Bull. Nov. Comp. Center, Comp. Science, 21 (2004), 69–84

The cellular pipelined algorithm architecture for 1D diffusion simulation using residue number system*

Valentina P. Markova

Abstract. This paper is oriented to the cellular pipelined algorithm architecture for the 1D diffusion simulation. The finite difference representation of 1D diffusion is written in terms of the residue number system. The cellular pipelined algorithm architectures for the main operations with a minimum period are developed. The parallel Substitution Algorithm is used for the algorithm design and modeling. The time complexity of the algorithm presented is obtained.

1. Introduction

In [1], an attempt has been undertaken to use the residue (modular) number system (RNS) for the numerical simulation. The idea of using the RNS is not new [2–4]. Many authors have demonstrated the potential of the modular arithmetic for realizing high-speed digital signal processing (digital filtering, convolution, correlation, and the DFT and the FFT computations). A high speed is attained due to parallel, carry-free arithmetic (addition, subtraction and multiplication) and shorter length of each remainder as distinct from an initial integer. In addition, the RNS arithmetic is exact (without overflow) and therefore free of a round-off error. This makes modular arithmetic an attractive platform for implementation of high-precision, and high-speed computations.

In [1], the computational characteristics (stability, accuracy, time complexity) of the residue number system have been investigated on an example of the 1D diffusion simulation. For this purpose, the finite difference representation of the diffusion equation, where time, space, and a certain physical value are discrete, has been written in terms of the RNS. To overcome the difficulties associated with the RNS division, in the finite difference representation, two strategies have been introduced: transfer of a remainder to the next step and representation of the diffusion number as a fraction.

Numerical simulation has been performed on Pentium III, MVS 1000/M. In our studies, a solution in the floating-point numbers is used for comparison. The experimental results have shown that the RNS computations are stable over a wide range of values for a diffusion number as opposed to the floating-point computations. The RNS provides an acceptable accuracy of

*Supported by Russian Academy of Science, Basic Research of No. 17 (2004).

solution. As expected, the time complexity of simulation of a diffusion process in the RNS on general-purpose computers is above the time complexity of the floating-point diffusion simulation. Indeed, general-purpose computers do not support the modular arithmetic. Some reduction in the time complexity (60%) has been obtained using the OpenMP and the MMX. A significant improvement of the time complexity can be gained due to the design of specialized computing devices.

In this paper, we present the cellular pipelined algorithm architecture for the 1D diffusion simulation. A good time complexity is attained due to the following.

- Parallel data processing in all the strips obtained by the domain decomposition.
- Parallel processing of arithmetic operations in all the moduli.
- Pipelining at both the initial data and the computation process levels.
- Using the table look-up operation.
- Loading the initial data, transformation of intermediate results and their moving in parallel.

The Parallel Substitution Algorithm (PSA) [5] is used for the design of the algorithm. The PSA is a model of the fine-grained parallelism, integrating the concepts of a cellular automaton and the Markov algorithm. Unlike other cellular models, the PSA properties and expressive capabilities allow one to represent any complex algorithm. Moreover, there is a one-to-one correspondence between the PSA and an automata net, thus forming the basis for the architectural design.

This paper is organized as follows. The second section describes the main operations in the Residue Number System. In the third section, the 3D cellular pipelined algorithm architectures for main operations (addition, subtraction, multiplication, and division) are described and their time complexities are evaluated. The cellular pipelined algorithm architecture for the 1D diffusion and its time complexity are given in the fourth section.

2. The RNS arithmetic

2.1. The RNS representation

Let $\mathcal{P} = \{p_0, p_1, \ldots, p_{k-1}\}$ be a set of the pairwise relatively prime integers (*the moduli set*). The interval $[0, M)$, $M = \Pi_{i=0}^{k-1} p_i$, determines *the dynamic range* of the system. Then any integer $X \in [0, M)$ has a unique *RNS representation* or the *RNS number* [2–4] given by

$$\langle X\rangle_{\mathcal{P}} \longrightarrow (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}) = \boldsymbol{X},$$

where $\boldsymbol{x}_j = \langle X\rangle_{p_j} = X \bmod p_j$ is *the j-th remainder* of X modulo p_j. The remainder $\boldsymbol{x}_j$ is calculated in the following way

$$\langle X\rangle_{p_j} = \boldsymbol{x}_j = X - \lfloor X/p_j \rfloor p_j,$$

where $\lfloor Y \rfloor$ denotes the largest integer smaller or equal to Y. If $X < 0$, then

$$\langle -X\rangle_{\mathcal{P}} = \langle M - X\rangle_{\mathcal{P}} = \langle \tilde{X}\rangle_{\mathcal{P}} \longrightarrow (\tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{k-1}) = \tilde{\boldsymbol{X}}.$$

As distinct from the weighted NS, the RNS representation has two properties:

- all remainders are independent,
- the length of each remainder is smaller than that of an initial integer.

These two properties of the RNS representation provide a parallel, high-speed, and carry-free arithmetic. In addition, the RNS arithmetic is exact and therefore free of the round-off error.

2.2. The RNS arithmetic

Let $\mathcal{P} = \{p_0, p_1, \ldots, p_{k-1}\}$ be a moduli set (here the moduli are located in increasing order), $M = \Pi_{j=0}^{k-1} p_j$. Let $\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$ and $\boldsymbol{Y} = (\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{k-1})$ be the two RNS representations of the integers X and Y, $X \in [0, M)$, $Y \in [0, M)$. Then the RNS representation of the integer $Z = X \circ Y$, $Z \in [0, M)$, is given by

$$\boldsymbol{X} \circ \boldsymbol{Y} = \boldsymbol{Z} = (\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{k-1}),$$

where $\circ$ denotes addition, subtraction, or multiplication,

$$\boldsymbol{z}_j = \langle \boldsymbol{x}_j \circ \boldsymbol{y}_j\rangle_{p_j} = \begin{cases} \boldsymbol{x}_j \circ \boldsymbol{y}_j & \text{if } 0 \le \boldsymbol{x}_j \circ \boldsymbol{y}_j < p_j, \\ \boldsymbol{x}_j \circ \boldsymbol{y}_j + p_j & \text{if } \boldsymbol{x}_j \circ \boldsymbol{y}_j < 0, \\ \boldsymbol{x}_j \circ \boldsymbol{y}_j - p_j & \text{if } \boldsymbol{x}_j \circ \boldsymbol{y}_j > p_j, \end{cases}$$

for all $j = 0, 1, \ldots, k-1$.

