Towards Optimizing Cloud Computing Using Residue Number System

N Kucherov\(^1\), E Kuchukova\(^1\), A Tchernykh\(^2,3,4\), V Kuchukov\(^1\) and M Babenko\(^1\)

\(^1\)North-Caucasus Federal University, Stavropol, Russia
\(^2\)CICESE Research Center, Ensenada, BC, México
\(^3\)South Ural State University, Chelyabinsk, Russia
\(^4\)Ivannikov Institute for System Programming, Moscow, Russia

E-mail nkucherov@ncfu.ru

Abstract. In the work methods for building systems of distributed data storage based on the system of residual classes are considered. The use of direct conversion of data from the positional system of calculation to the residue number system will have a large computational complexity, the use of modules of a particular type allows you to solve this problem. The operation of scaling and expansion of the base system, which is necessary to restore the number of stored parts in case of failure of one or more cloud servers, is considered.

1. Introduction

All the computational structure is closely related to the computational system in which it works. The number system is understood as a set of methods of notation of numbers, or more precisely, a way of coding of elements of some final model of real numbers by words of one or more alphabets.

Any representation of numbers in the working range is only a component of the corresponding machine arithmetic and cannot be considered separately from it. The arithmetic properties of this or that number system are first of all determined by the character of inter-bit links appearing in the course of operations over code words. Researches have shown that within the limits of the usual position system of notation (PSS), it is impossible to achieve considerable acceleration of performance of operations. It is explained by the fact that in the PSS the value of any digit of a number, except for the junior digit, which is the result of a double arithmetic operation, depends not only on the value of the operands of the same name but also on all junior digits, i.e., the PSS has a strictly consistent structure. Today, when improving the technology of production of computational means and introducing new, more effective ways of organizing and carrying out calculations, preference is given to computational structures possessing abilities for parallel processing of information. Non-position codes have these features with the parallel structure, which allows you to realize the idea of paralleling operations at the level of performing elementary arithmetic operations. This idea originated in the middle of the 50-s of the last century when Czech scientists in [1] their research in the field of non-positional number systems consider the representation of numbers as a set of residues from the division into selected natural modules - the bases of the system. Such a number system became known as the residual class system (SKS) or the modular number system (MSS). Following the Czech scientists, the possibility of using this system in the computer has been considered in the studies of American scientists Aiken, Simon, and Garner.

2. Application of residual class system for data storage in the cloud environment
When storing data in the cloud environment, there are situations in data access denial, errors, or deterioration of service performance, and sometimes even long interruptions in their work. For this and other reasons, a continuous stream of failures, errors, and malfunctions inevitably occurs in the distributed data processing. To increase reliability and reliability of data processing and storage, it is reasonable to apply distributed data storage schemes, based on the principles of modular arithmetic. The main tool to increase reliability indicators is the introduction of flexible redundancy into the system.

The data distribution schemes proposed in [2, 3] provide data availability and distribution. The redundant system of residual classes (RNS) represents initial numbers as residues about the set of modules. Thus, the number is broken down into smaller numbers, which are independent.

RNS is a non-positional number system that allows you to split long numbers into a series of independent digits of small length, which will enable you to speed up the calculations and organize their parallelism. The main advantage of RNS is the ability to perform addition and multiplication operations very quickly compared to all other number systems, which causes great interest in RNS in those areas in which large amounts of computation of these operations are required.

The RNS is defined by the method of mutual-size modules \( \beta = \{p_1, p_2, ..., p_n\} \). The positive number \( X \) in COK according to these modules is represented as a tuple of numbers \( X = (x_1, x_2, ..., x_n) \), where \( x_i = |X| p_i = X \mod p_i \) for \( i = 1, 2, ..., n \). This representation of number \( X \) is the only one if \( 0 \leq X < P \), where \( P = \prod_{i=1}^{n} p_i \) and is called the RNS range.

