*Bull. Nov. Comp. Center*, Comp. Science, 9 (1998), 45–57
© 1998 NCC Publisher

# Cellular algorithm architecture for long integers multiplication in arrays of restricted size

V. Markova

A new cellular algorithm architecture for multiplication of two long binary integers in arrays of restricted size is presented. The new algorithm is based on "divide and conquer" technique and performed in terms of a model of fine-grained parallelism – Parallel Substitution Algorithm. Time complexity of the new algorithm for multiplication of $n$-bit binary integers product is $(1.25n + \log_2 n + 14)$ steps, which is less to that of the parallel version of the Karatzuba algorithm constructed on the same techniques.

## Introduction

In the design of high-speed algorithms it occurs rather frequently that the size of a task (the operand length) is larger than the size of a computing array (the number of processor elements). In this case, designing algorithms becomes more complicated because of an additional problem of the coordination of the task size and the array size.

To solve a task of "big" size, we use the well-known "divide and conquer" technique [1]. It consists in the following. The task of "big" size is broken into tasks of "smaller" size, formulated in a similar way. At first, tasks of "smaller" size are solved and then the solution to the "big" task is found as a composition of the obtained solutions. Moreover, for solving tasks of "smaller" size this technique can be applied recursively. It is known that "divide and conquer" technique allows one to reduce the time complexity of the algorithm. For example, due to this technique the time complexity of integer multiplication was reduced from the traditional estimation $O(n^2)$ to $O(n^{\log 3})$ in the Karatzuba algorithm [1] and to $O(n \log_2 n \log\log_2 n)$ in the Schonhage–Strassen algorithm [1].

In this paper, we present a new cellular algorithm architecture for multiplication of two long binary integers in arrays of restricted size and give the time complexity of the new algorithm.

By *architecture* of cellular algorithm we mean the space-time arrangement of the data, operations and their interaction. Algorithm architecture maps a structure of data interaction and, similarly to device architecture has, some levels of the description which form hierarchy. Each level of the architecture description is defined by a degree of a detail. The architecture of algorithm may be represented graphically and analytically.

The new algorithm is based on "divide and conquer" technique and is designed within the framework of a model of fine-grained parallelism – Parallel Substitution Algorithm [2, 3]. The product of two $n$-bit binary integers $X$ and $Y$ is formed as $XY2^{-n/2}$. The scheme for computation of the shifted product includes four multiplications of $n/2$-bit integers, three additions and two shifts $n/2$ bits. For this scheme realization we use a cellular algorithm for computing a sum of products of binary integers (further referred to as *the basic* algorithm) with a period of four steps [3].

The new algorithm multiplies two $n$-bit integers in time $(1.25n+\log_2 n+14)$. For comparison, the parallel version of the Karatzuba algorithm requires $(1.75n + 2\log_2 n + 15)$ steps. (Let us recall that its computation scheme includes there multiplications, six additions, and two shifts $n$ and $n/2$ bits, moreover, the multipliers of the third pair are two sums of $n/2$-bit integers.) The reduction of the time complexity of the presented algorithm is achieved due to the following. Firstly, the algorithm forms the shifted product, it reduces the number of shifts in half. Secondly, the algorithm generates four products. It allows to constant the basic algorithm architecture. In the Karatzuba algorithm, to be able to multiply the third pair multipliers should be calculated in seven steps. Even though a fast carry-look-ahead adder is used, the summation takes at least $\log_2 n$ steps. Hence, the summation is done in time only for $n < 16$, otherwise the multiplicand loading rate should be changed.

The new algorithm is exceeds in complexity the basic algorithm which computes product of two integers in time $(n + \log_2 n + 3)$. So, "divide and conquer" technique does not decrease the time complexity of the parallel algorithm as compared with the sequential one. It is explained by the following, the basic algorithm multiplies with such a high speed that decreasing the time complexity due to the reduction of the operand length by half ($\sim 0.5n$) is less than the time needed to realize a composition of four $n/2$ bit integer products ($\sim 0.75n$).

The article is organized as follows. In the second section a necessary background for organization of long integers multiplication is given. The third section is devoted to cellular algorithm for computing a sum of products. The new cellular algorithm architecture for multiplication of two long binary integers and its time complexity are discussed in the fourth section.

## 1. Background

In this section the necessary material for understanding the new cellular algorithm architecture is given. It includes the Karatzuba algorithm (an illustration of "divide and conquer" technique) and a carry-save adder [4] (an effective way of the accelerated summation).

