Bull. Nov. Comp. Center, Comp. Science, 32 (2011), 77–92

Memory organization with parallel access to information and its application for image processing

Alina Glodovski, Feodor Murzin, Tatiana Murzina

Abstract. In this paper, a computer memory system intended for storing an arbitrary sequence of multidimensional arrays is described. This memory system permits parallel access to the cuts distinguished in a given array by fixing one of the coordinates and to a large set of parallelepipeds which are the same dimension subarrays of the given arrays. A model of an automatic system for analysis of dynamical images containing multiple moving objects is also investigated. In the main, a set of point objects is considered. The peculiarity of the proposed system is the use of parallelism at all stages: information perceiving, storing, and processing.
Key words: *Memory organization, parallel access to information, image processing, objects tracking*

1. Introduction

Many different memory systems are used in computer design. Some of them allow parallel access to information [1, 2] but, as a rule, these memory systems are highly specialized and provide users with limited possibilities. Therefore, even the newest and very high speed computers use memory systems that provide sequential access to information or memory systems admitting only a small degree of parallelism.

In this paper, a generalization of some approaches to memory organization used earlier [3–5] is considered. As a result, a memory system is briefly described for storing an arbitrary sequence of multidimensional arrays. Certainly, the total size of arrays should not exceed the whole memory size. In this memory system, it is possible to implement parallel access to cuts separated in a given array by fixing one of the coordinates and to a large set of parallelepipeds, which are subarrays of the same dimension [6].

Here we consider arrays rather than other data types because in this case the problems being arisen have the purest (refined) form, which facilitates their decision. Besides, this data type plays the main role in creating computers oriented to numerical computations and image processing.

A model of an automatic system for analysis of dynamical images containing multiple moving objects is also investigated. Thereby our investigations are based on the works [7–9]. The main functions of the considered system are: transformation of a luminous flux into a two-dimensional matrix of signals intended for further processing; discovery of objects, estimation

of their coordinates, directions and velocities relative to the coordinate system of the gauge; tracking of objects in the feedback regime; and output of data in a form convenient for a user. The peculiarity of the proposed system is the use of the parallel memory. Parallelism is used at all stages: information perceiving, storing, and processing.

2. General principles of parallel access memory organization

Let $n_1, \ldots, n_k$ be positive integers. A set

$$A(n_1, \ldots, n_k) = \{\langle i_1, \ldots, n_k\rangle : \wedge_{j=1}^{k}(0 \leq i_j < n_j)\}$$

is called an array. This non-standard usage of the term "array" will be useful further. In fact, in computer science, a function of the form

$$A^* : A(n_1, \ldots, n_k) \to W,$$

is usually called an array, where W is a set of any type (integer, real, complex, ...) or a set of words in some alphabet. We shall write below A instead of $A(n_1, \ldots, n_k)$, omitting brackets together with their content. An arbitrary set $S \subseteq A$ will be called a segment.

Suppose we have N memory modules. A module will be implied as a chip of semiconductor memory or magnetic bubble memory. Such a module has an address line, a data line and a read/write signal line. The memory capacity of all modules is supposed to be the same and equal to K. An array $P = A(N, K)$ will be called a memory space. Without loss of generality, we can suppose that only one bit of information is stored into each address.

Let $A_1, \ldots, A_s$ be a sequence of multidimensional arrays, respectively, and $\mathcal{G}_1, \ldots, \mathcal{G}_s$ be sets of segments in $A_1, \ldots, A_s$. The functions $f_1, \ldots, f_s$ satisfy the following conditions:

$dom f_i = A_i$,

$range f_i \subseteq P$,

$i \neq j \to range f_i \cap range f_j = \emptyset$.

Here *dom* is the domain and *range* is the range of values. So the sequence $\langle f_1, \ldots, f_s\rangle$ determines the distribution of arrays $A_1, \ldots, A_s$ in the memory P. Suppose also that f_i is injective, i. e. for every $\overline{a}, \overline{b} \in dom f_i$ if $\overline{a} \neq \overline{b}$, then $f_i(\overline{a}) \neq f_i(\overline{b})$.

Definition

A sequence $\langle f_1, \ldots, f_s\rangle$ is called universal relative to $\langle \mathcal{G}_1, \ldots, \mathcal{G}_s\rangle$ if and only if $\forall i \forall S \in \mathcal{G}_i \forall \overline{s}_0, \overline{s}_1 \in S(\overline{s}_0 \neq \overline{s}_1 \to pr_1 f_i(\overline{s}_0) \neq pr_1 f_i(\overline{s}_1))$, where pr_1 is a projection of the pair on the first coordinate.

Let us consider an example. Suppose that an array (a_{ij}),

$$(0 \le i < K, 0 \le j < N)$$

is stored in the memory in natural order, i. e. an element a_{ij} is placed in j-th module on i-th address. In this case it is possible to access the line $i = Const$ for one clock period by placing the same address $\alpha = i$ into all modules simultaneously. Extraction of the column $j = Const$ is a more complex task. It takes time proportional to K. A similar problem appears in the case when a three-dimensional array is placed in the memory with the help of standard methods and we are trying to access two-dimensional cuts or small three-dimensional subarrays.

Universality of $\langle f_1, \ldots, f_s \rangle$ relative to $\langle \mathcal{G}_1, \ldots, \mathcal{G}_s \rangle$ means every segment $S \in \mathcal{G}_i (1 \le i \le s)$ to be well distributed in the memory in the sense that all elements of the segment are stored in different modules. It is exactly what permits reading/writing the segment S as a whole simultaneously at any single instant of time. And hereby the addresses $\{pr_2 f_i(\overline{s}) : \overline{s} \in S\}$ are placed into the modules $\{pr_1 f_i(\overline{s}) : \overline{s} \in S\}$.

