entropy

MDPI

*Article*

# An Efficient CRT-Base Power-of-Two Scaling in Minimally Redundant Residue Number System

**Mikhail Selianinau and Yuriy Povstenko ***

Department of Mathematics and Computer Sciences, Faculty of Science and Technology, Jan Dlugosz University in Czestochowa, al. Armii Krajowej 13/15, 42-200 Czestochowa, Poland
* Correspondence: j.povstenko@ujd.edu.pl; Tel.: +48-343-612-269

**Abstract:** In this paper, we consider one of the key problems in modular arithmetic. It is known that scaling in the residue number system (RNS) is a rather complicated non-modular procedure, which requires expensive and complex operations at each iteration. Hence, it is time consuming and needs too much hardware for implementation. We propose a novel approach to power-of-two scaling based on the Chinese Remainder Theorem (CRT) and rank form of the number representation in RNS. By using minimal redundancy of residue code, we optimize and speed up the rank calculation and parity determination of divisible integers in each iteration. The proposed enhancements make the power-of-two scaling simpler and faster than the currently known methods. After calculating the rank of the initial number, each iteration of modular scaling by two is performed in one modular clock cycle. The computational complexity of the proposed method of scaling by a constant $S_l = 2^l$ associated with both required modular addition operations and lookup tables is estimeted as $k$ and $2k + 1$, respectively, where $k$ equals the number of primary non-redundant RNS moduli. The time complexity is $\lceil \log_2 k \rceil + l$ modular clock cycles.

**Keywords:** residue number system; Chinese Remainder Theorem; modular arithmetic; rank of a number; power-of-two scaling



**Citation:** Selianinau, M.; Povstenko, Y. An Efficient CRT-Base Power-of-Two Scaling in Minimally Redundant Residue Number System. *Entropy* **2022**, *24*, 1824. https://doi.org/10.3390/e24121824

Academic Editors: Jun Chen and Sadaf Salehkalaibar

Received: 23 November 2022
Accepted: 10 December 2022
Published: 14 December 2022



## 1. Introduction

Nowadays, high-performance computing is progressing extremely rapidly. This makes qualitatively new demands to designed number-theoretic methods and computational algorithms. That is why creating fundamentally new and efficient computing tools for fast and reliable parallel data processing is especially important. Modular computational structures occupy a special place among them. Modular arithmetic, i.e., the arithmetic of RNS, creates their mathematical basis.

The inherent parallelism and carry-free properties of RNS provide a high potential for accelerating arithmetic operation compared with conventional weighted number systems (WNS). The main advantage of RNS consists of its unique ability to decompose large integer numbers into a set of small residues and to process them in parallel in independent modular channels.

A steadily growing interest in RNS arithmetic as a unique means of carrying out high-speed calculations stimulates developments focused on providing a fundamentally new performance level when carrying out large volumes of time-consuming calculations. The modular arithmetic has attracted the considerable attention of researchers and developers in number-theoretic methods [1–3], computer technology [4,5], digital signal and image processing [3,5–8], cryptography [5,8–10], computer networks and communication systems [3,5], and other areas [9].

In this regard, one of the most promising ways in the specified area is the development of high-speed parallel modular computational structures as well as the enhancement of their functionality and optimization. In this case, the main optimization criteria are the

minimum redundancy of data coding, the execution time minimization of implemented computational procedures, and the throughput maximization of the corresponding computational structures.

As is known, compared with WNS, the residue code of a number does not explicitly contain information about its integer value. Therefore, in RNS arithmetic, the implementation of operations, which require the estimating of the integer value of a number by its residue code, i.e., the evaluation of number position in the operating range, encounters specific difficulties. Such operations, in contrast to modular ones, are called non-modular.

The positional characteristics of the residue code such as core function, the rank of a number, interval index, and others, and the associated forms of number representation are of great importance for designing algorithms of non-modular operations [1,5,7,8]. The computational complexity of calculating the used positional characteristics ultimately determines the efficiency of the corresponding configuration of RNS arithmetic.

Division is one of the most complex arithmetic operations. Even in computers operating in a positional system, this operation stands apart, and its execution requires much more time than most elementary operations. In RNS arithmetic, the hardships with division operations are related to the non-modular character of this operation. This means that the residue of the quotient concerning primary RNS modulus is determined not only by the dividend and divisor residues, and it is necessary to get the additional information, in one form or another, about the integer values of the dividend and divisor [1,7]. It is no coincidence that many publications are devoted to the problem of modular division, for example [11–19].

Along with the general division, modular scaling, i.e., the division of the RNS number by a constant, is a commonly used operation [3,5,8,10]. This operation plays a fundamental role in constructing residue arithmetic algorithms and is of great practical importance. The need for scaling is due to several tasks, for example, to round the floating point numbers with the residue representation of the mantissa and to reduce the dynamic range in digital signal processing and long-word-length cryptography. In addition, scaling by a power of two is often one of the integral steps of more complex non-modular operations, for example, in the method of general modular division [19,20].

Thus, developing novel approaches and methods for fast scaling is highly important in high-performance computing based on parallel algorithmic structures of RNS, especially for high-speed implementing digital signal processing applications and public-key cryptosystems. That should make it possible to widely use modular arithmetic in various priority areas of science and technology.

In the paper, we present a new approach to the power-of-two scaling based on using minimal redundancy of residue code, the rank form of a number, and fast calculation of the rank characteristic at each iteration of the scaling procedure. Compared with the conventional non-redundant RNS, the proposed method makes it possible to optimize and speed up the non-modular scaling operation and concurrently reduces its computational complexity to a large extent.

The paper is structured as follows. Section 2 discusses the basic theoretical concepts of the research. Section 3 presents the known approaches to rank calculation. Sections 4 and 5 describe the RNS scaling algorithms, and the mathematical basis of the rank calculation in the bisection scaling method. Section 6 presents a novel power-of-two scaling algorithm and a numerical example. Section 7 provides discussion and Section 8 concludes the paper.

## 2. The Basic Concepts of RNS Arithmetic

Abstract algebra and number theory [21,22] constitute the theoretical foundation of RNS arithmetic.Traditionally, the apparatus of congruences is used for the mathematical formalization of an RNS with integer ranges. At the same time, Euclid's Division Lemma plays a fundamental role in building an RNS of the concerned type. For the ring **Z** of integers, it is formulated as follows.

**Lemma 1** (**Euclid's Division Lemma**). *For any $X \in \mathbf{Z}$ and a positive integer $m$, there exists a unique pair of integers $Q, R$ such that*

$$X = Qm + R, \tag{1}$$

*where $R \in \mathbf{Z}_m = \{0, 1, \ldots, m-1\}$.*

On the set $\mathbf{Z}$ of integers, a non-redundant RNS is defined using pairwise prime moduli $m_1, m_2, \ldots, m_k$ $(k > 1)$ by the mapping $\mathbf{Z} \to \mathbf{Z}_{m_1} \times \mathbf{Z}_{m_2} \times \cdots \times \mathbf{Z}_{m_k}$, which assigns to each $X \in \mathbf{Z}$ the $k$-tuple $(\chi_1, \chi_2, \ldots, \chi_k)$ of least nonnegative residues $\chi_i = |X|_{m_i}$ of dividing $X$ by $m_i$ $(i = 1, 2, \ldots, k)$. At the same time, the notation $X = (\chi_1, \chi_2, \ldots, \chi_k)$ is used.

The residue code $(\chi_1, \chi_2, \ldots, \chi_k)$ corresponds to the set of all integers $X$ satisfying the system of simultaneous linear congruences

$$\begin{cases} X \equiv \chi_1 \pmod{m_1}, \\ X \equiv \chi_2 \pmod{m_2}, \\ \qquad \cdots \\ X \equiv \chi_k \pmod{m_k} \end{cases} \tag{2}$$

The following statement is true [9,23,24].

**Theorem 1** (**Chinese Remainder Theorem**). *Let the moduli $m_1, m_2, \ldots, m_k$ be pairwise prime, and let $M_k = \prod_{i=1}^{k} m_i$, $M_{i,k} = M_k / m_i$, $\mu_{i,k} = \left| M_{i,k}^{-1} \right|_{m_i}$ $(i = 1, 2, \ldots, k)$. Then the system of congruences (2) has a unique solution, the class of residues modulo $M_k$, defined by the congruence*

$$X \equiv \sum_{i=1}^{k} M_{i,k} \mu_{i,k} \chi_i \pmod{M_k}. \tag{3}$$

The practical application of the RNS assumes that each residue code $(\chi_1, \chi_2, \ldots, \chi_k)$ must correspond only to one integer number. Therefore, certain sets of representatives of residue classes are used as the number range to ensure required single-valued correspondence. Since in the given RNS it is possible to represent $M_k$ integers, the set $\mathbf{Z}_{M_k} = \{0, 1, \ldots, M_k - 1\}$ is usually used in computer applications as an RNS operating range.

Because of the above, we define modular coding as a mapping $\mathbf{\Phi}_{RNS} : \mathbf{Z}_{M_k} \to \mathbf{Z}_1 \times \mathbf{Z}_2 \times \cdots \times \mathbf{Z}_{m_k}$, which assigns a residue code $(\chi_1, \chi_2, \ldots, \chi_k)$ to each $X \in \mathbf{Z}_{M_k}$.

The decoding mapping $\mathbf{\Phi}_{RNS}^{-1} : \mathbf{Z}_1 \times \mathbf{Z}_2 \times \cdots \times \mathbf{Z}_{m_k} \to \mathbf{Z}_{M_k}$ based on the CRT (3) executes according to the rule

$$X = \left| \sum_{i=1}^{k} M_{i,k} \mu_{i,k} \chi_i \right|_{M_k}. \tag{4}$$

Applying Euclid's Division Lemma (1), we can write

$$\mu_{i,k} \chi_i = \left| \mu_{i,k} \chi_i \right|_{m_i} + \left\lfloor \frac{\mu_{i,k} \chi_i}{m_i} \right\rfloor m_i = \chi_{i,k} + \left\lfloor \frac{\mu_{i,k} \chi_i}{m_i} \right\rfloor m_i, \tag{5}$$

where $\chi_{i,k}$ is a normalized residue modulo $m_i$:

$$\chi_{i,k} = \left| \mu_{i,k} \chi_i \right|_{m_i} (i = 1, 2, \ldots, k), \tag{6}$$

$\lfloor x \rfloor$ denotes the largest integer less than or equal to $x$.

Substituting (5) into (4), and taking into consideration (6), we have

$$X = \left| \sum_{i=1}^{k} M_{i,k}\chi_{i,k} + M_k \sum_{i=1}^{k} \left\lfloor \frac{\mu_{i,k}\chi_i}{m_i} \right\rfloor \right|_{M_k}$$

that is equivalent to

$$X = \left| \sum_{i=1}^{k} M_{i,k}\chi_{i,k} \right|_{M_k}. \tag{7}$$

Since the summands in (7) have narrower change bounds, the use of (4), which is a normalized analog of (1), is preferable for constructing RNS arithmetic.