Example 1. Let $\mathcal{P} = \{3, 5, 7, 11\}$, $M = 1155$, $\boldsymbol{X} = (1, 2, 1, 0)$ $(X = 22)$, $\boldsymbol{Y} = (0, 0, 4, 5)$ $(Y = 60)$. Then we have $\boldsymbol{X} + \boldsymbol{Y} = (1, 2, 5, 5)$, $\boldsymbol{X} - \boldsymbol{Y} = (1, 2, -3, -5) = (1, 2, 4, 6)$, $\boldsymbol{X} \cdot \boldsymbol{Y} = (0, 0, 4, 0)$.

Division. Let $\mathcal{P} = (p_0, p_1, \ldots, p_{k-1})$ be a moduli set, $\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$ be a RNS dividend, $\boldsymbol{p}_i = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{i-1}, 0, \underbrace{p_i, \ldots, p_i}_{k-1-i})$ be the RNS divisor.

If the number X is divided by p_i, then $\boldsymbol{x}_i = 0$, otherwise, the number $X' = X - \boldsymbol{x}_i$ is used as dividend. The division is carried out in two steps.

At **the first step**, the algorithm generates the first approximation of the quotient

$$\hat{\boldsymbol{Z}} = (\hat{\boldsymbol{z}}_0, \hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_{i-1}, 0, \hat{\boldsymbol{z}}_{i+1}, \ldots, \hat{\boldsymbol{z}}_{k-1}),$$

where $\hat{\boldsymbol{z}}_j = \langle \frac{\boldsymbol{x}_j}{\boldsymbol{\pi}_j} \rangle_{p_j} = \langle \boldsymbol{x}_j \cdot \langle \frac{1}{\boldsymbol{\pi}_j} \rangle_{p_j} \rangle_{p_j}$, $j \neq i$, $\frac{1}{\boldsymbol{\pi}_j}$ is the multiplicative inverse of $\boldsymbol{\pi}_j$ modulo p_j. To avoid uncertainty, the digit $\hat{\boldsymbol{z}}_i$ is equated to zero. Here $\hat{\boldsymbol{z}}_j = \boldsymbol{z}_j$ for all j, $j \neq i$.

The obtained quotient $\hat{\boldsymbol{Z}}$ can belong to one of the intervals $l^m_{M_i}$, $M_i = M/p_i$, $m = 0, \ldots, p_i - 1$, that are derived from splitting the range $[0, M)$ to p_i parts. Each interval $l^m_{M_i}$ contains M_i numbers. The first number of the m-th interval takes the form

$$\boldsymbol{G}_m = (0, 0, \ldots, 0, \underbrace{m}_{i}, 0, \ldots, 0).$$

Here $\boldsymbol{g}_{mj} = 0$ for all $j \neq i$, because the number $\boldsymbol{G}_m$ is a multiple of each module p_j. As far as $Z \leq M/p_i$, the quotient $\boldsymbol{Z}$ always belongs to the interval $[0, M_i)$.

At **the second step**, the algorithm determines the digit value $\boldsymbol{z}_i$. The determination is based on the fact that both numbers $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$ are at the same distance from the beginnings of appropriate intervals, i.e.,

$$\boldsymbol{Z} - \boldsymbol{0} = \hat{\boldsymbol{Z}} - \boldsymbol{G}_m.$$

Hence, $\boldsymbol{z}_i = \langle 0 - m_i \rangle_{p_i}$.

The number $\boldsymbol{G}_m$ is determined according to $\hat{\boldsymbol{Z}}$ with the help of nullivization. The idea of nullivization consists in successive increase of the amount of zero remainders in the RNS representation of an integer (initial or intermediate). It begins with the least nonzero remainder except the i-th remainder (the modulo p_i is the divisor). The nullivization is carried out in $(k-1)$ steps. It can be expressed as

$$\hat{\boldsymbol{Z}} \to (0, \boldsymbol{g}^1_1, \ldots, \boldsymbol{g}^1_{k-1}) = \boldsymbol{G}^0 \to (0, 0, \boldsymbol{g}^2_2, \ldots, \boldsymbol{g}^2_{k-1}) = \boldsymbol{G}^1 \to \cdots \to \boldsymbol{G}_m.$$

Here the moduli p_0, p_1 are not divisors. The number $\boldsymbol{G}^j$ is defined as follows:

$$\boldsymbol{G}^j = \begin{cases} \boldsymbol{G}^{j-1} & \text{if } \boldsymbol{g}^{j-1}_j = 0 \text{ for all } j \neq i, \\ \boldsymbol{G}^{j-1} - \boldsymbol{M}^{p_j}_{\boldsymbol{g}^{j-1}_j} & \text{otherwise,} \end{cases}$$

where $\boldsymbol{G}^{j-1} = (0, \ldots, 0, \boldsymbol{g}_j^{j-1}, \ldots, \boldsymbol{g}_{k-1}^{j-1})$, $j < i$, is a result of the $(j-1)$th step of nullivization, a number $\boldsymbol{M}_{\boldsymbol{g}_j^{j-1}}^{p_j} = (0, \ldots, 0, \boldsymbol{g}_j^{j-1}, \ldots, \boldsymbol{g}_{k-1}^{j})$ is a nullivization constant. This constant is the number that sets up into zero the jth remainder of the number $\boldsymbol{G}^{j-1}$ and saves zero remainders obtained at the previous steps. For each p_j, the quantity of the nullivization constants equals $p_j - 1$.

Example 2. Let $\mathcal{P} = \{3, 5, 7, 11\}$, the number $\boldsymbol{X} = (0, 4, 2, 1)$ $(X = 309)$ be a dividend, and the modulus $\boldsymbol{p}_1 = (3, 0, 5, 5)$ be a divisor. Since the remainder $\boldsymbol{x}_1 = r_1 = 4$, then the number X is not divided by p_1. In such a case, the RNS number $\boldsymbol{X'} = \boldsymbol{X} - \boldsymbol{r}_1$ is used as a dividend. Here $\boldsymbol{r}_1 = \langle r_1 \rangle_{\mathcal{P}} = \langle 4 \rangle_{\mathcal{P}} = (1, 4, 4, 4)$. Then $\boldsymbol{X'} = (0, 4, 2, 1) - (1, 4, 4, 4) = (-1, 0, -2, -3) = (2, 0, 4, 8)$.

