mathematics

MDPI

Article

Construction of Akushsky Core Functions Without Critical Cores

Vladislav Lutsenko [1], Mikhail Babenko [1,2,*] and Maxim Deryabin [3,*]

[1] North Caucasus Center for Mathematical Research, North-Caucasus Federal University, 355017 Stavropol, Russia; vvlutcenko@ncfu.ru
[2] Control/Management and Applied Mathematics, Ivannikov Institute for System Programming, 109004 Moscow, Russia
[3] Samsung Advanced Institute of Technology, Suwon 16678, Republic of Korea
* Correspondence: mgbabenko@ncfu.ru (M.B.); maxim.deryabin@gmail.com (M.D.)

Abstract: The residue number system is widely used in cryptography, digital signal processing, image processing systems, and other areas where high-performance computing is required. One of the main tools used in the residue number system is the Akushsky core function. However, its use is limited due to the existence of so-called critical cores. This study aims to develop Akushsky core functions that effectively eliminate the occurrence of critical cores, thereby enhancing their applicability in real-world scenarios. We introduce a fundamental approach to critical core detection that reduces the average time for critical core detection by 99.48% compared to the brute force algorithm. The results of our analysis indicate not only a substantial improvement in the speed of core detection but also an enhancement in the overall performance of systems utilizing the Akushsky core function. Our findings provide important insights into optimizing residue number systems and encourage further exploration into advanced computational techniques within this domain.

Keywords: residue number system; Akushsky core function; critical cores; monotonicity of the core function; non-modular operations; brute force algorithm

MSC: 11A07; 11Y16; 68Q25

check for updates

Citation: Lutsenko, V.; Babenko, M.; Deryabin, M. Construction of Akushsky Core Functions Without Critical Cores. *Mathematics* **2024**, *12*, 3399. https://doi.org/10.3390/math12213399

Academic Editor: Ding Wang

Received: 29 August 2024
Revised: 6 October 2024
Accepted: 28 October 2024
Published: 30 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The development of the algorithmic base of computing systems requires the development and implementation of innovative approaches to the organization and execution of computational tasks. Along with modern technologies such as quantum computing [1] and AI accelerators [2], parallel processing remains one of the key strategies for improving performance. This area focuses on modular computational structures based on the Residue Number System (RNS). In contrast to conventional techniques such as fixed point, RNS offers distinctive benefits due to its non-positional nature and parallel properties derived from number theory and abstract algebra. The primary benefit of RNS is the elimination of carryovers during arithmetic operations, which has the potential to significantly accelerate computational processes.

RNS is known to have several advantages over the conventional Weighted Number System (WNS) in the design and realization of high-performance computing applications, devices, and systems [3]. Since its introduction in the mid-1950s to the present day, RNS arithmetic has attracted continuous attention from researchers in the field.

(1) computer technology [3],
(2) number-theoretic methods [4,5],
(3) blockchain [6],
(4) homomorphic encryption [7],
(5) digital signal and image processing [8,9],
(6) communication systems [10],

(7) highly reliable cloud environments [11],
(8) neural networks [12,13].

The main advantage of RNS is the ability to decompose numbers of large length into a set of residues of smaller length, which are processed in parallel in independent modular channels. The parallelism inherent in RNS avoids carryovers obtained by addition, subtraction, and multiplication, which usually take a long time in WNS. In this regard, modularity and the absence of carryovers make computation fast and efficient. It is important to note that with the active development of quantum computing and AI accelerators, RNS remains a relevant and efficient tool for tasks requiring high computational speed and parallelism.

In addition to modular operations without carry-over, RNS arithmetic also uses non-modular operations such as inverse number conversion, basis expansion, number comparison, number sign detection, overflow detection, common division, scaling, and others.

The execution of these operations is complex and requires a large number of computations, which limits the application of RNS and its use for fast computations. Modern researchers actively study the problem of efficient implementation of non-modular operations.

The search for positional characteristics that would reduce the computational complexity of non-modular operations in RNS led researchers Akushsky I.Y., Burtsev V.M., and Pak I.T. to construct a new design called the Akushsky core function [14].

Akushsky core function can be effectively used to determine the sign of a number [15], number comparison [16], scaling [17], general division [18], and other non-modular operations.

However, when using the Akushsky core function, the so-called problem of critical cores arises [14], which significantly limits the applicability of each particular core function since it requires the specification of its values. In this paper, we propose methods for determining critical cores, which will allow more efficient realization of non-modular operations in RNS.

The key novelty of this work is the development of methods for determining critical cores, which opens new possibilities for efficient implementation of non-modular operations in RNS. In addition to the fundamental significance of the proposed methods, they open new perspectives for the application of RNS in practical areas. These methods may find their application in highly loaded computing systems, such as artificial intelligence systems and homomorphic encryption, where high speed and accuracy of arithmetic operations are required. Progress in this area may also contribute to the development of cloud computing, blockchain technologies, data protection systems, and telecommunications, providing more efficient and reliable performance of computational tasks.

The paper has the following structure: Section 2 discusses the RNS. Section 3 presents the basics of the Akushsky core function. Section 4 presents methods for determining critical cores. Finally, Section 5 analyzes the effectiveness of the proposed methods. The conclusion summarizes the results obtained.

2. Basic Concepts of Modular Arithmetic

The basis of the RNS is the set of moduli $\{p_1, p_2, \ldots, p_n\}$ where each module is $p_i \geq 2$ $(i = 1, 2, \ldots, n)$ and $gcd(p_i, p_j) = 1$. By default, the moduli are ordered in ascending order, that is, $p_1 < p_2 < \ldots < p_n$.

The product of the moduli $P = \prod_{i=1}^{n} p_i$ is called the dynamic range of the RNS.

An integer $X \in [0, P)$ is represented as an n-dimensional vector composed of the smallest non-negative residue from dividing the corresponding number by p_i:

$$X = (x_1, x_2, \ldots, x_n),$$

where $x_i = X mod\ p_i$, which is also denoted as $x_i = |X|_{p_i}$.

Consider a RNS with the basis $\{5, 6\}$. In this basis, it is possible to mutually unambiguously represent numbers from the half-interval $[0, 30)$ since $P = 5 \cdot 6 = 30$.

Table 1 shows the correspondences between numbers from the WNS and the RNS.

Table 1. Representation of numbers for RNS with basis $\{5, 6\}$.