Of course, there may be no universal sequence relative to $\langle \mathcal{G}_1, \ldots, \mathcal{G}_s \rangle$. Then the sets $\mathcal{G}_i^t (1 \le t \le m)$ such that $\mathcal{G}_i^t \subseteq \mathcal{G}_i, \cup \mathcal{G}_i^t = \mathcal{G}i$ may be found and there does exist a universal sequence relative to every tuple $\langle \mathcal{G}_1^t, \ldots, \mathcal{G}_s^t \rangle$.

So we can conclude that a certain addressing method allows parallel access to certain segments only. It may be impossible to implement a parallel access to all the segments we are interested in. Therefore, it is useful sometimes to change an addressing method at the run time.

Note that the question of complexity of an algorithm for f_i computation is also important. A memory control system is outlined in Figure 1.

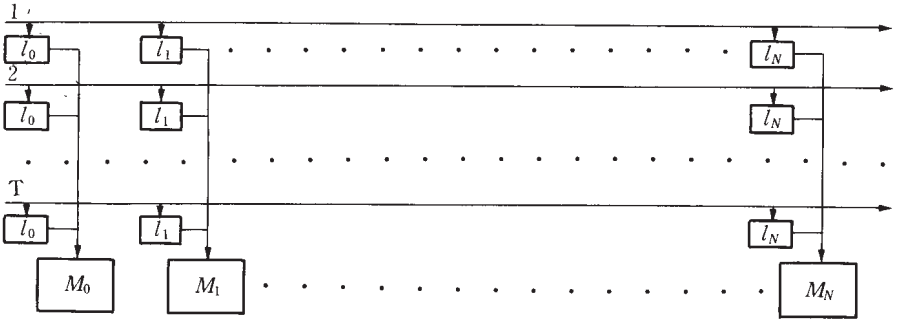

Figure 1. The scheme of the memory control system

Here M_i are memory modules and l_i are logical circuitries. The pairs $\{f_i(\overline{s}) : \overline{s} \in S\}$ are transferred through the horizontal lines. Hereby, if we want to take the elements of the segment S in a fixed order $\overline{s}_1, \ldots, \overline{s}_T$, then

it is necessary to input $f_i(\overline{s}_1)$ into the first line, $f_i(\overline{s}_2)$ into the second one and so on. These lines also include the read/write lines.

Assume that $f_i(\overline{s}_t) = \langle \nu_t, \alpha_t \rangle$. The logical circuitry l_m connects the address line and data line with a module when $m = \nu_t$. It means that ν_t is equal to the number of the module connected to a given logical circuitry. Otherwise connection does not occur.

So in any sense this control system runs as a telephone exchange station. Note that this control scheme may be easily integrated by means of the VLSI-technology.

In the papers another control scheme is used. Let S be a segment. Then, from universality of the sequence $\langle f_1, \ldots, f_s \rangle$, it follows that every module contains no more than one element from S. Let $\alpha^*[S](n)$ be equal to an address of an element from S in a module n, if such an element exists. If such elements do not exist, then $\alpha^*[S](n)$ is equal to some fixed address.

An access circuitry has two blocks. The first one forms addresses according to the function $\alpha^*[S]$. All addresses are put in simultaneously, each one into its own module. As an output, we obtain a segment. But its elements appeared to be rearranged just as a fixed natural order of elements may be desirable. The second block carries out necessary correction, which is also desirable to be done simultaneously. When writing, the correction is performed before writing and, when reading, the correction is performed after it. This scheme of the memory control is shown in Figure 2.

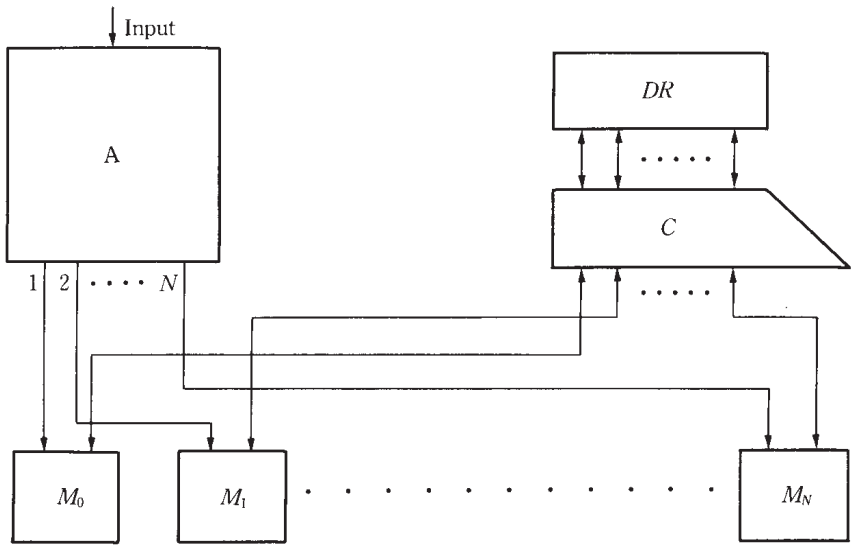

Figure 2. Controlling memory with a corrector

Here M_i are memory modules, A is an address circuitry, C is a corrector, DR is a data register. The name of the segment S and a read/write signal

are passing into the line Input. The addresses $\alpha^*[S](1), \ldots, \alpha^*[S](N)$ are passing through the lines $1, 2, \ldots, N$, respectively.

However, such a scheme seems to be less convenient for creating general-purpose computers, because for arrays of dimension ≥ 3 the correction algorithms are very complicated even for simple sets of segments needed in practice. But this scheme may be successfully used for processing two-dimensional arrays.