First, we calculate the remainders of the quotient $\hat{\boldsymbol{Z}} = (\hat{\boldsymbol{z}}_0, \hat{\boldsymbol{z}}_1, \hat{\boldsymbol{z}}_2, \hat{\boldsymbol{z}}_3) = \frac{(2,0,4,8)}{(2,0,5,5)}$, except $\hat{\boldsymbol{z}}_1$:

$$\hat{\boldsymbol{z}}_0 = \boldsymbol{z}_0 = \left\langle \frac{2}{2} \right\rangle_3 = \left\langle 2 \cdot \left\langle \frac{1}{2} \right\rangle_3 \right\rangle_3 = \langle 2 \cdot 2 \rangle_3 = 1,$$
$$\hat{\boldsymbol{z}}_2 = \boldsymbol{z}_2 = \left\langle \frac{4}{5} \right\rangle_7 = \left\langle 4 \cdot \left\langle \frac{1}{5} \right\rangle_7 \right\rangle_7 = \langle 4 \cdot 3 \rangle_7 = 5,$$
$$\hat{\boldsymbol{z}}_3 = \boldsymbol{z}_3 = \left\langle \frac{8}{5} \right\rangle_{11} = \left\langle 8 \cdot \left\langle \frac{1}{5} \right\rangle_{11} \right\rangle_{11} = \langle 8 \cdot 9 \rangle_{11} = 6.$$

As a result, $\hat{\boldsymbol{Z}} = (1, 0, 5, 6)$ $(\hat{Z}_{10} = 985)$. Further, we carry out the nullivization. For the set $\mathcal{P} = \{3, 5, 7, 11\}$, the sets of nullivization constants (M^3, M^7, and M^{11}) are tabulated in the table. In order that $\boldsymbol{g}_0^0 = 0$ be obtained, we choose the constant $\boldsymbol{M}_1^3 = (1, 1, 1, 1)$ from the set $\boldsymbol{M}^3$, since the digit $\hat{\boldsymbol{z}}_0$ equals 1. As a result, we have the integer $\boldsymbol{G}^0 = \hat{\boldsymbol{Z}} - \boldsymbol{M}_1^3 = (1, 0, 5, 6) - (1, 1, 1, 1) = (0, -1, 4, 5) = (0, 4, 4, 5)$. The nullivization constant $\boldsymbol{M}_4^7$ sets up the second remainder into "zero": $\boldsymbol{G}^1 = \boldsymbol{G}^0 - \boldsymbol{M}_4^7 =$

$\boldsymbol{M}^3$	$\boldsymbol{M}^7$	$\boldsymbol{M}^{11}$
$\boldsymbol{M}_1^3 = (1, 1, 1, 1)$	$\boldsymbol{M}_1^7 = (0, 0, 1, 4)$	$\boldsymbol{M}_1^{11} = (0, 0, 0, 1)$
$\boldsymbol{M}_2^3 = (2, 2, 2, 2)$	$\boldsymbol{M}_2^7 = (0, 4, 2, 9)$	$\boldsymbol{M}_2^{11} = (0, 4, 2, 0)$
	$\boldsymbol{M}_3^7 = (0, 3, 3, 3)$	$\boldsymbol{M}_3^{11} = (0, 3, 0, 3)$
	$\boldsymbol{M}_4^7 = (0, 3, 4, 7)$	$\boldsymbol{M}_4^{11} = (0, 2, 0, 4)$
	$\boldsymbol{M}_5^7 = (0, 2, 5, 1)$	$\boldsymbol{M}_5^{11} = (0, 1, 0, 5)$
	$\boldsymbol{M}_6^7 = (0, 1, 6, 6)$	$\boldsymbol{M}_6^{11} = (0, 0, 0, 6)$
		$\boldsymbol{M}_7^{11} = (0, 4, 0, 7)$
		$\boldsymbol{M}_8^{11} = (0, 3, 0, 8)$
		$\boldsymbol{M}_9^{11} = (0, 2, 0, 9)$
		$\boldsymbol{M}_{10}^{11} = (0, 1, 0, 10)$

$(0,4,4,5)-(0,3,4,7)=(0,1,0,-2)=(0,1,0,9)$. Finally, we obtain the last integer $\boldsymbol{G}^2=\boldsymbol{G}^1-\boldsymbol{M}_9^{11}=(0,1,0,9)-(0,2,0,9)=(0,4,0,0)$. The number $\boldsymbol{G}^2$ points out that the number $\hat{\boldsymbol{Z}}$ belongs to the fourth initial interval. Hence, the digit $\boldsymbol{z}_1=(0-4) \bmod 5=1$ and the quotient $\boldsymbol{Z}=(1,1,5,6)$ $(Z=65)$.

3. The cellular algorithm architectures for the main RNS operations

3.1. The cellular algorithm architecture for the RNS addition

Let $x_j,\ y_j,\ p_j$, $j=0,1,\ldots,k-1$, be $2'$s-complement representations of the initial remainders $\boldsymbol{x}_j$, $\boldsymbol{y}_j$ and modulo p_j, respectively.

A cellular algorithm for the RNS addition is carried out in two steps. First, the algorithm calculates the sum (x_j+y_j) for all p_j, $p_j\in\mathcal{P}$ in parallel by the $2'$s-complement addition. At the second, the remainder of the obtained binary sum with respect to p_j is formed for all p_j in parallel, i.e., the transformation $x_j+y_j\to\langle x_j+y_j\rangle_{p_j}$ is performed.

The straightforward modulo p transformation of a binary integer is reduced to dividing the integer by modulo p. However, the division of a large number can be slow and does not fit for high-speed calculations. To avoid a division, we use the generalization of the usual *casting out of nine* rule for the binary number system [6]. The idea consists in the following.