Equation (7) is called the CRT-form of representing the integer $X = (\chi_1, \chi_2, \ldots, \chi_k)$ from the RNS number range $\mathbf{Z}_{M_k}$.

The mapping $\mathbf{\Phi}_{RNS}$ is an isomorphism concerning the basic arithmetic operations. The operation $\circ \in \{+, -, \times\}$ on arbitrary elements $A$ and $B$ given by their residue codes $A = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $B = (\beta_1, \beta_2, \ldots, \beta_k)$ is carried out by the rule

$$\begin{aligned} A \circ B &= (\alpha_1, \alpha_2, \ldots, \alpha_k) \circ (\beta_1, \beta_2, \ldots, \beta_k) = \\ &= \left( |\alpha_1 \circ \beta_1|_{m_1}, |\alpha_2 \circ \beta_2|_{m_2}, \ldots, |\alpha_k \circ \beta_k|_{m_k} \right), \end{aligned} \tag{8}$$

where $\alpha_i = |A|_{m_i}, \beta_i = |B|_{m_i}, i = 1, 2, \ldots, k$.

In the RNS, according to (8), the modular addition, subtraction, and multiplication are performed independently for each modulus $m_i$ $(i = 1, 2, \ldots, k)$. It must be noted that (8) is correct only if the result $A \circ B$ of the arithmetic operation does not go beyond the RNS number range, i.e., if $A \circ B \in \mathbf{Z}_{M_k}$.

The RNS inherent code parallelism illustrated by (8), which consists of the decomposition of arithmetic operations on integers $A$ and $B$ into independent small word length operations on the like digits $\alpha_i$ and $\beta_i$ of residue code, is the main advantage of modular arithmetic compared with the arithmetic of weighted number systems (WNS). Realizing this advantage to the fullest extent is a key strategic goal of all computer applications in the RNS.

As is known, in contrast to the positional code, the residue code $(\chi_1, \chi_2, \ldots, \chi_k)$ of the number $X$ does not explicitly contain information about its value. Therefore, the implementation in the RNS arithmetic operations that require calculating the so-called positional characteristics which give information about the numbers location in the RNS range $\mathbf{Z}_{M_k}$ encounters specific difficulties. Such procedures, in contrast to modular ones, are called non-modular.

The efficiency factor of RNS arithmetic, to a decisive extent, is determined by the optimality of the applied non-modular procedures. At the same time, the main factor that has the most impact on the quality indicators of algorithms for non-modular operations is the computational complexity of calculating the positional characteristics of the residue code and related integer representation forms.

As for Equation (7), its direct application as the general form of integers for building non-modular procedures is practically impossible due to the complexity of straightforward implementation, especially in the case of large $M_k$. At the same time, the use of the specific positional characteristics enables us to obtain from (7) the relevant forms of integer representation, which have good implementation properties and make it possible to overcome the problem of time-consuming addition operations modulo $M_k$.

As follows from (7), the difference

$$\sum_{i=1}^{k} M_{i,k}\chi_{i,k} - X$$

is a multiple of $M_k$. Hence, the following equality holds

$$X = \sum_{i=1}^{k} M_{i,k}\chi_{i,k} - \rho_k(X)M_k. \tag{9}$$

The positional characteristic $\rho_k(X)$ is called a rank of the number $X$. In essence, the rank $\rho_k(X)$ is a CRT reconstruction coefficient that indicates how many times the upper bound $M_k$ of the number range is exceeded when the integer value of the number $X$ is calculated by its residue code $(\chi_1, \chi_2, \ldots, \chi_k)$.

Equation (9) is called a rank form of the integer $X$.

From (9), it also follows that the rank $\rho_k(X)$ is a quotient of the integer division of $X_k$ by $M_k$.

Hence, we obtain

$$\rho_k(X) = \left\lfloor \frac{1}{M_k} \sum_{i=1}^{k} M_{i,k}\chi_{i,k} \right\rfloor = \left\lfloor \sum_{i=1}^{k} \frac{\chi_{i,k}}{m_i} \right\rfloor. \tag{10}$$

Therefore, since $\chi_{i,k} \in \mathbf{Z}_{m_i}$ $(i = 1, 2, \ldots, k)$, the inequality $0 \le \rho_k(X) \le k - 1$ holds.

Compared with (7), Equation (9) does not contain time-consuming reduction modulo $M_k$. Therefore, designing non-modular procedures in RNS arithmetic on the basis of the rank form has a substantial lead over the canonical CRT implementation.

## 3. The Approaches Currently Used to Calculate the Rank of a Number

First, the rank of a number as a main RNS integral characteristic has been studied in [1], and later in [2]. The rank evaluation algorithm consists of a slow $k$-step iterative procedure of sequential additions large modulo of specific constants defined by the chosen RNS moduli-set $\{m_1, m_2, \ldots, m_k\}$. Moreover, the upper bound of the rank $r(X)$ depends on the values of the weights $\mu_{1,k}, \mu_{2,k}, \ldots, \mu_{k,k}$ (see (4)), and can be sufficiently large for most moduli-sets suitable for practical use. If we assume that the processing of such long $L$-bit word-length numbers $L = (\lceil \log_2 M_k \rceil)$ is comparable in time with $k$ operations on the small residues, then the complexity of this method is equal to $O(k^2)$. Because of that, the given approach to the rank calculation is time-consuming and practically unacceptable for high-performance computing due to its computational complexity, especially when using huge $M_k$.

The so-called "extra modulus method" for rank calculation has been proposed in [25]. It rearranges the canonical CRT implementation to an exact integer equation, i.e., the same form as (9). To be able to retrieve the value of the CRT reconstruction coefficient, i.e., the rank of a number, the extra-modulus $m_e$ must satisfy the following conditions: $m_e > k$, and $m_e$ is any integer prime to $M_k$. In this way, the slow and challenging addition modulo $M_k$ in the straightforward CRT implementation is replaced by subtraction and multiplication modulo $m_e$. Thus, we have an extra modular channel for rank calculation. This method works well and correctly when it assumes that proper redundant residue $|X|_{m_e}$ is available. Hence, the "extra modulus method" is suitable for the base extension operation. At the same time, when the number under consideration results from the modular addition or subtraction operation [26], it cannot be used owing to eventual overflow or underflow, respectively. Thus, in such a case, the exact value of $|X|_{m_e}$ is not available. Therefore, this method is not applicable for sign determination and magnitude comparison of two numbers in RNS.

A different approach to evaluating the CRT reconstruction coefficient is proposed in [27–29]. The main idea of the so-called "fractional domain method" consists in the representation of the reconstruction coefficient $r$ as an integer part of a sum of at most $k$ proper fractions (see (10)). The value $r$ is recursively estimated by approximating terms of a fraction $\chi_{i,k}/m_i$. To avoid division by the modulus $m_i$ in the fraction, the denominator $m_i$ is replaced by $2^n$ $(m_i < 2^n)$, while the numerator $\chi_{i,k}$ is approximated by its most significant

$v$ bits $(v < n)$ $(i = 1, 2, \ldots, k)$. Since the division by powers of 2 is equivalent to simple shifts, then the calculation of $r$ can be implemented by addition only.

The main drawbacks of this method consist of the following. First of all, full-precision fractional computations are required. In any case, such calculations are slower than operating on smaller word-length, and the full-precision fractional bits require substantial storage. On the other hand, the number of iterations required is of the order of the bit-length needed for the approximation. For example, the method employing a fractional interpretation of the CRT [27] needs a very high precision of $\lceil \log_2(kM_k) \rceil$ bits. The method proposed in [28,29] uses a sequential bit-by-bit manner for evaluating reconstruction coefficient $r$. The iterative structure of this method makes it very slow in the case of large word-length numbers.

There are also approaches to reconstruct the integer value of RNS number based on the CRT by using special moduli-sets with a limited number of moduli such as $m = 2^n + d$ $(d \in \{-1, 0, 1\})$ [5,8]. The main drawback of these methods consists of a small number of the selected moduli, typically from three to five. Such moduli sets are suitable for the efficient implementations of digital signal processing algorithms but completely not applicable for the processing of large numbers which are widely used in cryptography.

In recent decades, the CRT algorithm, corresponding forms of number representation, and the methods of integer reconstruction by residue code have been intensively studied, especially concerning their application in high-performance computing. The major efforts are aimed at reducing the computational complexity of calculating the main integral characteristics of residue code.

There are some new approaches for calculating an approximate value of the rank of a number which allow us to reduce the computational complexity of complicated non-modular operations in RNS arithmetic [30–32]. The method proposed in [30] is based on the so-called interval floating-point characteristic which provides information about the range of changes in the relative value of RNS representation. Generally, it enables us to perform effectively such operations as magnitude comparison, sign determination, and overflow detection. The concept of an approximate value of the rank of a number is introduced in [31]. This approach allows us to reduce the computational complexity of the decoding from residue code to binary representation and decrease the size of the required coefficients. Based on the properties of the approximate value and arithmetic properties of RNS, a new method for error detection, correction, and controlling computational results has been proposed. In [32], a new original general-purpose technique for CRT basis extension and scaling in RNS using floating-point arithmetic for the rank estimation is proposed for a homomorphic encryption scheme. The main algorithmic improvements focus on optimizing decryption and homomorphic multiplication in the RNS using the CRT to represent and manipulate the large coefficients in the ciphertext polynomials.

The rank positional characteristic has been thoroughly investigated in [33,34]. As shown, the rank $\rho_k(X)$ has a simple structure, high modularity of calculation, and a small range of changes. At the same time, the rank $\rho_k(X)$ is a sum of two small numbers, namely, the inexact rank $\widehat{\rho}_k(X) < k$ and two-valued rank correction $\Delta_k(X) \in \{0, 1\}$:

$$\rho_k(X) = \widehat{\rho}_k(X) + \Delta_k(X), \tag{11}$$

where

$$\widehat{\rho}_k(X) = \left\lfloor \frac{1}{m_k} \sum_{i=1}^{k} R_{i,k}(\chi_i) \right\rfloor \tag{12}$$

and

$$R_{i,k}(\chi_i) = \left\lfloor \frac{m_k \chi_{i,k}}{m_i} \right\rfloor = \left\lfloor \frac{m_k \left| \mu_{i,k} \chi_i \right|_{m_i}}{m_i} \right\rfloor \ (i = 1, 2, \ldots, k-1), \tag{13}$$

$$R_{k,k}(\chi_k) = \chi_{k,k} = \left| \mu_{k,k} \chi_k \right|_{m_k}. \tag{14}$$

In conventional non-redundant RNS, as it follows from (11)–(14), the calculation of the inexact rank $\widehat{\rho}_k(X)$ is reduced to a summation of $k$ small residues $R_{1,k}(\chi_1)$, $R_{2,k}(\chi_2)$,

..., $R_{k,k}(\chi_k)$ modulo $m_k$ taking into account the number of the overflows occurring during the modular addition operations. At the same time, as demonstrated in [34], the main computational cost is associated with the estimation of the rank correction $\Delta_k(X)$. Its evaluation requires concurred modular addition operations in all independent modular channels corresponding to primary RNS moduli $m_1, m_2, \ldots, m_k$. These computations can be implemented easily by the pre-computation and lookup table techniques. As a result, the total number of required modular addition operations and lookup tables for rank $\rho_k(X)$ calculation are $\left(k^2+5k-10\right)/2$ and $\left(k^2+k-2\right)/2$, respectively.