## 1.1. Karatzuba algorithm

Let $X$ and $Y$ be two $n$-bit binary integers (for simplicity $n$ is a degree of integer 2). The multiplicand and the multiplier are broken into two equal parts $a$ and $b$, $c$, and $d$, respectively (Figure 1). Then the product $P$ can be expressed as

$$P = XY = \left(a2^{n/2} + b\right)\left(c2^{n/2} + d\right) = ac2^n + (ad + bc)2^{n/2} + bd. \qquad (1)$$

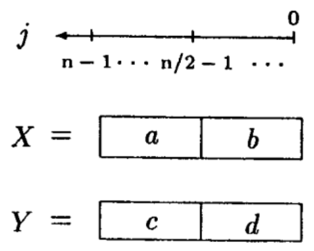

**Figure 1**

Equality (1) gives the scheme for calculation of $n$-bit integers product with the help of four multiplications of $n/2$-bit integers, several additions and shifts. To calculate each of four products the given scheme can be applied recursively. If we denote $ac$, $bd$ and $(a+b)(c+d)$ as $v$, $w$ and $u$, respectively, then equality (1) can be transformed to the following form:

$$P = XY = v2^n + (u - v - w)2^{n/2} + w. \qquad (2)$$

Here $u$ is the product of $(n/2+1)$-bit integers. The substitution of three multiplications for four ones has allowed to improve an asymptotical estimate of the time complexity up to $O(n^{\log 3})$ instead of $O(n^2)$. It is necessary to add that the choice of scheme (2) was made by the author of the algorithm [1] with the assumption that multiplication is more complex than addition.

## 1.2. Carry-save adder

A parallel counter (3,2) is the base of a carry-save adder (CSA). A (3,2)-counter is a usual single-bit full adder, i. e., a combinatorial network, which receives three bits of equal weight as inputs and produces a 2-bit word corresponding to their sum as output. The first bit is the carry and the second bit is the sum. (Both bits are calculated by addition rules in the binary number system which, as it is known, has not natural parallelism as

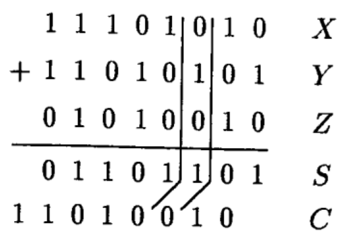

**Figure 2**

distinct from other number systems.) It is conservation, rather than carry propagation from the less significant bit to the more significant bit, that allows CSA to reduce any three binary integers $X$, $Y$, and $Z$ into two binary integers (the sum $S$ and the carry $C$) in one step (Figure 2), i. e., in parallel. Further the obtained result will be called *the two-row code* of a sum.

CSA as an effective way of the accelerated summation is used in fast multiplication schemes in different ways. In reality, these are the CSA array and the Wallace tree [5]. The former reduces a partial products set into the two-row code product in time $O(\log_3 n)$, the latter – in time $O(n)$.

## 2. Cellular algorithm architecture for computing a sum of products

In this section, we present in some detail the cellular algorithm architecture for computing a sum of products, since this algorithm (further referred to as *the basic* one) dictates a computation scheme of the new algorithm. The time complexity of the basic algorithm is given.

### 2.1. Basic algorithm architecture

Figure 3 shows the cellular algorithm architecture for computing a sum of products of $n$-bit binary integers

$$P = \sum_{i=0}^{m-1} P_i = \sum_{i=0}^{m-1} X_i Y_i. \qquad (3)$$

Here the arrays $ya$ and $xa$ store the initial data (the multipliers and the multiplicands). The first pair $(X_0, Y_0)$ is placed in the arrays $pa1_x$ and $ya_y$. The former is the 0-th row of $pa1$, $n$ low-order bits of the row are the significant bits, the others – are equal to zero. The latter is the rightmost column of $ya$. In Figure 3 the arrays $ya_y$, $pa1_x$ are marked. The arrays $pa1$, $pa2$ and $pa3$ are intended for computing the product $P$ according to (3). $pa1$ calculates $m$ products in the form of the two-row codes. The array $pa2$ (CSA) accumulates the two-row code of the product $P$ which is transferred