The operations of addition, subtraction and multiplication in the RNS for the numbers \( A = (a_1, a_2, ..., a_n) \) and \( B = (b_1, b_2, ..., b_n) \) are determined by the formulas:

\[
A \pm B = (|a_1 \pm b_1| p_1, ..., |a_n \pm b_n| p_n),
\]

\[
A \times B = (|a_1 \times b_1| p_1, ..., |a_n \times b_n| p_n).
\]

Equalities (1) and (2) show the parallel nature of RNS, free from bitwise transfers. Also, the numbers \( a_i \) and \( b_i \) have a much smaller number of digits than the original numbers \( A \) and \( B \).

In modern technology, one of the most popular properties of algorithms is their parallelism. This fact is due to the development of many parallel systems, from multiprocessor clusters to embedded systems for particular purposes.

The most common way to reconstruct the positional value of a number based on its residual representation is the Chinese remainder theorem (CRT), the classical form of which we will designate as CRT. Let the number \( X \) be given in the form \((x_1, x_2, ..., x_n)\) in the RNS by modules \((p_1, p_2, ..., p_n)\). Then:

\[
X = \sum_{i=1}^{n} \left[ \frac{P_i^{-1}}{p_i} \right] P_i x_i
\]

where \( P_i = P/p_i, \left[ P_i^{-1} \right]_{p_i} \) – the multiplicative inversion \( P_i \) modulo \( p_i \) for \( i = 1, ..., n \).

This method is computationally complex since it leads to calculations that fall outside the range of \( P \), and its implementation requires the operation of calculating the remainder of the division by a large number of \( P_i \), which significantly complicates the calculation scheme.

Thus, arithmetic operations in the modular code are performed independently for each of the modules, which indicates the parallelism of this system. This circumstance determines the possibility of independent processing, i.e., bitwise execution of operations. This property eliminates the need to “occupy” or “carry over” a high-order unit, which leads to the appearance of codes with a parallel structure. This allows you to parallelize the algorithms when performing arithmetic operations.

Studies of the new system have revealed several advantages. First, this is a maximum concurrency. To assess the level of parallelism of the number system, a special indicator is introduced:

\[
\prod(p) = \frac{n(p)}{k},
\]
where $k$ – system code length, $n(v)$ – number of bitwise concurrency indicators $\pi_1, \pi_2, \ldots, \pi_k$, not less than a given threshold $v \left( \frac{1}{k} \leq v \leq 1 \right)$, at that $\pi_i = 1 - \frac{n_i}{k} (i = 1, k)$, where $n_i$ – the maximum possible number of pairs of digits $(x_j, y_j)$ ($j = 1, 2, \ldots, i - 1, i + 1, \ldots, k$), that affect the value of the sum $Z = X + Y$ during its formation in the language of this code. For RNS, the parallelism index takes the maximum possible value $\prod(1) = 1$. This indicates the absence of inter-discharge bonds in the number recorded in the RNS.

Features of the RNS are the implementation of the principle of conveyor information processing, high accuracy, reliability, and the ability to self-correct. Of course, this system is not without drawbacks. These include the impossibility of visual comparison of numbers, the absence of signs that the results of operations go beyond the range, the limited action of the system by the sphere of integer positive numbers, the receipt in all cases of the exact result of the operation, and the difficulty of performing non-modular operations. But these shortcomings are not insurmountable. To date, algorithms for performing non-modular operations have been developed and continue to be developed, with priority being given to the development of algorithms that meet the requirement of parallel processing of information, a search is being made for the most convenient integral characteristics of the modular code that facilitate the implementation of non-modular operations.

3. Algorithms for transferring from a positional number system to a system of residual classes

3.1. Direct conversion using modular exposure

In [5, 6], the direct conversion method was described. They refer to this method as “modular exponentiation.” Basically, in this method, various residuals of degrees 2 (i.e. $2^x \mod m_i$) are obtained using logical functions. This will be illustrated first with an example. Consider the search $2^{5s_2s_1s_0}$ mod 13, where the exponent is a 4-bit binary word. We can write this expression as

$$2^{5s_2s_1s_0} \mod 13 = 2^{8s_3+4s_2s_1s_0} \mod 13 = 256^{s_3}16^{s_2}4^{s_1}2^{s_0} \mod 13 = (255s_3 + 1)(15s_2 + 1)4^{s_1}2^{s_0} \mod 13 =$$