This paper considers the case when the memory and the system of processor elements are separated. In bibliography, one can find investigations of different systems (systolic multiprocessors, processors on the basis of cellular logic) in which the memory and processor elements are merged together. We do not consider this approach to be perspective for designing high performance computers, although it is useful for creating specialized chips.

The memory organization scheme considered above and the ideas of a cellular logic may be used together. Let the sequences $\langle f_1, \ldots, f_s \rangle$ and $\langle f'_1, \ldots, f'_s \rangle$, be universal relative to $\langle \mathcal{G}_1, \ldots, \mathcal{G}_s \rangle$ and $\langle \mathcal{G}'_1, \ldots, \mathcal{G}'_s \rangle$, respectively. Suppose $\sigma_i : P \to P, (1 \leq i \leq s)$ to be such that for every i there is a commutative diagram

$$\begin{array}{ccc} A_i & \xrightarrow{id} & A_i \\ f_i \downarrow & & \downarrow f'_i \\ P & \xrightarrow{\sigma_i} & P \end{array} .$$

Then the first distribution of arrays in a memory may be turned into the second one by execution of the operators σ_i with the help of cellular logic. In any sense, memory organized in this way resembles a liquid crystal, which may be in different states. External effects may turn it from one state into another. Each state allows access to fixed information to be realized. Theoretically, we can consider the dynamic case when the states change continuously. And information needed is extracted in one or several steps during this process, though the question of practical realization of such a memory seems to be very complicated.

3. Some permutations and their properties

Let a, b, n be integers, and $a, b \geq 0, n > 0$. Let us introduce some notations:
a/n is a quotient resulting from integer division of a by n,
$a//n$ is a remainder resulting from integer division of a by n,
$a \oplus b(modn) = (a + b)//n$ is a sum of a and b modulo n,
$[a]_n = n \cdot (a/n) = a - (a//n)$.
If $1 \leq j \leq k$ and $\langle i_1, \ldots, i_k \rangle$ is an arbitrary sequence, then
$pr_j(i_1, \ldots, i_k) = i_j$.
Let $A = A(n_1, \ldots, n_k)$ be an array, $k \geq 2$. Define segments

$P_j(t) = pr_j^{-1}(t) \cap A = \{\langle i_1, \ldots, i_k\rangle \in A : i_j = t\}$,

$BL[r_1, \ldots, r_k](i_1^0, \ldots, i_k^0) = \{\langle i_1, \ldots, i_k\rangle \in A : \wedge_{j=1}^k (i_j^0 \le i_j < i_j^0 + r_j)\}$.

For every j, t the first segment is a $(k-1)$-dimensional cut separated by the condition $i_j = t$. The second segment is a k-dimensional parallelepiped of $r_1 \times \ldots \times r_k$ size. The point $\langle i_1^0, \ldots, i_k^0\rangle$ is its apex having the least coordinates.

In further considerations we will assume that the condition

$C0$. $n_1 \le n_2, \ldots, n_k$ is valid.

Define a function $\lambda : A \to A$ by equalities

$\lambda(i_1, \ldots, i_k) = \langle \lambda_1(i_1, \ldots, i_k), \ldots, \lambda_k(i_1, \ldots, i_k)\rangle$,

$\lambda_1(i_1, \ldots, i_k) = i_1$,

$\lambda_j(i_1, \ldots, i_k) = i_j \oplus i_1 (mod n_j), 2 \le j \le k$.

It is evident that the function λ is a permutation on A.

We will define the next function in the case when, besides C0, the following conditions are also valid.

$C1$. $\wedge_{j=1}^k (n_j // r_j) = 0$,

$C2$. $r_1 = \Pi_{j=2}^k (n_j / r_j)$.

Define a function $\rho : A \to A$ by equalities:

$\rho(i_1, \ldots, i_k) = \langle \rho_1(i_1, \ldots, i_k), \ldots, \rho_k(i_1, \ldots, i_k)\rangle$,

$\rho_1(i_1, \ldots, i_k) = i_1$,

$\rho_j(i_1, \ldots, i_k) = i_j \oplus \mu_j(i_1) (mod n_j), 2 \le j \le k$,

$\mu_j(i) = r_j(i / \nu_{j+1}), 2 \le j < k$,

$\mu_k(i) = r_k(i // \nu_k)$,

$\nu_j = \Pi_{t=j}^k n'_t, n'_t = (n_t / r_t)$.

It is easy to see that ρ is also a permutation on A.

We will define the following function $\sigma : A \to A$ for the case when the conditions $C0 - C2$ are valid.

Let

$\sigma(i_1, \ldots, i_k) = \langle \sigma_1(i_1, \ldots, i_k), \ldots \sigma_k(i_1, \ldots, i_k)\rangle$,

$\sigma_1(i_1, \ldots, i_k) = i_1$,

$\sigma_j(i_1, \ldots, i_k) = [\rho_j(i_1, \ldots, i_k)]_{r_j} + (i_j \oplus \tau_j(i_1) (mod r_j)), 2 \le j \le k$.

$\tau_j(i) = (i / \nu_j)\nu_{j+1} + i // nu_{j+1}, 2 \le j < k$,

$\tau_k(i) = i / \nu_k$.

Analogously, the function σ is a permutation on A.

4. Data addressing

Now suppose we have two injective functions

$\varepsilon_0 : A(n_2, \ldots, n_k) \to \{i : 0 \le i < N\}$,

$\varepsilon_1 : \{i : 0 \le i < n_1\} \to \{i : 0 \le i < K\}$.

