http://doi.org/10.35784/iapgos.5615 *received: 27.11.2023 | revised: 29.02.2024 | accepted: 07.03.2024 | available online: 31.03.2024*

# ARCHITECTURAL AND STRUCTURAL AND FUNCTIONAL FEATURES OF THE ORGANIZATION OF PARALLEL-HIERARCHICAL MEMORY

**Leonid Timchenko[1], Natalia Kokriatska[1], Volodymyr Tverdomed[1], Iryna Yu. Yepifanova[2], Yurii Didenko[1], Dmytro Zhuk[1], Maksym Kozyr[1], Iryna Shakhina[3]**

[1]State University of Infrastructure and Technology, Kyiv, Ukraine, [2]Vinnytsia National Technical University, Vinnytsia, Ukraine, [3]Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University, Vinnytsia, Ukraine

***Abstract.*** *Parallel hierarchical memory (PI memory) is a new type of memory that is designed to improve the performance of parallel computing systems. PI memory is composed of two blocks: a mask RAM and a tail element RAM. The mask RAM stores the masks that are used to encode the information, while the tail element RAM stores the actual information. The address block of the PI memory is responsible for generating the physical addresses of the cells where the tail elements and their masks are stored. The address block also stores the field of addresses where the array was written and associates this field of addresses with the corresponding external address used to write the array. The proposed address block structure is able to efficiently generate the physical addresses of the cells where the tail elements and their masks are stored. The address block is also able to store the field of addresses where the array was written and associate this field of addresses with the corresponding external address used to write the array. The proposed address block structure has been implemented in a prototype PI memory. The prototype PI memory has been shown to be able to achieve significant performance improvements over traditional memory architectures. The paper will present a detailed description of the PI transformation algorithm, a description of the different modes of addressing organization that can be used in PI memory, an analysis of the efficiency of parallel-hierarchical memory structures, and a discussion of the challenges and future research directions in the field of PI memory.*

**Keywords**: parallel hierarchical memory, PI memory, address block, mask RAM, tail element RAM, performance improvement

## ARCHITEKTONICZNE, STRUKTURALNE I FUNKCJONALNE CECHY RÓWNOLEGŁO-HIERARCHICZNEJ ORGANIZACJI PAMIĘCI

***Streszczenie***. *Równoległa pamięć hierarchiczna (pamięć PI) jest nowym typem pamięci zaprojektowanym w celu poprawy wydajności równoległych systemów obliczeniowych. Pamięć PI składa się z dwóch bloków: maski RAM i ogon RAM. Maska RAM przechowuje maski używane do kodowania informacji, podczas gdy ogon RAM przechowuje rzeczywiste informacje. Blok adresowy pamięci PI jest odpowiedzialny za generowanie fizycznych adresów komórek, w których przechowywane są elementy końcowe i ich maski. Blok adresowy przechowuje również pole adresu, w którym tablica została zapisana i kojarzy to pole adresu z odpowiednim adresem zewnętrznym użytym do zapisu tablicy. Proponowana struktura bloku adresowego jest w stanie efektywnie generować fizyczne adresy komórek, w których przechowywane są elementy ogonowe i ich maski. Blok adresowy może również przechowywać pole adresu, w którym tablica została zapisana i powiązać to pole adresu z odpowiednim adresem zewnętrznym użytym do zapisu tablicy. Zaproponowana struktura bloku adresowego została zaimplementowana w prototypie pamięci PI. Wykazano, że prototyp pamięci PI jest w stanie znacznie poprawić wydajność w porównaniu z tradycyjnymi architekturami pamięci. W artykule zostanie przedstawiony szczegółowy opis algorytmu konwersji PI, opis różnych trybów adresowania, które mogą być używane w pamięci PI, analiza wydajności równoległo-hierarchicznych struktur pamięci oraz omówienie wyzwań i przyszłych kierunków badań w dziedzinie pamięci PI.*

**Słowa kluczowe**: równoległa pamięć hierarchiczna, pamięć PI, blok adresowy, maska RAM, ogon RAM, poprawa wydajności

## Introduction

In recent years, much effort has been made to create parallel computing systems that use the power of the Internet and the availability of computers in many homes and businesses. The main advantage of these approaches is that they provide a cheap environment for parallel computing for those who cannot afford the costs of supercomputers and hardware parallel processing. However, most of the solutions on offer are not very flexible in the use of available resources and are very difficult to install and configure [21]. When designing high-performance computers and systems, it is necessary to make many trade-offs, such as the sizes and technologies for each level of the hierarchy.

Analyzing previous work [9, 12, 14] on parallel-hierarchical structures, we can identify the main disadvantages of these systems [8, 10, 15].

For example, the lack of a method for evaluating peak performance as the number of instructions executed by the computer per unit of time (MIPS, Million Instruction Per Second) gives only a general idea of the speed, since it does not take into account the specifics of specific programs (it is difficult to predict how many and which instructions a user program will be displayed by the processor) [2, 5, 20].

Parallel computing systems are excessively expensive. According to the law of diminishing returns, the performance of a computer grows proportionally to the square of its cost; as a result, it is much more profitable to obtain the required computing power by purchasing one high-performance processor than by using several less powerful processors [4, 9, 10].

When organizing parallelism, performance losses grow too quickly. According to the Minsky hypothesis (Marvin Minsky), the acceleration of calculations achieved when using a parallel system is proportional to the binary logarithm of the number of processors (with 1000 processors, the possible acceleration is only 10).