$$= (3s_3s_2 + 8s_3 + 2s_2 + 1)4^{s_1}2^{s_0} \mod 13$$

Further, for various values $s_1, s_0$ we can calculate the term in brackets. For example, for $s_1 = 0, s_0 = 0$, $2^{5s_2s_1s_0} \mod 13 = (3s_3s_2 + 8s_3 + 2s_2 + 1) \mod 13$. Further, for the four variants of bits $s_3 \in s_2$, namely, 11, 10, 01, 00, the value $2^{5s_2s_1s_0} \mod 13$ can be estimated as 1, 9, 3, 1, respectively. Thus, the logical function $g_0$ can be used to represent $2^{5s_2s_1s_0} \mod 13$ для $s_1 = 0, s_0 = 0$, considering the bit values as

$$g_0 = 8s_3 \bar{s}_2 + 2s_3s_2 + 1$$

Similarly, other functions corresponding to $s_1 s_0$, t. e. 01, 10, 11, can be obtained as

$$g_1 = 4(s_2 \oplus s_3) + 2(\bar{s}_3 + s_2) + s_3 \bar{s}_2$$

$$g_2 = 8(s_2 \oplus s_3) + 4(\bar{s}_3 + s_2) + 2s_3 \bar{s}_2$$

$$g_3 = 8(\bar{s}_3 + s_2) + 4s_3 \bar{s}_2 + 2(s_3 \oplus s_2) + (s_3 \oplus s_2)$$

Note that the logic elements used to generate and combine the minimum terms in the $g_i$. Functions can be shared between modules. For example, $2^{11} \mod 13$ can be obtained from $g_3$ (так как $s_1 = s_0 = 1$), substituting $s_3 = 1, s_2 = 0$ как 7. The architecture consists of supplying the input power “i,” which requires $2^i \mod 13$. The two least significant bits $i$, namely, $x_1, x_0$ are used to select the nibble output using four 4:1 multiplexer of the remainder corresponding to the function $g_j$, depending on the values of $s_3 \in s_2$-бит. Thus, for each degree 2, the remainder will be selected using a set of multiplexers, and all these residues will need to be added mod 13 to the modulo adder tree to get the final remainder. Fully parallel architecture or consecutive parallel architectures can be used to achieve trade-offs between areas and time.

3.2. Direct converters for modules of type $(2^n \pm k)$
In [7, 8], the conversion of a binary code to an RNS for modules of type $2^n \pm k$ for a set of four modules \{2$^{-}n-1, 2^n+1, 2^n-3, 2^n+3$\} with a dynamic range of 4$n$ bits. This 4$n$-bit binary word can be considered as four $n$-bit fields $W_3, W_2, W_1$ and $W_0$, giving in the case of the module $2^n - k$,

$$W_{2n} - k = (W_3 k^3 + W_2 k^2 + W_1 k + W_0)_{2^n-k}$$

(8) since $2^n \mod (2^n - k) = k$.

Therefore, using three factors $(n \times p, n \times 2p$ and $n \times n$) to multiply $W_3, W_2$ and $W_1$ correspondingly на $k^3, k^2$ and $k$, respectively, the reduction can be carried out in stages. The summation in (8) gives a $(2n + 2)$-bit word, which is again considered as three $n$-bit fields and then reduces to an $(n + p + 1)$-bit word using two factors $(2 \times 2p, n \times p)$, followed by an adder, where, $p = \lceil \log_2 k \rceil$. Two more steps reduce the word length from $(n + p + 1)$ bits to $(n + 2)$ bits and $(n + 2)$ bits to $n$ bits using the factor $(p + 1) \times p$ and the factor $p \times p$, and modulo adder.

In the case of the module $(2^n + k)$ the calculation is performed as follows:

$$W_{2n} + k = (-W_3 k^3 + W_2 k^2 - W_1 k + W_0)_{2^n+k}$$

(9) since $2^n \mod (2^n + k) = -k$ and $2^n \mod (2^n + k) = -k^3$. Please note that (9) can be rewritten as follows:

$$W_{2n} + k = (W_3 k^3 + W_2 k^2 + W_1 k + W_0 + c)_{2^n+k}$$

(10) where $c = (k^3(k + 1) + 3k(k + 1)) \mod (2^n + k)$.