$0 \overset{RNS}{\to} (0, 0)$	$1 \overset{RNS}{\to} (1, 1)$	$2 \overset{RNS}{\to} (2, 2)$	$3 \overset{RNS}{\to} (3, 3)$	$4 \overset{RNS}{\to} (4, 4)$
$5 \overset{RNS}{\to} (0, 5)$	$6 \overset{RNS}{\to} (1, 0)$	$7 \overset{RNS}{\to} (2, 1)$	$8 \overset{RNS}{\to} (3, 2)$	$9 \overset{RNS}{\to} (4, 3)$
$10 \overset{RNS}{\to} (0, 4)$	$11 \overset{RNS}{\to} (1, 5)$	$12 \overset{RNS}{\to} (2, 0)$	$13 \overset{RNS}{\to} (3, 1)$	$14 \overset{RNS}{\to} (4, 2)$
$15 \overset{RNS}{\to} (0, 3)$	$16 \overset{RNS}{\to} (1, 4)$	$17 \overset{RNS}{\to} (2, 5)$	$18 \overset{RNS}{\to} (3, 0)$	$19 \overset{RNS}{\to} (4, 1)$
$20 \overset{RNS}{\to} (0, 2)$	$21 \overset{RNS}{\to} (1, 3)$	$22 \overset{RNS}{\to} (2, 4)$	$23 \overset{RNS}{\to} (3, 5)$	$24 \overset{RNS}{\to} (4, 0)$
$25 \overset{RNS}{\to} (0, 1)$	$26 \overset{RNS}{\to} (1, 2)$	$27 \overset{RNS}{\to} (2, 3)$	$28 \overset{RNS}{\to} (3, 4)$	$29 \overset{RNS}{\to} (4, 5)$

The advantage of modular arithmetic over conventional binary arithmetic is that addition, subtraction, and multiplication operations are performed in parallel at the residue level. This allows us to efficiently perform modular operations on integers $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ и $B = (\beta_1, \beta_2, \ldots, \beta_n)$ independently in each modular channel, respecting the computational rules:

$$A \circ B = (\alpha_1, \alpha_1, \ldots, \alpha_n) \circ (\beta_1, \beta_2, \ldots, \beta_n) = \left(|\alpha_1 \circ \beta_1|_{p_1}, |\alpha_2 \circ \beta_2|_{p_2}, \ldots, |\alpha_n \circ \beta_n|_{p_n} \right). \quad (1)$$

where $\alpha_i = |A|_{p_i}$ и $\beta_i = |B|_{p_i}, i = 1, 2, \ldots, n$ and $\circ \in \{+, -, \times\}$.

In other words, arithmetic operations on operands in the form of vectors are divided into modular channels, each of which works with operands not exceeding the corresponding modulo. In addition, modular channels are completely independent of each other.

Let us look at an example of addition in the RNS.

Example 1. (Addition in the RNS). *Add two numbers $A = 7$ and $B = 8$ in the basis $\{5, 6\}$. Their representation in the given basis is given in Table 1. Let us use Equation (1) for addition:*

$$A + B = (|2 + 3|_5, |1 + 2|_6) = (0, 3).$$

Carry-free execution of modular operations is one of the key advantages of arithmetic in RNS [19]. This means that the results of addition and multiplication operations remain within a given range of values for each modulo. This greatly simplifies and speeds up the process of performing addition and multiplication operations compared to WNS.

The advantage of residue arithmetic is especially clear when performing computational procedures consisting of a sequence of modular arithmetic operations. In this case, a set of moduli is chosen in such a way that the final results are always within an acceptable range of values for any input operands. Intermediate results may fall outside the range, but this is not a problem.

To convert a number from RNS to WNS, the traditional approach involves the use of the Chinese Residue Theorem (CRT) [9].

$$X = \left| \sum_{i=1}^{n} P_i \cdot x_i \cdot \left| \frac{1}{P_i} \right|_{p_i} \right|_P = \sum_{i=1}^{n} P_i \cdot x_i \cdot \left| \frac{1}{P_i} \right|_{p_i} - r_X \cdot P. \quad (2)$$

where P_i coefficients are defined as $\frac{P}{p_i}$. Multiplicative inversion of P_i modulo p_i is denoted as $\left| \frac{1}{P_i} \right|_{p_i}$ satisfying the requirement $\left| \left| \frac{1}{P_i} \right|_{p_i} \cdot P_i \right|_{p_i} = 1$ and can be computed using methods similar to the Euclidean algorithm [20], and the r_X—is the rank of the number reflecting how many times the range value must be subtracted from the resulting sum to restore it within the dynamic range.

Example 2. (Inverse number conversion using CRT). *In RNS with basis* $\{5,\ 6\}$ *reconstruct the number* $(0,\ 3)$ *resulting from Example 1.*

First, let us find P_i*:*

$$P_1 = \frac{P}{p_1} = 6;\ P_2 = \frac{P}{p_2} = 5.$$

Next, we find the values of $\left|\frac{1}{P_i}\right|_{p_i}$*:*

$$\left|\frac{1}{P_1}\right|_{p_1} = 1;\ \left|\frac{1}{P_2}\right|_{p_2} = 5.$$

Then, using (2), we obtain

$$X = |6{\cdot}0{\cdot}1 + 5{\cdot}3{\cdot}5|_{30} = |75|_{30} = 15.$$

The result corresponds to Table 1, so the number was recovered correctly. Calculating the rank of a number will be discussed in the next section.

3. Akushsky Core Function

3.1. Application

Akushsky core function is a method for determining the positional characteristic of a number in RNS. The other most common methods for determining the positional characteristic are CRT, Approximate Method (AM) [9,11], Mixed-Radix Conversion (MRC) [10], Diagonal Function (DF) [21], and Pirlo Function (PF) [22].

However, Akushsky core function has some advantages due to the ability to select weights for a particular problem and the ability to determine the positional characteristic without inverse conversion. In [16], it is shown that the Akushsky core function for even ranges improves number sign detection by a factor of 1.93–15.3 and number comparison by a factor of 1.55–11.35 overall methods described previously. In [23], a number comparison method in RNS using core function is modeled; it was shown that the proposed method gains 31% in time and 37% in device area on average over the known AM. The paper [24] presents an efficient division method based on AM and the Akushsky core function.

In [11], Akushsky's core function is applied to build reliable distributed storage systems for the Internet of Things. In [25], the core function is used for digital image filtering. Thus, the Akushsky core function can be used in all RNS applications.

3.2. Mathematical Foundations

Consider RNS using a set of moduli $\{p_1,\ p_2, \ldots,\ p_n\}$ with dynamic range $P = \prod_{i=1}^{n} p_i$ and given residuals $(x_1,\ x_2, \ldots,\ x_n)$ for a number X such that $0 \le X \le (P-1)$.

The Akushsky core function is defined by the following equation:

$$C(X) = C_X = \sum_{i=1}^{n} w_i \cdot \left\lfloor \frac{X}{p_i} \right\rfloor. \tag{3}$$

or

$$C(X) = X \cdot \sum_{i=1}^{n} \frac{w_i}{p_i} - \sum_{i=1}^{n} \frac{x_i \cdot w_i}{p_i}. \tag{4}$$

where integers w_i are constants determined by the choice of the interpolation point. The constants w_i specify the weight of each of the partial $\left\lfloor \frac{X}{p_i} \right\rfloor$ in (3), thus defining the core function and giving it different properties.