Counter-argument. The given acceleration estimate is true for parallelizing certain algorithms. However, there are a large number of tasks, the parallel solution of which achieves close to 100% use of all available processors of a parallel computing system.

Sequential computers are constantly being improved. According to the well-known Moore's law, the complexity of sequential microprocessors doubles every 18 months, so the required performance can also be achieved on "ordinary" sequential computers.

However, the use of parallelism allows obtaining the necessary acceleration of calculations without waiting for the development of new faster processors. The efficiency of parallelism strongly depends on the characteristic properties of parallel systems. All modern sequential electronic computers operate in accordance with the classical von Neumann scheme; parallel systems differ significantly in architecture and the maximum effect from the use of parallelism can be obtained by full use of all the features of the hardware (consequence – the transfer of parallel algorithms and programs between different types of systems is difficult, and sometimes impossible).

For decades of operation of sequential electronic computers, a huge amount of software has been accumulated, oriented at sequential electronic computers; its processing for parallel computers is practically unrealistic.

There is a limit on the acceleration of calculation in the parallel – hierarchical implementation of the algorithm in comparison with the sequential one.

Parallel-hierarchical memory (PHM) is a promising approach to addressing the challenges of memory organization in parallel computing systems. PHM systems are organized in a hierarchy of levels, with each level providing different levels of performance and capacity. This allows the system to adapt to the needs of different applications. In this paper, we present an overview of the architectural and structural and functional features of the organization of PHM systems.

## 1. Examples of network parallel-hierarchical structures

The network structure allows to simulate the principle of operation of a distributed PI network and, due to spatial separation over time, processes information in a deterministic pyramidal PI network (Fig. 1). The network, built on the basis of PI transformation, consists of a set of subnetworks (Fig. 1) for forming features about the states of the spatio-temporal environment (PTE), the structure of which is homogeneous and consists of a number of interdependent hierarchical levels.

The network operation algorithm is universal and consists in the PI formation of sets of common and different states of the PTE. Generalization of all types of sensory information occurs at the very final stage of the transformation outside the hierarchical processing of each type of sensory information. So, the process of generalization between different types of sensory information begins only when the construction according to a certain set of features is completed [9].

On the figure 1 there is an example of transformed states on each level: ☐ – set of shared states on each level, ☒ – define common state on separate level.

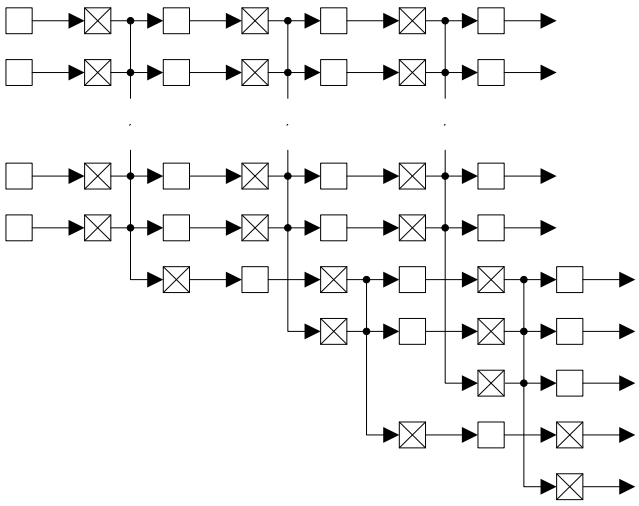


*Fig. 1. Model of neural structur based on the PI transformation block*

To associate the masks with their corresponding branches at each level, after their sequential writing starting from the lower branches, two additional bytes of information need to be written. This information includes the number of masks generated at that level and the highest branch number from which the last mask is selected at that level. This information is also generated in the PI transformation block.

On the figure 2 you see the structure of a parallel hierarchical memory (PI memory). The core of the structural organization of PI memory consists of three main components: the storage medium (mask RAM, tail element RAM), the PI transformation network, and the address block. The storage medium is composed of two blocks, each of which is a conventional RAM with a sequential data access structure. The word size of the mask RAM corresponds to the memory's access width, while the word size of the tail element RAM matches the bit depth of the image frame [1, 5].

The PI transformation network is responsible for encoding the information during the write operation and decoding it during the read operation from the memory.

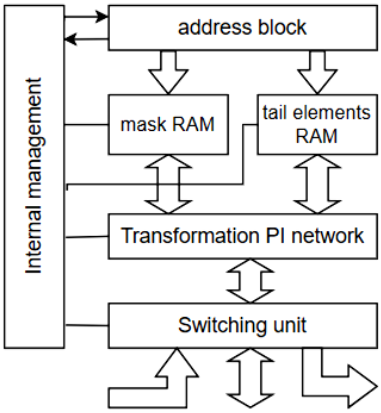



*Fig. 2. Structure of PI memory with PI transforming network*

## 2. Organization of the architecture of the address block parallel to the hierarchical memory

The address block of the PI memory implements the functions of generating internal addresses for the storage medium blocks based on the external address and the state of the PI transformation network.

In a linear sequential memory, the memory capacity ($N$), which represents the number of addressable data elements (cells), plays a crucial role in determining the address size. The dependency $H_{addr} = INT(log_2 N + 0.5)$, is based on the $H_{addr}$ – number of binary digits in the address code, $INT$ – represents the integer part operation. The address structure may contain additional fields used for control information, such as specifying base or index register numbers, address modification indicators, etc. Thus, the description of accessing conventional memory typically has a two-component structure $[C, adr]$, including $C$ – control code and $adr$ – address code [6, 7].

