Bull. Nov. Comp. Center, Comp. Science, 17 (2002), 51–61
© 2002 NCC Publisher

Multilayer cellular pipelined algorithm architecture for complex scalar product computation

V. Markova

A new multilayer cellular pipelined algorithm architecture for the complex scalar product computation is presented. The time complexity is evaluated. The initial data and results are quaterimaginary numbers. The design is performed in terms of a model of distributed computation – Parallel Substitution Algorithm.

1. Introduction

Complex multiplication is the main operation in digital signal processing. One possible method for increasing speed of complex multiplication is through the use of non-conventional number systems, specifically, the Knuth (or the quaterimaginary) number system [1].

The Knuth number system (NS) is defined as a positional NS with the image radix $r = 2i$ and the base $D = \{0, 1, 2, 3\}$. The most interesting property of this number system is the possibility of representing a complex number as single vector (a quaterimaginary number). This property is the key to a high-speed performance of complex multiplication. In fact, in this case a complex multiplication requires only one multiplication distinctly from three multiplications and three additions for conventional positional NS.

However, in spite of its attractiveness the Knuth number system has not found much application. There are three reasons for that:

- The singularities of the conversion of the binary numbers in the equivalent quaterimaginary ones.
- The necessity of a sequential modification of intermediate results (the partial products and their sums).
- The absence of a multi-valued hardware.

The second disadvantage is associated with the fact that the result of any arithmetic operation over the quaterimaginary numbers may occur not to be a quaterimaginary one. This means that an intermediate result contains not only the digits belonging to the base (they are further referred to as *the staff* digits) but also, at least, one not belonging to the base (*the unstaff* digits). For the intermediate result be involved in the computation process, a modification of all unstaff digits (*the number-modification*) should be done beginning with the least significant digit. This modification is similar to the sequential carry propagation along a number. The time required for the number-modification is half the length of a modified number.

In [2], the first attempt has been undertaken to get rid of the above disadvantage. The 2D algorithm for complex multiplication is a cellular version of the classical algorithm. At the first stage of the 2D complex multiplication algorithm, the partial products (PP) and the modification of the results obtained are carried out in parallel. We employ the parallel number-modification as opposed to the classical algorithm. At the second stage, a pairwise summation of the quaterimaginary numbers is performed concurrently with the parallel modification of the intermediate results. These processes are repeated until the last sum is calculated. This algorithm multiplies two n-digit quaterimaginary numbers in time $T \sim 3.5n$. This estimation is obtained with the help of an experimental simulating system.

In [3], the above estimate has been improved. The algorithm has covered the classical form. As distinct from the 2D algorithm [2], here the 2-layer summation is used. The idea of the 2-layer summation consists in the following. The staff and the unstaff digits in each pair of the intermediate results are processed concurrently in two layers of a 3D array. This is considered to mean that the pairwise summation starts without waiting the intermediate results to be modified.

A new pipelined multilayer cellular algorithm for the complex scalar product computation is proposed in this paper. High-speed calculation is attained due to abandonment of the classical form for multiplication and deep pipelining at both the data and the computation process levels.

The Parallel Substitution Algorithm (PSA) [3, 4] is used for designing an algorithm. The PSA is the fine-grained parallelism model, which integrates the concepts of the cellular automaton and the Markov algorithm. Unlike other cellular models, the PSA properties and expressive capabilities enables us to represent any complex algorithm. Moreover, there is one-to-one correspondence between the PSA and the automata net, thus forming the basis for the architectural design.

This paper is organized as follows. The first section describes the main operations in the Knuth number system. In the third section, a 3D multilayer cellular pipelined algorithm architecture for the complex scalar product computation is described, its time complexity being evaluated.

2. The Knuth arithmetic

The Knuth number system (NS) is a positional NS with the image radix $r = 2i$ and the base $D = \{0, 1, 2, 3\}$. The most interesting property of this number system is the possibility of representing a complex number as a single number. This property is the key to the high-speed performance of complex multiplication. Here we deal with only *Gaussian integers* (complex numbers with integer parts).