Let $p=2^l\pm1$ and $Z=b_{n-1}b_{n-2}\ldots b_0$ be an initial n-bit number. Split the number Z to any numbers of l bits, i.e., $Z=a_{l-1}a_{l-2}\ldots a_0$, where $0<a_i<2^l$ for all i. Then

$$\langle Z\rangle_p=\left\langle\sum_{i=0}^{l-1}a_i(-1)^i\right\rangle_p \quad \text{if} \quad p=2^l+1, \tag{1}$$

and

$$\langle Z\rangle_p=\left\langle\sum_{i=0}^{l-1}a_i\right\rangle_p \quad \text{if} \quad p=2^l-1. \tag{2}$$

Example 3. Let $p=17=2^4+1$, $Z=100110110001$ (1201). Since $l=4$, the number Z is partitioned into three numbers: 3-bit, 4-bit, and 4-bit each, starting with the most significant bit. As a result, we have $a_2=100$ (8), $a_1=1011$ (11), and $a_0=0001$ (1). Then, according to (2),

$$\langle Z\rangle_{17}=\langle 1-11+8\rangle_{17}=\langle -4\rangle_{17}=13.$$

Since $x_j<p_j$ and $y_j<p_j$, $p_j=2^{l_j}\pm1$, then the length of the binary sum (x_j+y_j), does not exceed (l_j+1) bits. Hence, the second number, obtained from partitioning the integer (x_j+y_j), consists of one carry out

bit. So, the following rule results from equations (1) and (2) for modulo p_j transformation of a binary integer.

Rule.

- If $0 \leq x_j + y_j < p_j$, then $\langle x_j + y_j \rangle_{p_j} = x_j + y_j$.
- If $x_j + y_j > p_j$, then there will be a carry out of the leftmost bit, and $\langle x_j + y_j \rangle_{p_j}$ is obtained by adding 1 to the binary sum if $p = 2^l - 1$ and by subtracting 1 from the binary sum if $p = 2^l + 1$.
- If $x_j + y_j = p_j$, then in this case, the result will be $11 \ldots 1$, which is converted in $2'$s-complement representation.

Figure 1 shows an example of the RNS addition. The cellular algorithm architecture for the RNS addition is given in Figure 2a. As the main RNS operations (addition, subtraction, and multiplication) are carried out in each modulus p_j in parallel, the algorithm architecture for each of such operations is done for one modulus.

The algorithm calculates $\langle x_j + y_j \rangle_{p_j}$ in the arrays p_j^1 and p_j^2 of a uniform size equal to $(l_j{+}1) \times \lfloor \log_2 l_j \rfloor$. The binary sum $x_j{+}y_j$ is generated in p_j^1. The

$$\langle 12 + 13 \rangle_{15} = \quad \langle 25 \rangle_{15} = 10 \quad \rightarrow + \begin{array}{r} 1100 \\ 1001 \end{array} \rightarrow 1\ 1001 \rightarrow 1010$$

Figure 1. An example of the RNS addition

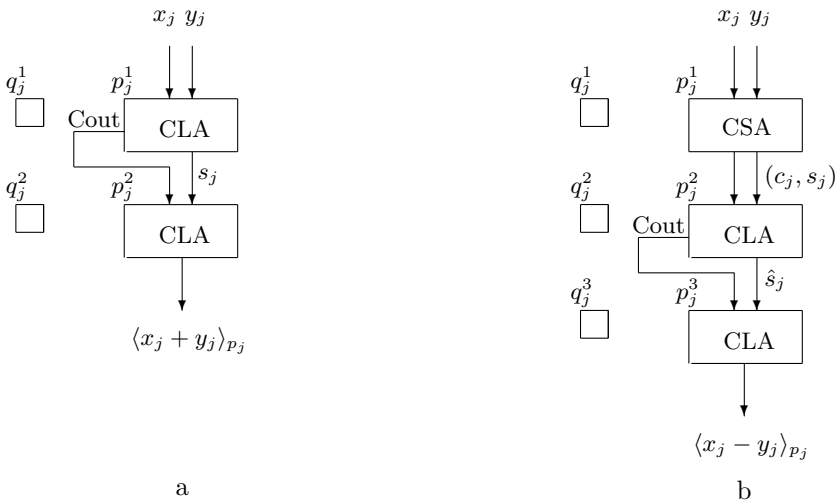

Figure 2. Cellular algorithm architecture for RNS addition (a) and RNS subtraction (b)

remainder of the obtained sum modulo p_j is formed in p_j^2 according to the rule. All the data of loading and processing in both arrays are accomplished by the control of the arrays q_j^1, q_j^2, respectively. To provide pipelining of the basic algorithm (Section 3.2), two fast conventional carry look-ahead adders (CLA) are used for a binary addition and the modulo p transformation of a binary sum.

The time complexity of a cellular algorithm for the RNS addition is defined by the time complexity of CLA. In the worst case, $t_{\text{ad}} = 2t_{\text{CLA}} = 2\log_2 p_{k-1} + 4 = 2l_{k-1} + 4$, where p_{k-1} is maximum modulo from $\mathcal{P}$, the period, p_{ad} (the time between two successive calculations of sums), is equal 1 step.

3.2. The cellular algorithm architecture for the RNS subtraction

The cellular algorithm for the RNS subtraction is reduced to addition, in which the second addend is a negative integer. As $\langle -\boldsymbol{y}_j \rangle_{p_j} = \langle p_j - \boldsymbol{y}_j \rangle_{p_j}$, then $-y_j = p_j + y'_j$, where y'_j is the integer $(-y)_j$ in $2'$s-complement representation. Figure 3 shows an example of the RNS subtraction.

$$
\begin{array}{rcccccccc}
\langle 12-13\rangle_{15} = \langle -1\rangle_{15} = 14 & \rightarrow & + & \begin{array}{r} 0.1100 \\ (p=15) \rightarrow \ 0.1111 \\ 1.0010 \\ 2'\text{s complement of } (-13) \rightarrow \ 1 \end{array} & \rightarrow & + & \begin{array}{r} 0.1100 \\ 0.0010 \end{array} & \rightarrow & 1110
\end{array}
$$

Figure 3. An example of RNS subtraction

Unlike the cellular algorithm architecture for the RNS addition, the algorithm architecture for the RNS subtraction has an additional array d_j for each p_j. In this array, a sum of four integers x_j, p_j, y''_j, and 1, where y''_j is $1'$s-complement representation of $-(y_j)$, is generated in the form of the two-row code (c_j, s_j), using a carry-save adder (CSA). As a result, the algorithm calculates the RNS subtraction in the time $t_{\text{sub}} = t_{\text{ad}} + 6$ (four steps are needed for loading the initial data and the inversion of the second addend, the CSA-addition takes 2 steps), $p_{\text{sub}} = 1$.