For the considered PI memory, the structure of the access description follows the conventional form and includes the address code. Unlike the cell address structure, here the address of the array or array package is used instead of the cell address. The control code in the address structure incorporates information about the memory operation mode: a) standard mode with the ability to access any cell, and b) array mode [3, 18].

Let's examine the structure of the address block (Fig. 3) in the PI memory that stores information arrays. The tasks of the address block are as follows:

1) During array write operation, generate physical addresses of the cells where the tail elements and their masks are stored.
2) After completing the write operation, store the field of addresses where the array was written and associate this field of addresses with the corresponding external address used to write the array.

3) During read operation using the external address value, determine the field of physical addresses where the array is stored and generate the necessary data physical addresses in the required sequence [1, 13].

The proposed address block structure, presented in Fig. 3, performs this function. To illustrate the operation of the suggested address structure in the PI memory, figure 3 also shows the control scheme and the mask RAM and single-element RAM blocks. The address block of the PI memory is outlined with a dashed line.

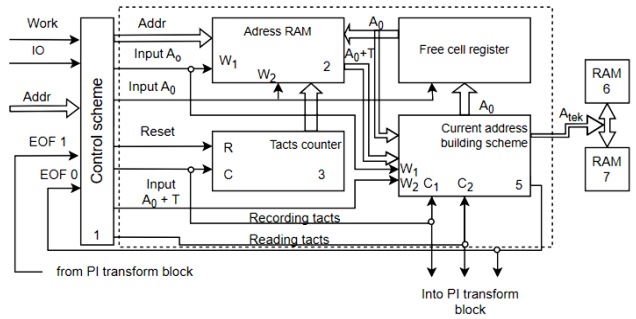



*Fig. 3. Functional scheme of PI memory address block*

To write an array to the PI transformation network, the array itself is digitally input to the data bus, and the control signals "Work" (selection of the memory), "I/O" (write information), and "Addr" (address) are sent to the control circuit. The control circuit routes the content of the "Addr" bus to the input of Address RAM 1 [5, 6]. Address RAM 1 has a capacity of $2 \times k$ cells, where $k$ – is the address bus width. The control circuit generates the "Write Address" signal, which writes the address of the first available cell in Address RAM 1 and the current address generation circuit 4 from the free cell register 3. This address will be the same for Address RAM 6 and Address RAM 7 because the number of tail elements for decomposing one array using the PI transformation method is equal to the number of masks for that array. In other words, each selected element will have its corresponding mask. In the circuit, the first available cell address is required to generate the current physical addresses for writing the encoded array to Address RAM 6 and Address RAM 7.

In Block 2, the value $A_0$ is written to the cell with an address equal to the address on the "Addr" bus. This establishes the connection between the external address on the "Addr" bus and the internal addresses of the storage environment. Then, the control scheme generates a sequence of pulses on the "Write Clocks" line to perform the array write operation [8, 9, 21]. Using these pulses, Control Scheme 4 generates the current address by incrementing the value of $A_0$. The Clock Counter 2 counts the number of pulses on the "Write Clocks" line. When the decomposition of the array in the PI transformation block is completed, a signal "End of Write" is sent to the control scheme, and the pulses on the "Write Clocks" line cease. With the control signal "Write Time", generated by the control scheme, the content of Clock Counter 2 is written to Block 1 under the same address on the "Addr" bus, and the new address of the first available cell from Block 4 is written to Block 3. Then, the "Reset" signal resets Clock Counter 2, completing the write procedure. During the read operation, the control signals "Work", "Wr/Rd", and the corresponding address from the "Addr" bus are applied to the control scheme [7, 25].

The control scheme 1 generates the "Write" signal $A_0 + T$, which writes the values $A_0$ and $T$ from Address RAM 2 to Current Address Generation Scheme 5, corresponding to the code on the "Addr" bus. In Scheme 5, the first current address is formed by adding the values $A_0$ and $T$. The control scheme generates a series of pulses on the "Read Clocks" line. These pulses are sent to the PI transformation block and Scheme 5, where for each pulse, the values $A_0$ and $T$ are decremented by one. This process reads the array from Address RAM 6 and Address RAM 7. With each pulse on the "Read Clocks" line, when the value becomes zero, Scheme 5 generates the "End of Read" signal, which, when received by the control scheme, causes the cessation of pulses on the "Read Clocks" line [7, 19]. The decoding of the read array is completed in the PI transformation block, and the read process is finished.

When organizing the address block of the PI memory, significant difficulties arise due to the significant difference in addressing between the tail element memory block and the mask memory block [9, 31]. This is because the number of masks in processing an array packet, according to the PI transformation method, is much larger than the number of tail elements.

For algorithms where only one tail element can appear at any given time (parallel-sequential processing, selecting only the first tail element at each level, etc.), the addressing of the tail element RAM can be sequential. In this case, one bit in the masks is allocated as a flag indicating the presence or absence of a tail element for that mask. If the mask RAM contains information about the current transformation step and the number of masks at each level of that step, the flag indicating the presence of a tail element is not necessary [19, 23]. A mask corresponds to a tail element when that mask is the only one at its level in the current transformation step.

For algorithms where all tail elements are considered in the encoding result, and multiple tail elements can occur at different levels in a particular transformation step, their storage in memory is done sequentially, starting from the lowest level. During decoding, masks are selected starting from the highest level, and tail elements are read in reverse order, also starting from the highest levels [7, 17].