Due to the intermediate reduction steps, to reduce the $(2n+2)$-bit word to $(n + p + 1)$ bits and the next $(n + p + 1)$ bits to $n$ bits, the correction factor is $c = k^3(k + 1) + 3k(k + 1)$. The converter for the module $(2^n + k)$ needs three steps, while for the module $(2^n - k)$ four steps are required. Mautino and others [8] also offer a multiplier implementation by adding only shifted versions of the input data instead of using a hardware multiplication block.

In cryptography, as well as in RNS, it is often required to obtain a scalable remainder, that is, $(\frac{x}{2^n})_m$ [9]. This can be achieved by sequentially dividing by $2 \mod m$. For example, $(\frac{13}{2^{19}})_19$ can be obtained by first calculating $(\frac{13}{2})_{19} = (\frac{13+19}{2})_{19} = 16$. Further, $(\frac{13}{2^{7}})_{19} = (\frac{16}{2})_{19} = 8$, and $(\frac{13}{2^{21}})_{19} = (\frac{8}{2})_{19} = 4$. The procedure of scaling $x$ by 2 involves adding the module $m$ in the case where the least significant bit of $x$ is 1 and dividing by two (ignoring the least significant bit or shifting one bit to the right). If the least significant bit of $x$ is equal to zero, it is enough to simply divide it into two (ignoring the least significant bit or shift to the right).

The Montgomery algorithm [10] allows one to estimate $(\frac{x}{2^n})_m$, by considering $\alpha$-bits at a time (also called a higher-base implementation). Here we want to find the multiple of $m$, that needs to be added to $x$, to make it exactly divisible by $2^n$. First, we need to calculate $\beta = -(m)_{2^n}$. Further, knowing the word $Z$, corresponding to $\alpha$ LSB from $x$, we need to calculate $Y = x + (Z\beta)_{2^n} \times m$, which will be exactly divisible by $2^n$. The division is carried out by shifting to the right by $\alpha$ bit.

Рассмотрим $x = (1010011101)_2 = 333$, мы хотим найти $(\frac{x}{2^{16}})_{23}$. Мы находим $\beta = (\frac{-1}{23})_{16} = 9$.

We know that $\alpha = 13$ (4-bit low-order bits $x$). Thus, we need to calculate $Y = 333 + (13 \times 9)_{16} \times 23 = 333 + 5 \times 23 = 448$, which is exactly divisible by 16, to get 28. Taking one bit at a time, we would need four steps (333 + 23)/2 = 178, 178/2 = 89, (89 + 23)/2 = 56, and 56/2 = 28. The procedure can be extended to find scaling by an arbitrary power of $2 \mod m$. The Montgomery method can be expanded to multiplication with the difference that every time a partial product is added, the least significant bits of the result are used in the calculation.

4. Algorithms for transferring from a system of residual classes to a positional number system

4.1. Convert CRT-based RNS to binary
The binary number $X$, corresponding to the given residuals $(x_1, x_2, x_3, \ldots, x_n)$ in the RNS \{\(m_1, m_2, m_3, \ldots, m_n\)\} can be obtained using CRT in the form

$$X = \left( x_1 \left( \frac{1}{M_1} \right)_{m_1} \right) M_1 + \left( x_2 \left( \frac{1}{M_2} \right)_{m_2} \right) M_2 + \ldots + \left( x_n \left( \frac{1}{M_n} \right)_{m_n} \right) M_n \mod M \quad (11)$$

where $M_i = M/m_i$ for $i = 1, 2, \ldots, n$ and $M = M_1 M_2 \ldots M_n$. Note that below we denote $p \mod q$ как $(p)_q$. The quantities $\frac{1}{M_j} \mod m_j$ are known as the inverse numbers $M_j$ mod $m_j$, defined in such a way that

$$\left( M_j \left( \frac{1}{M_j} \right) \right)_{m_j} = 1.$$  

The sum in (11) can be much larger than $M$, and the reduction of mod $M$ to obtain $X$ is a cumbersome process. The advantage of CRT is that the residuals $x_i$ can be weighted in parallel, and the results are summed up with the subsequent recovery of mod $M$.