2.1. The Knuth representation

The Knuth (or quaterimaginary) number system is a positional NS with the image radix $r = 2i$ and the base $D = \{0, 1, 2, 3\}$.

In the Knuth NS, any Gaussian integer $g = a+bi$ corresponds to *a Knuth representation* or *a quaterimaginary number* as sequence of the coefficients

$$\mathbf{g} = \mathbf{g}_{n-1} \cdots \mathbf{g}_0 {\cdot} \mathbf{g}_{-1}, \tag{1}$$

where $\mathbf{g}_j \in D$ for all $j \neq -1$, and $\mathbf{g}_{-1} \in \{0, 2\}$. The digits with even indices in (1) specify the real part of a Gaussian integer, the digits with odd indices – the imaginary one.

$$\begin{aligned} A &= 8 \to A_4 = 20.0 \to A_{-4} = 120.0 \\ B &= -5 \to B_4 = -2.2 \to B_{-4} = 12.2 \end{aligned}$$

$$\begin{array}{rcccccccl} 8_{-4} = & 1 & & 2 & & 0 & & & \\ & \Downarrow & & \Downarrow & & \Downarrow & & & \\ g = 8 - 5i \Rightarrow & \mathbf{g}_4 & \mathbf{g}_3 & \mathbf{g}_2 & \mathbf{g}_1 & \mathbf{g}_0 {\cdot} & \mathbf{g}_{-1} & & \Rightarrow 11220.2 = \mathbf{g} \\ & & \Uparrow & & \Uparrow & & \Uparrow & & \\ -5_{-4} = & & 1 & & 2 & & 2 & & \end{array}$$

Figure 1

The conversion algorithm $g \Rightarrow \mathbf{g}$ is shown in Figure 1. At first, both parts of the integer $g = 8-5i$ are transformed to the quaternary numbers, then to the quaterimaginary ones. The obtained representations are concentrated into the single vector $\mathbf{g}$ in such a way, that digits of the real part are put at the even positions, and the digits of imaginary part are put at the odd ones. The length of the quaterimaginary number is equal to the length of the greatest part of the Gaussian integer in the binary NS.

The conversion $\mathbf{g} \Rightarrow g$ is executed in the usual way as in any positional number system. For example, the inverse conversion of the quaterimaginary number $\mathbf{g}$ (see Figure 1) is computed as the following sum

$$g = [0(-4)^0 + 2(-4)^1 + 1(-4)^2] + 2i[2(-4)^{-1} + 2(-4)^0 + 1(-4)^1] = 8 - 5i.$$

2.2. The Knuth arithmetic

Let us consider the main operations in the Knuth number system. Let $\mathbf{g} = \mathbf{g}_{n-1} \dots \mathbf{g}_1\mathbf{g}_0.\mathbf{g}_{-1}$ and $\mathbf{h} = \mathbf{h}_{n-1} \dots \mathbf{h}_1\mathbf{h}_0.\mathbf{h}_{-1}$ be two n-digit quaterimaginary numbers.

The quaterimaginary sum $\mathbf{s} = \mathbf{s}_{n+1} \dots \mathbf{s}_1\mathbf{s}_0.\mathbf{s}_{-1}$ is produced in two steps. At the first step, for each pair of digits $(\mathbf{g}_j, \mathbf{h}_j)$, $j = -1, 0, \dots, n-1$, the sum modulo (-4) is obtained as

$$(\mathbf{g}_j + \mathbf{h}_j) \bmod (-4) = c_{j+2} + \mathbf{v}_j,$$

where $c_{j+2} \in \{0, -1\}$ and $\mathbf{v}_j \in D$ are called *the intermediate carry* digit and *the intermediate sum* digit, respectively. Further, the digits belonging to the base are referred to as the *staff* ones. Distinctly from other number systems the carry is transferred to two positions ahead of a current intermediate sum digit. At the second step, the final sum digit s_j is calculated by the usual arithmetic addition