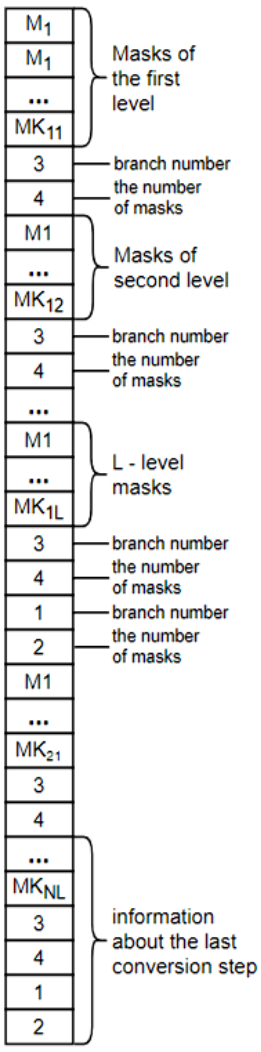



*Fig. 4. Structure of information composition in the RAM mask by memory cell*

The process of writing masks to the PI memory is further complicated by the fact that, according to the PI transformation algorithm, masks are generated in parallel at multiple levels, and each level can have multiple masks. A regular storage environment is used to store the mask codes. Therefore, at each transformation step, the mask codes are captured in buffer registers and are sequentially rewritten from the lower levels to the mask RAM, starting with the lowest levels and branches [12, 16].

The described association procedure is valid only when all masks belong to neighboring branches. It should be noted that on any level, there may be a situation where masks from one transformation step are not present on all branches located between the initial and final branches. In such a case, additional information about the location of masks within the level needs to be stored in the mask RAM. The arrangement of information in the mask RAM cells for one array packet is presented in figure 4 [13, 22, 29].

Masks have different sizes, so it is inefficient to allocate cells with pre-determined sizes to store them. The following structure is proposed for mask addressing:

a) The four most significant bits of the mask indicate the length of the mask in half-bytes.
b) The mask bits themselves follow, occupying the number of half-bytes indicated in the mask size. The length of the mask is determined by the PI transformation block.

Figure 5 illustrates the bit-wise structure for storing one mask.

By using this structure, the size of the area where the mask is stored can be varied widely. The minimum size for storing a mask would be 1 byte. At the same time, you can record $4 \le k \le 60,\ mod\, {}^{k}/_{4} = 0$.

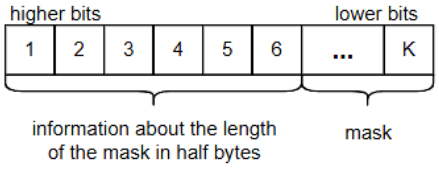



*Fig. 5. Structure of information composition for separate mask (mod – euclidean division)*

Here, half a byte is used to indicate the length of the mask, and the other half a byte represents the code of the mask. The maximum size for the mask area is 15 (4 + 4) = 64 bits, or 8 bytes. In this case, 4 bits indicate the mask length, and the remaining 60 bits represent the mask code. In regular memory, the word size is 16 bits, and the addressing allows accessing individual bits.

During the write mode, the addressing block of the PI memory sequentially increments the addresses for writing the masks until the "End of Write" signal is received from the PI transformation block. The same process is used to generate addresses for writing the tail elements into their respective RAM [17, 23, 26].

During the read mode, the addressing block of the PI memory uses the mask length information and the information stored in cells 1, 2, 3, 4 (Fig. 5) to generate physical addresses for the mask RAM. The masks are selected in sequence, starting from the highest levels and branches. For the tail element RAM, the addressing block generates addresses by decrementing the current address by "one" when selecting a mask from the mask RAM that requires a tail element.

The organization of the addressing part becomes simpler if the encoding of the array packet is done in a parallel-sequential manner, where each level is processed in parallel while the levels themselves are processed sequentially. In this case, an additional buffer memory is required to store arrays from the previous level.

Indeed, there can be different modes of addressing organization depending on the transformation algorithm based on the PI transformation method and the corresponding structural organization of the address block and the entire memory system [16]. This topic is a separate subject of research.

The approach presented here outlines a general method of addressing the tail elements and masks in a parallel memory based on the PI transformation principle.

When designing a PI memory, the development of the address block should consider the requirements imposed on it, such as performance, hardware costs, cost, power consumption, etc. [13, 18]. Therefore, in a specific implementation of a particular address block algorithm, there is a question of what should be performed at the hardware level and what can be offloaded to software. To make a decision in each specific case, a special analysis is required.

The generalized structural diagram of a parallel memory implemented according to the PI transformation algorithm is presented in Fig. 6. The PI transformation block not only performs encoding and decoding of the original information arrays but also controls the memory units (RAM) and generates the addresses for data retrieval. RAM1 stores the values of tail elements for each level, while RAM2 stores the masks generated during the transformation process in the PI transformation block.

This generalized structural diagram illustrates the main components and connections in a PI memory system, which can be further adapted and expanded for specific implementations and requirements.