The main property of the core function is that its maximum range is variable and can be much smaller than the number P depending on the choice of weights. For example, we can use an arbitrary value of $C(P)$ as $C(P)$, provided that it has the necessary properties

to solve our particular problem. This value is called the range of the core function and is defined by the expression.

$$C(P) = \sum_{i=1}^{n} w_i \cdot P_i. \tag{5}$$

Given that $P_i \equiv 0(mod\ p_j)$ for any $i \neq j$, the constants w_i of this function can be determined from the relation

$$w_i \equiv \left| \left| \frac{1}{P_i} \right|_{p_i} \cdot C(P) \right|_{p_i}. \tag{6}$$

Note that (6) specifies some class of deductions for each i, and the numbers w_i may turn out to be negative or positive in each case.

Let us introduce the value $B_i = P_i \cdot \left| \frac{1}{P_i} \right|_{p_i}$—the so-called RNS orthogonal bases. Core functions for orthogonal RNS bases

$$C(B_i) = \frac{B_i \cdot C(P)}{P} - \frac{w_i}{p_i}. \tag{7}$$

Form (4) is not suitable for calculating core function values in practical applications, as we will usually only have access to the residuals of x_i. Using CRT, we can change the way we calculate the core function value of X based on its residuals.

The value of the core function $C(X)$ given by the weights $w_1, w_2, \ldots, w_n$, provided $0 \leq C(X) < C(P),\ X \in [0, P)$, can be calculated using the equation

$$C(X) = \left| \sum_{i=1}^{n} C(B_i) \cdot x_i \right|_{C(P)}. \tag{8}$$

However, in the general case, if certain w_i, $C(X)$ may not satisfy the condition $0 \leq C(X) < C(P)$ for $X \in [0, P)$, then the problem of so-called critical cores arises, which prevents the use of (8).

Using the rank of the number r_X from (2), then (8) can be rewritten as

$$C(X) = \sum_{i=0}^{n} x_i \cdot C(B_i) - r_X \cdot C(P). \tag{9}$$

According to CRT, the rank of a number can be calculated as follows:

$$r_X = \left\lfloor \frac{\sum_{i=1}^{n} x_i \cdot B_i}{P} \right\rfloor. \tag{10}$$

The number X can be calculated using the Akushsky core function as

$$X = \frac{P}{C(P)} \cdot \left(C(X) + \sum_{i=1}^{n} \frac{x_i \cdot w_i}{p_i} \right). \tag{11}$$

Another advantage of using the Akushsky core function is that the constants $C(B_i)$ involved in the calculations are small since they are smaller than $C(P)$. To simplify or avoid the cumbersome division by $C(P)$ required in (11), we can choose as $C(P)$ a value equal to a power of two. The following examples show the inverse conversion procedure using the Akushsky core function.

Example 3. (Inverse number conversion using core function). *Consider the set of moduli* $\{5, 6\}$. *Let us choose weights* $w_1 = -3$ *and* $w_2 = 5$ *for the core function. Then* $P = 30$, $P_1 = 6,\ P_2 = 5$. *The values* $\left| \frac{1}{P_i} \right|_{p_i}$ *are equal to* $\left| \frac{1}{P_1} \right|_{p_1} = 1,\ \left| \frac{1}{P_2} \right|_{p_2} = 5$.

We will find $B_i = P_i \cdot \left| \frac{1}{P_i} \right|_{p_i}$:

$$B_1 = P_1 \cdot \left| \frac{1}{P_1} \right|_{p_1} = 6;\ B_1 = P_2 \cdot \left| \frac{1}{P_2} \right|_{p_2} = 25,$$

Next we find $C(P)$

$$C(P) = -3 \cdot 6 + 5 \cdot 5 = 7,$$

Let us find $C(B_i)$ *using (7)*

$$C(B_1) = 2;\ C(B_1) = 5.$$

Consider a number in the RNS equal to $(0,\ 3)$. *Using (8)*

$$C(X) = |0 \cdot 2 + 3 \cdot 5|_7 = 1,$$

Next, X can be derived from (11) as

$$X = \frac{30}{7}\left(1 + \frac{0 \cdot (-3)}{5} + \frac{3 \cdot 5}{6}\right) = 15.$$

Example 4. (Inverse number conversion using core function being critical). *Let the system be given by the same set of moduli; respectively, all parameters of the system remain the same. Let us carry out the procedure* of *number conversion* $(0,\ 5)$.

$$C(X) = |0 \cdot 2 + 5 \cdot 5|_7 = 4,$$

$$X = \frac{30}{7}\left(4 + \frac{0 \cdot (-3)}{5} + \frac{5 \cdot 5}{6}\right) = 35.$$

However, this is not true since the number $(0,\ 5)$ *corresponds to a five in WNS. Thus, we can conclude that a critical core has arisen for this number. It is necessary to use Formula (9) to clarify the value of the core.*

Find r_X *using (11)*

$$r_X = \left\lfloor \frac{0 \cdot 6 + 5 \cdot 25}{30} \right\rfloor = 4.$$

Then

$$C(X) = (0 \cdot 2 + 5 \cdot 5) - 4 \cdot 7 = -3,$$

and

$$X = \frac{30}{7}\left(-3 + \frac{0 \cdot (-3)}{5} + \frac{5 \cdot 5}{6}\right) = 5.$$

Thus, the core function is a universal positional characteristic. However, its practical use is complicated due to the need to solve two important problems. The first problem is related to the accurate calculation of the values of the core function. Since by definition the weights w_i of the core function can be negative, the values of this function can potentially lie outside the range $[0,\ C(P))$. This problem significantly limits the applicability of each particular core function, as it requires the specification of its values. On the other hand, number comparison algorithms based on the core function require solving the problem of choosing monotone core functions.

4. Methods for Constructing Akushsky Core Functions Without Critical Cores

4.1. The Problem of Core Function Monotonicity

The monotone Akushsky core function can be used to implement the number comparison operation. Let us consider the monotonicity conditions for the Akushsky function. For this purpose, we prove the following theorem.

Theorem 1. *The Akushsky function is monotonically increasing if and only if all its coefficients w_i are non-negative.*

Proof of Theorem 1. Calculate the value of $C(X-1)$

$$C(X-1) = (X-1)\sum_{i=1}^{n}\frac{w_i}{p_i} - \sum_{i=1}^{n}\frac{|x_i-1|\cdot w_i}{p_i} = X\cdot\sum_{i=1}^{n}\frac{w_i}{p_i} - \sum_{i=1}^{n}\frac{w_i}{p_i} - \sum_{i=1}^{n}\frac{|x_i-1|\cdot w_i}{p_i}. \quad (12)$$

Using (12), we calculate the value of $C(X) - C(X-1)$, we obtain

$$C(X) - C(X-1) = \sum_{i=1}^{n}\frac{w_i}{p_i} + \sum_{i=1}^{n}\frac{|x_i-1|\cdot w_i}{p_i} - \sum_{i=1}^{n}\frac{x_i\cdot w_i}{p_i} = = \sum_{i=1}^{n}\left(1 + |x_i-1|_{p_i} - x_i\right)\cdot\frac{w_i}{p_i}. \quad (13)$$