3.3. The cellular algorithm architecture for the RNS multiplication

The cellular algorithm for the RNS multiplication is performed in two steps. At the first step, the algorithm calculates binary products $z_j = x_j y_j$ for all p_j, $p_j \in \mathcal{P}$, in parallel. For this purpose, a cellular pipelined algorithm with a very short period (four steps) from [7] is used. At the second step, the transformations $x_j y_j \rightarrow \langle x_j y_j \rangle_{p_j}$ are carried out for all p_j in parallel, according to equations (1) and (2). For this purpose, the product z_j is

partitioned into two numbers: a_j^0 and a_j^1 if $p_j = 2^{l_j} - 1$. The number a_j^0 consists of l_j low-order digits of z_j, the number a_j^1 consists of l_j high-order digits of z_j. If $p = 2^{l_j} + 1$, the product z_j is partitioned into three numbers, since the product length is over $2l$ bits. The number a_j^2 includes only one bit.

Example 4. Let $\boldsymbol{x} = 7$ $(x = 111)$, $\boldsymbol{y} = 7$ $(y = 111)$, and let $p = 9 = 2^3 + 1$. $\langle xy \rangle_9 = \langle \mathbf{7} \times \mathbf{7} \rangle_9 = \langle 49 \rangle_9 = 5 = \langle 111 \times 111 \rangle_9 = \langle 110001 \rangle_9$. Further, the transformations are shown in Figure 4.

$$\langle 7 \times 7 \rangle_9 = 4 \rightarrow \quad 110001 \rightarrow + \begin{array}{r} 001 \\ \\ -110 \end{array} \rightarrow \quad + \begin{array}{r} 0.0001 \\ \\ 0.1001 \end{array} \rightarrow 100$$

$$2'\text{s complement of } (-6) \rightarrow \begin{array}{r} 1.1001 \\ 1 \end{array}$$

Figure 4. An example of RNS multiplication

Let $p_j = 2^{l_j} - 1$. The algorithm architecture for modulo $(2^{l_j} - 1)$ multiplication is given in Figure 5a. The initial data (y_j, x_j) are stored in the array y_j and in the 0-th row of the array p_j^1 of the size $l_j \times 1$ and $l_j \times (2l_j + 1)$, respectively; l_j low-order bits of the 0-th row of p_j^1 are significant bits, the others are equal to zero. The first step of the cellular algorithm forms the two-row code (c_j, s_j) of the product z_j in p_j^1 using fast carry-save technique for summation of partial products. Here generation and addition of partial products are done in parallel. Then the two-row codes (c_j, s_j) are transferred to the array p_j^2 of $\log_2 l_j \times (2l_j + 1)$ size to calculate the sum z_j by a carry-look-ahead adder. At the second step, the algorithm calculates the remainder $\langle x_j y_j \rangle_{p_j}$ in the array p_3. Data loading and processing in the arrays p_j^1, p_j^2, and p_j^3 are accomplished by the control of the corresponding arrays q_j^1, q_j^2, and q_j^3, respectively.

Let $p_j = 2^{l_j} + 1$. The algorithm architecture for modulo $2_j^l - 1$ multiplication is given in Figure 5b. The first step of this algorithm is done like to the algorithm architecture for modulo $p_j = 2^{l_j} - 1$ multiplication. At the second step, the algorithm calculates the sum $(a_j^0 - a_j^1 + a_j^2)$ in the array p_j^3 of $4 \times (l_j + 3) \times 2$ size. The first layer of p_j^3 is intended for storage of the two numbers: the modulo p_j and 1. p_j is placed in the 2-nd row, 1 is placed in the 3-rd row. At first, the number a_j^0 is loaded into the 0-th row of the array p_j^3. Then shifting the number a_j^0 one row to the bottom of the array, loading the number a_j^1 into the 0-th row of p_j^3, and loading the modulo p_j and 1 into the same rows of the 0-th layer are carried out in parallel. In this case, each digit of the number a_j^1 has been inverted. If $a_j^2 = 1$, then 1 in the third row of the 0-th layer of p_j^3 is shifted 1 bit to the left. After this, the

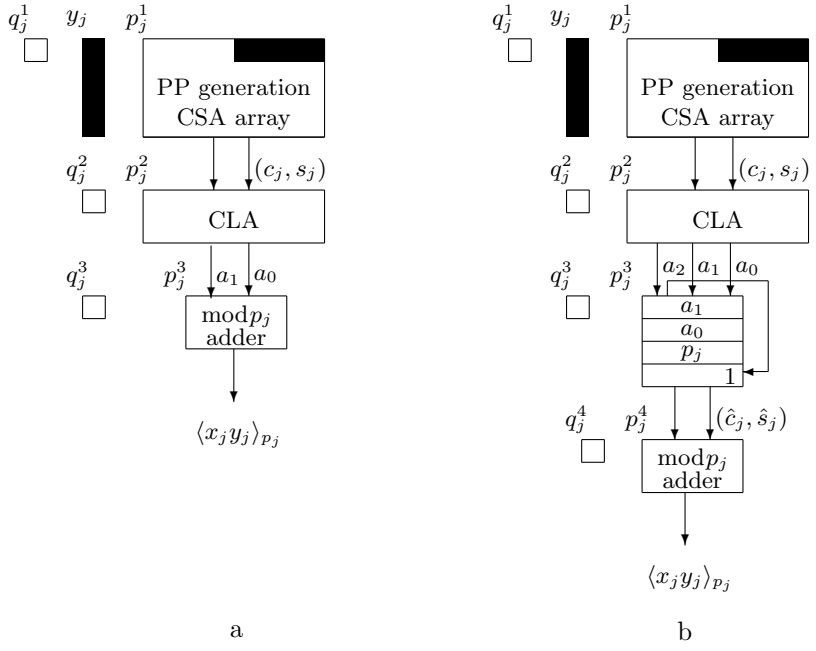

Figure 5. Cellular algorithm architectures for modulo $(2^{l_j} - 1)$ multiplication (a) and modulo $(2^{l_j} + 1)$ multiplication (b)

algorithm adds four integers, using the CSA. The obtained two-row code (c_j, s_j) is transferred into the array p_j^4 for modulo p_j addition.