Example. Using CRT, find the binary number corresponding to the remainders (1, 2, 3) in the set of modules (3, 5, 7).

Note that $M = 105$, $M_1 = 35$, $M_2 = 21$ и $M_3 = 15$. So we have $(1/M_1) \mod 3 = 2$, $(1/M_2) \mod 5 = 1$, $(1/M_3) \mod 7 = 1$. Thus, the result is $X = \left( 35 \times ((1 \times 2) \mod 3) + 21 \times ((2 \times 1) \mod 5) + 15 \times ((3 \times 1) \mod 7) \right) \mod 105 = (70 + 42 + 45) \mod 105 = 52$. It can be noted that in this example, a single subtraction of 105 is necessary to reduce the result mod 105.

CRT can be effectively used in the case of three and four sets of modules, for example $\{2^n - 1, 2^n, 2^n + 1\}$, $\{2^{2n} - 1, 2^n, 2^{2n} + 1\}$ и $\{2^{3n} - 1, 2^{2n}, 2^{3n} + 1\}$. The $n$ the bits of the decoded number $X$ are available directly in the form of the remainder corresponding to $2^n$. Module and the modulo reduction needed at the end for the product of the remaining modules can be effectively implemented in the case of the first three sets of modules. For some of these sets of modules, the modules have a wide length in the range from $n$ to $2n$ bits, which may be a disadvantage, since a larger module determines the cycle time of the RNS processor instructions. For regular sets of modules, CRT may require the use of specialized module reduction hardware and require ROM-based implementations to obtain $x'_i = \left( x_i \left( \frac{1}{M_i} \right)_{m_i} \right)$ values and factors to calculate $x'_i M_i$.

4.2. Convert mixed-radix system based RNS to binary conversion

The MRC method is sequential and includes modulo subtraction and modulo multiplication by multiplicative inversions of one module concerning other modules. In MRC, the decoded number is expressed as

$$B = x_1 + d_1 m_1 + d_2 m_2 m_1 + d_3 m_2 m_3 m_1 + \ldots + d_{j-1} m_1 m_2 m_3 \ldots m_{j-1} \quad (12)$$

where $0 \leq d_i \leq (m_i + 1) - 1$ for $j$ modules RNS. The parameters $d_i$ are known as numbers in a mixed positional system.

At each step, one digit $d_i$ in the mixed-radix system. In the end, the MRC digits are weighted by (12) to obtain the final decoded number. There is no need for an absolute modulo reduction.

Note that at each step, the remainder corresponding to one module is subtracted so that the result is precisely divided by this module. Multiplication with the multiplicative inverse completes this division. The last step requires the multiplication of large numbers, for example $z_3 m_2 m_3$ in the example with three modules, and the addition of the resulting products using adders with saving transfer with subsequent. But in the case of MRC, a final reduction in absolute value is not required, since the result is always less than $M = m_1 m_2 m_3$. In the case of a large number of RNS modules, for example $\{m_1, m_2, m_3, \ldots, m_n\}$, different multiplicative inversions must be known a priori and different products
of modules \(m_{(k-1)}m_k, m_{(k-2)}m_{(k-1)}m_k\) etc. Thus, the conversion time of the RNC to binary is 
\[(n-1)\Delta_{\text{modsub}} + (n-1)\Delta_{\text{modmul}} + \Delta_{\text{mult}} + \Delta_{\text{CSA}(n-2)} + \Delta_{\text{CPA}},\]
where modsub and modmul mean the operations of subtraction and multiplication modulo, CSA\((k-2)\) means \((k-2)\) evel of CSA, and mult – ordinary multiplication. Note that the MRC algorithm can be pipelined.