As shown in [34], the minimum redundancy residue code enables optimization of the rank calculation. It assumes the extension of non-redundant residue code $(\chi_1, \chi_2, \ldots, \chi_k)$ of the number $X$ by the redundant residue $\chi_0 = |X|_{m_0}$ concerning extra modulus $m_0 = 2$, i.e., by adding the parity of the number $X$ to its residue representation. Therefore, in minimally redundant RNS, the number $X \in \mathbf{Z}_{M_k}$ is represented by its minimally redundant residue code $(\chi_0, \chi_1, \ldots, \chi_k)$. So, the total residue code length increases by only one bit.

The main advantage of minimally redundant RNS compared with non-redundant analogs consists of a significant simplification of calculating the rank correction $\Delta_k(X)$ and, accordingly, the rank $\rho_k(X)$.

The use of minimum redundancy residue code makes it possible to replace in (11) the rank correction $\Delta_k(X)$, which evaluation is time-consuming and requires performing addition operations in all modular channels, with a trivially calculated binary attribute $\delta_k(X) \in \{0,1\}$. At the same time,

$$\rho_k(X) = \widehat{\rho}_k(X) + \delta_k(X) \tag{15}$$

and

$$\delta_k(X) = \left| \chi_0 + \sum_{i=1}^{k} \psi_i + \widehat{\rho}_0 \right|_2, \tag{16}$$

where $\chi_0 = |X|_2$, $\psi_i = \left|\chi_{i,k}\right|_2 = \left|\left|\mu_{i,k}\chi_i\right|_{m_i}\right|_2$, and $\widehat{\rho}_0 = |\widehat{\rho}_k(X)|_2$.

Compared with non-redundant analogs, the use of minimally redundant RNS enables us to reduce significantly the complexity of the rank $\rho_k(X)$ calculation both in terms of required modular addition operations and lookup tables. At the same time, the corresponding computational cost is $k$ modular addition operations and $k$ one-input lookup tables. The time complexity depends only on the number of primary RNS moduli and equals $T_{rank} = \lceil \log_2 k \rceil$ modular clock cycles.

As shown in [34], the transition to the minimum redundant residue code enables a decrease in the computational complexity of the rank calculation from the order $O\left(k^2/2\right)$ to $O(k)$ concerning required modular addition operations and lookup tables. Thus, the computational complexity reduction factor increases with the number $k$ of non-redundant moduli $m_1, m_2, \ldots, m_k$ and asymptotically approaches the threshold $k/2$.

The use of minimally redundant RNS ensures significant optimization of calculating the rank $\rho_k(X)$ of the number $X$. Moreover, this is also applied to the implementation of the CRT algorithm and, correspondingly, to the execution of various non-modular operations based on it. First of all, that is caused owing to the extreme simplicity evaluation of two-valued characteristic $\delta_k(X) \in \{0,1\}$ as well as the modular structure of the main calculation equation for inexact rank $\widehat{\rho}_k(X)$ (see (12)). This circumstance enables radical simplifying the calculation of the rank $\rho_k(X)$ in minimally redundant RNS in comparison with conventional non-redundant RNS and, consequently, makes it possible to construct faster and optimal with respect to computational complexity variants of RNS arithmetic.

Therefore, the application of minimally redundant residue representation takes priority over conventional non-redundant RNS arithmetic to implement the scaling procedures based on the rank form of a number.

## 4. The Main Types of Scaling Algorithms in RNS Arithmetic

In the conventional WNS, the power-of-two scaling is performed simply by right shifting. In the RNS, compared with WNS, this procedure has substantial difficulty because it is not easily implementable due to its non-positional nature.

The classical power-of-two scaling method consists of the residue code conversion to binary representation, scaling in the conventional WNS, and converting the result back to the RNS.

Unlike the WNS, the residue code does not contain explicit information about the integer value of the represented number. Therefore, in addition to its usual purpose, which consists of limiting the undesirable growth of calculation results, the scaling in RNS is also used to detect the position of integers in a particular range (i.e., to evaluate their values), rounding, and solving other similar tasks. This operation is often used in more complex non-modular procedures such as general modular division. Many different scaling algorithms, which do not require RNS-to-binary conversion, have been presented in the literature. A detailed review of the known modular scaling methods is presented in [8].

The essence of the modular scaling operation is to obtain some integer approximation $\widehat{X} = (\widehat{\chi}_1, \widehat{\chi}_2, \ldots, \widehat{\chi}_k)$ $(i = 1, 2, \ldots, k)$ to the fraction $X/S$, where $X = (\chi_1, \chi_2, \ldots, \chi_k)$ is an arbitrary element of the RNS number range $\mathbf{Z}_{M_k}$, and $S$ is a constant factor (scale). The fraction $X/S$ is usually approximated by the integers $\lfloor X/S \rfloor$ and $\lceil X/S \rceil$ ($\lfloor x \rfloor$ and $\lceil x \rceil$ are the floor and ceiling function of $x$, respectively).

The most important aspect of the scaling problem in RNS is to ensure the high flexibility of the created algorithmic tools. That implies adoption of the set $S = \{S_0, S_1, \ldots, S_{\Lambda-1}\}$ of scales $S_l > 1$ $(l = 0, 1, \ldots, \Lambda - 1)$ which is usually chosen based on the criterion for the minimum calculating error under a given constraint on the number of scaling factors.

All known scaling techniques can be classified into four main categories:

1. scaling by the product of some RNS moduli [32,35–38],
2. scaling by an integer from the RNS number range [39,40],
3. scaling by a common fraction [41],
4. scaling by a power of two [42–44].

In the first group, many scaling methods take the scaling factor $S$ as a product of $l$ moduli, i.e., of the form $S = M_l$ $(l = 1, 2, \ldots, k-1)$ [35–38]. That makes it easier to obtain the residues $\widehat{\chi}_{l+1}, \widehat{\chi}_{l+2}, \ldots, \widehat{\chi}_k$ of the approximation $\widehat{X}$ to the fraction $X/S$. The remaining residues $\widehat{\chi}_1, \widehat{\chi}_2, \ldots, \widehat{\chi}_l$ can be calculated sufficiently lightly within the framework of procedures based on one of the base extension algorithms [2,35,45]. Due to the small word length of residues, the pre-computation and lookup table techniques are suitable for modular scaling.

In [35], the base extension algorithm uses the reverse conversion of residue code to mixed-radix representation. The method proposed in [36] requires a redundant modulus to evaluate the CRT reconstruction coefficient, i.e., the rank of a number, to complete the base extension procedure. In [38], the suggested approach is entirely based on a lookup tables technique, while all the required tables have two inputs. At the same time, the memory costs are too high when the number of chosen moduli is sufficiently large. The method proposed in [37] enables one to carry out base extension and exact scaling without some system redundancy only by using additional lookup tables.

The CRT-base technique for modular scaling by an integer has been suggested in [39]. Here, the main idea is to approximate the CRT calculating relation for reconstructing the integer value of RNS numbers. This enables the substitution of large modulo $M_k$ addition in the canonic CRT-decoding scheme by smaller word-length modular addition operations. In [40], the proposed method uses minimum redundancy for modular scaling by arbitrary positive scales. The distinctive feature of the algorithm consists of using the interval index as a positional characteristic of residue code. At the same time, the interval index can be calculated fast and lightly by modular addition of small residues in the $k$th modular channel corresponding to the modulus $m_k$ from the RNS moduli-set $\{m_1, m_2, \ldots, m_k\}$.

In the case of arbitrary rational scale $S$, an efficient basis for modular scaling is the approach presented in [41]. The main feature is that for the scales of the form $S = p/q$, the numbers $p$ and $q$ can take any integer values for which the fraction $qX/p$ does not exceed the upper bound of the RNS number range. In addition, both the number $qX$ and the results of intermediate calculations may not satisfy the specified requirement.

The scaling methods in the fourth group implement division by constants of the form $S = 2^l$ $(l = 1, 2, \ldots, \Lambda)$, $\Lambda \leq \lfloor \log_2 M_k \rfloor$ [7,42,43]. General approaches to solving this task are based mainly on the bisection method. It consists of calculating the recurrence relation $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ for $j = 0, 1, \ldots, l-1$. In this case, $X^{(0)} = X$, and $X^{(l)} = \lfloor X/2^l \rfloor$. The residue $\widehat{\chi}_i^{(j+1)}$ $(i = 1, 2, \ldots, k)$ of approximation $X^{(j+1)}$ is determined as

$$\chi_i^{(j+1)} = \begin{cases} \left| \frac{1}{2}\chi_i^{(j)} \right|_{m_i} & \text{if } X^{(j)} \text{ is even,} \\ \left| \frac{1}{2}\left(\chi_i^{(j)} - 1\right) \right|_{m_i} & \text{if } X^{(j)} \text{ is odd,} \end{cases} \tag{17}$$

while all the primary moduli $m_1, m_2, \ldots, m_k$ are coprime odd numbers. The last condition ensures that 2 and $m_i$ $(i = 1, 2, \ldots, k)$ are relatively prime numbers, and, correspondingly, the existence of a modular multiplicative inverse of 2, i.e., the number $\left|2^{-1}\right|_{M_k} = \left(\left|2^{-1}\right|_{m_1}, \left|2^{-1}\right|_{m_2}, \ldots, \left|2^{-1}\right|_{m_k}\right)$. As followed from (17), the scaling by 2 requires the parity detection of the number $X^{(j)}$, $j = 0, 1, \ldots, l-1$. So, there is a need for a base extension operation to extra modulus equal 2.

An iterative algorithm for scaling by the factor $S = 2^l$ proposed in [42] is implemented in $l$ steps. At the same time, the parity of the intermediate results is checked at each iteration using the base extension operation suggested in [25]. In [43], the power-of-two scaling technique is applied to realize a digital filter in quadratic RNS. The scaling algorithm presented in [44] focuses on arbitrary moduli sets with large dynamic ranges and requires only machine-precision integer and floating-point operations. At the same time, it is used for software implementation of rounding and exponent alignment procedures in a multiple-precision RNS-based arithmetic library for parallel CPU-GPU systems.

Many modular scaling algorithms use special moduli sets with a limited number of moduli. A detailed review of some of these methods is given in [8]. The most commonly used moduli sets for efficient RNS scalers are $\{2^{2n+1}+1, 2^{2n+1}, 2^{2n+1}-1\}$, $\{2^n - 1, 2^{n+p}, 2^n + 1\}$, $\{2^{n+1}-1, 2^n, 2^n - 1\}$, $\{2^{n+p}, 2^n - 1, 2^{n-1} - 1\}$ among others [46–51]. The main drawback of such approaches is imposing very restrictive constraints on the moduli sets. They are certainly suitable for implementing scaling tasks in digital signal processing but, at the same time, they do not fit for scaling and other non-modular operations on numbers belonging to large dynamic ranges which are widely used in long-word-length cryptography.