They generate a function

$\varepsilon = \varepsilon_0 \times \varepsilon_1 : A(n_1, \ldots, n_k) \to P = A(N, K)$, which can be defined by the formula

$\varepsilon(i_1, i_2, \ldots, i_k) = \langle \varepsilon_0(i_2, \ldots, i_k), \varepsilon_1(i_1) \rangle$.

Let us introduce functions

$\lambda^*(i_1, \ldots, i_k) = \varepsilon\lambda^{-1}(i_1, \ldots, i_k)$,

$\rho^*(i_1, \ldots, i_k) = \varepsilon\rho^{-1}(i_1, \ldots, i_k)$,

$\sigma^*(i_1, \ldots, i_k) = \varepsilon\sigma^{-1}(i_1, \ldots, i_k)$.

We will consider below the following sets of segments:

$\mathcal{CUT}_j = \{P_j(t) : 0 \le t < n_j\}$,

$\mathcal{CUT} = \cup_{j=1}^{k} \mathcal{CUT}_j$,

$\mathcal{BL}(r_1, \ldots, r_k) = \{BL[r_1, \ldots, r_k](i_1^0, \ldots, i_k^0) : \wedge_{j=1}^{k}(0 \le i_j^0 \le n_j - r_j)\}$,

$\mathcal{RBL}(r_1, \ldots, r_k) =$

$= \{BL[r_1, \ldots, r_k](i_1^0, \ldots, i_k^0) :\in \mathcal{B}l(r_1, \ldots, r_k) : i_1^0 // n_k' = 0\}$.

The following theorems are valid.

Theorem 1.

The sequence $\langle \lambda^* \rangle$ is universal relative to $\langle \mathcal{CUT} \rangle$.

Theorem 2.

The sequence $\langle \rho^* \rangle$ is universal relative to

$\langle \mathcal{CUT}_1 \cup \mathcal{BL}(r_1, \ldots, r_k) \rangle$.

Theorem 3.

The sequence $\langle \sigma^* \rangle$ is universal relative to

$\langle \mathcal{CUT} \cup \mathcal{RBL}(r_1, \ldots, r_k) \rangle$.

The following theorem shows that if parallel access to a large number of segments is demanded, then the corresponding universal function may be not existing.

Theorem 4.

Assume there exist at least two j such that $r_j < n_j$, together with the conditions $C1$, $C2$ and the condition

$C3$. $\Pi_{j=2}^{k} n_j = N$.

Then there is no $f : A(n_1, \ldots, n_k) \to P$ such that $\langle f \rangle$ is universal relative to

$\langle \mathcal{CUT} \cup \mathcal{BL}(r_1, \ldots, r_k) \rangle$.

The proofs of all these theorems are given in [6].

5. Some examples

Let us now consider several methodical examples.

Example 1

Let $(a_{i,j}), (0 \le i, j < 4)$ be a two-dimensional array. Then $n_1 = n_2 = 4$ and $\lambda_1(i_1, i_2) = i_1, \lambda_2(i_1, i_2) = i_2 \oplus i_1 (mod 4)$. Suppose $\varepsilon_1, \varepsilon_2$ are identical functions. The following table shows the elements in the memory.

	M_0	M_1	M_2	M_3
$\alpha = 0$	a_{00}	a_{01}	a_{02}	a_{03}
$\alpha = 1$	a_{11}	a_{12}	a_{13}	a_{10}
$\alpha = 2$	a_{22}	a_{23}	a_{20}	a_{21}
$\alpha = 3$	a_{33}	a_{30}	a_{31}	a_{32}

It is easy to see that the elements of every row are stored in different modules. And the elements of every column are stored in different modules. We have $\lambda^*(i_1, i_2) = \langle \lambda_1^{-1}(i_1, i_2), \lambda_2^{-1}(i_1, i_2)\rangle = \langle i_1, i_2 \oplus (4 - i_1)(mod4)\rangle$. Suppose we use the memory control system outlined in Figure 1. For example, we want to read/write the first row $\langle a_{10}, a_{11}, a_{12}, a_{13}\rangle$ or the first column $\langle a_{01}, a_{11}, a_{21}, a_{31}\rangle$. Then in the first case the sequence
$\{\lambda^*(1,0), \lambda^*(1,1), \lambda^*(1,2), \lambda^*(1,3)\} = \{\langle 1,3\rangle, \langle 1,0\rangle, \langle 1,1\rangle, \langle 1,2\rangle\}$.
should be inputed in the horizontal lines. And in the second case we should input the sequence
$\{\lambda^*(0,1), \lambda^*(1,1), \lambda^*(2,1), \lambda^*(3,1)\} = \{\langle 0,1\rangle, \langle 1,0\rangle, \langle 2,3\rangle, \langle 3,2\rangle\}$.
These sequences may be generated and inputed in the lines in parallel.

Example 2

Let $(a_{ij}), (0 \le i, j, k < 3)$ be a three-dimensional array. Then
$n_1 = n_3 = n_3 = 3$, $\lambda_1(i_1, i_2, i_3) = i_1$,
$\lambda_2(i_1, i_2, i_3) = i_2 \oplus i_1(mod3)$,
$\lambda_3(i_1, i_2, i_3) = i_3 \oplus i_1(mod3)$.

The following diagrams show the standard distribution of the elements of an array and the distribution according to the permutation λ.

Let $\varepsilon_0(i_2, i_3) = 3i_2 + i_3$ and $\varepsilon_1(i_1) = id$ be an identical function. The following table shows the distribution of elements in the memory according to the function λ^*.