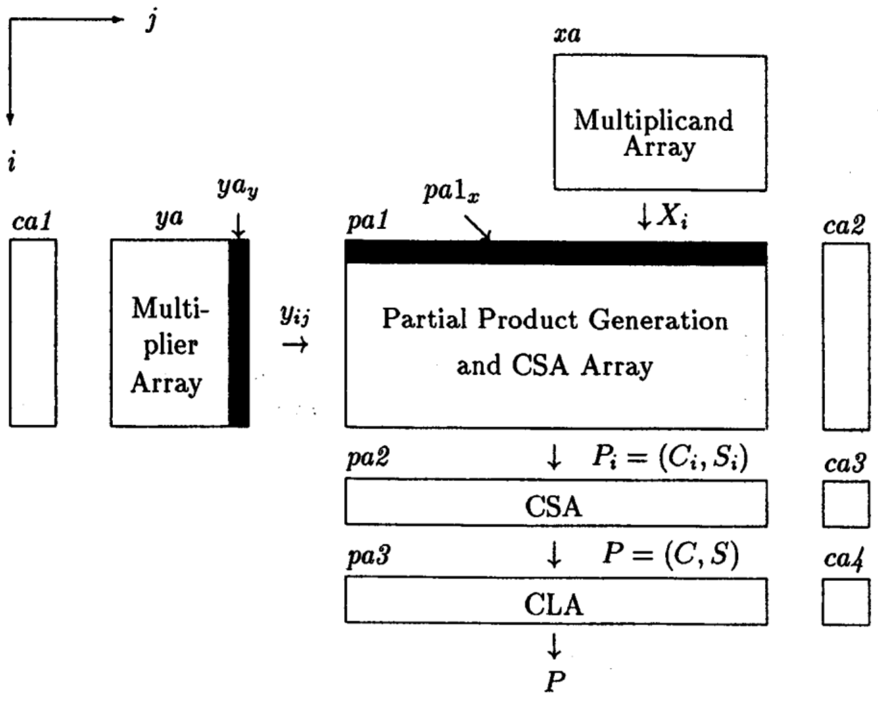

**Figure 3**

into the array $pa3$ (carry-look-ahead adder (CLA)) [5] for summation of two final integers. Data loading is accomplished by the control of the arrays $ca1$, and $ca2$, data processing is accomplished by the control of the arrays $ca2$, $ca3$, and $ca4$.

The computation process in the algorithm is organized as follows. A new multiplier digit is loaded in $ya$ at each step beginning from the least significant bit, placed in the 0-th row of $ya$. A new multiplicand is loaded in the 0-th row of $p1$ at 4 step intervals. (Hence, at the instant of $i$-th multiplicand loading, in the array $ya_y$ $i$-th multiplier is represented by the first four bits). $i$-th product as the two-row code $(C_i, S_i)$ is formed in the array $pa1$ as a result of the following three operations:

- one row shift of the multiplicand to the bottom,
- the generation of a partial product (PP),
- the reduction of integer triple (the partial products or (and) the result of their addition).

Figure 4 gives the graphical representation of the process of computing in the array $pa1$. Let us consider the process of the first product calculation.