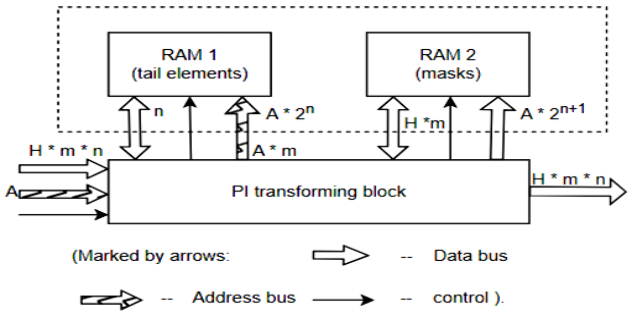



*Fig. 6. Structural scheme of PI memory (A – bit size address bus, m – array size, n – word size in array)*

The address bus width of the RAM is shown for the case when the PI transformation block performs transformations according to the expressions (15), (16), (17), and the level processing is done sequentially.

## 3. Analysis of the efficiency of parallel-hierarchical memory structures

Let us derive an expression that allows calculating the number of cycles required for the complete write (read) operation in a PI memory with a word dimension of $H \cdot m$ [16, 17, 27].

The number of levels until the packet of arrays fully converges, excluding only one initial single element on each level, is calculated using the formula:

$$U = H \cdot m - \sum_{i=1}^{u} \sum_{j=1}^{W_i} r_{i,j} + \sum_{i=1}^{u} \sum_{j=1}^{W_i} L_{i,j} + 1 = \\ = H \cdot m - \sum_{i=1}^{u} \sum_{j=1}^{W_i} (r_{i,j} - L_{i,j}) + 1 \quad (1)$$

where $U$ is the number of levels formed as a result of the PI processing of arrays, $i$ – is the ordinal number of the level, $W_i$ – is the number of branches (original arrays) for the $i$-th level, $j$ – is the branch number on the current level, $r_{i,j}$ and $L_{i,j}$ are the respective counts of identical elements and groups with identical elements on the $i$-th level of the $j$-th branch.

The term 1 in formula (1) accounts for the first level where there is no initial tail element in the PI transformation network. When excluding all tail elements from further processing at each level, formula (1) takes the following form:

$$U = H \cdot m - \sum_{i=1}^{u}\sum_{j=1}^{W_i}\left(r_{i,j} - L_{i,j}\right) - \sum_{i=1}^{u}\sum_{j=1}^{W_i} z_{i,j} + 1 \quad (2)$$

where $z_{i,j}$ – number of tail elements in $j$-th branch of $i$-th level (value of $z_{i,1}$ does not account for separated element $a_{1,1}$).

The number of cycles required to write (read) the initial tail elements in RAM1 is equal to the number of levels minus one, and it is generally less than the number of cycles required to write (read) the masks in RAM2, especially for certain algorithms [10, 27, 28]. Therefore, the determining factor influencing the performance of the PI memory is the time it takes to write (read) the masks obtained from encoding using the PI transformation method. The following are the formulaic dependencies for calculating the number of cycles required to write (read) the masks for all levels:

a) for the first level:

$$T_1 = m - \min_{j=1,W_1}\left(r_{1j} - L_{1j}\right) \quad (3)$$

b) second level:

$$T_2 = \max_{j=1,W_2}\left(\omega_{2j} + (j-1) - \left(r_{2j} - L_{2j}\right)\right) \quad (4)$$

where $\omega_{2j}$ is the number of words (elements) in the initial array of the $j$-th branch of the 2nd level. Since each subsequent branch on all levels, except the first, starts processing one step later than the previous one, and this step corresponds to the cycle of writing (reading) the mask, the term *(j – 1)* in expression (4) represents the shifting process. The shifting starts from the second branch, so one is subtracted from $j$. Thus, expression (4) determines the maximum length of the decomposition among all branches of the second level, taking into account the shifting of each subsequent branch and the convolutions $\left(r_{2j} - L_{2j}\right)$ in each branch, if applicable [17, 19, 30].

c) for the sequential levels will be correct formula (4).

$$T_i = \max_{j=1,W_i}\left(\omega_{ij} + j + L_{ij} - r_{ij} - 1\right)$$
$$i = \{1,2,\dots,u\}, \quad j = \{1,2,\dots,W_i\} \quad (5)$$

Number of leaves on each level counts by formulas:

$$W_1 = H$$
$$W_2 = T_1$$
$$W_3 = T_2 - 1 \quad (6)$$
$$\dots$$
$$W_i = T_{i-1} - 1, \; i = \{3,4,\dots,u\}$$

If all tail elements are taken into account as a result of expansion, then expressions (6) will take the form:

$$W_1^э = H$$
$$W_2^э = T_1 - z_{1j}, \quad z_{1j} = \{0,1,2,\dots,m-1\}$$
$$W_3^э = T_2 - \sum_{j=1}^{W_2} z_{2j} - 1 \quad (7)$$
$$\dots$$
$$W_i^э = T_{i-1} - \sum_{j=1}^{W_{i-1}} z_{i-1j} - 1, \; i = \{3,4,\dots,u\}$$

With sequential processing of levels, the total number of cycles for writing (reading) masks is:

$$T_\Sigma^{pos} = \sum_{i=1}^{u} T_i = m - \min_{j=1,W_1}\left(r_{1j} - L_{1j}\right) + \sum_{i=2}^{u}\max_{j=1,W_i}\left(\omega_{ij} + j + L_{ij} - r_{ij} - 1\right) \quad (8)$$

With a parallel encoding process, the number of cycles for writing (reading) an information array with a dimension of H·m words will be:

$$T_\Sigma^{pair} = U + max\left(\omega_{uj} + j + L_{uj} - r_{uj} - 1\right) - 2 = U + \max_{j=1,W_u}\left(\omega_{uj} + j + L_{uj} - r_{uj}\right) - 3 \quad (9)$$

The subtraction of two in the first part of expression (9) is done to exclude the first and last levels. The first level does not involve a cycle of writing, and the last level is already accounted for in the term that determines the maximum length of the last level.