$$\mathbf{v}_j + c_j = s_j \qquad \text{for all } j = -1, 0, \dots, n+1.$$

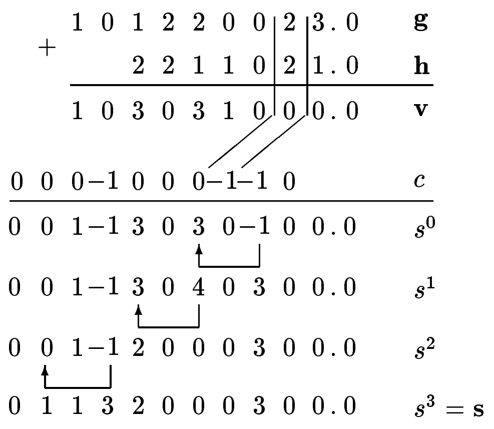
It is easily seen that s_j may occur not to belong to the set D. Such digits are further called *the unstaff* digits. In order that an intermediate result becomes the quaterimaginary number, modification of unstaff digits (the calculation of new values) should be performed according to Rule 1 beginning with the least significant digit. Further, the modification of an unstaff digit to the base is referred to as *the digit-modification*. The calculation of new values of the intermediate result digits is similar to the carry propagation along the number. The conversion process of $s^0 \Rightarrow s^1 \Rightarrow \dots s^i \Rightarrow \dots \Rightarrow \mathbf{s}$ according to Rule 1, is further called *the sequential number-modification.*

Rule 1. Let s^i be an intermediate result. Then for any pair of digits (s^i_j, s^i_{j+2}), $j = -1, 0, \dots, n+1$, $s^i_j \notin D$, the following computation is done:

- If $s^i_j < 0$, then $s^{i+1}_j = s^i_j + 4$ and $s^{i+1}_{j+2} = s^i_{j+2} + 1$.
- If $s^i_j \geq 4$, then $s^{i+1}_j = s^i_j - 4$ and $s^{i+1}_{j+2} = s^i_{j+2} - 1$.

In worst case, the time required for the sequential number-modification of n-digit intermediate result is $n/2$ steps.

Example. Let $\mathbf{g} = (227 - 34i)_{2i} = 101220023.0$, $\mathbf{h} = (-111 - 30i)_{2i} = 2211021.0$. Figure 2 shows computation of their quaterimaginary sum. Three steps are needed for the number-modification of the indeterminate result $s^0 = 001\,{-1}3030\,{-1}00.0$ to the quaterimaginary number $\mathbf{s} = 01132103300.0$. It is easy to check up that the sum obtained in the Knuth number system agrees with directly computed sum. Indeed, the inverse



```
        1 0 1 2 2 0 0|2|3 . 0     g
  +
              2 2 1 1 0|2|1 . 0     h
        ---------------------------
        1 0 3 0 3 1 0|0|0 . 0     v

  0 0 0-1 0 0 0-1-1 0             c
  ---------------------------------
  0 0 1-1 3 0 3 0-1 0 0 . 0       s^0
  0 0 1-1 3 0 4 0 3 0 0 . 0       s^1
  0 0 1-1 2 0 0 0 3 0 0 . 0       s^2
  0 1 1 3 2 0 0 0 3 0 0 . 0       s^3 = s
```

Figure 2

conversion of the quaterimaginary number **s** is $[0(-4)^0 + 3(-4)^1 + 0(-4)^2 + 2(-4)^3 + 1(-4)^4] + 2i[0(1/4)^{-1} + 0(-4)^0 + 0(-4)^1 + 0(-4)^2 + 3(-4)^3 + 1(-4)^4] = 116 - 64i$.

The quaterimaginary product $\mathbf{p} = \mathbf{p}_{2n-1} \dots \mathbf{p}_1\mathbf{p}_0.\mathbf{p}_{-1}$ is calculated in a classic manner first by obtaining n partial products and then performing their summation.