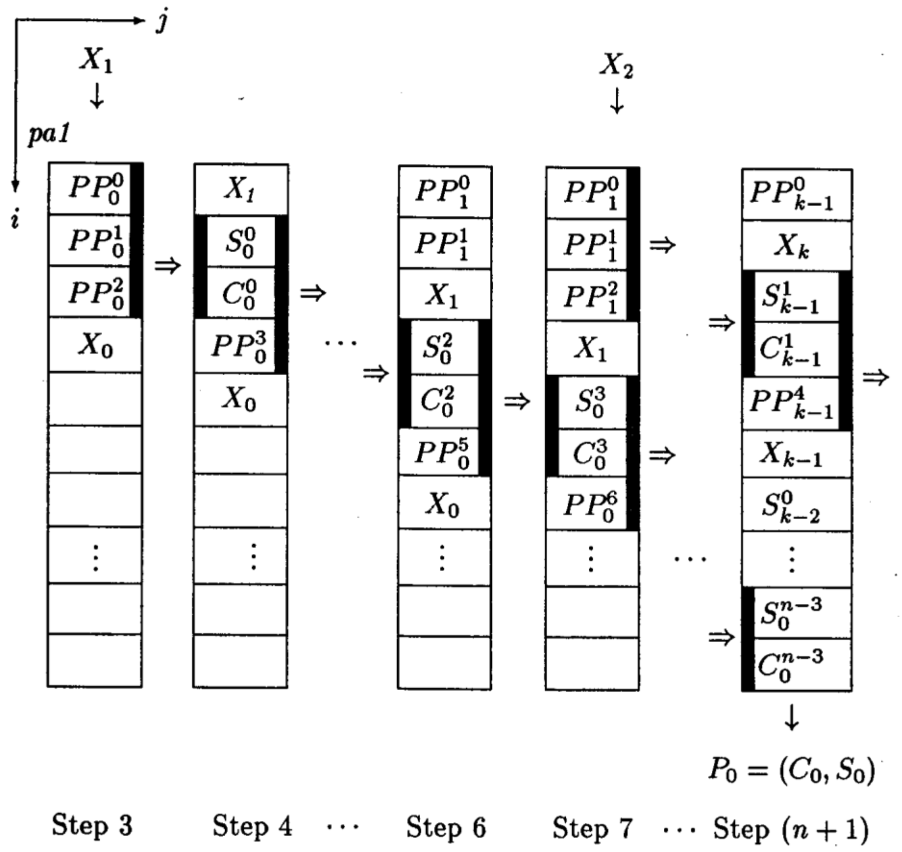

**Figure 4**

At the first three steps, the algorithm shifts the multiplicand and generates the partial products. As you can see from Figure 4, the result of the 3-rd step includes three partial products $PP_0^0$, $PP_0^1$, $PP_0^2$, and the multiplicand $X_0$ accommodated in the first four rows. At the 4-th step, this quadruple is processed into a new quadruple by three operations (multiplicand shift, $PP$ generation, and reduction) carried out in parallel. The new quadruple is shifted one row to the bottom and contains the following data: the two-row code $(C_0^0, S_0^0)$, the partial product $PP_0^3$ and the multiplicand $X_0$. As a result, the 0-th row leaves the process of calculation of the first product and is ready to take the new multiplicand, thus providing the deep pipelining of the basic algorithm.

So, the computation front propagates one row per step to the bottom of the array $pal$. At $(n-1)$-th step $X_0$ achieves the last row and after two steps the first product $P_0 = (C_0^{n-3}, S_0^{n-3}) = (C_0, S_0)$ is calculated in two latter rows.

At the 4-th step, the multiplicand $X_1$ is loaded in the 0-th row and beginning the 5-th step the basic algorithm forms two products in parallel. The second product is obtained in 4 step intervals and so on.

### 2.2. Time complexity

So, the algorithm calculates the first product in the form of two-row code at $(n+1)$-th step, the second product – in 4 step intervals, i. e., at $(n+5)$-th step. Computing the sum of $(C_1, S_1)$ and $(C_0, S_0)$, placed in the array *pa*2, takes 3 steps. After one step, i. e., at $(n+9)$ step, two-row code of the third product is ready in the array *pa*1. Hence, the code of the sum (3) is obtained in the array *pa*2 at $((n+1)+4(m-1)+3)$-th step. The final summation of two integers using a fast carry-look-ahead adder requires $(2+\log_2 n)$ steps.

Thus, this algorithm has the following time complexity:

- *the latency* $T_c = n + 1$ (a time to calculate the first product);
- *the period* $T_p = 4$ (a time between two successive calculations of products), which equals to the multiplicand loading rate;
- *the execution time* $T_e = n + \log_2 n + 4m + 2$ (a total time to generate the result).

As you can see, the basic algorithm has a very short period. It is achieved, firstly, by pipelining at both the initial data and the computation process levels, secondly, by loading of the initial data, transformation of the intermediate results and their moving in parallel, and, third, by using a fast carry-cave technique. This algorithm calculates the product of two $n$-bit integers in time $(n+\log_2 n+3)$. It is faster than that of high-speed sequential algorithms of the Karatzuba and the Schonhage–Strassen $(O(n^{\log_2 3})$ and $O(n \log_2 n \log\log_2 n)$, respectively).

## 3. Cellular algorithm architecture for long integers multiplication

In this section, we present the new cellular algorithm architecture for multiplication of long integers. The time complexity is obtained. At first, we describe the idea of the new algorithm.

### 3.1. New algorithm idea

The algorithm calculates the product of $n$-bit integers $X$ and $Y$ (see Figure 1) as

$$P' = X'Y',$$

where $X'$ and $Y'$ – the new multipliers, whose components are defined as follows:

$$x'_{j'} = x_{j-n/4}, \quad y'_{j'} = y_{j-n/4}, \quad j = 0, 1, \ldots, n-1.$$

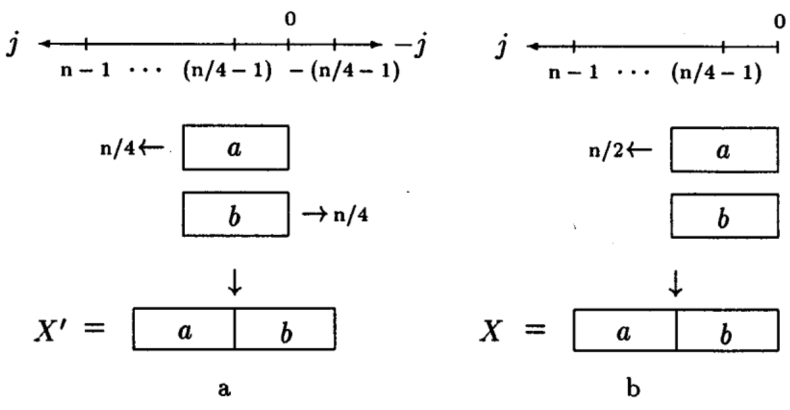

**Figure 5**

Graphically $X'$ and $Y'$ represent the integers $X$ and $Y$ shifted $n/4$ bits to the right along the axis $j$.

The substitution of the initial multipliers for the shifted multipliers is associated with the desire to speed-up the multiplication. Let us return to the sum (1). As mentioned in the introduction, for its calculation the base algorithm with a period in four steps is used. Hence, the speed-up in calculation (1) can be achieved due to

- the reduction of the shifted products number,
- the reduction of the number of shifts.

To reduce the shifted products number and the number of shifts, the values of the degrees of integer 2 in the sum (1) should be reduced to minimum. It means that the integers $a$ and $b$, $c$, and $d$ in the binary representation of multipliers should have the degrees of integer 2 with the opposite signs. From this follows the rule for the construction of new multipliers: the integers $a$ and $b$, $c$, and $d$ should be shifted to the opposite sides with respect to "0" (Figure 5a), unlike to the original multipliers $X$ and $Y$ (Figure 5b). Moreover, the shift should be done to equal numbers of bits and so that the obtained multipliers $X'$ and $Y'$ might be $n$-bit long. That is the way the integers $a$ and $b$, $c$, and $d$ are shifted $n/4$ bit to the opposite sides. (In Figure 5 only one multiplier is shown.)

Taking into account that $X' = X2^{-n/4}$ and $Y' = Y2^{-n/4}$

$$P' = X'Y' = X2^{-n/4}Y2^{-n/4} = XY2^{-n/2} = P2^{-n/2},$$

i. e., the product $P'$ is equal to product $P$, shifted $n/2$ bits to the right. So, the new algorithm forms product $P'$ as the sum of four products

$$P' = P2^{-n/2} = ac2^{n/2} + bd2^{-n/2} + ad + bc. \tag{4}$$

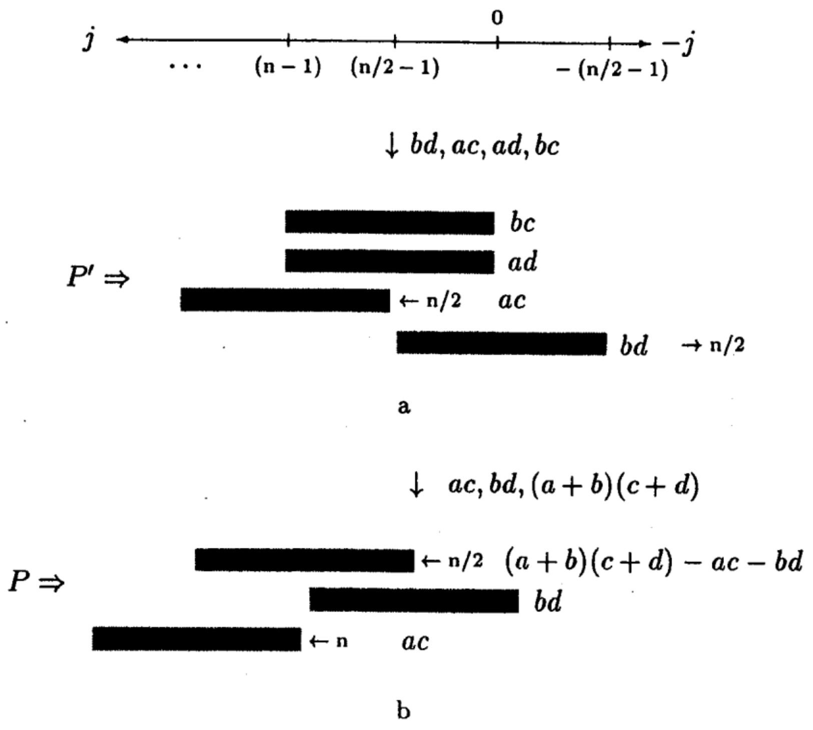

**Figure 6**

Graphically scheme (4) is given in Figure 6a. As a comparison, Figure 6b shows scheme (2) (the Karatzuba algorithm). Below we list the distinguishing features of the new algorithm.