For regular sequential memory, writing (reading) a two-dimensional array of words will require cycles:

$$T^{st} = H \cdot m \quad (10)$$

From the analysis of expressions (1), (8), (9) and (10) it follows that $T_\Sigma^{pair}$ less than $T^{ST}$ for $\sum_{i=1}^{u}\sum_{j=1}^{W_i}\left(r_{i,j} - L_{i,j}\right) - \max_{j-1,W_u}\left(\omega_{uj} + j + L_{uj} - r_{uj}\right) + 2$, a $T_\Sigma^{pos}$ depending on the variables $K$, i, $\omega$ and $W$ can be either less or more $T^{st}$.

The capacity of the PI memory (Fig. 6) depends on the type of $F^*$ and $Q^*$ transformation and the method of mask generation during array processing.

The capacity of RAM1 is relatively small with a fixed word size of $n$ bits. RAM2, on the other hand, has a much larger capacity, and the word size varies from $m$ to $H$ bits for the first and second levels, to 2 bits for the last level. This is achieved through additional technical and software means, compromises, and flexible addressing that allows access to memory cells of different lengths. The number of levels and branches, as seen from (8) and (9), directly affects the memory performance and required capacity [24, 31].

The PI memory (Fig. 5) allows storing $2^A$ two-dimensional arrays with a dimension of $H \cdot m$, and during reading, it can generate the entire original array in parallel. It can be used, for example, to store information about image fragments with dimensions $H \cdot m$ samples. The PI transformation block in the PI memory is a complex device, which may not always be justified in practice for hardware implementation of parallel memory. A simpler transformation algorithm can be used for encoding a one-dimensional array, i.e., processing a single branch using the PI transformation method [1, 24, 28]. A parallel-hierarchical RAM with pre-processing of a one-dimensional array can have a single RAM block (Fig. 7), into which the selected element value (data-1) and the mask of that element (data-2) are simultaneously written (read) under the same address [11].

The number of clock cycles for writing (reading) in this case is determined by the formula:

$$T^{brn} = m - r + L \quad (11)$$

where $r$ – number of the same words in the array; $L$ – number of pairs with the same words.

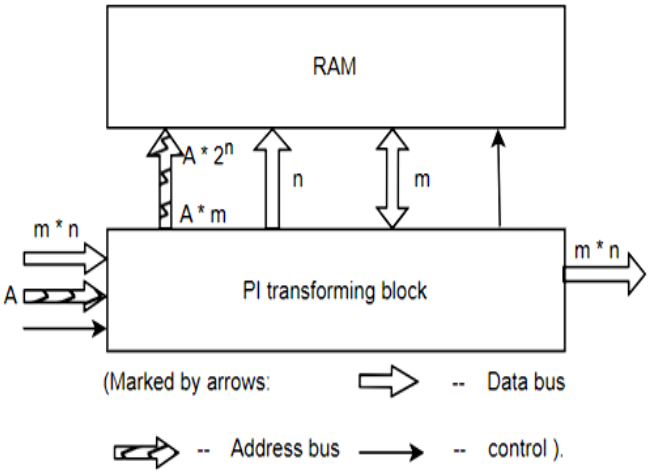



*Fig. 7. Structural scheme of PI memory (A – Address bus registry size; m – address' registry size; n – arrays' word registry size)*

## 4. Research results

Comparing formulas (10) and (11), it can be seen that the PI memory with preliminary array transformation is more efficient compared to regular memory by the amount of $r - L$.

To store information about one array in the PI memory, the required memory volume is:

$$O^{brn} = T^{brn} \cdot n + T^{brn} \cdot m = T^{brn} \cdot (n + m) = (m - r + L)(m + n) \quad (12)$$

Required value for usual memory:

$$O^{st} = m \cdot n \tag{13}$$

Subtracting (12) from (13), we get:

$$P = O^{st} - O^{st} = r \cdot (n + m) - L \cdot (n + m) - m^2 \tag{14}$$

From (14) as consequence:

$$\begin{aligned} r \cdot (n + m) - L \cdot (n + m) - m^2 = 0, \ \text{to} \ O^{st} = O^{brn} \\ r \cdot (n + m) - L \cdot (n + m) - m^2 > 0, \ \text{to} \ O^{st} > O^{brn} \\ r \cdot (n + m) - L \cdot (n + m) - m^2 < 0, \ \text{to} \ O^{st} < O^{brn} \end{aligned} \tag{15}$$

From equation (15), it follows that the volume of the PI memory compared to traditional memory structures will be smaller, the higher the values of $r$ and $n$, and the smaller the values of $L$ and $m$.