Distribution of elements according to λ^*

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
$\alpha = 0$	a_{000}	a_{001}	a_{002}	a_{010}	a_{011}	a_{012}	a_{020}	a_{021}	a_{022}
$\alpha = 1$	a_{111}	a_{112}	a_{110}	a_{121}	a_{122}	a_{120}	a_{101}	a_{102}	a_{100}
$\alpha = 2$	a_{222}	a_{220}	a_{221}	a_{202}	a_{200}	a_{201}	a_{212}	a_{210}	a_{211}

It is easy to see that the elements of every cut $i_j = Const$ are stored in different modules. The numbers of modules and the addresses needed for reading/writing an arbitrary fixed cut may be generated and inputted in parallel.

Example 3

Let $n_1 = n_2 = 16$, $r_1 = r_2 = 4$ and $(a_{ij}), (0 \le i, j < 16)$ be a two-dimensional array.

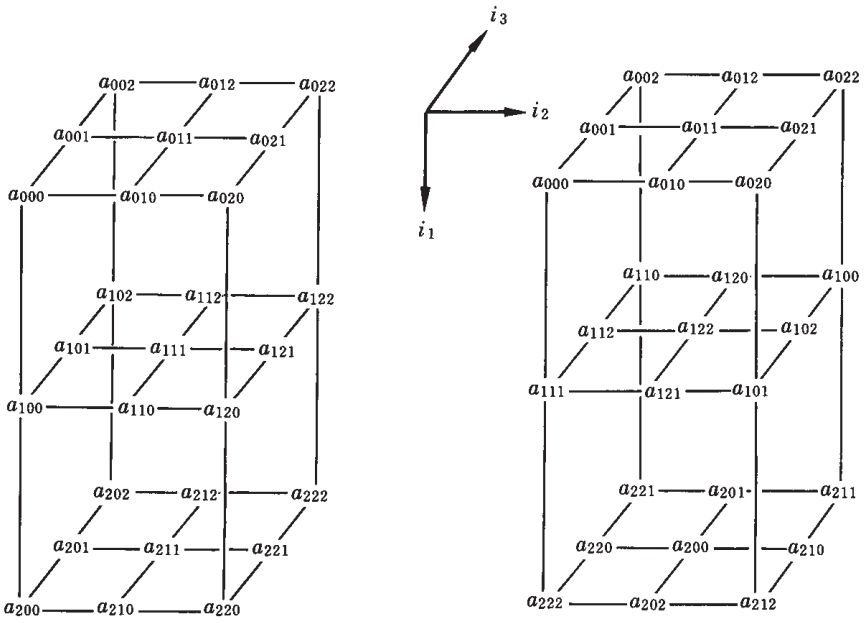

Figure 3. Distribution of the elements of a three-dimensional array

Then we have following formulas:
$\sigma_1(i_1, i_2) = i_1, \rho_2(i_1, i_2) = i_2 \oplus 4(i_1//4)(mod16)$,
$\sigma_2(i_1, i_2) = [\rho_2(i_1, i_2)]_4 + (i_2 \oplus (i_1/4)(mod4))$.
The conditions $C1 - C2$ are also valid because $16//4 = 0$ and $4 = 16//4$. Suppose $\varepsilon_1(i_1) = id$ are identical functions. The following table shows the distribution of elements in the memory according to the function σ^*.

In this case we have a possibility to perform parallel access to all rows and columns. Moreover, there is a parallel access to squares of size 4×4 from every layer
$4s \leq i_1 < 4(s+1)$, $(0 \leq s < 4)$. All is easy for the memory control system outlined in Figure 1.

Suppose we use the memory control system outlined in Figure 2. For example, correction needed for the rows is cyclical transfers of intervals of length 4 and cyclical transfers inside intervals. For high dimensions, the correction algorithms are complicated. It is often not clear how in these cases a correction may be done in a parallel mode. Even in the two-dimensional case the corrector hardware costs are essential.

6. Parallel system for analysis of images

A model of an automatic system for analysis of dynamical images containing multiple moving objects is investigated, in the main, a set of point objects