- The products $ad$ and $bc$ are not shifted at all. The products $ac$ and $bd$ are shifted only $n/2$ bits to the opposite sides.
- The algorithm computes four products. The abandonment of the scheme, where the third (last) product is the product of two sums of $n/2$-bit integers (see Figure 6b) is motivates by a very short multiplicand loading rate. As we can see from Figure 4 to begin multiplying the 3-rd pair, the sums $(a+b)$ and $(c+d)$ must be already calculated which is be done not more than in 7 steps. Even if we use a fast CLA the addition can be done in time only for $n \leq 16$. Otherwise, the loading rate must be reduced. Moreover, the computing of two sums in the array $pa3$ makes the multiplier loading more difficult.

## 3.2. New algorithm architecture

Let $X$ and $Y$ be $n$-bits multipliers. It is required to calculate their product in an array of the size $n' \times (2n' - 1)$, where $n' = n/2$. The multiplication

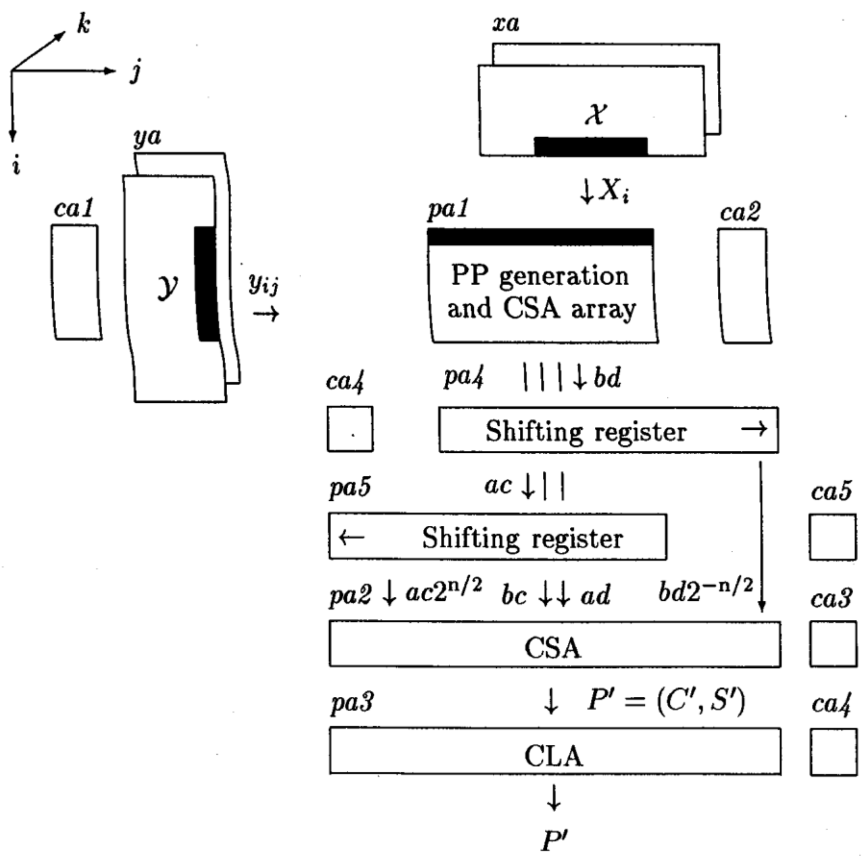

**Figure 7**

algorithm architecture is given in Figure 7. Its difference from the basic algorithm is as follows.

- Presence of two additional arrays *pa*4 and *pa*5 of the size $2 \times (1.5n - 1)$ carrying out shift of the products *ac* and *bd* $n/2$ bits to opposite side under the control of the arrays *ca*4 and *ca*5.
- The arrays *xa* and *ya* of the sizes $(4 \times n \times 2)$ and $(n \times 4 \times 2)$, respectively. The 0-th layer of each array is the processing one, the 1-st layer is the controlling one. In the initial state the arrays *xa* and *ya* store two copies of the multipliers $X$ and $Y$ respectively which then are shaped into initial data $\mathcal{X}$ and $\mathcal{Y}$ for computing the product $P'$ according to (4). The shaping $X, X \Rightarrow \mathcal{X}$ and $Y, Y \Rightarrow \mathcal{Y}$ are described below.