Each k-th partial product $p_k = (p_{k,n+1} \dots p_{k,1}p_{k,0}.p_{k,-1})$, $k = -1, 0, \dots, n-1$, is obtained in two steps. At the first steps, for each pair (g_k, h_j), $j = -1, 0, \dots, n-1$, the product modulo (-4) is defined

$$(\mathbf{g}_j \cdot \mathbf{h}_k) \bmod (-4) = r_{k,j+2} + \mathbf{z}_{k,j},$$

where $r_{k,j+2} \in \{0, -1, -2\}$ and $\mathbf{z}_{k,j} \in D$ are called *the intermediate carry* digit and *the intermediate product* digit, respectively.

At the second step, the partial product digit $p_{k,j}$ is calculated by the usual arithmetic addition

$$\mathbf{z}_{k,j} + r_{k,j} = p_{k,j} \qquad \text{for all } j = -1, 0, \dots, n+1.$$

If the obtained partial product is not a quaterimaginary number, then the number-modification should be performed according to Rule 1. The final result (the quterimaginary product) **p** is calculated by summation of all partial products using the quaterimaginary addition.

As one can see, the quaterimaginary product requires only one multiplication distinctly from three multiplications and three additions for a conventional positional NS. One of the reasons which restrains the use of the Knuth number system is the sequential number-modification.

In [3], the parallel number-modification was presented. It is done according to Rule 2 in time $O(\log n)$.

Rule 2. Let s^i be an intermediate result. For each pair (s^i_j, s^i_{j+2}), $j = -1, 0, \dots, n+1$, the following computation is performed in parallel

1. If $s^i_j < 0$, then $s^{i+1}_{j+2} = \tilde{s}^i_{j+2} + 1$, where

$$\tilde{s}^i_{j+2} = \begin{cases} s^i_{j+2} + 4 & \text{if } s^i_{j+2} < 0, \\ s^i_{j+2} & \text{if } s^i_{j+2} \in D, \\ s^i_{j+2} - 4 & \text{if } s^i_{j+2} > 3. \end{cases} \tag{2}$$

2. If $s^i_j > 3$, then $s^{i+1}_{j+2} = \tilde{s}^i_{j+2} - 1$ where $\tilde{s}^i_{j+2}$ is calculated according to (2).
3. If $s^i_j \in D$, then $s^{i+1}_{j+2} = \tilde{s}^i_{j+2}$ where $\tilde{s}^i_{j+2}$ is calculated according to (2).

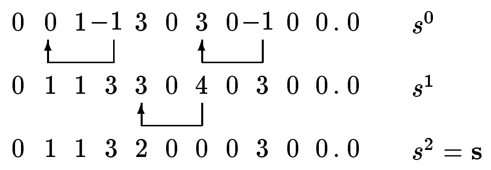

0 0 1 −1 3 0 3 0 −1 0 0.0 s^0
0 1 1 3 3 0 4 0 3 0 0.0 s^1
0 1 1 3 2 0 0 0 3 0 0.0 $s^2 = \mathbf{s}$

Figure 3

An example of the parallel number-modification of the sum s^0 from the example performed according to Rule 2 is shown in Figure 3.

3. Multilayer cellular pipelined algorithm architecture for complex scalar product computation

In this section, we present a new 3D cellular pipelined algorithm architecture for the complex scalar product computation and evaluate its time complexity (*the period* (T_p) *the latency* (T_l) and *the execution time* (T_e)). In this case, T_p is the time between two successive calculations of products, T_l is the time needed to generate the first product, and T_e is the total time to generate the result. At first, we will describe the main principles of a new algorithm.

3.1. Main principles of the new algorithm

Let $\mathbf{G} = (\mathbf{g}_{m-1}, \mathbf{g}_{m-2}, \dots, \mathbf{g}_0)$ and $\mathbf{H} = (\mathbf{h}_{m-1}, \mathbf{h}_{m-2}, \dots, \mathbf{h}_0)$ be two vectors, whose components are n-digit quaterimaginary numbers. It is required to calculate the scalar product of the two vectors $\mathbf{G}$ and $\mathbf{H}$

$$\mathbf{P} = \mathbf{G} \times \mathbf{H} = \sum_{i=0}^{m-1} \mathbf{p}_i = \sum_{i=0}^{m-1} \mathbf{g}_i \mathbf{h}_i. \tag{3}$$

Let us design the algorithm architecture, based on the belief that only a single 3D array can be used for computing the products $\mathbf{g}_i\mathbf{h}_i$. This means that in spite of a special parallelism (3), the high-speed sequential computation of the product $\mathbf{P}$ can be attained due to the following:

- abandonment of the classical form for multiplication and
- deep pipelining at both the data and the computation process levels.