The time complexity of mod $\mathcal{P}$ multiplication is defined by the time complexity of last remainder calculation and is equal to the following sum

$$t_{\text{mul}} = t_{\text{csa}} + t_{\text{cla}} + t_{(1)} + t_{\text{lsum}}.$$

Here t_{csa} is the time needed to generate a set of partial products and to form a two-row code of the product z_{k-1}, it takes $l_{k-1} + 2$ steps, t_{cla} is the time for summation of the two integers c_{k-1} and s_{k-1} at the CLA, $t_{\text{cla}} = \log_2 2p_{k-1} + 2 = l_{k-1} + 3$, $t_{(1)}$ (four steps) is the time to load the numbers in the array p_j^3, and to calculate the sum (1), $t_{\text{lsum}} = t_{\text{ad}}$ is the time required for the last modulo summation $(2l_{k-1} + 4)$. As a result, the algorithm calculates the modulo $\mathcal{P}$ product in the time $4l_{k-1} + 12$. The period of this algorithm is four steps.

3.4. The cellular algorithm architecture for the RNS division

The algorithm architecture is given in Figure 6. The modulo p_i is the divisor. The initial data are placed as follows. The table for the first approximation of the quotient $\hat{z}_j$ is stored in the array t_j^1. The table contains $(p_j - 1)$ $2'$s

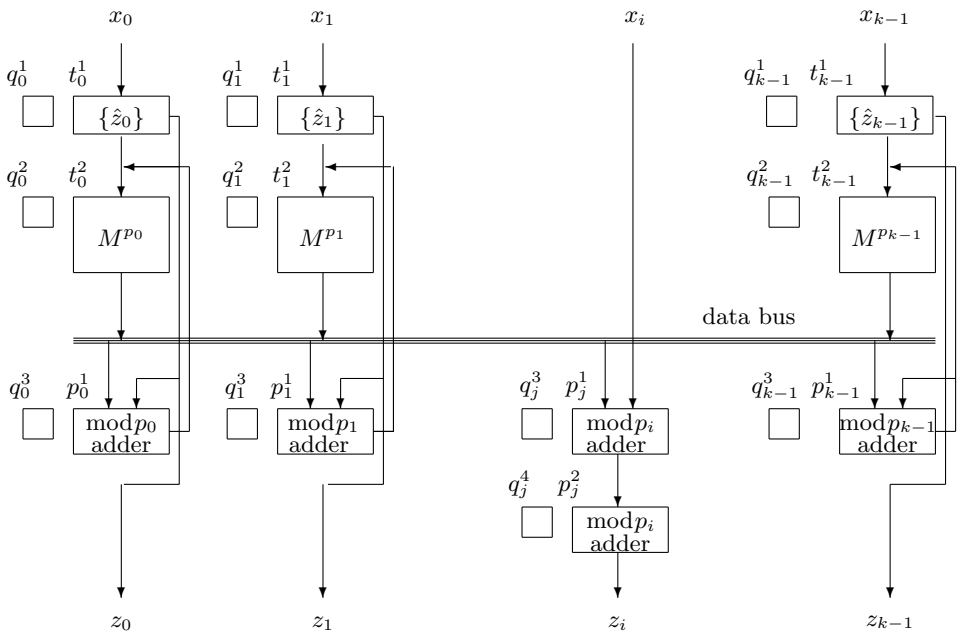

Figure 6. Cellular algorithm architecture for RNS division

complement numbers. The nullivization constant table M^{p_j} is placed in the array t_j^2. The quantity of the constant M^{p_j} equals $(p_j - 1)$.

At the first step, the cellular algorithm chooses the values $\hat{z}_j$ for all x_j, $j \neq i$, from the tables t_j^1 in parallel, and places them in the arrays p_j^0. The values obtained are remainders of the quotient, i.e., $\hat{z}_j = z_j$ for all j, $j \neq i$.

At the second step, a successive nullivization procedure is carried out beginning with the 0-th remainder. The cellular algorithm chooses a nullivization constant $M_l^0 = (m_{l0}^0, m_{l1}^0, \ldots, m_{l(k-1)}^0)$ from the array t_1^0 so, that $m_{l0}^0 = z_0$, and places this constant into the data bus. The control arrays q_j^1, q_j^2 accompany the fetch constants from the tables t_j^1 and t_j^2, and accommodation the nullivization constants into the data bus.

The values m_{lj}^0 are chosen in parallel for all p_j, $p_j \in \mathcal{P}$, and transferred to the arrays p_j^2, accompanied by the arrays q_j^3. Further, the algorithm calculates the values $g_j^1 = z_j^0 - m_{lj}^0$ for all p_j in the arrays p_j^2. As a result, the 0th remainder is equal to zero. The nulivization process is produced in $(k-2)$ steps. In response to this process, all the jth remainders, $j \neq i$, are set up into zero. The value $z_i = \langle 0 - g_i^{k-2} \rangle_{p_i}$ is formed by modulo p_i subtraction in the array p_i^3.

The algorithm carries out the modular division in the time

$$t_{\text{div}} = t_{\text{f}} + (k-1)(2t_{\text{f}} + t_{\text{sub}}) + t_{\text{sub}},$$

where t_{f} is the fetch time. Since $t_{\mathrm{f}} \ll t_{\mathrm{sub}}$, it may be omitted from the formula. Then $t_{\mathrm{div}} = (k-1)t_{\mathrm{sub}} + t_{\mathrm{sub}} = 2kl_{k-1} + 10k$. For a small number of moduli, the division time complexity unessentially exceeds in complexity the multiplication algorithm $(4l_{k-1} + 12)$. However, a period of the division algorithm is more essential ($p_{\mathrm{div}} = (k-2)t_{\mathrm{sub}}$) due to the communications among intermediate results at every step of the nulivization process.

4. The cellular pipelined algorithm architecture for 1D diffusion simulation using the residue number system

4.1. The RNS representation of 1D diffusion

It is known, that a finite difference representation of the 1D diffusion as a result of using an explicit scheme of the time and the spatial discretization, takes the following form

$$u_i^{t+1} = u_i^t + \frac{1}{d}(u_{i-1}^t + u_{i+1}^t - 2u_i^t) = u_i^t + \frac{1}{d}L(u_i^t), \tag{3}$$

where $t = 0, 1, \ldots$, u_i^t, $i = 0, 1, \ldots, N$, is a value of the integer function u at the nodes in 1D lattice, $\frac{1}{d}$ is a diffusion number.

To obtain an exact realization of division in scheme (3) by the RNS, two strategies are used.