## 5. A Novel Approach for Calculating the Rank of a Number Resulting from Scaling by 2

In RNS, the rank $\rho_k(X) \in \mathbf{Z}_k = \{0, 1, \ldots, k-1\}$ is a principal positional characteristic since all the non-modular operations, such as magnitude comparison, sign determination, overflow detection, general division, scaling, residue-to-binary conversion, and others, can be implemented on its basis. Because the rank $\rho_k(X)$ enables estimation of the integer value of the RNS-number $X$, then the development of efficient methods and algorithms for its calculating is of primary importance in building efficient variants of RNS arithmetic and, accordingly, high-performance modular computational structures.

Let us show that the rank form (9) of the number representation in residue arithmetic creates a basis for constructing relatively fast and sufficiently simple iterative algorithms for the implementation of division by constant $S_l = 2^l$ $(l = 1, 2, \ldots, \Lambda, \Lambda \leq \lfloor \log_2 M_k \rfloor)$. In this case, the following theorem is fundamental for solving the problem of modular scaling by powers of 2.

**Theorem 2.** *Let in RNS with pairwise prime odd moduli $m_1, m_2, \ldots, m_k$ the arbitrary number $X = (\chi_1, \chi_2, \ldots, \chi_k)$ from the range $\mathbf{Z}_{M_k}$ having rank $\rho_k(X)$ be given. Then the rank of the integer $\widehat{X} = \lfloor X/2 \rfloor$ satisfies the equation*

$$\rho_k\left(\widehat{X}\right) = \begin{cases} \frac{1}{2}\left(\rho_k(X) + \sum_{i=1}^{k} \psi_i\right) & \text{if } X \text{ is even,} \\ \frac{1}{2}\left(\rho_k(X) - \rho_k(1) + \sum_{i=1}^{k} \omega_i + \sum_{i=1}^{k} \varphi_i\right) & \text{if } X \text{ is odd,} \end{cases} \tag{18}$$

*where*

$$\psi_i = \left|\chi_{i,k}\right|_2 = \left|\left|\mu_{i,k}\chi_i\right|_{m_i}\right|_2, \tag{19}$$

$$\omega_i = \begin{cases} 0 \text{ if } \chi_{i,k} \geq \mu_{i,k}, \\ 1 \text{ if } \chi_{i,k} < \mu_{i,k}, \end{cases} \tag{20}$$

$$\varphi_i = \begin{cases} \left|\psi_i + \omega_i\right|_2 & \text{if } \left|\mu_{i,k}\right|_2 = 0, \\ \overline{\left|\psi_i + \omega_i\right|_2} & \text{if } \left|\mu_{i,k}\right|_2 = 1, \end{cases} \tag{21}$$

*$\rho_k(1)$ is the rank of the number 1, and $\overline{x}$ denotes the negation of the Boolean value $x$.*

**Proof.** As follows from the rank form (9), the number 1 in a given RNS has the following form

$$1 = \sum_{i=1}^{k} M_{i,k}\mu_{i,k} - \rho_k(1)M_k.$$

Therefore, we can write

$$\widehat{X} = \lfloor X/2 \rfloor = \frac{1}{2}(X - |X|_2) =$$

$$= \frac{1}{2}\left(\sum_{i=1}^{k} M_{i,k}\chi_{i,k} - \rho_k(X)M_k - |X|_2\left(\sum_{i=1}^{k} M_{i,k}\mu_{i,k} - \rho_k(1)M_k\right)\right) =$$

$$= \frac{1}{2}\left(\sum_{i=1}^{k} M_{i,k}\left(\chi_{i,k} - |X|_2\mu_{i,k}\right) - M_k\left(\rho_k(X) - |X|_2\rho_k(1)\right)\right). \tag{22}$$

Then, in accordance with Euclid's Division Lemma (1), from (22) we have

$$\widehat{X} = \frac{1}{2}\left(\sum_{i=1}^{k} M_{i,k}\left(\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i} + \left\lfloor(\chi_{i,k} - |X|_2\mu_{i,k})/m_i\right\rfloor m_i\right) - \right.$$

$$\left. - M_k\left(\rho_k(X) - |X|_2\rho_k(1)\right)\right).$$

Thus,

$$\widehat{X} = \sum_{i=1}^{k} \frac{1}{2}M_{i,k}\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i} - \frac{1}{2}M_k\left(\rho_k(X) - |X|_2\rho_k(1) - \right.$$

$$\left. - \sum_{i=1}^{k}\left\lfloor(\chi_{i,k} - |X|_2\mu_{i,k})/m_i\right\rfloor\right). \tag{23}$$

Since for each least nonnegative residue $\chi \in \mathbf{Z}_m$ modulo an arbitrary odd modulus $m$, there is a unique formal quotient $|\chi/2|_m$, and

$$|\chi/2|_m = (\chi + m|\chi|_2)/2$$

(see, for example, [1]), then

$$\widehat{\chi}_{i,k} = \left|\frac{1}{2}\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_{m_i} = \frac{1}{2}\left(\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i} + m_i\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2\right).$$

Therefore,

$$\frac{1}{2}\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i} = \widehat{\chi}_{i,k} - \frac{1}{2}m_i\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2.$$

Taking this into account, from (23) we get

$$\widehat{X} = \sum_{i=1}^{k} M_{i,k}\widehat{\chi}_{i,k} - \frac{1}{2}\left(\rho_k(X) - |X|_2\rho_k(1) - \sum_{i=1}^{k}\left\lfloor\left(\chi_{i,k} - |X|_2\mu_{i,k}\right)/m_i\right\rfloor + \right.$$
$$\left. + \sum_{i=1}^{k}\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2\right).$$

Hence, according to the rank form of number representation (9), we conclude that the following equation for the rank $\rho_k\left(\widehat{X}\right)$ of the number $\widehat{X}$ is valid:

$$\rho_k\left(\widehat{X}\right) = \frac{1}{2}\left(\rho_k(X) - |X|_2\rho_k(1) - \sum_{i=1}^{k}\left\lfloor\left(\chi_{i,k} - |X|_2\mu_{i,k}\right)/m_i\right\rfloor + \right.$$
$$\left. + \sum_{i=1}^{k}\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2\right). \tag{24}$$

If the number $X$ is even, then $|X|_2 = 0$, so that

$$\left\lfloor\left(\chi_{i,k} - |X|_2\mu_{i,k}\right)/m_i\right\rfloor = \left\lfloor\chi_{i,k}/m_i\right\rfloor = 0$$

and

$$\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2 = \left|\chi_{i,k}\right|_2 = \psi_i$$

$(i = 1, 2, \ldots, k)$. Therefore, in this case, Equation (24) takes the form

$$\rho_k\left(\widehat{X}\right) = \frac{1}{2}\left(\rho_k(X) + \sum_{i=1}^{k}\psi_i\right)$$

which corresponds to (18).

If the number $X$ is odd, then $|X|_2 = 1$, and it is easy to check that

$$\left\lfloor\left(\chi_{i,k} - |X|_2\mu_{i,k}\right)/m_i\right\rfloor = \left\lfloor\left(\chi_{i,k} - \mu_{i,k}\right)/m_i\right\rfloor = -\omega_i,$$

while

$$\left|\left|\chi_{i,k} - |X|_2\mu_{i,k}\right|_{m_i}\right|_2 = \left|\left|\chi_{i,k} - \mu_{i,k}\right|_{m_i}\right|_2 = \varphi_i$$

$(i = 1, 2, \ldots, k)$, where $\omega_i$ and $\varphi_i$ are two-valued quantities determined by (19) and (20), respectively. In this case, Equation (24) takes the form

$$\rho_k\left(\widehat{X}\right) = \frac{1}{2}\left(\rho_k(X) - \rho_k(1) + \sum_{i=1}^{k}\omega_i + \sum_{i=1}^{k}\varphi_i\right)$$

which also corresponds to (18).

The theorem is proved. □

As it follows from Theorem 2, the rank $\rho_k\left(\widehat{X}\right)$ of the number $\widehat{X} = \lfloor X/2 \rfloor$ can be calculated rapidly and easily only taking into account the known value of the rank $\rho_k(X)$ of the initial number $X$. This circumstance makes it possible to optimize and significantly speed up the execution of the power-of-two scaling operation. In this case, it is not necessary at each iteration to calculate the rank of the number, which is the intermediate result of scaling, by its residue code. At the same time, the complete operation of rank calculation is necessary only for the initial number $X$ at the preliminary stage of the scaling procedure.

## 6. A Novel Power-of-Two Modular Scaling Based on the Rank Positional Characteristic in Minimally Redundant RNS

Theorem 2 implies the following step algorithm for power-of-two scaling in minimally redundant RNS with primary pairwise prime odd modules $m_1, m_2, \ldots, m_k$, extra modulus $m_0 = 2$, and scales of the form $S_l = 2^l$ $(l = 1, 2, \ldots, \Lambda, \Lambda = \lfloor \log_2 M_k \rfloor)$.

**S.1.** Based on the minimum redundant residue code $(\chi_0, \chi_1, \ldots, \chi_k)$ of the original number $X$, the rank $\rho_k(X)$ is calculated following to (12)–(16). In addition, it is assumed that $X^{(0)} = X$, $\chi_i^{(0)} = \chi_i$ $(i = 0, 1, \ldots, k)$, $\chi_{i,k}^{(0)} = \chi_{i,k} = \left|\mu_{i,k}\chi_i\right|_{m_i}$ $(i = 1, 2, \ldots, k)$, and $j = 0$.

**S.2.** For the residue number $X^{(j)} = \left(\chi_0^{(j)}, \chi_1^{(j)}, \ldots, \chi_k^{(j)}\right)$, the integer

$$\Delta^{(j)} = \begin{cases} \sum_{i=1}^{k} \psi_i^{(j)} \text{ if } \chi_0^{(j)} = 0, \\ \sum_{i=1}^{k} \omega_i^{(j)} + \sum_{i=1}^{k} \varphi_i^{(j)} - \rho_k(1) \text{ if } \chi_0^{(j)} = 1 \end{cases} \tag{25}$$

is calculated, where

$$\psi_i^{(j)} = \left|\chi_{i,k}^{(j)}\right|_2, \tag{26}$$

$\omega_i^{(j)}$ and $\varphi_i^{(j)}$ are obtained by formulas similar to (19) and (20), namely:

$$\omega_i^{(j)} = \begin{cases} 0 \text{ if } \chi_{i,k}^{(j)} \geq \mu_{i,k}, \\ 1 \text{ if } \chi_{i,k}^{(j)} < \mu_{i,k}, \end{cases} \tag{27}$$

$$\varphi_i^{(j)} = \begin{cases} \left|\psi_i^{(j)} + \omega_i^{(j)}\right|_2 \text{ if } \left|\mu_{i,k}\right|_2 = 0, \\ \overline{\left|\psi_i^{(j)} + \omega_i^{(j)}\right|_2} \text{ if } \left|\mu_{i,k}\right|_2 = 1, \end{cases} \tag{28}$$

$i = 1, 2, \ldots, k$.