Clearly, the process pipelining is based on functional decomposition of task (3). In our case, the pipeline has two stages. The first stage generates the products. The second stage accumulates the sum as

$$P^i = P^{i-1} + \mathbf{g}_i\mathbf{h}_i. \tag{4}$$

The data pipelining is classified as two types: the initial data pipelining and data reduction. *The initial data pipelining or data forwarding* is defined as loading and moving of the initial data inside a cellular array. The *Data reduction* is interpreted as high-speed summation and moving a result in the direction of the computation front. Currently, there are many reduction schemes.

In Figures 4 and 5, two reduction schemes for multiplying 12-digit integers are shown. Here the symbols r/z and p stand for the intermediate

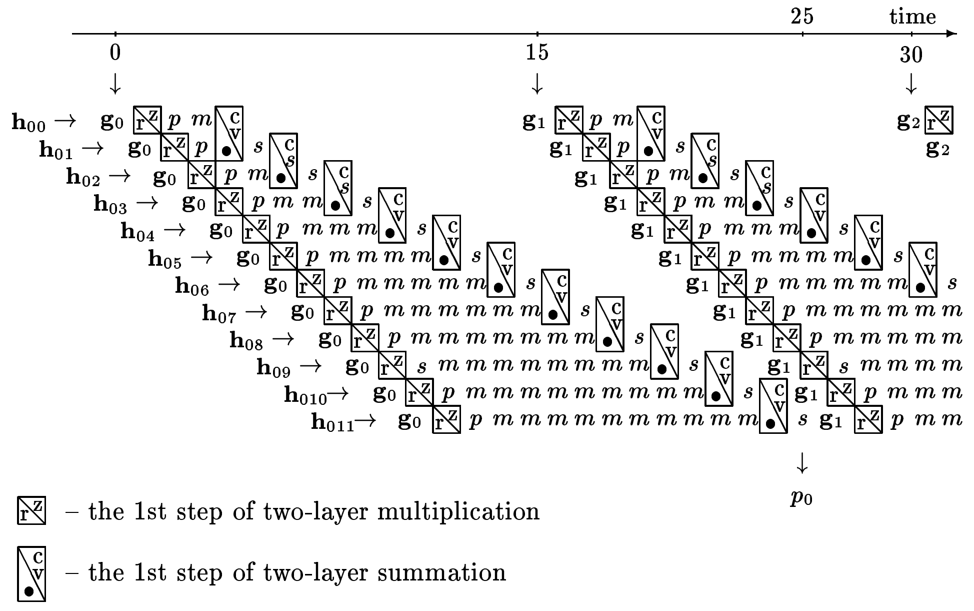


Figure 4

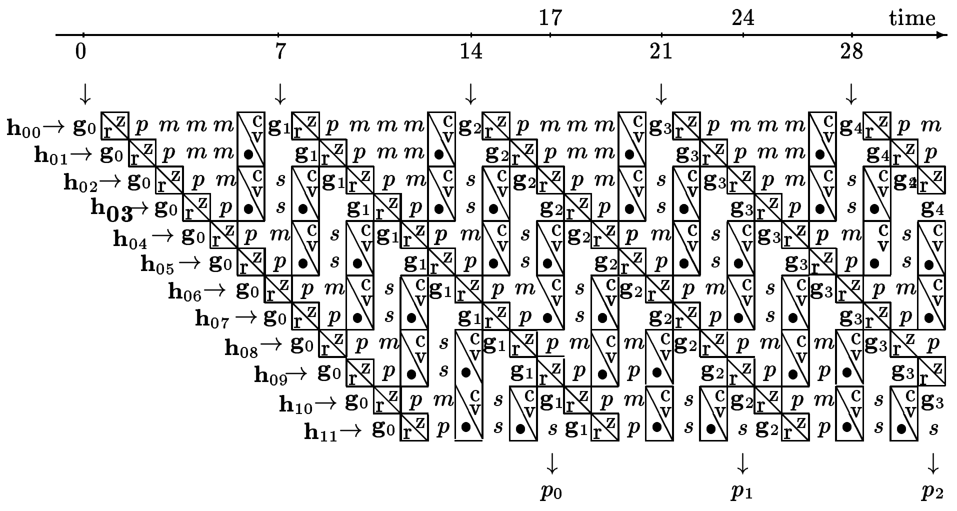


Figure 5

carry/the intermediate product and product, and m denotes the integer, which is modified. The symbols c/v and s stand for the intermediate carry / the intermediate sum, sum, respectively, and • denotes the integer with the unstaff digits. In both schemes the multipliers are loaded digit serially, the least significant bit first. The multiplicands are loaded digit paralleling in w step intervals, $w = T_p$. The value of w will be estimated bellow.

Let us consider the calculation of the first product in Figure 4. Beginning with the 4th step, a single summation of the neighboring intermediate results without preceding reduction is done every 1st step. (Here a pair of the neighboring intermediate results is written in the neighboring rows with even and odd indices.) As a result, the intermediate results are reduced one row per two