4.3. Convert mixed-radix system based RNS to binary conversion for a special type of modules
Scientists have studied three sets of modules \(\{m_1, m_2, m_3\} = \{2n, 1, 2n, 2n-1\}\). The inverse transformer for this set of modules based on the CRT described uses the expressions

\[ X = \left\{ \frac{m_1}{2} + \left(\frac{m_3m_2}{2}\right)x_1 + \left(\frac{m_1m_3}{2}\right)x_3 - m_1m_3x_2 \right\} \mod M \text{ for } (x_1 + x_3) \text{ not even} \]
\[ X = \left\{ \left(\frac{m_2m_3}{2}\right)x_1 + \left(\frac{m_1m_2}{2}\right)x_3 - m_1m_3x_2 \right\} \mod M \text{ for } (x_1 + x_3) \text{ even} \]

where \(M = 2n(4n^2 - 1)\).

5. Range extension
The expansion of the base system is one of the main non-modular operations in the RNS. This operation is necessary when performing a number of other operations, for example, when the operating division of numbers, when calculating positional characteristics, when detecting overflow of addition or multiplication of numbers. Also, when detecting and correcting errors, it is necessary to add one or more control bases.

The problem of expanding the base system can be formulated as follows: find the residual representation of the number on a new base (new bases), if the representation of the number on other bases is known, i.e., find the remainder of division by number, if the rest of division by other numbers is known.

One way to expand the base system is to translate the number into a positional number system and calculate the remainder of the division by a new module. It must be admitted that this path is not rational in terms of the number of operations.

Another method of expanding the base system allows you to determine the digit of a number on a new basis, based on such positional characteristics of the number as the rank of the number, the trace of the number. Let the base system \(p_1, p_2, ..., p_n\) with a range \(P\), orthogonal bases \(B_1, B_2, ..., B_n\), whose weights \(m_1, m_2, ..., m_n\) be given again. By definition \(B_i = m_i \frac{p_i}{p_i}, i = 1, n\). Let the number \(A = (\alpha_1, \alpha_2, ..., \alpha_n)\) be given in this system. We expand the base system by adding the base \(p_{n+1}\), then the range of the system becomes \(P' = p_{n+1} \cdot P\), orthogonal bases of the system \(B'_1, B'_2, ..., B'_{n+1}\), their weights \(m'_1, m'_2, ..., m'_{n+1}\) and \(B'_i = m_i \frac{p_{n+1}p_i}{p_i}, i = 1, n + 1\). The task is to determine the digit \(\alpha_{n+1}\) of the number \(A\) from the base \(p_{n+1}\).

\[ m'_{n+1}\alpha_{n+1} + r_A \equiv q(\mod p_{n+1}) \]

Formula (15) is the formula for expanding the range of numbers.

For the practical implementation of this formula, proceed as follows:

1. The parameters of the central and extended systems (orthogonal bases, their weights, minimal pseudo-orthogonal numbers with their ranks, and multiplicities) are calculated.

2. The number \(A = (\alpha_1, \alpha_2, ..., \alpha_n)\) is composed of the minimum pseudo-orthogonal numbers \(M\alpha_i, i = 1, p_j - 1, j = 1, n - 1\), with ranks \(r_{\alpha_i}\), which are uniquely are determined by the selected base system \(p_1, p_2, ..., p_n\). As a result, the number \(M = (\alpha_1, \alpha_2, ..., \alpha_n, S_A)\), is obtained, where \(S_A\) – is the trace of the number, and the sum rank theorem finds its rank:

\[ r_{MA} = \sum_{j=1}^{n_1} r_{\alpha_i} - m_n n_{MA}, \]

where \(n_{MA}\) – number of transitions modulo \(p_n\).
3. Extension $M_A$ by extension formula (15). Using the rank value $r_{M_A}$, calculated by the formula (16), we obtain the number $A' = (a_1, a_2, ..., a_n, S_A, a_{n+1})$, which differs from the desired number $A$ by the numbers two last reasons.