Let us calculate the value of the expression $1 + |x_i-1|_{p_i} - x_i$ depending on x_i and p_i. We obtain:

$$1 + |x_i-1|_{p_i} - x_i = \begin{cases} p_i, \ if \ x_i = 0, \\ 0, \ if \ x_i \neq 0. \end{cases} \quad (14)$$

Substituting (14) into (12), we obtain

$$C(X) - C(X-1) = \sum_{i=1, x_i=0}^{n} w_i. \quad (15)$$

To prove this theorem, we will apply the method of contradiction: suppose there exists a monotonically increasing Akushsky function with negative weight coefficients w_i. Let the number of negative coefficients be $s > 0$, and let them be indexed by $\{i_1, i_2, \ldots, i_s\}$. Consider the number $X = \prod_{j=1}^{s} p_{i_j}$. Using (12), we calculate the value of $C(X) - C(X-1)$ and obtain:

$$C(X) - C(X-1) = \sum_{j=1}^{s} w_{i_j}. \quad (16)$$

Since in Equation (16) the right-hand side $\sum_{j=1}^{s} w_{i_j} < 0$, it follows that $C(X) < C(X-1)$. This implies that the Akushsky function cannot be strictly monotonic. We have arrived at a contradiction. Therefore, for the Akushsky core function to be monotonic, it is necessary and sufficient that all of its coefficients are non-negative. □

Note that the Akushsky function cannot be strictly monotonic due to the fact that the change of its value for neighboring numbers occurs only in the presence of a zero residue in the larger number. Thus, the Akushsky function with non-negative coefficients serves as a positional characteristic of a number in the RNS.

Equation (16) defines a core function with a stepped structure. This structure consists of "levels" at which the value of the function remains unchanged. As X increases, all x_i increase, leading to a transition to the next level of the core function. This property is used in comparison algorithms that compare objects based on the value of the core function. Figure 1a shows an example of a core function with non-negative coefficients. Such a core function is monotonic and is divided into "levels". The number of levels can be estimated based on the magnitude of the remainder value for one of the moduli. In Figure 1b, the contour lines of the core function corresponding to these levels are shown. The Akushsky core function, for which $w_i > 0$, transitions to the next level each time $x_i = 0$ for any of the moduli p_n, where $i = 1, 2, \ldots, n$.

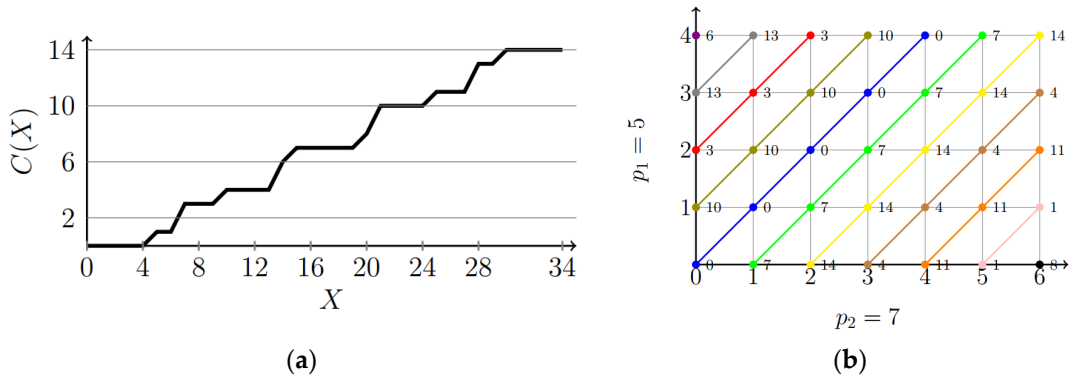


Figure 1. Plots (**a**) level lines (**b**) core function with $w_1 = 1$, $w_2 = 2$ for RNS with bases $p_1 = 5$, $p_2 = 7$; different colors represent different levels of the core function.

The core function can be used in comparison algorithms. For example, the diagonal function widely used in the literature [21] is a variant of the core function with $w_i = 1$ for all $i = 1, 2, \ldots, n$ and which contains no critical cores.

On the other hand, if we set $w_i = 0$ for all $i = 1, 2, \ldots, n-1$ and $w_n = 1$, we obtain a core function with the range $C(P) = P_n$. Thus, the expression $P_i(X) = \left\lfloor \frac{X}{p_n} \right\rfloor$ [22] is a special case of $C(X)$. This is the so-called Pirlo function, and it also does not contain critical cores. In fact, the result of such a function is the quotient of division by p_n. P_n is the smallest of P_i, $i = 1, 2, \ldots, n$. Given that the coefficients of a core function suitable for comparing numbers must be non-negative, according to Theorem 1, a core function with modulo $C(P) = P_n$ is a core function with the minimum possible modulo. Since $C_{P_n}(X)$ is the quotient of the division of X by p_n, the comparison of numbers with the help of this function is reduced to the comparison of their quotients. If the parts are equal, then the remainders modulo p_n are compared. If the parts of division of two numbers by modulo p_n are equal, then it is necessary to compare the remainders of division of these numbers by modulo p_n. It can be shown that in the case of using numbers smaller than P_n as the range $C(P)$, the comparison becomes impossible.

If we analyze the behavior of the core function at different values of numbers, we can notice that, in general, this function is not monotonic and cannot be used as a positional characteristic. This means that the value of the core function can change in an arbitrary way when the values of the numbers change. It is well seen from the graph in Figure 2, which reflects the values of the core function with coefficients $w_1 = -3$ and $w_2 = 5$ for the RNS with bases $p_1 = 5$ and $p_2 = 6$.

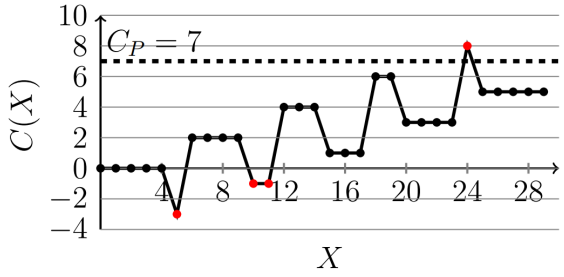


Figure 2. The core function with coefficients $w_1 = -3$ and $w_2 = 5$ for the RNS with bases $p_1 = 5$ and $p_2 = 6$ with critical cores demarked by red dots.

This graph shows that the core function has two inflection points, which means that it is not monotonic. In addition, the core function can take negative values, which also makes it unsuitable for comparing numbers in RNS because it cannot uniquely determine which number is greater or less than the other.

Figure 2 also demonstrates an important problem, the problem of critical cores, which is observed when using the Akushsky core function. In the above example, the critical cores will be at points X equal to 5, 10, 11, where the core function will be negative, and at point X equal to 24, where the value of the core function exceeds the value of $C(P)$.