step to the bottom of the processing array. So, according to this reduction scheme the algorithm calculates the first product at the $(2n + 2)$-th step, the second product – in $(n + 3)$ step intervals. Hence, loading the multiplicand $\mathbf{g}_1$ is performed at the $(n + 3)$-th step.

As distinct from the reduction scheme in Figures 4 and 5 two summations of the neighboring intermediate results are carried out in parallel. The second step of summation of the first pair is merged with shifting the obtained sum one position down. As a result, two rows leave the process of calculation of the product and are ready to take a new multiplicand. Hence, loading the $\mathbf{g}_1$ is performed at the 7th step and the latency equals $(n + 6)$.

So, the computation process is distributed over a 3D cellular array according to the following principles:

1. The multiplicand is shifted one row to the bottom.
2. The multipliers are loaded digit serially, the least significant bit first.

The multiplicands are loaded digit paralleling at 7 step intervals according to the scheme in Figure 5.

3. The generation of partial products is carried out in two layers.
4. The summation of the staff digits and the processing of the unstaff ones in each pair of intermediate results are done concurrently in two layers.
5. The parallel number-modification of the intermediate results.
6. The generation of the partial products, the number-modification of the intermediate results, the data reduction and the initial data loading are performed concurrently.

3.2. New algorithm architecture

The 3D cellular pipelined algorithm for the complex scalar product computation is carried out in the arrays H, G, P_1, P_2, C_H, C_G, C_{P2} (Figure 6).