4. If $S_A = a_n$, then $A' = A$, i.e., $A' –$ required increase in number $A$.

5. If $S_A 
eq a_n$, then add to number $A'$ such of the minimum pseudo-orthogonal numbers as $\bar{M}_{\beta n n} = \{0, ..., 0, \beta_n, S_{\beta n}\}$ the multiplicity $k_{\beta n n}$, where $\beta_n \equiv (a_1 - S_A)(mod p_n)$, turns the base $p_n$ number into $a_n$. The result is the number $A^{(2)} = A' + \bar{M}_{\beta n n} = (a_1, a_2, ..., a_n, a_{n+1})$.

6. If the diversity $k_{\beta n n} \leq p_n - (n - 1)$, then the number $A^{(2)}$ is the desired extension of the number $A$, since the number $A'$, not exceeding $(n - 1) \frac{p}{p_n}$ was added $\bar{M}_{\beta n n}$, not exceeding $(p_n - (n - 1)) \frac{p}{p_n}$, i.e. $A^{(2)}$ does not exceed $P$, i.e., values of the 1st interval.

7. If $k_{\beta n n} > p_n - (n - 1)$, then the number $A^{(2)}$ can be located either in the last $(n - 1) \frac{p}{p_n}$ parts of the 1st interval $[0, P)$, either in the lower $(n - 2) \frac{p}{p_n}$ parts of 2nd interval $[P, 2P)$, and then the desired number is: $A^{(3)} = (a_1, ..., a_n, (a_{n+1}^2 - 1)mod (p_n + 1))$.

When analyzing the considered method of expanding base systems, it can be noted that it relies on positional characteristics such as rank and number trail, which require the calculation of a number of parameters (orthogonal bases, their weights, minimal pseudo-orthogonal numbers, their ranks with multiplicities), the calculation of which can seem cumbersome. However, all these parameters are determined only by the foundations of the system, and therefore can be calculated in advance. Moreover, using the indicated positional characteristics, other non-modular procedures can be organized, which allows one to construct machine arithmetic based on these characteristics.

6. Conclusion

The work considered methods for constructing distributed data storage systems based on a system of residual classes. The use of direct data conversion from a positional number system to a system of residual classes will have great computational complexity; the use of modules of a special kind allows solving this problem. The operation of scaling and expanding the base system, which is necessary to restore the number of stored parts in case of failure of one or more cloud servers, is considered. To restore data, you must use methods that use modules of a special kind.

Acknowledgments

This work was supported by a grant from the President of the Russian Federation MK-24.2020.9, MK-341.2019.9, SP-2236.2018.5, and RFBR according to the research project № 20-37-70023

References

[1] Szvoboda A 1962 Computer progress in Czechoslovakia Digitale Informationswandler/Digital Information Processors/Dispositifs traitant des informations numeriques (Springer) pp 543–574
[2] Asmuth C and Bloom J 1983 IEEE transactions on information theory 29 208–210
[3] Mignotte M 1982 How to share a secret Workshop on Cryptography (Springer) pp 371 {375
[4] Tchernykh A, Schwiegelsohn U, Talbi E g and Babenko M 2019 Journal of Computational Science 36 100581
[5] Koc C K, Acar T and Kaliski B S 1996 IEEE micro 16 26–33
[6] McLoone M, McIvor C and McCanny J V 2004 Coarsely integrated operand scanning (cios) architecture for high-speed Montgomery modular multiplication Proceedings. 2004 IEEE International Conference Field-Programmable Technology (IEEE Cat. No. 04EX921) (IEEE) pp 185–191
[7] Harris D, Krishnamurthy R, Anders M, Mathew S and Hsu S 2005 An improved unified scalable radix-2 Montgomery multiplier 17th IEEE Symposium on Computer Arithmetic (ARITH’05) (IEEE) pp 172–178
[8] Kelley K and Harris D 2005 Very high radix scalable montgomery multipliers *Fifth International Workshop on System-on-Chip for Real-Time Applications (IWSOC’05)* (IEEE) pp 400–404
[9] Meehan S, O’ Neil S and Vaccaro J 1990 *IEEE Transactions on circuits and systems* 37 799–803
[10] Montgomery P L 1985 *Mathematics of computation* 44 519–521