**S.3.** The digits $\chi_1^{(j+1)}$, $\chi_2^{(j+1)}$, …, $\chi_k^{(j+1)}$ of the minimally redundant residue code and the rank $\rho_k\left(X^{(j+1)}\right)$ of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ are determined, respectively, according to the rules

$$\chi_i^{(j+1)} = \left|\frac{1}{2}\left(\chi_i^{(j)} - \chi_0^{(j)}\right)\right|_{m_i} \quad (i = 1, 2, \ldots, k), \tag{29}$$

$$\rho_k\left(X^{(j+1)}\right) = \frac{1}{2}\left(\rho_k\left(X^{(j)}\right) + \Delta^{(j)}\right). \tag{30}$$

**S.4.** The redundant residue $\chi_0^{(j+1)} = \left|(X^{(j+1)}\right|_2$ is calculated according to equation following from the rank form (9)

$$\chi_0^{(j+1)} = \left|\sum_{i=1}^{k} \psi_i^{(j+1)} + \rho_0^{(j+1)}\right|_2, \tag{31}$$

where $\psi_i^{(j+1)} = \left|\chi_{i,k}^{(j+1)}\right|_2 = \left|\left|\mu_{i,k}\chi_i^{(j+1)}\right|_{m_i}\right|_2$ and $\rho_0^{(j+1)} = \left|\rho_k\left(X^{(j+1)}\right)\right|_2$. In essence, it determines the parity of the number $X^{(j+1)}$.

If $j = l-1$, then the number $X^{(j+1)} = X^{(l)} = \lfloor X/2^l \rfloor$ is the required number, and the scaling process ends. Otherwise, the variable $j$ is incremented by one ($j = j+1$), and the jump to step S.2 is carried out.

For its hardware implementation, the most important feature of the above recursive scaling algorithm is that the specified operations on steps S.2, S.3, and S.4 can be combined in time and carried out within one modular clock cycle. Due to this circumstance, after obtaining the rank $\rho_k(X)$, each iteration of RNS number scaling by 2, i.e., each shift of its integer value by one bit to the right, is performed in one modular clock cycle.

Since the calculating process of the rank $\rho_k(X)$ has a pipeline structure, with the appropriate organization of computations the described scaling procedure at low hardware costs provides a reasonably high speed.

It follows from the above that all the necessary calculations within the scaling algorithm can be implemented using tabular computational structures.

For example, the calculation of the inexact rank $\widehat{\rho}_k(X)$ of the initial number $X$ is reduced to a summation of the sets of small residues $\langle R_{1,k}(\chi_1), R_{2,k}(\chi_2), \ldots, R_{k,K}(\chi_k)\rangle$ modulo $m_k$. Simultaneously, we take into account the number of occurred overflows when performing these modular addition operations (see (12)–(14)). Therefore, we need $k$ one-input lookup tables to store the given set, while the bit length of recorded residues is $\lceil \log_2 m_k \rceil$ $(l = 1, 2, \ldots, k)$. At the same time, the estimation of two-valued rank correction $\delta_k(X)$ (see (16)) requires the set $\langle \psi_1, \psi_2, \ldots, \psi_k\rangle$ of least significant bits of normalized residues $\chi_{i,k}$ $(i = 1, 2, \ldots, k)$ of the number $X$ (see (6)).

Similarly, the sets of binary flags $\langle \psi_1^{(j)}, \psi_2^{(j)}, \ldots, \psi_k^{(j)}\rangle$, $\langle \omega_1^{(j)}, \omega_2^{(j)}, \ldots, \omega_k^{(j)}\rangle$ and also $\langle \varphi_1^{(j)}, \varphi_2^{(j)}, \ldots, \varphi_k^{(j)}\rangle$ (see (26)–(28)) enable us to obtain the integer $\Delta^{(j)}$ required for rank calculating in the corresponding iterations of scaling procedure $(j = 0, 1, \ldots, l-1)$. All these binary sets can also be recorded in the appropriate lookup tables.

Thus, the content of the $i$th lookup table corresponding to the input residue $\chi_i^{(j)}$ has the form $\langle R_{i,k}\left(\chi_i^{(j)}\right), \psi_i^{(j)}, \omega_i^{(j)}, \varphi_i^{(j)}\rangle$ $(i = 1, 2, \ldots, k)$, $(j = 0, 1, \ldots, l-1)$.

Below we present the proposed scaling method in the form of a pseudo-code algorithm.

Let us evaluate the computational complexity of the proposed iterative power-of-two scaling method. As follows from the above, Algorithm 1 requires total $T_{scal} = T_{rank} + T_{iter} \times l$ modular clock cycles. According to [33,34], in minimally redundant RNS, the time complexity of calculating the rank $\rho_k(X)$ of the initial number $X$ depends only on the number $k$ of primary RNS moduli and can be evaluated as $T_{rank} = \lceil \log_2 k \rceil$. At the same time, all calculations within each iteration, consisting in obtaining both the minimally redundant residue code $\left(\chi_0^{(j+1)}, \chi_1^{(J+1)}, \ldots, \chi_k^{(j+1)}\right)$ and the rank $\rho_k\left(X^{(j+1)}\right)$ of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ $(j = 0, 1, \ldots, l-1)$, can be performed in one modular clock cycle by using lookup table technique. Therefore, $T_{iter} = 1$. Hence, the algorithm time complexity $T_{scal} = \lceil \log_2 k \rceil + l$ modular clock cycles.

**Algorithm 1:** Power-of-two scaling in minimally redundant RNS

**Input:** $X = (\chi_0, \chi_1, \ldots, \chi_k)$, $S_l = 2^l$

**Output:** $X^{(l)} = \lfloor X/2^l \rfloor = \left(\chi_0^{(l)}, \chi_1^{(l)}, \ldots, \chi_k^{(l)}\right)$

// compute the rank of the number $X$:

$\rho_k(X) = \widehat{\rho}_k(X) + \delta_k(X)$;

$X^{(0)} = X$;

$j = 0$;

// iterations based on the bisection method:

**while** $j < l - 1$ **do**

  // get the sets of binary flags:

  $\psi^{(j)} = \langle \psi_1^{(j)}, \psi_2^{(j)}, \ldots, \psi_k^{(j)} \rangle$;

  $\omega^{(j)} = \langle \omega_1^{(j)}, \omega_2^{(j)}, \ldots, \omega_k^{(j)} \rangle$;

  $\varphi^{(j)} = \langle \varphi_1^{(j)}, \varphi_2^{(j)}, \ldots, \varphi_k^{(j)} \rangle$;

  // compute the rank correction $\Delta^{(j)}$ according to a parity of the number $X^{(j)}$:

  **if** $\chi_0^{(l)} = 0$ **then**

    $\Delta^{(j)} = \sum_{i=1}^{k} \psi_i^{(j)}$;

  **else**

    $\Delta^{(j)} = \sum_{i=1}^{k} \omega_i^{(j)} + \sum_{i=1}^{k} \varphi_i^{(j)} - \rho_k(1)$;

  **end**

  // compute the residue code of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$:

  **for** $i = 1$; $i \le k$ **do**

    $\chi_i^{(j+1)} = \left| \frac{1}{2}\left(\chi_i^{(j)} - \chi_0^{(j)}\right) \right|_{m_i}$;

  **end**

  // get the set of binary flags:

  $\psi^{(j+1)} = \langle \psi_1^{(j+1)}, \psi_2^{(j+1)}, \ldots, \psi_k^{(j+1)} \rangle$;

  // compute the rank of the number $X^{(j+1)}$:

  $\rho_k\left(X^{(j+1)}\right) = \frac{1}{2}\left(\rho_k\left(X^{(j)}\right) + \Delta^{(j)}\right)$;

  // get the least significant bit of the rank $\rho_k\left(X^{(j+1)}\right)$:

  $\rho_0^{(j+1)} = \left| \rho_k\left(X^{(j+1)}\right) \right|_2$;

  // compute the parity of the number $X^{(j+1)}$:

  $\chi_0^{(j+1)} = \left| \sum_{i=1}^{k} \psi_i^{(j+1)} + \rho_0^{(j+1)} \right|_2$;

  // increment the iteration counter:

  $j = j + 1$;

**end**

**return** $X^{(l)} = \lfloor X/2^l \rfloor$;

To illustrate the power-of-two scaling of the number $X = (\chi_0, \chi_1, \ldots, \chi_k)$ based on the rank form (9) in the proposed minimally redundant RNS, we present below a numerical example.

Let us consider the RNS with the primary moduli $m_1 = 5$, $m_2 = 7$, $m_3 = 9$, and $m_4 = 11$, taking into account the excess modulus $m_0 = 2$.

**Example 1.** *Suppose we wish to scale the number* $X = 1731$ *having the minimally redundant residue code* $(\chi_0, \chi_1, \chi_2, \chi_3 \chi_4) = (1, 1, 2, 3, 4)$ *by the constant* $S_3 = 2^3 = 8$.

*Therefore, the number of required iterations is* $l = 3$.

*Before describing the proposed scaling algorithm, we give below the required primitive constants used in the RNS under consideration. So, we have*

$M_4 = 3465$,
$M_{1,4} = 693,\ M_{2,4} = 495,\ M_{3,4} = 385,\ M_{4,4} = 315$,
$\mu_{1,4} = 2,\ \ \mu_{2,4} = 3,\ \ \mu_{3,4} = 4,\ \ \mu_{4,4} = 8$,
$\rho_4(1) = 2$.

***S.1. The rank calculation of the initial number.***

*First, having the non-redundant residue code* $(1,2,3,4)$ *of the number* $X$*, by using lookup tables, we obtain the following sets of residues and least-significant bits, respectively,*

$\langle R_{1,4}(\chi_1), R_{2,4}(\chi_2), R_{3,4}(\chi_3), R_{4,4}(\chi_4)\rangle = \langle 4,9,3,10\rangle$,
$\langle \psi_1, \psi_2, \psi_3, \psi_4\rangle = \langle 0,0,1,0\rangle$.

*Let us show in more detail how these values were obtained, according to* (6), (19), *and* (13), (14), *respectively, before storing in the lookup tables:*

$\chi_{1,4} = |2\cdot 1|_5 = 2,\ \ \psi_1 = |2|_2 = 0$,
$\chi_{2,4} = |3\cdot 2|_7 = 6,\ \ \psi_2 = |6|_2 = 0$,
$\chi_{3,4} = |4\cdot 3|_9 = 3,\ \ \psi_3 = |3|_2 = 1$,
$\chi_{4,4} = |8\cdot 4|_{11} = 10,\ \ \psi_4 = |10|_2 = 0$,
$R_{1,4}(\chi_1) = \lfloor (11\cdot 2)/5\rfloor = 4$,
$R_{2,4}(\chi_2) = \lfloor (11\cdot 6)/7\rfloor = 9$,
$R_{3,4}(\chi_3) = \lfloor (11\cdot 3)/9\rfloor = 3$,
$R_{4,4}(\chi_4) = |8\cdot 4|_{11} = 10$.