In this case, the values of the core function will lie in some interval $[-\gamma_1, C(P) + \gamma_2]$. This means that the core function will not take values that lie outside this interval. The critical values make the equation $C(X) = |\sum_{i=1}^{n} C(B_i) \cdot x_i|_{C(P)}$ inapplicable, allowing us to efficiently compute the values of the core function.

We will call critical cores from the range $[-\gamma_1, 0)$ as lower critical cores and cores from the range $[C(P),\ C(P) + \gamma_2]$ as upper critical cores.

The problem of critical cores can be solved by choosing suitable values of the coefficients w_i. However, in the general case, this can be a challenging task.

There are approaches to monitor critical cores and correct core function values in them, but in practice this is inefficient. These approaches require complex computations and do not always lead to the desired result. A more pressing issue is the construction of core functions that do not contain critical cores. Such core functions will be simpler and more efficient to use.

Thus, constructing core functions that do not contain critical cores reduces to determining whether there are lower or upper critical cores.

4.2. Determining Critical Cores Using the Brute Force Algorithm

The lower and upper critical cores can be found using (4) or (9) and searching X in the interval $[0, P)$. However, for large values of P, this method will be impossible due to the huge time consumption. Therefore, we propose to search X in the interval $[0, p_n)$ to find the lower critical cores, i.e.,

$$lower\ critical\ cores = \begin{cases} 1,\ if\ \ C(X) < 0, \\ 0, if\ 0 \le C(X) < C(P). \end{cases} \tag{17}$$

And to find the upper critical cores, we will search X in the interval $[P - p_n, P)$. Then

$$upper\ critical\ cores = \begin{cases} 1,\ if\ \ C(X) \ge C(P), \\ 0, if\ 0 \le C(X) < C(P). \end{cases} \tag{18}$$

Algorithm 1 shows the pseudocode of the proposed approach.

Algorithm 1. Detection of critical cores using the brute force algorithm

Input: $\{p_1, p_2, \ldots, p_n\}$, $\{w_1, w_2, \ldots, w_n\}$, $P = \prod_{i=1}^{n} p_i$, $C(P)$ for $i = \overline{1, n}$.
Output: *lower_critical_cores*, *upper_critical_cores*
1. *lower_critical_cores*, *upper_critical_cores* $= 0$
2. **for** $X = 0$, $X < p_n$, $X++$ **do**: // For each number from 0 to p_n
2.1 $X \overset{RNS}{\rightarrow} (x_1, x_2, \ldots, x_n)$ // Converting the number to RNS
2.2 $C(X) = X \cdot \sum_{i=1}^{n} \frac{w_i}{p_i} - \sum_{i=1}^{n} \frac{x_i \cdot w_i}{p_i}$ // Calculate the core function
2.3 **if** $C(X) < 0$ **then**:
2.3.1 *lower_critical_cores* $= 1$ // The presence of lower critical cores
2.3.2 **break**
3. **for** $X = P - p_n$, $X < P$, $X++$ **do**:
3.1 $X \overset{RNS}{\rightarrow} (x_1, x_2, \ldots, x_n)$
3.2 $C(X) = X \cdot \sum_{i=1}^{n} \frac{w_i}{p_i} - \sum_{i=1}^{n} \frac{x_i \cdot w_i}{p_i}$
3.3 **if** $C(X) \ge C(P)$ **then**:
3.3.1 *upper_critical_cores* $= 1$ // The presence of upper critical cores
3.3.2 **break**
4. **return** *lower_critical_cores*, *upper_critical_cores*

Let us consider the application of this algorithm in Example 5.

Example 5. (Detection of critical cores using the brute-force algorithm). *Let also be given a system with bases* $p_1 = 5,\ p_2 = 6$. *And weights* $w_1 = -3,\ w_2 = 5$. *The parameters* P, $C(P)$ *are already known, and they are 30 and 7, respectively.*

By looping over X *in the interval* $[0, p_n)$, *we reach* $X = 5, 5 \overset{RNS}{\to} (0,\ 5)$ *for the given number.*

$$C(X) = 5 \cdot \left(\frac{-3}{5} + \frac{5}{6} \right) - \left(\frac{0 \cdot (-3)}{5} + \frac{5 \cdot 5}{6} \right) = -3$$

Since $C(X) < 0$, *the system with given weights contains lower critical cores.*

By looping over X *in the interval* $[P - p_n, P)$, *we take* $X = 24,\ 24 \overset{RNS}{\to} (4,\ 0)$ *for the given number.*

$$C(X) = 24 \cdot \left(\frac{-3}{5} + \frac{5}{6} \right) - \left(\frac{4 \cdot (-3)}{5} + \frac{0 \cdot 5}{6} \right) = 8.$$

Since $C(X) > C(P)$, *it means that at these weights the system has upper critical cores.*

4.3. Method for Determining Critical Cores Using the Fundamental Approach

Let us formulate conditions to determine whether the core function has critical cores based on the values of w_i, $i = 1,\ 2, \ldots, n$ for RNS with moduli $p_1, p_2, \ldots, p_n$ ordered in ascending order.

Theorem 2. *For the core function* $C(X)$ *defined by the coefficients* $w_1, w_2, \ldots, w_n$ *to contain no critical cores, the following conditions must be satisfied for all* $k = 1,\ 2, \ldots, n$:

1. $\sum_{i=1,}^{n} w_i > 0$.
2. *Absence of lower critical cores:* $C(p_k) = \sum_{i=1}^{n} w_i \left\lfloor \frac{p_k}{p_i} \right\rfloor \geq 0$.
3. *Absence of upper critical cores:* $\sum_{i=1}^{n} \left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) \cdot w_i - w_k > 0$.

Proof of Theorem 2. Let us start with the assumption that there are no lower critical cores. Based on (15), we can notice that the value of the core function changes only when zeros appear in the representation of a number in the RNS. Zeros in the RNS representation of the number X appear sequentially when its value increases. The first such case occurs when there is a zero in the remainder modulo p_1. This means that for there to be no lower critical core, it is necessary at least that $w_1 \geq 0$. If this condition is satisfied, the next change will only occur when zero in the residue modulo p_2. The possible change in core function in this case will be equal to $\left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) \cdot w_1 + w_2$. The first summand reflects the accumulated number of values of w_1 before the condition $|X|_{p_2} = 0$. For there to be no lower critical core, it is necessary that $\left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) \cdot w_1 + w_2 \geq 0$. Continuing the reasoning further, we obtain that for the core function to have no critical cores from below, it is sufficient that

$$\sum_{i=1}^{k} \left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) \cdot w_i + w_k \geq 0,$$

for $k = 1,\ 2,\ \ldots,\ n$. A similar condition is obtained by the requirement $C(p_k) \geq 0$, which is transformed to

$$C(p_k) = \sum_{i=1}^{n} w_i \left\lfloor \frac{p_k}{p_i} \right\rfloor = \sum_{i=1}^{k} w_i \left\lfloor \frac{p_k}{p_i} \right\rfloor \geq 0.$$

Let us consider the condition of absence of upper critical cores.