a_0^0	a_1^0	a_2^0	a_3^0	a_4^0	a_5^0	a_6^0	a_7^0	a_8^0	a_9^0	a_{10}^0	a_{11}^0	a_{12}^0	a_{13}^0	a_{14}^0	a_{15}^0
a_4^1	a_5^1	a_6^1	a_7^1	a_8^1	a_9^1	a_{10}^1	a_{11}^1	a_{12}^1	a_{13}^1	a_{14}^1	a_{15}^1	a_0^1	a_1^1	a_2^1	a_3^1
a_8^2	a_9^2	a_{10}^2	a_{11}^2	a_{12}^2	a_{13}^2	a_{14}^2	a_{15}^2	a_0^2	a_1^2	a_2^2	a_3^2	a_4^2	a_5^2	a_6^2	a_7^2
a_{12}^3	a_{13}^3	a_{14}^3	a_{15}^3	a_0^3	a_1^3	a_2^3	a_3^3	a_4^3	a_5^3	a_6^3	a_7^3	a_8^3	a_9^3	a_{10}^3	a_{11}^3
a_1^4	a_2^4	a_3^4	a_0^4	a_5^4	a_6^4	a_7^4	a_4^4	a_9^4	a_{10}^4	a_{11}^4	a_8^4	a_{13}^4	a_{14}^4	a_{15}^4	a_{12}^4
a_5^5	a_6^5	a_7^5	a_4^5	a_9^5	a_{10}^5	a_{11}^5	a_8^5	a_{13}^5	a_{14}^5	a_{15}^5	a_{12}^5	a_1^5	a_2^5	a_3^5	a_0^5
a_9^6	a_{10}^6	a_{11}^6	a_8^6	a_{13}^6	a_{14}^6	a_{15}^6	a_{12}^6	a_1^6	a_2^6	a_3^6	a_0^6	a_5^6	a_6^6	a_7^6	a_4^6
a_{13}^7	a_{14}^7	a_{15}^7	a_{12}^7	a_1^7	a_2^7	a_3^7	a_0^7	a_5^7	a_6^7	a_7^7	a_4^7	a_9^7	a_{10}^7	a_{11}^7	a_8^7
a_2^8	a_3^8	a_0^8	a_1^8	a_6^8	a_7^8	a_4^8	a_5^8	a_{10}^8	a_{11}^8	a_8^8	a_9^8	a_{14}^8	a_{15}^8	a_{12}^8	a_{13}^8
a_6^9	a_7^9	a_4^9	a_5^9	a_{10}^9	a_{11}^9	a_8^9	a_9^9	a_{14}^9	a_{15}^9	a_{12}^9	a_{13}^9	a_2^9	a_3^9	a_0^9	a_1^9
a_{10}^{10}	a_{11}^{10}	a_8^{10}	a_9^{10}	a_{14}^{10}	a_{15}^{10}	a_{12}^{10}	a_{13}^{10}	a_2^{10}	a_3^{10}	a_0^{10}	a_1^{10}	a_6^{10}	a_7^{10}	a_4^{10}	a_5^{10}
a_{14}^{11}	a_{15}^{11}	a_{12}^{11}	a_{13}^{11}	a_2^{11}	a_3^{11}	a_0^{11}	a_1^{11}	a_6^{11}	a_7^{11}	a_4^{11}	a_5^{11}	a_{10}^{11}	a_{11}^{11}	a_8^{11}	a_9^{11}
a_3^{12}	a_0^{12}	a_1^{12}	a_2^{12}	a_7^{12}	a_4^{12}	a_5^{12}	a_6^{12}	a_{11}^{12}	a_8^{12}	a_9^{12}	a_{10}^{12}	a_{15}^{12}	a_{12}^{12}	a_{13}^{12}	a_{14}^{12}
a_7^{13}	a_4^{13}	a_5^{13}	a_6^{13}	a_{11}^{13}	a_8^{13}	a_9^{13}	a_{10}^{13}	a_{15}^{13}	a_{12}^{13}	a_{13}^{13}	a_{14}^{13}	a_3^{13}	a_0^{13}	a_1^{13}	a_2^{13}
a_{11}^{14}	a_8^{14}	a_9^{14}	a_{10}^{14}	a_{15}^{14}	a_{12}^{14}	a_{13}^{14}	a_{14}^{14}	a_3^{14}	a_0^{14}	a_1^{14}	a_2^{14}	a_7^{14}	a_4^{14}	a_5^{14}	a_6^{14}
a_{15}^{15}	a_{12}^{15}	a_{13}^{15}	a_{14}^{15}	a_3^{15}	a_0^{15}	a_1^{15}	a_2^{15}	a_7^{15}	a_4^{15}	a_5^{15}	a_6^{15}	a_{11}^{15}	a_8^{15}	a_9^{15}	a_{10}^{15}

.

$$a_j^i \equiv a_{ij}$$

is considered. Some computer experiments are also presented.

The main functions of the considered system are as follows:

- transformation of a luminous flux into a two-dimensional matrix of signals intended for further processing;
- discovery of objects, estimation of their coordinates;
- directions and velocities relative to the coordinate system of the gauge;
- tracking of objects in the feedback regime; output of data in a form convenient for a user. The peculiarity of the proposed system is the use of parallelism at all stages: information perceiving, storing and processing. Everything is based on using the parallel memory described above.

The structural scheme of the system is presented below

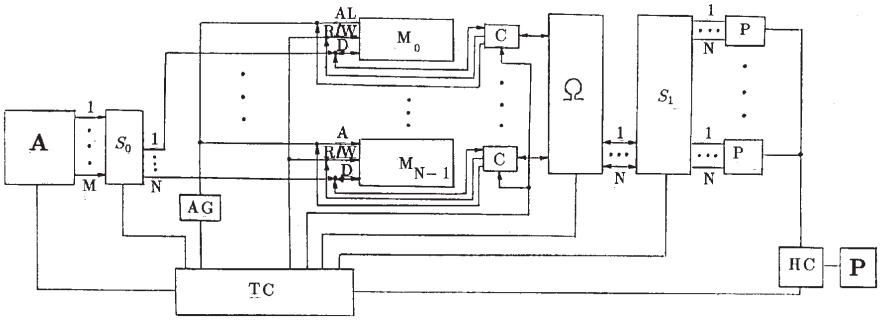

Figure 4. The structural scheme of the system

A is a photo-receiving matrix, S_0, S_1 are gate schemes, AG is a generator of addresses, M_i are memory modules, C are controllers, Ω is a commutator network,

P_i are processors, TC is a controller controlled by a clock-pulse generator, HC is a host-computer, P is peripheral equipment, D is a line of data, A is an address line, R/W is a read/write line.

The luminous signal is perceived by the photo receiving matrix processing as a charge coupling device. The dimension of the matrix $M \times M$ is quite big, for example $M = 1024$. The gate scheme S is placed on the same crystal. At present chips have not too much pins. Therefore, the first aim is to narrow the information flow. Suppose $N \ll M$ and M is divisible by N.

Moreover, S makes a permutation of data according to some algorithm. The scheme S is controlled by the controller TC, which is controlled by the clock-pulse generator. Simultaneously, TC activates the block AG generating the addresses.

It generates an address, which is the same for all modules, and forms the read/write signal.

Thus, information on some N points (pixels) is transferred simultaneously into N modules of memory and placed there in some nonstandard manner. Suppose that capacity of every memory module is $K \geq M^2/N$. It is sufficient to transfer all matrices of the size $M \times M$ after several steps. It is easy to see that a mapping of an array of size $M \times M$ into a rectangular array of size $K \times N$ appear.