*Further, using the set of residues* $\langle 4,9,3,10\rangle$*, according to* (12)*, we calculate the inexact rank*

$$\widehat{\rho}_4(X) = \lfloor (4+9+3+10)/11\rfloor = \lfloor 26/11\rfloor = 2,$$

*and also take its parity bit*

$$\widehat{\rho}_0 = |\widehat{\rho}_4(X)|_2 = 0.$$

*Then, taking into account that* $\chi_0 = 1$*, using the set* $\langle \psi_1, \psi_2, \psi_3, \psi_4\rangle = \langle 0,0,1,0\rangle$ *and* $\widehat{\rho}_0$*, according to* (16)*, we find two-valued correction*

$$\delta_4(X) = |1 + (0+0+1+0) + 0|_2 = |2|_2 = 0.$$

*As a result, according to* (16)*, we get the exact rank of the initial number* $X$

$$\rho_4(X) = \widehat{\rho}_4(X) + \delta_4(X) = 2 + 0 = 2.$$

*To verify the obtained result, using the rank form* (9)*, we find*

$$X = \sum_{i=1}^{4} M_{i,4}\chi_{i,4} - \rho_4(X)M_4 = 693\cdot 2 + 495\cdot 6 + 385\cdot 3 + 315\cdot 10 - 2\cdot 3465 = 1731.$$

*In addition, it is assumed that* $j = 0$, $X^{(0)} = X$, $\chi_i^{(0)} = \chi_i\ \left(i = \overline{0,4}\right)$, $\chi_{i,4}^{(0)} = \chi_{i,4}$, $\psi_i^{(0)} = \psi_i\ \left(i = \overline{1,4}\right)$.

***Iteration 1.***

***S.2.1.*** *Since* $\chi_0^{(0)} = 1$*, using the sets of binary flags (see* (27) *and* (28)*)*

$\langle \omega_1^{(0)}, \omega_2^{(0)}, \omega_3^{(0)}, \omega_4^{(0)}\rangle = \langle 0,0,1,0\rangle$,
$\langle \varphi_1^{(0)}, \varphi_2^{(0)}, \varphi_3^{(0)}, \varphi_4^{(0)}\rangle = \langle 0,1,0,0\rangle$,
*according to* (25)*, we calculate the quantity*

$$\Delta^{(0)} = \sum_{i=1}^{4}\omega_i^{(0)} + \sum_{i=1}^{4}\varphi_i^{(0)} - \rho_k(1) = 1 + 1 - 2 = 0.$$

***S.3.1.*** *We calculate the non-redundant residue code and the rank of the number* $X^{(1)} = \lfloor X^{(0)}/2\rfloor$*, according to* (29) *and* (30)*, respectively:*

$\left(\chi_1^{(1)}, \chi_2^{(1)}, \chi_3^{(1)}, \chi_4^{(1)}\right) = (0,4,1,7)$,

$$\rho_4\left(X^{(1)}\right) = \tfrac{1}{2}\left(\rho_4\left(X^{(0)}\right) + \Delta^{(0)}\right) = \tfrac{1}{2}(2+0) = 1,$$
$$\rho_0^{(1)} = \left|\rho_4\left(X^{(1)}\right)\right|_2 = 1.$$

***S.4.1.*** *Using the set* $\langle \psi_1^{(1)}, \psi_2^{(1)}, \psi_3^{(1)}, \psi_4^{(1)} \rangle = \langle 0,1,0,1 \rangle$ *corresponding to the non-redundant residue code* $(0,4,1,7)$ *and taking into account that* $\rho_0^{(1)} = 1$*, according to* (31)*, we find*

$$\chi_0^{(1)} = |(0+1+0+1)+1|_2 = 1.$$

*Hence, as a result of Iteration 1, we have the minimally redundant residue code* $(1,0,4,1,7)$ *of the number* $X^{(1)} = \lfloor X^{(0)}/2 \rfloor = \lfloor 1731/2 \rfloor = 864$.

***Iteration 2.***

***S.2.2.*** *Since* $\chi_0^{(1)} = 1$*, using the following sets of binary flags*

$\langle \omega_1^{(1)}, \omega_2^{(1)}, \omega_3^{(1)}, \omega_4^{(1)} \rangle = \langle 1,0,0,1 \rangle$,
$\langle \varphi_1^{(1)}, \varphi_2^{(1)}, \varphi_3^{(1)}, \varphi_4^{(1)} \rangle = \langle 1,0,0,0 \rangle$,
*we have*

$$\Delta^{(1)} = \sum_{i=1}^{4} \omega_i^{(1)} + \sum_{i=1}^{4} \varphi_i^{(1)} - \rho_k(1) = 2+1-2 = 1.$$

***S.3.2.*** *We calculate the non-redundant residue code and the rank of the number* $X^{(2)} = \lfloor X^{(1)}/2 \rfloor$*:*

$$\left(\chi_1^{(2)}, \chi_2^{(2)}, \chi_3^{(2)}, \chi_4^{(2)}\right) = (2,5,0,3),$$
$$\rho_4\left(X^{(2)}\right) = \tfrac{1}{2}\left(\rho_4\left(X^{(1)}\right) + \Delta^{(1)}\right) = \tfrac{1}{2}(1+1) = 1,$$
$$\rho_0^{(2)} = \left|\rho_4\left(X^{(2)}\right)\right|_2 = 1.$$

***S.4.2.*** *Using the set* $\langle \psi_1^{(2)}, \psi_2^{(2)}, \psi_3^{(2)}, \psi_4^{(2)} \rangle = \langle 0,1,0,0 \rangle$ *corresponding to the non-redundant residue code* $(2,5,0,3)$ *and taking into account that* $\rho_0^{(2)} = 1$*, we obtain*

$$\chi_0^{(2)} = |(0+1+0+0)+1|_2 = 0.$$

*Hence, as a result of Iteration 2, we have the minimally redundant residue code* $(0,2,5,0,3)$ *of the number* $X^{(2)} = \lfloor X^{(1)}/2 \rfloor = \lfloor X^{(0)}/4 \rfloor = \lfloor 1731/4 \rfloor = 432$.

***Iteration 3.***

***S.2.3.*** *Since* $\chi_0^{(2)} = 0$*, using the set*

$\langle \psi_1^{(2)}, \psi_2^{(2)}, \psi_3^{(2)}, \psi_4^{(2)} \rangle = \langle 0,1,0,0 \rangle$,
*according to* (25)*, we find*

$$\Delta^{(2)} = \sum_{i=1}^{4} \psi_i^{(2)} = 1.$$

***S.3.3.*** *We calculate the non-redundant residue code and the rank of the number* $X^{(3)} = \lfloor X^{(2)}/2 \rfloor$*:*

$$\left(\chi_1^{(3)}, \chi_2^{(3)}, \chi_3^{(3)}, \chi_4^{(3)}\right) = (1,6,0,7),$$
$$\rho_4\left(X^{(3)}\right) = \tfrac{1}{2}\left(\rho_4\left(X^{(2)}\right) + \Delta^{(2)}\right) = \tfrac{1}{2}(1+1) = 1,$$
$$\rho_0^{(3)} = \left|\rho_4\left(X^{(3)}\right)\right|_2 = 1.$$

***S.4.3.*** *Using the set* $\langle \psi_1^{(3)}, \psi_2^{(3)}, \psi_3^{(3)}, \psi_4^{(3)} \rangle = \langle 0,0,0,1 \rangle$ *corresponding to the non-redundant residue code* $(1,6,0,7)$ *and taking into account that* $\rho_0^{(3)} = 1$*, we get*

$$\chi_0^{(3)} = |(0+0+0+1)+1|_2 = 0.$$

*Hence, as a result of Iteration 3, we have the minimally redundant residue code* $(0,1,6,0,7)$ *of the number* $X^{(3)} = \lfloor X^{(2)}/2 \rfloor = \lfloor X^{(0)}/8 \rfloor = \lfloor 1731/8 \rfloor = 216$.

*As far as* $j = l-1 = 2$*, the scaling procedure ends, and the number* $X^{(3)}$ *is the desired solution.*

*To verify the obtained result, according to the rank form (9), we find*

$$X^{(3)} = \sum_{i=1}^{4} M_{i,4}\chi_{i,4}^{(3)} - \rho_4\left(X^{(3)}\right)M_4 = \ 693 \cdot 1 + 495 \cdot 4 + 385 \cdot 0 + 315 \cdot 1 - 1 \cdot 3465 =$$
$$= 3681 - 3645 = 216.$$

*The result is correct.*

The above example shows that the use of minimally redundant RNS enables us to optimize and speed up the power-of-two scaling procedure compared with the conventional non-redundant RNS to a large extent. First of all, that is caused by the extreme simplicity of calculating the inexact rank $\widehat{\rho}_k(X)$ and estimating two-valued characteristic $\delta_k(X)$ of the initial number $X$ as well as by the trivial operations for obtaining the rank $\rho_k\left(X^{(j)}\right)$ $(j = 0, 1, \ldots, l-1)$ at each iteration of the scaling procedure (see Theorem 2).

Therefore, the proposed minimally redundant residue representation takes priority over non-redundant analogs in optimization and speed-up of the scaling and other non-modular procedures based on the CRT implementation using a rank characteristic.

## 7. Discussion

Let us now discuss the theoretical and practical aspects of the approach proposed in this paper.

As followed from (17), the power-of-two scaling algorithm based on the bisection method requires the parity detection of the number $X^{(j)}$ $(j = 0, 1, \ldots, l-1)$ at each iteration. Therefore, fast calculating the residue concerning extra modulus $m_0 = 2$ is a significantly important task.

In conventional non-redundant RNS, the parity detection of the number $X^{(j)} = \left(\chi_1^{(j)}, \chi_2^{(j)}, \ldots, \chi_k^{(j)}\right)$ is usually based on estimating the integer value of $X^{(j)}$ by the use of specific positional characteristics. The generally accepted ones are the digits of mixed-radix representation, core function, the rank of a number, and interval index [1–3,5,8].

In RNS arithmetic, the parity check of a number refers to complicated non-modular operations requiring high computational costs. The computational complexity of this operation is comparable to the computational complexity of the reverse conversion from the residue code into the mixed-radix representation or to the calculation of the rank of a number.

Generally, in non-redundant RNS, the implementation of parallel parity check algorithm requires $O(k^2)$ modular addition operations [33,34]. So it can become computationally expensive for large values of $k$. Thus, for efficient implementation of the power-of-two scaling algorithm based on the bisection method, one needs to speed up and optimize the RNS parity detection technique.