*Shaping* $X, X \Rightarrow \mathcal{X}$ *and* $Y, Y \Rightarrow \mathcal{Y}$. According to the accepted order of calculation of the products (*bd*, *ac*, *bc* and *ad*) forming the product $P'$ (see Figure 6a), the arrays $\mathcal{X}$ and $\mathcal{Y}$ should take the following form $\mathcal{X} \doteq$

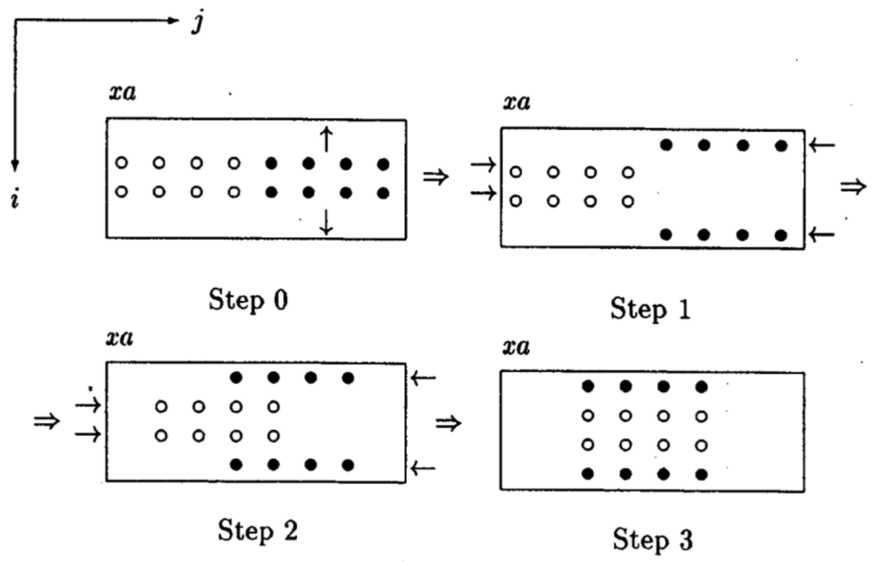

**Figure 8**

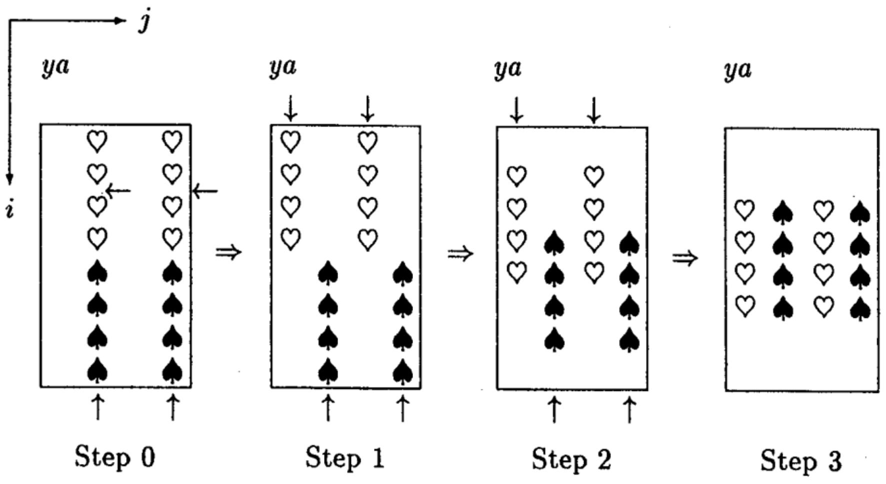

**Figure 9**

$(b, a, a, b)$ and $\mathcal{Y} = (d, c, d, c)$ respectively. The shaping schemes for 8-bit multipliers are given in Figures 8, 9 (the controlling layers are not shown). Here the symbols •, ∘, ♠ and ♡ stand for bits in the integers $a$, $b$, $c$, and $d$. The step 0 in Figures 8, 9 shows the arrays $xa$ and $ya$ in the initial states.

The shaping procedure $X, X \Rightarrow \mathcal{X}$ and $Y, Y \Rightarrow \mathcal{Y}$ is carried out under the control of the first layers of the arrays $xa$ and $ya$ and consists of the following two steps.

1. Splitting the multipliers into two parts. It is carried out by a simple shifting of the integers $a$ and $b$, $c$, and $d$ relative to each other (Step 1).

2. "Alignment" of the integers $a$, $b$, $c$, and $d$. The alignment is realized by shifting the integers during $n/4$ steps as follows: the integers $a$ and $d$ are shifted in the direction of the low-order bits, the integers $b$ and $c$ are shifted in the direction of the high-order bits.