Taking into account the condition $\sum_{i=1}^{n} w_i \cdot P_i = C(P)$ we note that when passing from the value $X = P - 1$ to the value P, the difference between the cores $C(P)$ and $C(P-1)$ will be the sum $w_1 + w_2 + \cdots + w_n$. If $C(P-1) \geq C(P)$ (i.e., $C(P-1)$ is a critical core),

then this sum does not exceed 0. Hence, the minimum condition for no critical cores from above is

$$\sum_{i=1}^{n} w_i > 0.$$

Continuing similar reasoning for critical cores from above, we can see that for the absence of critical cores from above, we must check that the value of the core function does not exceed the range $C(P)$ for all numbers smaller than P containing zero residues from division by RNS moduli. Hence, we must require that $C(P - p_k) < C_P$ for all $k = 1, 2, \ldots, n$.

Let us calculate $C(P - p_k)$

$$C(P - p_k) = (P - p_k)\frac{C_P}{P} - \frac{1}{P}\sum_{i=1}^{n} w_i \cdot P_i |p_i - p_k|_{p_i}.$$

If $i = k$, then $|p_i - p_k|_{p_i} = 0$. Note that for any i and k, $i \neq k$, from the set of natural numbers $1, 2, \ldots, n$

$$|p_i - p_k|_{p_i} = \left\lfloor \frac{p_k}{p_i} \right\rfloor \cdot p_i + p_i - p_k.$$

Hence

$$C(P - p_k) = C_P - p_k \cdot \frac{C_P}{P} - \frac{1}{P} \cdot \sum_{i=1, i\neq k}^{n} w_i \cdot P_i \cdot \left(\left\lfloor \frac{p_k}{p_i} \right\rfloor \cdot p_i + p_i - p_k \right).$$

Further, since $p_k - \left\lfloor \frac{p_k}{p_k} \right\rfloor p_k = 0$, then

$$C\,(P - p_k) = C_P - p_k \cdot \frac{C_P}{P} - \sum_{i=1, i\neq k}^{n} w_i + \frac{1}{P} \cdot \sum_{i=1}^{n} w_i \cdot P_i \left(p_k - \left\lfloor \frac{p_k}{p_i} \right\rfloor \right).$$

Considering that

$$C(p_k) = p_k \cdot \frac{C_P}{P} - \frac{1}{P} \cdot \sum_{i=1}^{n} w_i \cdot P_i \left(p_k - \left\lfloor \frac{p_k}{p_i} \right\rfloor \right),$$

we obtain

$$C(P - p_k) = C_P - C(p_k) - \sum_{i=1, i\neq k}^{n} w_i.$$

We require that $C(P - p_k) < C_P$, then

$$C_P - C(p_k) - \sum_{i=1, i\neq k}^{n} w_i < C_P.$$

Then

$$C(p_k) + \sum_{i=1, i\neq k}^{n} w_i > 0,$$

or

$$\sum_{i=1}^{n} w_i \left\lfloor \frac{p_k}{p_i} \right\rfloor + \sum_{i=1, i\neq k}^{n} w_i = \sum_{i=1}^{n} w_i \cdot \left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) - w_k > 0.$$

The theorem is proved. □

Algorithm 2 presents a fundamental approach to determining critical cores in a pseudocode format.

Algorithm 2. Detection of critical cores using a fundamental approach

Input: $\{p_1, p_2, \ldots, p_n\}$, $\{w_1, w_2, \ldots, w_n\}$, for $i = \overline{1,n}$.
Output: *lower_critical_cores*, *upper_critical cores*
1. $lower_critical_cores$, $upper_critical_cores = 0$
2. **for** $k = 1$, $k \leq n$, $k++$ **do**:
2.1 $C(p_k) = \sum_{i=1}^{n} w_i \left\lfloor \frac{p_k}{p_i} \right\rfloor$ // Calculation of the core function from moduli
2.2 **if** $C(p_k) < 0$ **then**:
2.2.1 $lower_critical_cores = 1$ // The presence of lower critical cores
2.2.2 *break*
3. **for** $k = 1$, $k \leq n$, $k++$ **do**:
3.1 $sums = 0$
3.2 **for** $i = 1, i \leq n$, $i++$ **do**:
3.2.1 $sums+ = \left(\left\lfloor \frac{p_k}{p_i} \right\rfloor + 1 \right) \cdot w_i$
3.3 $sums = sums - w_k$
3.4 **if** $sums \leq 0$ **then**:
3.4.1 $upper_critical_cores = 1$ // The presence of upper critical cores
3.4.2 *break*
4. **return** $lower_critical_cores$, $upper_critical_cores$

Let us consider the application of this algorithm in Example 6.

Example 6. (Detection of critical cores using a fundamental approach). *We also continue to consider RNS with bases* $p_1 = 5$, $p_2 = 6$. *And weights* $w_1 = -3$, $w_2 = 5$.

First, let's determine if the system has lower critical cores.

$$C(p_1) = -3 \cdot \left\lfloor \frac{5}{5} \right\rfloor + 5 \cdot \left\lfloor \frac{5}{6} \right\rfloor = -3.$$

Since $C(p_1) < 0$, *hence the system with these weights will have lower critical cores.*
Next, let us determine whether there exist upper critical cores. For $k = 1$, *we obtain*

$$\left(\left\lfloor \frac{5}{5} \right\rfloor + 1 \right) \cdot (-3) + \left(\left\lfloor \frac{5}{6} \right\rfloor + 1 \right) \cdot 5 + 3 = 2 > 0.$$

Thus, the system with these weights has upper critical cores.

5. Performance Evaluation

We analyze the algorithms proposed in Sections 4.2 and 4.3. The complexity of the algorithms is evaluated in Table 2.

Table 2. Complexity analysis.

Method	Time Complexity
Brute Force	$O(n^2)$
Fundamental Approach	$O(n^2)$

The two algorithms have the same asymptotic complexity but differ in the amount of processed data. In the case of the first algorithm, RNS numbers are processed, the number of which can reach tens of thousands; in the fundamental approach, the moduli of the system are enumerate; in classical cases, the number of moduli varies from three to six numbers. To confirm the properties, we implement the proposed algorithms in Python and compare their performance. The experiments are conducted on the Windows 10 Home Edition operating system on a computer with an Intel Core i7-7700HQ 2.80 GHz processor, 8 GB of 1196 MHz DDR4 RAM, and a 512 GB SSD.

The experiment is as follows; the study is conducted in two stages:

The first stage of the study is to investigate the performance of 6 sets, 7 to 12 modules, and 16-bit dimensionality (Table 3).

Table 3. Moduli sets for the first stage of the study.