In this article, the proposed approach to power-of-two scaling is based on using the rank of a number as the main RNS positional characteristic. Therefore, in our case, obtaining residue modulo $m_0 = 2$ is reduced to the calculation of the rank $\rho_k\left(X^{(j)}\right)$ with the following use of the rank form (9).

Hence,

$$\chi_0^{(j)} = \left|X^{(j)}\right|_2 = \left|\sum_{i=1}^{k}\left|M_{i,k}\chi_{i,k}^{(j)}\right|_2 + \left|\rho_k\left(X^{(j)}\right)M_k\right|_2\right|_2 = \left|\sum_{i=1}^{k}\psi_i^{(j)} + \rho_0^{(j)}\right|_2.$$

Thus, determining the parity of a number has a computational complexity identical to the complexity of rank calculating concerning the numbers of required modular addition operations $R_{MO}$ and lookup tables $R_{LUT}$. At the same time, obtaining the residue code $\left(\chi_1^{(j+1)}, \chi_2^{(j+1)}, \ldots, \chi_k^{(j+1)}\right)$ of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ needs $k$ additional lookup tables (see (17)).

Therefore, the computational cost of the iterative procedure of scaling by $S_l = 2^l$ consists of $S_{MO} = R_{MO} \times l$ modular addition operations and $S_{LUT} = R_{LUT} + k$ lookup tables, whereas the time complexity is $T_{scal} = T_{iter} \times l$ modular clock cycles, where $T_{iter}$ is a performance time of one iteration based on the bisection method.

Thus, in conventional non-redundant RNS, the computational cost of the canonical power-of-two scaling procedure based on the bisection method (17) and the rank calculation method described in [34] is estimated as

$$S_{MO} = R_{MO} \times l = \frac{1}{2}\left(k^2 + 5k - 10\right) \times l. \tag{32}$$

$$S_{LUT} = R_{LUT} + k = \frac{1}{2}\left(k^2 + 3k - 2\right). \tag{33}$$

The main advantage of the proposed approach to power-of-two scaling over the existing ones consists in the use of minimally redundant RNS and the novel method for calculating the rank of a number resulting from division by two (see Theorem 2) in each iteration of the scaling algorithm. This circumstance enables a significant reduction of the computational complexity of the scaling algorithm.

As follows from [34], the corresponding computational cost of calculating the rank $\rho_k(X)$ of the initial number $X$ is $R^{\star}_{MO} = k$ and $R^{\star}_{LUT} = k$ in terms of required modular addition operations and lookup tables, respectively. Furthermore, the performance time of the rank calculation is $T^{\star}_{rank} = \lceil \log_2 k \rceil$ modular clock cycles (see Section 3).

It is important to note that all calculations at each iteration are implemented using the lookup tables technique and the simplest combinational logic circuits.

As shown above, the minimally redundant residue code of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ $(j = 0, 1, \ldots, l-1)$ is yielded in only one modular clock cycle and needs the use of $k+1$ additional lookup tables. At the same time, the first $k$ of these lookup tables are used for obtaining the residue code $\left(\chi_1^{(j+1)}, \chi_2^{(j+1)}, \ldots, \chi_k^{(j+1)}\right)$, while the last lookup table gives us the rank $\rho_k\left(X^{(j+1)}\right)$ of the number $X^{(j+1)}$ (see (29) and (30)). So, at each iteration, there are no additional modular operations.

The total numbers of required modular addition operations and lookup tables are estimated, respectively, as

$$S^{\star}_{MO} = k \tag{34}$$

and

$$S^{\star}_{LUT} = 2k + 1, \tag{35}$$

The time complexity of the novel power-of-two scaling algorithm is $T^{\star}_{scal} = T^{\star}_{rank} + l = \lceil \log_2 k \rceil + l$ modular clock cycles.

Thus, the use of minimally redundant RNS and novel approach to rank calculation at each iteration of power-of-two scaling (see Theorem 2) enables significant decrease of the computational complexity. The corresponding reduction factors of the computational complexity, in terms of the required modular addition operations (see (32) and (34)) and lookup tables (see (33) and (35)), are

$$C_{MO}(k, l) = \frac{S_{MO}}{S^{\star}_{MO}} = \frac{\left(k^2 + 5k - 10\right)}{2k} \times l, \tag{36}$$

$$C_{LUT}(k) = \frac{S_{LUT}}{S^{\star}_{LUT}} = \frac{k^2 + 3k - 2}{4k + 2}. \tag{37}$$

Below, Tables 1 and 2 present these reduction factors.

**Table 1.** Dependence of the reduction factor $C_{LUT}(k)$ on the moduli number.

| Reduction Factor | Moduli Number | | | | | |
|---|---|---|---|---|---|---|
| | $k = 5$ | $k = 10$ | $k = 15$ | $k = 20$ | $k = 25$ | $k = 30$ |
| $C_{LUT}$ | 1.73 | 3.05 | 4.32 | 5.59 | 6.84 | 8.10 |

**Table 2.** Dependence of the reduction factor $C_{MO}(k, l)$ on the moduli number $k$ and scaling factor $S_l = 2^l$.

| Scaling Factor | | Moduli Number | | | | | |
|---|---|---|---|---|---|---|---|
| $S_l$ | $l$ | $k = 5$ | $k = 10$ | $k = 15$ | $k = 20$ | $k = 25$ | $k = 30$ |
| 8 | 3 | 12.00 | 21,00 | 29.00 | 36.75 | 44.40 | 52.00 |
| 16 | 4 | 16.00 | 28.00 | 38.67 | 49.00 | 59.20 | 69.33 |
| 32 | 5 | 20.00 | 35.00 | 48.33 | 61.25 | 74.00 | 86.67 |
| 64 | 6 | 24.00 | 42.00 | 58.00 | 73.50 | 88.80 | 104.0 |
| 128 | 7 | 28.00 | 49.00 | 67.67 | 85.75 | 103.60 | 121.33 |
| 256 | 8 | 32.00 | 56.00 | 77.33 | 98.00 | 118.40 | 138.67 |
| 512 | 9 | 36.00 | 63.00 | 87.00 | 110.25 | 133.20 | 156.00 |
| 1024 | 10 | 40.00 | 70.00 | 96.67 | 122.5 | 148.00 | 173.33 |

It should be noted that the use of the novel method for calculating the rank $\rho_k\left(X^{(j+1)}\right)$ of the number $X^{(j+1)} = \lfloor X^{(j)}/2 \rfloor$ $(j = 0, 1, \ldots, l-1)$ at each iteration of the scaling procedure (see Theorem 2) in non-redundant RNS, gives us the following computational cost

$$S^{'}_{MO} = R_{MO} = \frac{1}{2}\left(k^2 + 5k - 10\right), \tag{38}$$

$$S^{'}_{LUT} = R_{LUT} + (k+1) = \frac{1}{2}\left(k^2 + 3k\right). \tag{39}$$

Simultaneously, the time complexity is $T^{'}_{scal} = \lceil \log_2 k \rceil + l + 1$ modular clock cycles.

As can be seen, the reduction factors of the computational complexity of power-of-two scaling based on Theorem 2 in minimally redundant RNS compared with conventional non-redundant RNS are represented by the following fractions

$$C^{'}_{MO}(k) = \frac{S^{'}_{MO}}{S^{\star}_{MO}} = \frac{k^2 + 5k - 10}{2k}, \tag{40}$$

$$C^{'}_{LUT}(k) = \frac{S^{'}_{LUT}}{S^{\star}_{LUT}} = \frac{k^2 + 3k}{4k + 2}. \tag{41}$$

In this case, as follows from (40), the reduction factor $C^{'}_{MO}(k) = C_{MO}(k, 1)$ does not depend on the value $S_l = 2^l$ $(l = 0, 1, \ldots, \Lambda - 1)$. At the same time, $C^{'}_{LUT}(k) \approx C_{LUT}(k)$.

The dependence of the reduction factors $C^{'}_{MO}(k)$ and $C^{'}_{LUT}(k)$ on the number of primary RNS moduli $k$ is presented in Table 3.

**Table 3.** Dependence of reduction factors $C^{'}_{MO}$ and $C^{'}_{LUT}$ on the moduli number $k$.

| Reduction Factor | Moduli Number | | | | | |
|---|---|---|---|---|---|---|
| | $k = 5$ | $k = 10$ | $k = 15$ | $k = 20$ | $k = 25$ | $k = 30$ |
| $C^{'}_{MO}$ | 4.00 | 7.00 | 9.67 | 12.25 | 14.80 | 17.33 |
| $C^{'}_{LUT}$ | 1.81 | 3.10 | 4.84 | 5.61 | 6.86 | 8.11 |

Thus, the use of minimally redundant RNS and novel approach to calculating the rank of a number at each iterations of bisection method enables radically simplifying

the carrying out of power-of-two scaling compared with conventional non-redundant RNS. This circumstance enables us to construct faster and optimal in computational cost RNS-oriented complicated computing procedures which widely use scaling algorithms.

## 8. Conclusions

As shown in this paper, the use of minimum-redundancy residue code enables the construction of efficient scaling procedures based on the CRT due to optimizing the calculation of the rank of a number, a principal positional characteristic in RNS arithmetic.

At the beginning stage of the power-of-two scaling procedure, to calculate the rank of the initial number, we apply the approach for the rank calculation proposed by one of the authors in [33,34]. It is reduced to the summation of the small word-length residues $R_{1,k}(\chi_1)$, $R_{2,k}(\chi_2)$, …, $R_{k,k}(\chi_k)$, taking into account the number of occurred overflows during the modular addition operations modulo $m_k$, and fast calculation of two-valued rank correction $\delta_k(X) \in \{0, 1\}$ (see (12) and (16)).

We propose a novel approach to power-of-two scaling based on Theorem 2. Using minimal residue code redundancy, we have optimized and sped up the rank calculation and parity determination of the numbers that result from division by two at each iteration of the bisection method. Each iteration of modular scaling by two is performed in only one modular clock cycle. Thus, owing to the proposed improvements, the power-of-two scaling procedure becomes simplest and faster than the currently known methods.

The computational complexity of the proposed scaling method by constant $S_l = 2^l$ concerning required both modular addition operations and lookup tables is estimated as $k$ and $2k + 1$, respectively, where $k$ equals the number of primary non-redundant RNS moduli. The time complexity is $\lceil \log_2 k \rceil + l$ modular clock cycles.

The use of minimally redundant RNS and a novel approach to calculating the rank of a number at each iteration of the bisection method enables a significant decrease in the power-of-two scaling computational complexity. Corresponding reduction factors concerning the required modular addition operations and lookup tables are given in Tables 1–3.

The proposed approach to power-of-two scaling coincides with the development vector of modern high-performance computing using RNS arithmetic. It enables the implementation of an extensive class of tasks in various areas of science and technology, first of all in cryptography and digital signal processing.