Consider in more detail the properties of this mapping we need. Suppose in addition that N is a square, i. e. $N = n^2$. The mapping is constructed so that, for every square of size $n \times n$ from the initial array, the elements of this square will be placed in different memory modules. In general, they can be stored with different or same addresses. The second fact is not essential. The fact that the elements lie in different modules allows us to generate in parallel all necessary addresses and to transfer simultaneously all elements

of the square to one or several processors for execution. This means we use windows of size $n \times n$. Thereby every window is accessible.

Now we will describe the use of functions in more detail.

Define a function $\rho : A \to A$ by the equalities

$\rho(i,j) = \langle \rho_1(i,j), \rho_2(i,j) \rangle$,

$\rho_1(i,j) = i$,

$\rho_2(i,j) = j \oplus n^*(mod M), n^* = n \cdot (i//n)$.

Let us notice that the given function represents a special case of a function ρ considered earlier. Therefore we use the same symbol for its designation.

The inverse function $\rho^{-1} : A \to A$ has the form

$\rho^{-1}(i,j) = \langle \rho_1^{-1}(i,j), \rho_2^{-1}(i,j) \rangle$,

$\rho_1^{-1}(i,j) = i$,

$\rho_2^{-1}(i,j) = j \oplus \overline{n}(mod M), \overline{n} = M - n^*$.

Let us introduce a function $\varepsilon : A \to A$ by the equalities

$\varepsilon(i,j) = \langle \varepsilon_1(i,j), \varepsilon_2(i,j) \rangle$,

$\varepsilon_1(i,j) = iN' + j/N, N' = M/N$,

$\varepsilon_2(i,j) = j//N$.

Let us denote $\rho^*(i,j) = \varepsilon\rho^{-1}(i,j)$. It is easy to prove that ρ is an injective mapping from A to$A$. It is also obvious that

$\rho^*(i,j) = \langle \rho_1^*(i,j), \rho_2^*(i,j) \rangle$,

$\rho_k^*(i,j) = \varepsilon_k \rho_k^{-1}(i,j), (k = 1,2)$.

The set

$W(i^0, j^0) = \{\langle i,j \rangle \in A : i^0 \le i < i^0 + n, j^0 \le j < j^0 + n\}$

will be called a square window of size $n \times n$.

The set of all windows we will designate by

$\mathbf{W} = \{W(i^0, j^0) : 0 \le i^0, j^0 \le M - n\}$.

As usual, if f is a function, $domf$ is its domain, $Q \subseteq domf$, then $f \mid Q$ designates a restriction of f to the set Q.

Theorem 5.

For every $W \in \mathbf{W}$, the function $\rho_2^* \mid W$ is injective.

Here we omit the proof.

Now we consider a simple example for $N = 16$. Thereby every window of size 2×2 will be accessible in the parallel mode. We will also have a possibility to see how ρ^{-1} and ε work together.

The mapping described above can be executed directly on a photoreceiving matrix working on the principle of Charge Couple Device (CCD). For this purpose it is enough to have an additional ring register (see Figure 6) also working on the principle of CCD.

Each next line is read out in a working line which represents the bottom part of the ring register. Further, using the ring register, a cyclic shift of elements on a necessary number of positions is made. The result is placed in the line L. Another part of a ring is used as auxiliary.

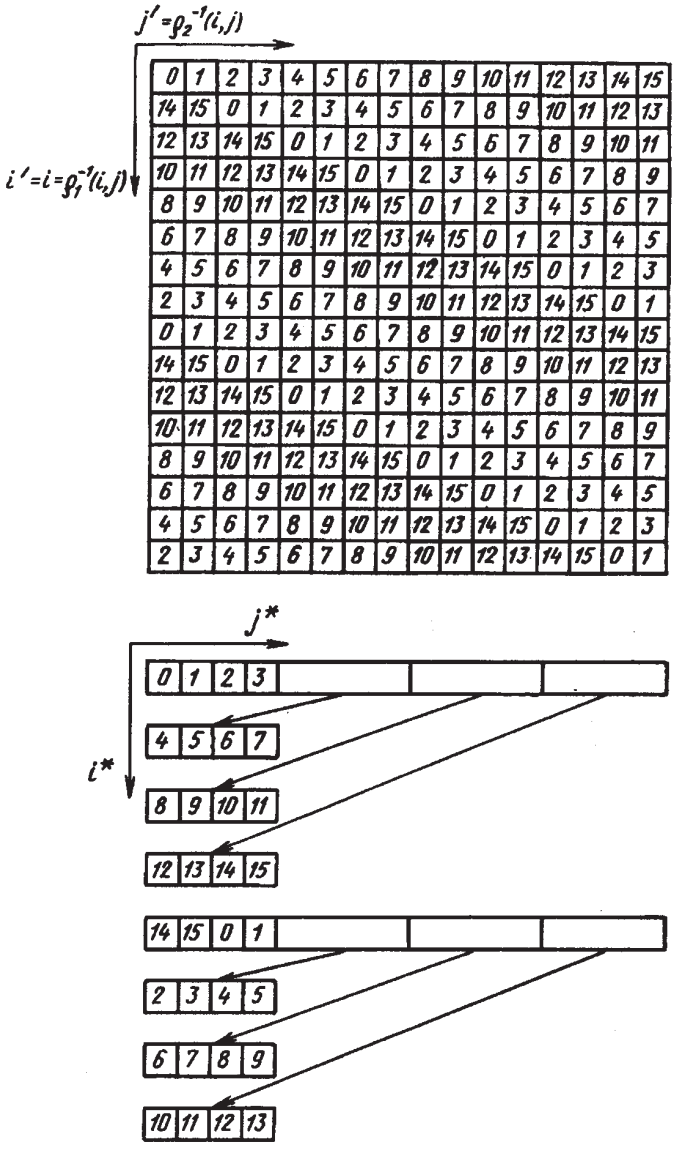

Figure 5. The function ρ^{-1} makes permutations inside lines;
the function ε cuts lines into pieces, then places them one under another