Representation of the diffusion number as fraction. Since the number $\frac{1}{d}$ is a fraction, by definition, the number d is represented as ratio between the integers $d = \frac{p_j}{d_1}$, $p_j \in \mathcal{P}$. Then expression (3) takes the following form

$$u_i^{t+1} = u_i^t + \frac{d_1 L(u_i^t)}{p_j}. \tag{4}$$

Transfer of a remainder to the next step. Let $t = 0$, $L(u_i^0) \bmod p_j = r_{ij}^1 \neq 0$. Then $d_1 L(u_i^0) = \left\lfloor \frac{L(u_i^0)}{p_j} \right\rfloor p_j + r_{ij}^1$, where the remainder r_{ij}^1 is transferred to the 1-st step and added to $d_1 L(u_i^1)$. According to this strategy, expression (4) is rewritten as

$$u_i^{t+1} = u_i^t + \frac{L_i^t}{p_j}, \tag{5}$$

where $L_i^t = \left\lfloor \frac{d_1 L(u_i^t) + r_{ij}^t}{p_j} \right\rfloor p_j$, $r_{ij}^{t+1} = (d_1 L(u_i^t) + r_{ij}^t) \bmod p_j$, $r_{ij}^0 = 0$.

Representation (5) in the RNS is

$$\boldsymbol{u}_i^{t+1} = \boldsymbol{u}_i^t + \frac{\boldsymbol{V}_i^t - \boldsymbol{R}_i^{t+1}}{\boldsymbol{p}_j} = \boldsymbol{u}_i^t + \frac{\hat{\boldsymbol{V}}_i^t}{\boldsymbol{p}_j} = \boldsymbol{u}_i^t + \boldsymbol{D}_i^t, \tag{6}$$

where $\boldsymbol{V}_i^t = (\boldsymbol{v}_{i0}^t, \ldots, \boldsymbol{v}_{ij}^t, \ldots, \boldsymbol{v}_{i(k-1)}^t) = \boldsymbol{d}_1 \boldsymbol{L}(u_i^t) + \boldsymbol{R}_i^t$, $\boldsymbol{R}_i^{t+1}$ is the RNS number of the residue $\boldsymbol{v}_{ij}^t$. The algorithm, realizing expression (6) is referred to as *the basic algorithm* for the 1D diffusion simulation.

4.2. The cellular basic algorithm architecture for 1D diffusion simulation

The basic algorithm calculates values u_{ij}^{t+1} at the i-th node of a 1D lattice at the t-th time step in parallel for all moduli and successively inside each module. Figure 7 shows the cellular basic algorithm architecture for modulo p_j. (The control arrays are omitted here.)