**Author Contributions:** Conceptualization, M.S.; investigation, Y.P.; methodology, M.S.; writing—original draft preparation, M.S.; writing—review and editing, Y.P. All authors have read and improved the final version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

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

## References

1. Akushskii, I.Y.; Juditskii, D.I. *Machine Arithmetic in Residue Classes*; Soviet Radio: Moscow, Russia, 1968. (In Russian)
2. Amerbayev, V.M. *Theoretical Foundations of Machine Arithmetic*; Nauka: Alma-Ata, Kazakhstan, 1976. (In Russian)
3. Omondi, A.R.; Premkumar, B. *Residue Number Systems: Theory and Implementation*; Imperial College Press: London, UK, 2007.
4. Szabo, N.S.; Tanaka, R.I. *Residue Arithmetic and Its Application to Computer Technology*; McGraw-Hill: New York, NY, USA, 1967.
5. Molahosseini, A.S.; de Sousa, L.S.; Chang, C.H. (Eds.) *Embedded Systems Design with Special Arithmetic and Number Systems*; Springer: Cham, Switzerland, 2017.
6. Soderstrand, M.A.; Jenkins, W.K.; Jullien, G.A.; Taylor, F.J. (Eds.) *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*; IEEE Press: New York, NY, USA, 1986.
7. Chernyavsky, A.F.; Danilevich, V.V.; Kolyada, A.A.; Selyaninov, M.Yu. *High-Speed Methods, and Systems of Digital Information Processing*; Belarusian State University: Minsk, Belarus, 1996. (In Russian)

8. Ananda Mohan, P.V. *Residue Number Systems. Theory and Applications*; Springer: Cham, Switzerland, 2016.
9. Ding, C.; Pei, D.; Salomaa, A. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*; World Scientific: Singapore, 1996.
10. Omondi, A.R. *Cryptography Arithmetic: Algorithms and Hardware Architectures*; Springer: Cham, Switzerland, 2020.
11. Chren, W.A., Jr. A new residue number division algorithm. *Comput. Math. Appl.* **1990**, *19*, 13–29. [CrossRef]
12. Chiang, J.-S.; Lu, M. A general division algorithm for the Residue Number System. In Proceedings of the 10th IEEE Symposium on Computer Arithmetic, Grenoble, France, 26–28 June 1991; IEEE Computer Society Press: Washington, DC, USA, 1991; pp. 76–83.
13. Lu, M.; Chiang, J.-S. A novel division algorithm for Residue Number Systems. *IEEE Trans. Comput.* **1992**, *41*, 1026–1032. [CrossRef]
14. Hitz, M.A.; Kaltofen, E. Integer division in residue number systems. *IEEE Trans. Comput.* **1995**, *44*, 983–989. [CrossRef]
15. Hiasat, A.A.; Abdel-Aty-Zohdy, H.S. Design and implementation of an RNS division algorithm. In Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, CA, USA, 6–9 July 1997; IEEE Computer Society Press: Washington, DC, USA, 1997; pp. 240–249.
16. Bajard, J.-C.; Didier, L.-S.; Muller, J.-M. A new Euclidean division algorithm for residue number systems. *J. VLSI Signal Process. Syst. Signal Image Video Technol.* **1998**, *19*, 167–178. [CrossRef]
17. Yang, Y.H.; Chang, C.C.; Chen, C.Y. A high-speed division algorithm in residue number system using parity-checking technique. *Int. J. Comput. Math.* **2004**, *81*, 775–780. [CrossRef]
18. Chang, C.-C.; Lai, Y.-P. A division algorithm for residue numbers. *Appl. Math. Comput.* **2006**, *172*, 368–378. [CrossRef]
19. Chang, C.-C.; Yang, J.-H. A division algorithm using bisection method in residue number system. *Int. J.Comput. Consum. Control (IJ3C)* **2013**, *2*, 59–66.
20. Hung, C.Y.; Parhami B. An approximate sign detection method for residue numbers and its application to RNS division. *Comput. Math. Appl.* **1994**, *27*, 23–35. [CrossRef]
21. Burton, D.M. *Elementary Number Theory*, 7th ed.; McGraw-Hill: New York, NY, USA, 2011.
22. Hardy, G.H.; Wright, E.M. *An Introduction to the Theory of Numbers*, 6th ed.; Oxford University Press: London, UK, 2008.
23. Knuth, D.E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed.; Addison-Wesley: Boston, MA, USA, 1998.
24. Shoup, V. *A Computational Introduction to Number Theory and Algebra*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2005.
25. Shenoy, A.P.; Kumaresan, R. Fast base extension using a redundant modulus in RNS. *IEEE Trans. Comput.* **1989**, *38*, 292–297. [CrossRef]
26. Phatak, D.S.; Houston, S.D. New distributed algorithms for fast sign detection in residue number systems (RNS). *J. Parallel Distrib. Comput.* **2016**, *97*, 78–95. [CrossRef]
27. Vu, T.V. Efficient implementations of the Chinese Remainder Theorem for sign detection and residue decoding. *IEEE Trans. Comput.* **1985**, *34*, 646–651.
28. Kawamura, S.; Koike, M.; Sano, F.; Shimbo, A. Cox-rower architecture for fast parallel Montgomery multiplication. In Proceedings of the EUROCRYPT'00: 19th International Conference on Theory and Application of Cryptographic Techniques, Bruges, Belgium, 14–18 May 2000; Springer: Berlin, Germany, 2000; pp. 523–538.
29. Nozaki, H.; Motoyama, M.; Shimbo, A.; Kawamura, S. Implementation of RSA algorithm based on RNS Montgomery multiplication. In Proceedings of the CHES 2001: Cryptographic Hardware and Embedded Systems, Third International Workshop, Paris, France, 6–14 May 2001; Springer: Berlin, Germany, 2001; pp. 364–376.
30. Isupov, K.; Knyazkov, V. Interval estimation of relative values in Residue Number System. *J. Circuits, Syst. Comput.* **2018**, *27*, 1850004. [CrossRef]
31. Chervyakov, N.; Babenko, M.; Tchernykh, A.; Kucherov, N.; Miranda-López, V.; Cortés-Mendoza, J.M. AR-RRNS: Configurable reliable distributed data storage systems for Internet of Things to ensure security. *Future Gener. Comput. Syst.* **2019**, *92*, 1080–1092. [CrossRef]
32. Halevi, S.; Polyakov, Y.; Shoup, V. An improved RNS variant of the BFV homomorphic encryption scheme. In *Topics in Cryptology–CT-RSA 2019, Proceedings of the Cryptographers' Track at the RSA Conference, San Francisco, CA, USA, 4–8 March 2019*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; Volume 11405, pp. 83–105.
33. Selianinau, M. An efficient implementation of the Chinese Remainder Theorem in minimally redundant Residue Number System. *Comput. Sci.* **2020**, *21*, 237–252. [CrossRef]
34. Selianinau, M. Computationally efficient approach to implementation of the Chinese Remainder Theorem algorithm in minimally redundant Residue Number System. *Theory Comput. Syst.* **2021**, *65*, 1117–1140. [CrossRef]
35. Jullien, G. Residue number scaling and other operations using ROM arrays. *IEEE Trans. Comput.* **1978**, *27*, 325–336. [CrossRef]
36. Shenoy, M.A.P.; Kumaresan, R. A fast and accurate RNS scaling technique for high speed signal processing. *IEEE Trans. Acoust. Speech Signal Process.* **1989**, *37*, 929–937. [CrossRef]
37. Barsi, F.; Pinotti, M. Fast base extension and precise scaling in RNS for look-up table implementation. *IEEE Trans. Signal Process.* **1995**, *43*, 2427–2430. [CrossRef]
38. Garsia, A.; Lloris, A. A look up scheme for scaling in the RNS. *IEEE Trans. Comput.* **1999**, *48*, 748–751. [CrossRef]
39. Griffin, M.; Sousa, M.; Taylor, F. Efficient scaling in the residue number system. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Glasgow, UK, 23–26 May 1989; IEEE Computer Society Press: Washington, DC, USA, 1989; pp. 1075–1078.

40. Vasilevich, L.N.; Kolyada, A.A. Scaling in residue number systems. *Cybern. Syst. Anal.*, **1989**, *25*, 610–615. [CrossRef]
41. Chernyavsky, A.F.; Kolyada, A.A.; Revinsky, V.V.; Selyaninov, M.Y.; Shabinskaja, E.V. Scaling methods in minimally redundant modular arithmetic. *Proc. Natl. Acad. Sci. Belarus Phys. Math. Ser.* **1998**, *4*, 132–137.
42. Meyer-Base, U.; Stouraitis, T. New power-of-2 RNS scaling scheme for cell-based IC design. *IEEE Trans. VLSI Syst.* **2003**, *11*, 280–283. [CrossRef]
43. Cardarilli, G.C.; Del Re, A.; Nannarelli, A.; Re, M. Programmable power-of-two RNS scaler and its application to a QRNS polyphase filter. In Proceedings of the IEEE International Symposium on Circuits and Systems, Kobe, Japan, 23–26 May 2005; IEEE Computer Society Press: Washington, DC, USA, 2005; pp. 1002–1005.
44. Isupov, K.; Knyazkov, V.; Kuvaev, A. Fast power-of-two RNS scaling algorithm for large dynamic ranges. In Proceedings of the 2017 IVth International Conference on Engineering and Telecommunication (EnT), Moscow, Russia, 29–30 November 2017; IEEE Computer Society Press: Washington, DC, USA, 2017; pp. 135–139.
45. Clemens, K.J. A modified definition of symmetric RNS improving scaling and overflow detection. *IEEE Trans. Circuits Syst.* **1985**, *32*, 412–413. [CrossRef]
46. Sousa, L. $2^n$ RNS scalers for extended 4-moduli sets. *IEEE Trans. Comput.* **2015**, *64*, 3322–3334. [CrossRef]
47. Mustapha K.S.; Bankas E.K. RNS scaling algorithm for a new moduli set $\{2^{2n+1}+1, 2^{2n+1}, 2^{2n+1}-1\}$. *Int. J. Comput. Appl.* **2017**, *165*, 21–28.
48. Hiasat, A. Efficient RNS scalers for the extended three-moduli set $\{2^n-1, 2^{n+p}, 2^n+1\}$. *IEEE Trans Comput.* **2017**, *66*, 1253–1260. [CrossRef]
49. Hiasat, A. New residue number system scaler for the three-moduli set $\{2^{n+1}-1, 2^n, 2^n-1\}$. *Computers* **2018**, *3*, 46. [CrossRef]
50. Hiasat, A. A scaler design for the RNS three-moduli set $\{2^{n+1}-1, 2^n, 2^n-1\}$ based on mixed-radix conversion. *J. Circuits. Syst. Comput.* **2020**, *29*, 2050041. [CrossRef]
51. Taheri, M.R.; Navi, K.; Molahosseini, A.S. Efficient programmable power-of-two scaler for the three-moduli set $\{2^{n+p}, 2^n-1, 2^{n+1}-1\}$. *ETRI J.* **2020**, *42*, 596–607. [CrossRef]