Number of Moduli	Modulo Set
7	{257, 263, 269, 271, 277, 281, 283}
8	{257, 263, 269, 271, 277, 281, 283, 293}
9	{257, 263, 269, 271, 277, 281, 283, 293, 307}
10	{257, 263, 269, 271, 277, 281, 283, 293, 307, 337}
11	{257, 263, 269, 271, 277, 281, 283, 293, 307, 337, 449}
12	{257, 263, 269, 271, 277, 281, 283, 293, 307, 337, 449, 887}

The second stage of the study is to investigate the performance of 6 sets, a commonly used set of special moduli $\{2^n - 1,\ 2^n,\ 2^n + 1\}$ [19], the dimensionality of the dynamic range from 24 to 64 bits (Table 4).

Table 4. Moduli sets for the second stage of the study.

Dynamic Range Size (bits)	Modulo Set
24	{255, 256, 257}
32	{2047, 2048, 2049}
40	{16,383, 16,384, 16,385}
48	{65,535, 65,536, 65,537}
56	{524,287, 524,288, 524,289}
64	{4,194,403, 4,194,304, 4,194,304}

Weights for the systems were generated randomly from the interval $[-5, 5]$.

To avoid time measurement errors, we take 10,000 measurements. Then, the maximum, minimum, and average values from 10,000 runs of each method are included in the results. In addition, we include the difference between the mean and minimum value. We consider the average values of the methods to determine the most efficient methods.

In the two-stage modeling, the temporal characteristics of each method were obtained. The results obtained are shown in Tables 5 and 6. The times are given in milliseconds.

Table 5. Operation time with dynamic number of moduli (ms).

Method	Number of Moduli	Determination of Lower and Upper Critical Cores			
		Max	Mean	Min	Mean − Min
Brute Force	7	50.9046	1.9157	0.0284	1.8873
	8	52.9824	4.2223	0.0343	4.188
	9	59.6002	5.4515	0.0385	5.413
	10	63.3836	5.7955	0.0417	5.7538
	11	83.3868	7.473	0.0448	7.4282
	12	111.4736	7.6149	0.0482	7.5667
Fundamental Approach	7	0.3187	0.0308	0.0077	0.0231
	8	0.2697	0.0323	0.008	0.0243
	9	0.4204	0.0385	0.0088	0.0297
	10	0.4115	0.0438	0.0096	0.0342
	11	0.4648	0.0537	0.0103	0.0434
	12	0.6458	0.0602	0.0111	0.0491

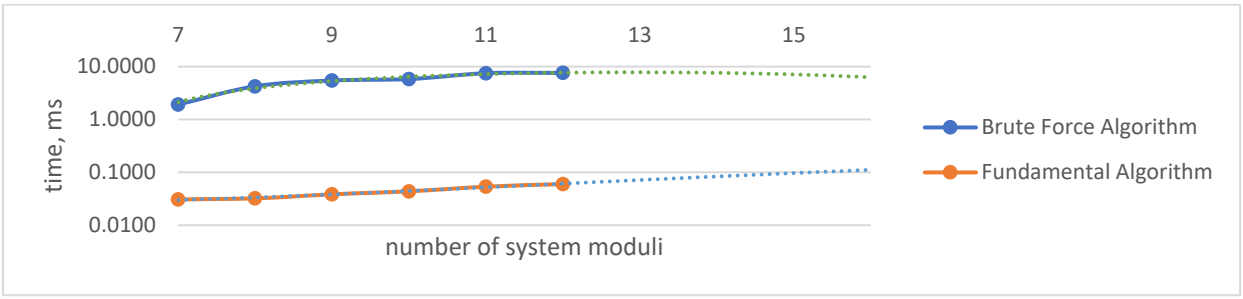
Table 6. Operation time with dynamic range of different lengths (ms).

Method	Dynamic Range Size (bits)	Determination of Lower and Upper Critical Cores			
		Max	Mean	Min	Mean − Min
Brute Force	24	13.0919	0.2885	0.0092	0.2793
	32	20.2188	1.5381	0.0157	1.5224
	40	21.8157	1.692	0.0147	1.6773
	48	22.8071	1.7088	0.0146	1.6942
	56	27.3549	2.3019	0.0147	2.2872
	64	33.3781	2.2711	0.0161	2.255
Fundamental Approach	24	0.0608	0.0078	0.004	0.0038
	32	0.0994	0.0084	0.0041	0.0043
	40	0.0965	0.0088	0.0041	0.0047
	48	0.1029	0.0096	0.0042	0.0054
	56	0.3976	0.0103	0.0044	0.0059
	64	0.0769	0.0121	0.0045	0.0076

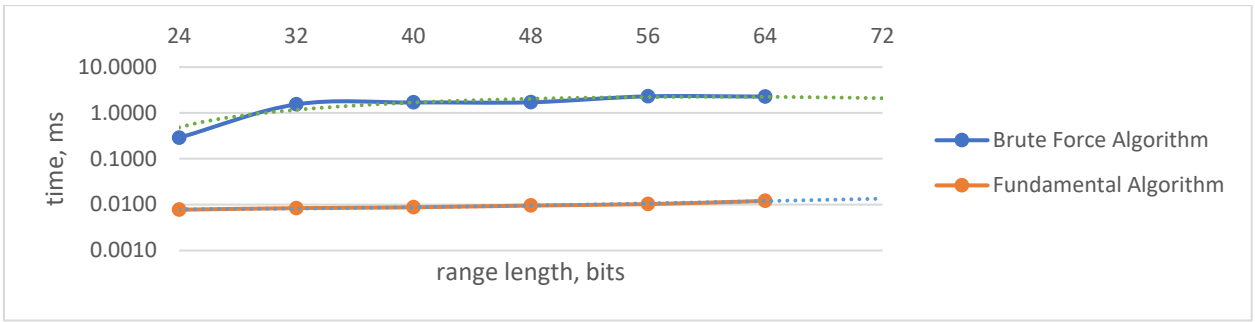
Table 5 shows that our proposed fundamental approach to determining critical cores is on average 99.2015% faster than the brute force algorithm. It is worth noting that the difference between the average maximum time of the fundamental algorithm and the brute force algorithm is 16,563.29%.

Table 6 shows that the fundamental approach is, on average, 99.4184% faster than the brute force algorithm.

For clarity, the graphs in Figure 3a,b showing the results of the average time obtained from the first and second stages of the study were plotted.



(**a**)



(**b**)

Figure 3. Comparison of the brute-force algorithm and the fundamental algorithm for determining critical cores with (**a**) dynamic moduli number p and (**b**) dynamic range width.

Reduced computational cost leads to energy savings and reduced power consumption, which makes our proposed fundamental approach promising for computing systems. In addition, the efficiency of the algorithm can lead to reduced hardware requirements, which further enhances its suitability for resource-constrained environments.

6. Conclusions

The residue number system is an efficient numerical system that is widely used in multiple applications. However, due to the non-positional structure, some operations, including comparison and inversion, are difficult over numbers represented in RNS. This work is aimed at addressing the problem of constructing a general positional characteristic useful for complex operations in RNS. Among all existing functions that are suitable to be a positional characteristic of the number represented in RNS, the Akushsky core function stands out since it has flexibility and allows for reduce the size of operands involved in the computations. The presented work solves the important problem of critical cores, which limits the Akushsky core function's possible application.