The efficiency of reducing the volume of PI memory can be assessed by the ratio:

$$R = \frac{O^{st}}{O^{brn}} = \frac{mn}{(m-r+L)(n+m)} \tag{16}$$

Consider a specific example: let there be an array with such parameters $m = 15$, $n = 8$, $r = 10$ and $L = 4$:

$$T^{brn} = 15 - 10 + 4 = 9\tau, T^{st} = 15\tau$$
$$O^{brn} = 9(15 + 8) = 207\text{bits}$$
$$O^{st} = 15 \times 8 = 120 \text{ bits}$$

For an array of the same dimension but with different parameters $m = 15$, $n = 8$, $r = 13$, $L = 3$:

$$T^{brn} = 15 - 13 + 3 = 5\tau, \ T^{st} = 15\tau$$
$$O^{brn} = 5(15 + 8) = 115\text{bits}$$
$$O^{st} = 15 \times 8 = 120 \text{ bits}$$

In the first case, the performance of the PI memory is higher than that of the standard memory, but the memory capacity is larger. In the second case, both the performance is higher and the memory capacity of the PI memory is smaller than that of the standard memory. In general, the number of clock cycles required for writing (reading) information using the proposed PI transformation method is within the range of:

$$\tau \leq T^{brn} \leq T^{st}$$

Therefore, the explored architectural features of the parallel-hierarchical memory classify it as a non-von Neumann computational structure. The proposed structure of the PI memory, focused on parallel and compact processing of data fields, enables real-time transformation. The researched architectural characteristics allow for efficient memory organization for array processing and transformation, which can be valuable in various applications requiring fast and parallel data processing.

## 5. Conclusions

In this paper we demonstrate the potential of parallel-hierarchical memory (PHM) systems to significantly improve the performance of parallel computing systems. The results show that PHM systems can achieve significant performance improvements over traditional memory systems, particularly for applications with a high degree of data parallelism.

Research also highlight the challenges that need to be addressed in order to realize the full potential of PHM systems. These challenges include the design of efficient memory access policies, the development of scalable and efficient memory management schemes, the development of fault-tolerant PHM systems, and the development of PHM systems that are compatible with existing programming models and languages.

Provided information is a foundation for further research on PHM systems. The results suggest that PHM systems have the potential to revolutionize the field of parallel computing, and the challenges identified in this paper provide a roadmap for future research in this area.

Presented results is significant because they demonstrate the potential of PHM systems to revolutionize the field of parallel computing.

## References


[1] Aboutabl A. E., Elsayed M. N.: A Novel Parallel Algorithm for Clustering Documents Based on the Hierarchical Agglomerative Approach. International Journal of Computer Science & Information Technology – IJCSIT 3(2), 2011, 152–163.

[2] Bisikalo O. et al.: Parameterization of the Stochastic Model for Evaluating Variable Small Data in the Shannon Entropy Basis. Entropy 25(2), 2023, 184.

[3] Bykov M. et al.: Neural network modelling by rank configurations. Proc. of SPIE 10808, 2018, 1080821.

[4] Kim S., Wunsch D. C.: A GPU based Parallel Hierarchical Fuzzy ART clustering. IJCNN IEEE, 2011, 2778–2782.

[5] Kohonen T.: Self Organization and Associative Memory: Third Edition. Springer-Verlag, New York, 1989.

[6] Kovtun V., Izonin I.: Study of the Operation Process of the E-Commerce Oriented Ecosystem of 5Ge Base Station, Which Supports the Functioning of Independent Virtual Network Segments. Journal of Theoretical and Applied Electronic Commerce Research 16(7), 2021, 2883–2897.

[7] Kukharchuk V. V. et al.: Features of the angular speed dynamic measurements with the use of an encoder. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Srodowiska – IAPGOS 12(3), 2022, 20–26.

[8] Kukharchuk V. V. et al.: Information Conversion in Measuring Channels with Optoelectronic Sensors. Sensors 22(1), 2022, 271.

[9] Kuusilinna K. et al.: Configurable parallel memory architecture for multimedia computers, Journal of Systems Architecture 47(14–15), 2002, 1089–1115.

[10] Kvyetnyy R. et al.: Inverse correlation filters of objects features with optimized regularization for image processing. Proc. SPIE 12476, 2022, 124760Q.

[11] Li Z., Li K., Xiao D., Yang L.: An Adaptive Parallel Hierarchical Clustering Algorithm. Perrott, R., Chapman, B.M., Subhlok, J., de Mello, R.F., Yang, L.T. (eds): High Performance Computing and Communications. HPCC 2007. Lecture Notes in Computer Science 4782. Springer, Berlin, Heidelberg 2007.

[12] Nere A., Lipasti M.: Optimizing Hierarchical Algorithms for GPGPUs. Master's Project Report. University of Wisconsin Madison, 2010.

[13] Orazayeva A. et al.: Biomedical image segmentation method based on contour preparation, Proc. SPIE 12476, 2022, 1247605.

[14] Osman A. A. M.: A Multi-Level WEB Based Parallel Processing System: A Hierarchical Volunteer Computing Approach. World Academy of Science, Engineering and Technology 13, 2006, 66–71.

[15] Pavlov S. V. et al.: The use of Bayesian methods in the task of localizing the narcotic substances distribution. International Scientific and Technical Conference on Computer Sciences and Information Technologies 2, 2019, 8929835, 60–63.

[16] Rajasekaran S.: Efficient Parallel Hierarchical Clustering Algorithms. IEEE Transactions on Parallel and Distributed Systems 16(6), 2005, 497–502.

[17] Romanyuk S. A. et al.: Using lights in a volume-oriented rendering. Proc. SPIE 10445, 2017, 104450U.

[18] Rose K.: Deterministic Annealing, Clustering and Optimization. Ph.D. Thesis, California Institute of Technology, Pasadena, 1991.

[19] Sobota B.: Parallel Hierarchical Model of Visualization Computing. Journal of Information, Control and Management Systems 5(2), 2007, 345–350.