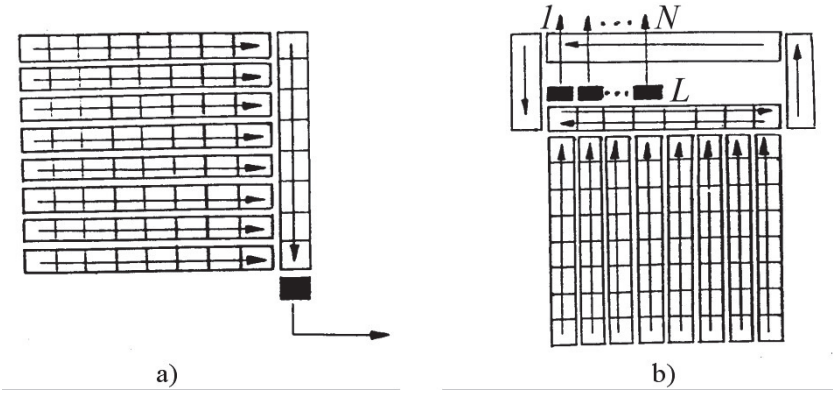

Figure 6. Schemes of data reading from photo-reception matrices:
a) the standard scheme;
b) the scheme with the ring register

Now suppose that there is a dynamical image containing multiple moving points. For every window in which the moving object appears, the system initiates an individual file in the memory of one of the processors. Further the system renovates the file contents. Processors perform simultaneously. First of all, this performance includes a forecast of the object movement direction. The forecast data are used by the system to perform access to the next window. Moreover, some results of every, processor may be transferred into the common line connected with the host-computer (it may be a personal computer). These results may be used for visualization of the situation. The visualization algorithms are implemented on the host-computer and serve for representation of information in a form for convenient a user. The movement directions may be shown by pointers. Velocities may be shown in the numerical form and so on. Usually every processor tracks its own object. When the number of moving points is greater than the number of processors, some processors (or even all) may track several objects. In this case, the reaction of the system is slowed down. The processors receive access to information with the help of the gate scheme S_1 and the commutator-network Ω. We suppose that, at the initial stage, information about objects location is received by processors from the discovery block. Further they work independently using comparison of information obtained at the previous and at the current stage.

7. Conclusion

Let us briefly consider the memory control scheme presented in Figure 1. This scheme provides us with the possibility of increasing both the number of memory modules and the number of address generators. The integrated circuitry implementation of this scheme should allow the access range and

capacity of the memory-processor channel to be essentially increased.

The detailed analysis of the memory control scheme shown in Figure 2 makes it clear that the hardware costs essentially increase with the growth of N and complication of the access mode in the case of hardware implementation of address calculation and corrector. However, we can distinguish a set of operations which are used for many interesting memory access modes: integer addition and subtraction, integer multiplication, displacements, and transfers.

In order to decrease the hardware costs and to increase the functional capabilities of the system, we propose to include a small special processor for address calculation for every memory module. The pipe-line mode of address processing, together with availability of long sequences of a single segment type, should allow us to combine high-speed characteristics of the hardware implementation with the possibility of working with segments of almost all usual types.

We can also say the following about the automatic system for dynamical image analysis: an additional special discovery block is necessary. The main problem of the discovery block is the search of new objects to transfer information about them into the tracking block. Certainly, if we deal with high velocities of objects and slow reaction of the system, it cannot track their movement. In these cases, it is possible to lose moving objects from the field of view.

References

[1] Colboum C.J., Heinrich K. Conflict-free access to parallel memories // J. of Parallel and Distributed Computing. 1992. – N 14. – P. 193–200.

[2] Das S.K., Finocchi I., Petreschi R. Conflict-free star-access in parallel memory systems // J. of Parallel and Distributed Computing. – 2006. – Vol. 66, Iss. 11. – P. 1431–1441.

[3] Van Voorhis D.C., Morrin T.H. Memory systems for image processing // IEEE Trans. on Computer. – 1978. – Vol. C-27, N 2. – P. 113–125.

[4] Van Voorhis D.C., Morrin T.H. Memory Systems for Image Processing. – Los Catos, CA, August, 1975. – (IBM Systems Communication Division / Working Paper 16 / A45).

[5] Van Voorhis D.C., Morrin T.H. United States Patent N3, 995, 253, November 30, 1976.

[6] Murzin F.A., Sluev V.A. A memory organization for parallel computers // New Generation Computing J. – 1988. – Vol. 6, N 1. – P. 3–18.

[7] Bratsev S.G., Murzin F.A., Nartov B.K., Puntus A.A. The conflict of complex systems. Models and control. – Moscow Aviation Inst., 1995. – ISBN 5-7035-0554-2. (In Russian)

[8] Bratsev S.G., Murzin F.A., Nartov B.K. A parallel automatic system for image processing // Computer Algebra and its Application in Mechanics. – 1992. – Nova Science Publishers, Inc. – P. 129–133.

[9] Bratsev S.G., Murzin F.A., Nartov B.K. Optimum targets search and dynamic image processing // Advances in Modeling & Analysis. – AMSE Press, France–Russia. – 1993. – Vol. 26, N 4. – P. 1–11.