This work demonstrates the approach to constructing Akushsky core functions for a residue number system, removing the limitations associated with critical core. By introducing a fundamental approach to identify critical cores, the proposed methodology shows a 99.48% reduction in average time compared to the brute-force algorithm. These results open new possibilities for applying Akushsky core functions to residue number systems and hence to all areas where high-performance computing is required.

In view of these results, further research will be aimed at optimizing the selection of weights for the Akushsky core function and at modifying existing methods for non-modular operations in RNS using the core function.

Author Contributions: Conceptualization, methodology, software, validation, research, manuscript—writing, V.L.; methodology, research, manuscript—writing, supervision, M.D.; conceptualization, methodology, research, manuscript—writing, supervision, M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Ministry of Science and Higher Education of the Russian Federation (Grant No. 075-15-2024-529).

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

References

1. Horowitz, M.; Grumbling, E. *Quantum Computing: Progress and Prospects*; The National Academies Press: Washington, DC, USA, 2019.
2. Reuther, A.; Michaleas, P.; Jones, M.; Gadepally, V.; Samsi, S.; Kepner, J. AI Accelerator Survey and Trends. In Proceedings of the 2021 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 20–24 September 2021; pp. 1–9.
3. Mohan, P.V.A. *Residue Number Systems*; Springer International Publishing: Cham, Switzerland, 2016; ISBN 978-3-319-41383-9.
4. Amerbayev, V.M. *Theoretical Foundations of Machine Arithmetic*; Nauka: Alma-Ata, Kazakhstan, 1976. (In Russian)
5. Krasnobayev, V.; Kuznetsov, A.; Yanko, A.; Kuznetsova, T. The Analysis of the Methods of Data Diagnostic in a Residue Number System. *Comput. Model. Intell. Syst.* **2020**, *2608*, 594–609. [CrossRef]
6. Guo, Z.; Gao, Z.; Mei, H.; Zhao, M.; Yang, J. Design and Optimization for Storage Mechanism of the Public Blockchain Based on Redundant Residual Number System. *IEEE Access* **2019**, *7*, 98546–98554. [CrossRef]
7. Su, Y.; Yang, B.-L.; Yang, C.; Zhao, S.-Y. ReMCA: A Reconfigurable Multi-Core Architecture for Full RNS Variant of BFV Homomorphic Evaluation. *IEEE Trans. Circuits Syst. Regul. Pap.* **2022**, *69*, 2857–2870. [CrossRef]
8. Cardarilli, G.C.; Nunzio, L.D.; Fazzolari, R.; Nannarelli, A.; Petricca, M.; Re, M. Design Space Exploration Based Methodology for Residue Number System Digital Filters Implementation. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 186–198. [CrossRef]
9. Chervyakov, N.I.; Molahosseini, A.S.; Lyakhov, P.A.; Babenko, M.G.; Deryabin, M.A. Residue-to-Binary Conversion for General Moduli Sets Based on Approximate Chinese Remainder Theorem. *Int. J. Comput. Math.* **2017**, *94*, 1833–1849. [CrossRef]
10. Omondi, A.R.; Premkumar, A.B. *Residue Number Systems: Theory and Implementation*; World Scientific: Singapore, 2007; ISBN 978-1-908979-11-7.
11. 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]
12. Valueva, M.V.; Nagornov, N.N.; Lyakhov, P.A.; Valuev, G.V.; Chervyakov, N.I. Application of the Residue Number System to Reduce Hardware Costs of the Convolutional Neural Network Implementation. *Math. Comput. Simul.* **2020**, *177*, 232–243. [CrossRef]

13. Roohi, A.; Taheri, M.; Angizi, S.; Fan, D. RNSiM: Efficient Deep Neural Network Accelerator Using Residue Number Systems. In Proceedings of the 2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD), Munich, Germany, 1–4 November 2021; pp. 1–9.
14. Akushsky, I.Y.; Akushsky, V.M.; Pak, I.T. About the New Positional Characteristic of the Non-Positional Code and Its Ap-plication. In *Theory of Coding and Optimisation of Complex Systems*; Nauka: Alma-Ata, Kazakhstan, 1977; pp. 8–16.
15. Shiriaev, E.; Kucherov, N.; Babenko, M.; Lutsenko, V.; Al-Galda, S. Algorithm for Determining the Optimal Weights for the Akushsky Core Function with an Approximate Rank. *Appl. Sci.* **2023**, *13*, 10495. [CrossRef]
16. Tchernykh, A.; Babenko, M.; Shiriaev, E.; Pulido-Gaytan, B.; Cortés-Mendoza, J.M.; Avetisyan, A.; Drozdov, A.Y.; Kuchukov, V. An Efficient Method for Comparing Numbers and Determining the Sign of a Number in RNS for Even Ranges. *Computation* **2022**, *10*, 17. [CrossRef]
17. Burgess, N. Scaling an RNS Number Using the Core Function. In Proceedings of the 16th IEEE Symposium on Computer Arithmetic, Santiago de Compostela, Spain, 15–18 June 2003; pp. 262–269.
18. Lutsenko, V.V.; Babenko, M.G.; Tchernykh, A.N.; Lapina, M.A. Optimization of a Number Division Algorithm in the Residue Number System Based on the Akushsky Core Function. *Proc. Inst. Syst. Program. RAS* **2023**, *35*, 157–168. [CrossRef] [PubMed]
19. Szabo, N.S.; Tanaka, R.I. *Residue Arithmetic and Its Applications to Computer Technology*; McGraw-Hill: New York, NY, USA, 1967.
20. Brown, W.S.; Traub, J.F. On Euclid's Algorithm and the Theory of Subresultants. *J. ACM* **1971**, *18*, 505–514. [CrossRef]
21. Dimauro, G.; Impedovo, S.; Pirlo, G. A New Technique for Fast Number Comparison in the Residue Number System. *IEEE Trans. Comput.* **1993**, *42*, 608–612. [CrossRef]
22. Pirlo, G.; Impedovo, D. A New Class of Monotone Functions of the Residue Number System. *Int. J. Math. Models Methods Appl. Sci.* **2013**, *7*, 802–809.
23. Babenko, M.; Tchernykh, A.; Chervyakov, N.; Kuchukov, V.; Miranda-López, V.; Rivera-Rodriguez, R.; Du, Z.; Talbi, E.-G. Positional Characteristics for Efficient Number Comparison over the Homomorphic Encryption. *Program. Comput. Softw.* **2019**, *45*, 532–543. [CrossRef]
24. Babenko, M.; Tchernykh, A.; Kuchukov, V. Improved Modular Division Implementation with the Akushsky Core Function. *Computation* **2022**, *10*, 9. [CrossRef]
25. Chervyakov, N.I.; Lyakhov, P.A.; Babenko, M.G. Digital Filtering of Images in a Residue Number System Using Finite-Field Wavelets. *Autom. Control Comput. Sci.* **2014**, *48*, 180–189. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.