[20] Sudarshan R. Lee S. E.: A Parallel Hierarchical Solver for the Poisson Equation, May 14, 2003,.

[21] Timchenko L. et al.: New methods of network modelling using parallel-hierarchical networks for processing data and reducing erroneous calculation risk. CEUR Workshop 2805, 2020, 201–212.

[22] Timchenko L. I., Kokriatskaia N. I., Pavlov S. V., Tverdomed V.: Method of indicators forecasting of biomedical images using a parallel-hierarchical network. Proc. of SPIE 11176, 2019, 111762Q.

[23] Timchenko L. I.: A multistage parallel-hierarchic network as a model of a neuronlike computation scheme. Cybern Syst Anal. 36, 2000, 251–267.

[24] Tolegen G., Toleu A., Mamyrbayev O., Mussabayev R.: Neural Named Entity Recognition for Kazakh. Lecture Notes in Computer Science 13452, 2023, 3–15.

[25] Tymkovych M. et al: Ice crystals microscopic images segmentation based on active contours. IEEE 39th International Conference on Electronics and Nanotechnology – ELNANO 2019, 493–496 [https://doi.org/10.1109/ELNANO.2019.8783332].

[26] Vasilevskyi O. et al.: A new approach to assessing the dynamic uncertainty of measuring devices. Proc. of SPIE 10808, 2018, 108082E.

[27] Vysotska O. V., Nosov K.: An approach to determination of the criteria of harmony of biological objects. Proc. of SPIE, 10808, 2018, 108083B

[28] Wójcik W., Pavlov S., Kalimoldayev M.: Information Technology in Medical Diagnostics II. Taylor & Francis Group, CRC Press, Balkema book, London 2019.

[29] Ybytayeva G. et al.: Creating a Thesaurus "Crime-Related Web Content" Based on a Multilingual Corpus. CEUR Workshop Proceedings 3396, 2023, 77–87.

[30] Zeki S.: A Vision of the Brain. Blackwell Scientific Publications, Oxford 1993.

[31] Zhao X., Guo Y., Feng Z., Hu S.: Parallel Hierarchical Cross Entropy Optimization for On-Chip Decap Budgeting. Design Automation Conference, Anaheim, CA, USA, 2010, 843–848.

**Prof. Leonid Timchenko**
e-mail: tumchenko_li@gsuite.duit.edu.ua

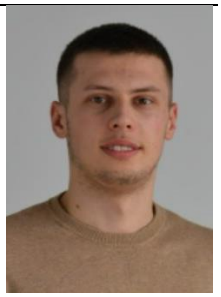
Doctor of Technical Science, professor. 56 articles published in Scopus, 227 citations in 112 articles (h-index = 8).

https://orcid.org/0000-0001-5056-5913

**Yurii Didenko**
e-mail: didenk.y.v@gmail.com

Post-graduate student at State University of Infrastructure and Technology.
Research interests: systems of artificial intelligence, image processing systems.

https://orcid.org/0009-0008-1033-4238



**Ph.D. Natalia Kokriatska**
e-mail: nkokriatskaia@gmail.com

Ph.D., associate professor. 34 articles published in Scopus, 119 citations in 82 articles (h-index = 7).

https://orcid.org/0000-0003-0090-3886

**Dmytro Zhuk**
e-mail: zhuk_do@ukr.net

Post-graduate student at State University of Infrastructure and Technology.
Research interests: systems of artificial intelligence, image processing systems.

https://orcid.org/0000-0001-8951-5542

**Ph.D. Volodymyr Tverdomed**
e-mail: tverdomed@gsuite.duit.edu.ua

Ph.D., associate professor, Director of Kyiv Institute of Railway Transport, State University of Infrastructure and Technology, Ukraine.
13 articles published in Scopus, 13 citations in 16 articles (h-index = 3).
Research interests: development of methods for diagnosing the technical condition and forecasting the duration of operational work of railway track elements and track devices.

https://orcid.org/0000-0002-0695-1304

**Maksym Kozyr**
e-mail: jettab3@gmail.com

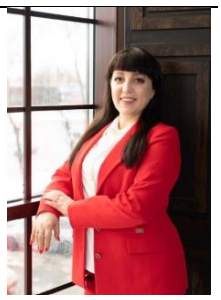
Ph.D. student. Artificial Intelligence Systems and Telecommunication Technologies Department, State University of Infrastructure and Technology, Ukraine. Research interests: systems of artificial intelligence, image processing systems.

https://orcid.org/0009-0007-2564-6552

**Prof. Iryna Yu. Yepifanova**
e-mail: yepifanova@vntu.edu.ua

Doctor of economic sciences, professor, Vice-rector by science work, Faculty of Management and Information Security of Vinnytsia National Technical University, academician of the Academy of Economic Sciences of Ukraine.
Scientific interests: financial support of innovative activities of domestic enterprises, enterprise potential, competitiveness, personnel management, digital economy, energy saving.

https://orcid.org/0000-0002-0391-9026



**Ph.D. Iryna Shakhina**
e-mail: rom.shahin@gmail.com

Ph.D. (in Pedagogy), associate professor at the Department of Innovation and Information Technologies in Education Vinnytsia Mykhailo Kotsiubynskyi State Pedagogical University, Vinnytsia, Ukraine.
Scientific direction: information technologies, image processing, innovation in pedagogic

https://orcid.org/0000-0002-4318-6189