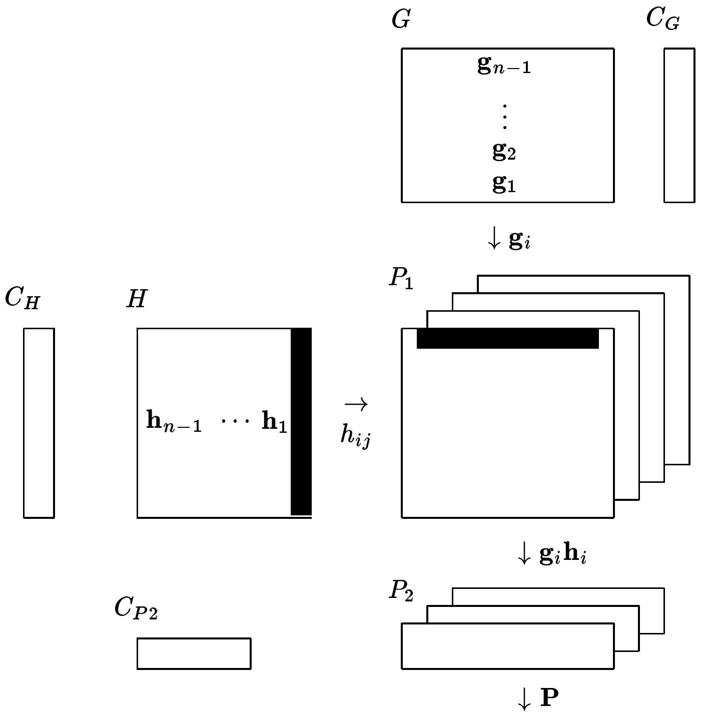


Figure 6

The 2D arrays H and G store the initial data (multipliers and multiplicands). The least significant digit of multiplier is placed in the 0th row of the array H. The least significant digit of the multiplicand is placed in the 2nd column of G. In Figure 4, the first pair of the initial data $(\mathbf{h}_0, \mathbf{g}_0)$ to be multiplied is marked. Data loading is accomplished by the control of the arrays C_H and C_G.

The 3D arrays P_1 and P_2 are intended for processing. The array P_1 has four layers. The 0th and the 1st layers constitute a pure processing field. The generation of the partial products is performed as follows. At the first step, the intermediate carry digits and the intermediate product ones are generated in the 0th and the 1st layers, respectively. At the second step, the obtained results are summed up and the partial products are placed in the 0th layer. If the obtained result is not an quaterimaginary integer, then it is modified iff it does not participate in summation.

The summation is carried out as follows. At the first step, for each pair of the staff digits the intermediate sum and carry digits are obtained and placed in the 1-th layer. At the same time, the unstaff digits from the odd row of the given pair are added to the unstaff ones of the even row. At the second step, the algorithm calculates the arithmetic sum of the unstaff digits and the quaterimaginary sum of the staff ones and loads in the 0th layer according to the scheme in Figure 5. Each product is generated in the last row of the 0th layer and then is transferred into the 0th layer of the array P_2. The data processing is done by control of the 2nd and the 3rd layers. The above processes are repeated until the last product is calculated. This moment is indicated by the array C_{P2}.

The array P_2 has three layers. The algorithm accumulates the sum (4) in two layers and modifies a result when it does not participate in the summation. The 2nd layer of the array P_2 controls the two-layer summation as well as the modification. The result (the quaterimaginary product $\mathbf{P}$) is considered to be obtained when the fact of termination is recognized by the array C_{P2}.

3.3. The time complexity

The presented 3D algorithm has the following time complexity:

- The latency $T_l = n + 5$.
- The period $T_p = 7$.
- The execution time is $T_e = (n+5) + 7(m-1) + 3 + \log n = n + 7m + 1 + \log n$, where m is the number of products.

This algorithm calculates a complex scalar product faster than that of ordinary algorithms. Of course, the cost of the hardware has not been assessed.

4. Conclusion

In this paper, we present the new 3D cellular pipelined algorithms for complex scalar product computation.

As would be expected, the 3D algorithm has a very short period (7 steps). This is achieved due to the following features:

- using the Knuth number system,
- deep pipelining at both the data and the computation process levels,
- the parallel number-modification of the intermediate results,
- the generation of the partial products, the number-modification of the intermediate results, data reduction, and loading the initial data in parallel.

However, it is difficult to estimate the proposed algorithms with respect to their ability to be embedded on a chip without doing a detailed design.

References

[1] Knuth D.E. An imaginary number system // Commun. ACM. – 1960. – Vol. 3. – P. 245–247.

[2] Markova V.P. The cellular Knuth algorithm for complex number multiplication // Parcella'94. Proceedings of the VI International Workshop on Parallel Processing by Cellular Automata and Arrays, Potsdam, Sept. 21–23, 1994. – P. 91–98.

[3] Markova V.P. Multilayer cellular algorithm for complex number multiplication // Proceedings of ASAP-95, Strasburg, France, July 16–20, 1995. – Los-Alamos, USA: IEEE Computer Society Press, 1995. – P. 290–297.

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