. M. B. Mera, and I. Verbauwhede, "Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication," *ACM Trans. Embedded Comput. Syst.*, vol. 2022, pp. 1–26, Nov. 2022.

[11] D. S. C. Putranto, R. W. Wardhani, H. T. Larasati, and H. Kim, "Space and time-efficient quantum multiplier in post quantum cryptography era," *IEEE Access*, vol. 11, pp. 21848–21862, 2023.

[12] M. Bodrato, "Towards optimal Toom–Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0," in *Proc. Int. Workshop Arithmetic Finite Fields*. Berlin, Germany: Springer, 2007, pp. 116–133.

[13] Z. Gu and S. Li, "A division-free Toom–Cook multiplication-based Montgomery modular multiplication," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 8, pp. 1401–1405, Aug. 2019.

[14] D. S. C. Putranto, R. W. Wardhani, H. T. Larasati, J. Ji, and H. Kim, "Depth-optimization of quantum cryptanalysis on binary elliptic curves," *IEEE Access*, vol. 11, pp. 45083–45097, 2023.

[15] M. Bodrato and A. Zanoni, "Integer and polynomial multiplication: Towards optimal Toom–Cook matrices," in *Proc. Int. Symp. Symbolic Algebr. Comput.*, Jul. 2007, pp. 17–24.

[16] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and Y.-K. Liu, "Status report on the third round of the NIST post-quantum cryptography standardization process," U.S. Dept. Commerce, NIST, Gaithersburg, MD, USA, Tech. Rep. NIST IR 8413, 2022.

[17] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 353–367.

[18] V. Lyubashevsky, "Fiat–Shamir with aborts: Applications to lattice and factoring-based signatures," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2009, pp. 598–616.

[19] D. Micciancio and O. Regev, "Post-quantum cryptography, chapter lattice-based cryptography," *Computing*, vol. 85, nos. 1–2, pp. 105–125, 2008.

[20] Z. Liu, K. R. Choo, and J. Grossschadl, "Securing edge devices in the post-quantum Internet of Things using lattice-based cryptography," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 158–162, Feb. 2018.

[21] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of CRYSTALS-Kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4648–4659, Nov. 2021.

[22] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic (ARITH)*, Jun. 2021, pp. 94–101.

[23] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, 1971.

[24] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, no. 2. Moscow, Russia: Russian Academy of Sciences, 1962, pp. 293–294.

[25] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 1999, pp. 388–397.

[26] D. S. Roche, "Space- and time-efficient polynomial multiplication," in *Proc. Int. Symp. Symbolic Algebr. Comput.*, Jul. 2009, pp. 295–302.

[27] A. Parent, M. Roetteler, and M. Mosca, "Improved reversible and quantum circuits for Karatsuba-based integer multiplication," in *Proc. 12th Conf. Theory Quantum Comput., Commun., Cryptogr. (TQC)*, Cham, Switzerland: Springer, 2017, pp. 7:1–7:15.

[28] C. Gidney, "Asymptotically efficient quantum Karatsuba multiplication," 2019, *arXiv:1904.07356*.

[29] G. Banegas, D. J. Bernstein, I. van Hoof, and T. Lange, "Concrete quantum cryptanalysis of binary elliptic curves," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 1, pp. 451–472, 2020. [Online]. Available: https://doi.org/10.46586/tches.v2021.i1.451-472

[30] C. H. Bennett, "Time/space trade-offs for reversible computation," *SIAM J. Comput.*, vol. 18, no. 4, pp. 766–776, Aug. 1989.

[31] R. Král'ovič, "Time and space complexity of reversible pebbling," in *Proc. Int. Conf. Current Trends Theory Pract. Comput. Sci.* Berlin, Germany: Springer, 2001, pp. 292–303.

[32] M. Amy, "Algorithms for the optimization of quantum circuits," M.S. thesis, Dept. Comput. Sci., Quantum Inf., Univ. Waterloo, Waterloo, ON, Canada, 2013.

[33] S. Kepley and R. Steinwandt, "Quantum circuits for $\mathbb{F}_{2^n}$-multiplication with subquadratic gate count," *Quantum Inf. Process.*, vol. 14, no. 7, pp. 2373–2386, May 2015.

[34] I. van Hoof, "Space-efficient quantum multiplication polynomials for binary finite fields with sub-quadratoc Toffoli gate count," *Quantum Inf. Comput.*, vol. 20, nos. 9–10, pp. 721–735, Aug. 2020, doi: 10.26421/qic20.9-10-1.

RINI WISNU WARDHANI (Graduate Student Member, IEEE) received the Diploma degree in cryptography from the National Cryptography Institute, the bachelor's degree in electrical engineering from Pancasila University, and the master's degree in electrical engineering from the University of Indonesia. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Pusan National University, South Korea. She was a Researcher with the National Cyber and Crypto Agency (BSSN), Indonesia. She was a Lecturer with the Hardware Security Program, National Cyber and Crypto Polytechnic (Poltek SSN), Indonesia. Her research interests include hardware security, information security, cryptography, and quantum computing.

DEDY SEPTONO CATUR PUTRANTO (Member, IEEE) received the Diploma degree in cryptography from the National Cryptography Institute, the bachelor's degree in electrical engineering from Pancasila University, Indonesia, the master's degree in electrical engineering from the University of Indonesia, and the Ph.D. degree in nanovision technology from Shizuoka University, Hamamatsu, Japan. He is currently a Postdoctoral Researcher with Pusan National University. He has experience as a Researcher and a Lecturer with the National Cyber and Crypto Agency (BSSN), Indonesia, and the National Cyber and Crypto Polytechnic (Poltek SSN), Indonesia. His research interests include hardware security, information security, cryptography, nanotechnology, and quantum computing.

HOWON KIM (Member, IEEE) received the bachelor's degree from the Kyungpook National University (KNU), and the Ph.D. degree from the Pohang University of Science and Technology (POSTECH). He is currently a Professor with the Department of Computer Science and Engineering, the Chief of Energy Internet of Things (IoT) with the IT Research Center (ITRC), and the Chief of the Information Security Education Center (ISEC), Pusan National University (PNU). Before joining PNU, he was with the Electronics and Telecommunications Research Institute (ETRI) as a Team Leader for ten years beginning, in December 1998. He was a Visiting Postdoctoral Researcher with the Communication Security Group (COSY), Ruhr-University Bochum, Germany, from July 2003 to June 2004.