It is obvious that the shaping procedure of the initial multipliers plus the loading of the integer $d$ in the 0-th row of the array $p1$ require $(n/4+2)$ steps.

## 3.3. Time complexity

From the calculation scheme of the product $P'$ (see Figure 6a) and the time complexity of the basic algorithm it follows that the sum of the products $bd2^{-n/2}$, $ad$, and $bc$ is already generated by the time of complecting the calculation of $ac2^{n/2}$. Hence, the time complexity of the new cellular algorithm for long integers multiplication is defined by the following sum

$$t_{\mathrm{e}} = t_{\mathrm{load}} + t_{ac2^{n/2}} + t_{\mathrm{CSA}} + t_{\mathrm{CLA}}.$$

Here $t_{\mathrm{load}}$ – the time needed to generate the arrays $\mathcal{X}$, $\mathcal{Y}$ and load the first multiplicand in the array $pa1$, it takes $(n/4+2)$ steps; $t_{ac2^{n/2}}$ – the time that the algorithm requires for computing the two-row code of the product $ac$ ($(n/2+5)$ steps), loading in the shifting register (2 steps) and shift $n/2$ steps to the right, hence, $(n+6)$ steps; $t_{\mathrm{CSA}}$ – the time needed to calculate the sum of $m$ products in the form of the two-row codes in the array $pa2$ (3 steps); $t_{\mathrm{CLA}}$ – the time needed to transfer the result from CSA into CLA (2 steps) and to sum two last integers ($\log_2 n$) steps).

As a result, our algorithm calculates the product of two $n$-bits integers in the array of restricted size in time $(1.25n+14+\log_2 n)$. The parallel version of the Karatzuba algorithm requires $(1.75n + 2\log_2 n + 12)$ steps. Thus, the time complexity of the new algorithm is less than that of the parallel version of the Karatzuba algorithm but it is more than the time required by the basic algorithm to multiply two $n$-bit integers in the array without size restriction ($(n + \log_2 n + 3)$ steps). In other words, the speed-up obtained due to reduction of the operand length by half is less than the time needed to realize the composition (shaping and shift) of four $n/2$ bit integer products. Indeed, the new algorithm computes four $n/2$ bit integer products is $\sim 0.5n$ steps, i. e., we obtain the speed-up of about 2, and the time complexity of its composition is $\sim 0.75n$ steps.

The inverse impact of "divide and conquer" technique relatively to the Karatzuba algorithm is explained as follows. Firstly, the initial algorithms with respect to which the speed-up are calculated, i. e., the traditional (sequential) algorithm and the cellular (parallel) algorithm have the different time complexities $O(n^2)$ and $(O(n)$, respectively. Secondly, in the Karaztuba algorithm, the time complexity of the composition is taken into account the complexity of shifts only.

## 4. Conclusion

In this paper, we present the new cellular algorithm architecture for multiplication of two long $n$-bit binary integers in the arrays of restricted size. The time complexity of this algorithm is $(1.25n + \log_2 n + 14)$ steps. It is less than the complexity of the parallel version of the Karatzuba algorithm. The speed-up is achieved due to the following.

- The algorithm forms the product of $n$-bit integers as $XY2^n$. It reduces the number of shifts by half.
- The algorithm computes four products. The increase in the number of products is resulted from this scheme is better adapted to the basic algorithm than the scheme with three products. There is no multiplier which is the sum of $n/2$-bit integers and hence the multiplicand loading rate is constant.

However it is necessary to note, that "divide and conquer" technique does not allow to reduce the time complexity of the cellular algorithm similarly in the sequential multiplication algorithm, and can be used only in the design of the arrays of restricted size.

## References

[1] Al.Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1996.

[2] S.M. Achasova, O.L. Bandman, V.P. Markova, and S.V. Piskunov, *Parallel Substitution Algorithm. Theory and Application*, World Scientific, 1994.

[3] V. Markova, S. Piskunov, and Yu. Pogudin, *Formal methods and tools for design of cellular algorithms and architectures*, Programmirovanie, No. 4, 1996, 24–36.

[4] C. Wallace, *A suggestion for a Faret multipliers*, IEEE Electron. Comput., **EC-34**, No. 3, 1961, 114–119.

[5] T.F. Ngai, M.J. Irvin, and S. Rawat, *Regular, area-time efficient carry lookahead adders*, J. Parallel and Distrib. Comput., **3**, No. 1, 1986, 92–105.