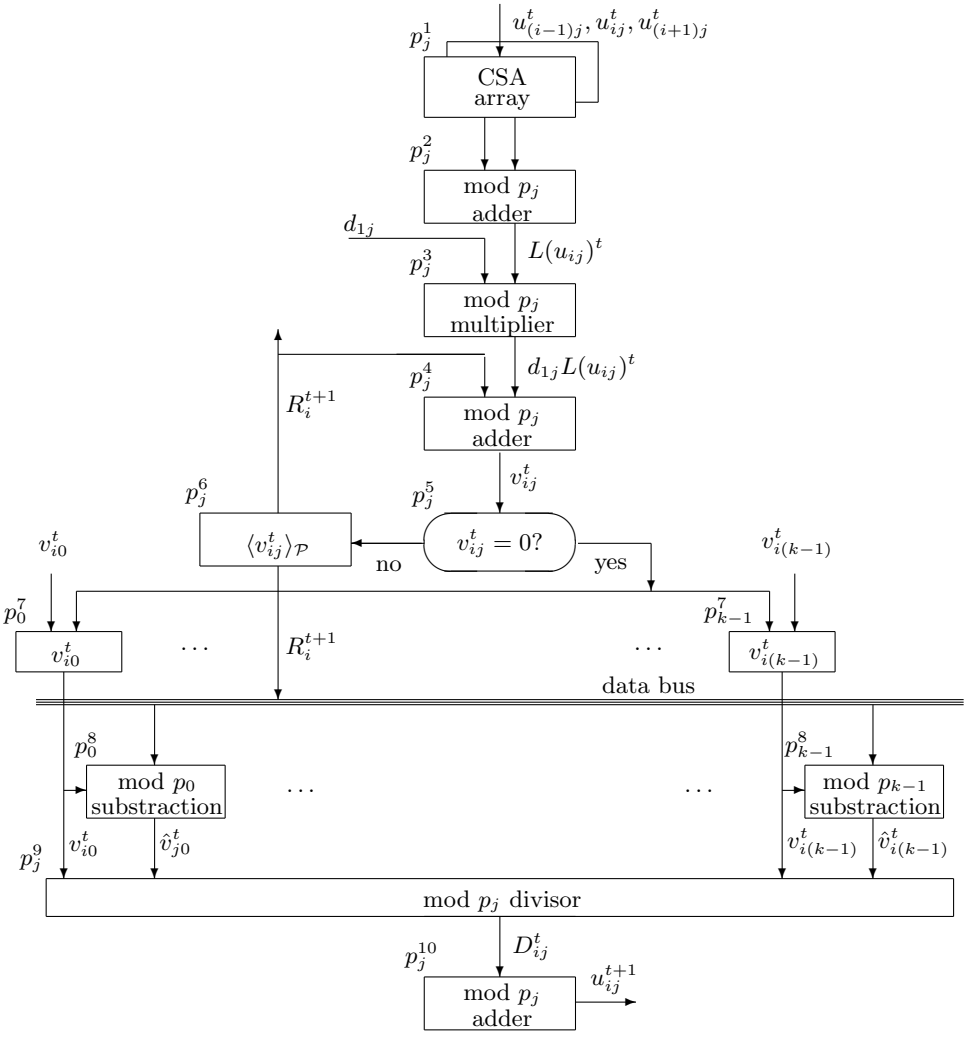

Figure 7. Cellular basic algorithm architecture for 1D diffusion simulation

At the first step, the algorithm forms the value $L(u_{ij}^t)$ in two arrays: p_j^1 of $5\times(l_j+4)\times 2$ and p_j^2 size. The first layer of the array p_j^1 stores the modulus p_j and 1 needed to calculate the value $L(u_{ij}^t)$. In the 0-th layer of the array p_j^1, two-row code (c_{ij}, s_{ij}) of the sum $(u_{i-1,j}^t+u_{i+1,j}^t-2u_{ij}^t)$ is generated and then transferred to modulo p_j adder (the array p_j^2) to calculate their sum. Then the formed sum is multiplied by d_{1j} in the array p_j^3. At the third step of the algorithm, the product $d_{ij}L(u_{ij})^t$ and the remainder r_{ij}^t obtained at $(t-1)$-st step are summed up in the array p_j^4. If $v_{ij}^t=0$, then modulo p_j division is carried out in the array p_j^9. Finally, the sum of the quotient D_{ij}^t and the j-th remainder of the function u_i^t is calculated in modulo p_j adder (the array p_j^{10}). If $v_{ij}^t\neq 0$, then $v_{ij}^t=r_{ij}^{t+1}$. The algorithm calculates the RNS number of the remainder v_{ij}^t, i.e., $R_i^{t+1}=\langle v_{ij}^t\rangle_{\mathcal{P}}=(r_{i0}^{t+1}, r_{i1}^{t+1}, \ldots, \underbrace{v_{ij}^t \ldots, v_{ij}^t}_{k-1-i})$ in the array p_j^6. Here $r_{im}^{t+1}=v_{im}^t-p_m$ for all $m<j$.

Remainders of the number R_i^{t+1} are transferred to the arrays p_j^4 for all p_j for their use in the calculation of the values V_i^{t+1} at the $(j+1)$th step. Furthermore, the number R_i^{t+1} is placed into the data bus to form the number $\hat{V}_i^t$ in the array p^8, where $\hat{v}_{im}^t=v_{im}^t-p_m$ for all $m\neq j$. Once the number $\hat{V}_i^t$ has been calculated, the modular division is performed in the array p_j^{10}. The basic algorithm architecture (Fig.7) is peculiar to the cellular processor architecture intended for the 1D diffusion simulation.

The time complexity of the basic algorithm is defined by the following sum

$$t_{\text{basic}}=t_{L(u)}+t_{\text{mul}}+2t_{\text{ad}}+2t_{\text{sub}}+t_{\text{div}}.$$

Here $t_{L(u)}$ is the time needed to obtain the modulo p_j sum of $L(u_{ij})^t=(u_{i-1,j}^t+u_{i+1,j}^t-2u_{ij}^t)$. It takes $6+t_{\text{ad}}=2l_{k-1}+12$ steps. As a result, the basic algorithm calculates the value u_i^{t+1} in the time $2l_{k-1}(k+9)+12k+62$. The period of the basic algorithm is equal to the period of the division algorithm.

4.3. The cellular algorithm architecture for 1D diffusion simulation

Let $U^0=(u_0^0, u_1^0, \ldots, u_{N-1}^0)$ be a set of values of the integer function u at the nodes in a 1D lattice. Let m be a number available for cellular processors realized in the basic algorithm for the 1D diffusion simulation. Obviously, a good time complexity estimation can be attained due to the following.

- Parallel implementation of a cellular algorithm using domain decomposition.
- Reduction of the basic algorithm period.

Parallel implementation of the cellular algorithm on a linear-connected processors is straightforward: for m processors, the simulation domain is split to m strips of equal size. Each processor stores this strip plus two notes from the neighboring strips. The algorithm is carried out in all the strips in parallel. Inside a strip, the algorithm is carried out sequentially for all $\frac{N}{m} - 1$ triplets $(u_{i-1}^t, u_i^t, u_{i+1}^t)$. At the end of each time step, the data in the overlapping nodes is updated by messages between the neighboring processors.

Reduction of the basic algorithm period is associated with increasing the number of devisers (D). This number is defined by the following relation: $p_{\text{div}} = \hat{p}_{\text{dif}} D$, where $\hat{p}_{\text{dif}}$ is a reduced period. As an example, let us take $k = 7$ and $\hat{p}_{\text{dif}} = 3p_{\text{mul}}$, where p_{mul} is the period of the multiplication algorithm (four steps), then $D = l + 5$. In this case, the initial data will be loaded at the 12-th step. In response to a deep pipelining algorithm the time complexity of one step of the algorithm is $t_{\text{basic}} + 12(\frac{N}{m} - 2) + t_{\text{up}}$ steps, where t_{up} is the time for data exchange between the neighboring processors. Parallel implementation of the cellular algorithm takes $K(t_{\text{basic}} + 12(\frac{N}{m} - 2)) + (K - 1)t_{\text{up}}$ steps, where K is the number of time steps.

5. Conclusion

In this paper, we propose the cellular algorithm architecture for the 1D diffusion simulation. For this purpose, the cellular pipelined algorithm architectures for computing the main RNS operation are proposed. The time complexity algorithm is $K(t_{\text{basic}} + 12(\frac{N}{m} - 2)) + (K - 1)t_{\text{up}}$ steps, where N is the number of notes in a 1D lattice, m is a number of strips. A good time complexity estimation is attained due to the following features:

- Processing data in all the strips in parallel.
- Processing of the arithmetic operations in all moduli in parallel.
- Deep pipelining at both the initial data and the computation process levels.
- Using the table look-up operation.
- Loading the initial data, transformation of intermediate results and their moving in parallel.

References

[1] Markova V. Using the residue number arithmetic for simulation of diffusion // Avtometria. – 2003. – Vol. 39, No. 3. – P. 60–71.

[2] Akyshskii I.J., Udizkii D.I. Computer Arithmetic in Residue Number System. – Moscow: Soviet Radio, 1968.

[3] Torgashov V.A. Residue Number System and Computer. – Moscow: Soviet Radio, 1973.

[4] Taylor F.J. Residue arithmetic: a tutorial with examples // IEEE Comput. Mag. – 1984. – Vol. 17, No. 5. – P. 50–62.

[5] Achasova S.M., Bandman O.L., Markova V.P., Piskunov S.V. Parallel Substitution Algorithm. – Singapore: World Scientific, 1994.

[6] Aho A., Hopcroft J., Ullman J. The Design and Analysis of Computer Algorithms. – Moscow: Mir, 1979.

[7] Markova V. Cellular algorithm architecture for long integers multiplication // NCC Bulletin. Series: Computer Science. – Novosibirsk: NCC Publisher, 1998. – Issue 9. – P. 45–57.