Associative Semantic Computers for Intelligent Computer Systems of a New Generation

Vladimir Golenkov and Daniil Shunkevich and Natalia Gulyakina
Valerian Ivashenko and Vadim Zahariev
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: {golen, shunkevich, guliakina, ivashenko, zahariev}@bsuir.by

***Abstract*—The paper considers the shortcomings of the currently dominant von Neumann architecture of computer systems as the basis for building intelligent computer systems of a new generation, analyzes modern approaches to the development of hardware architectures that eliminate some of these shortcomings, substantiates the need to develop fundamentally new hardware architectures, which are hardware version of the implementation of the interpretation platform for systems built on the basis of OSTIS Technology, — associative semantic computers.**

***Keywords*—OSTIS Technology, ostis-platform, platform independence, ontology, associative semantic computer.**

I. Introduction

For the development of *ostis-systems*, the use of modern software and hardware platforms focused on address access to data stored in memory is not always effective, since when developing intelligent systems, it is actually necessary to model nonlinear memory based on linear one. Improving the efficiency of problem solving by intelligent systems requires the development of specialized platforms, including hardware ones, focused on unified semantic models of information representation and processing. Thus, the main purpose of creating *associative semantic computers* is to increase the performance of ostis-systems.

II. Current state of developing computers for intelligent systems

The vast majority of modern software and hardware platforms used in the development of modern computer systems and, in particular, intelligent computer systems are based on the principles of the abstract *von Neumann machine*, or "von Neumann architecture" (see [1], [2]). Let us consider the basic principles underlying the *von Neumann machine*.

von Neumann machine
:= [abstract von Neumann machine]
∈ *abstract information processing machine*
⇒ *underlying principles**:
⟨
- [The information in memory is represented as a sequence of strings of characters in a binary alphabet ("0" or "1").]
- [The machine memory is a sequence of <u>addressable</u> memory cells.]
- [Any string of characters in the binary alphabet can be recorded to each cell. At the same time, the length of the lines for all addressable cells is the same (in the current standard of cells, called bytes, it is equal to 8 bits).]
- [Each memory cell uniquely corresponds to a bit string that denotes this cell and represents its address.]
- [To each type of elementary actions (operations) performed in the memory of the von Neumann machine, its identifier is uniquely assigned, which is also represented in memory as a bit string.]
- [Each specific operation (command) performed in memory is represented (specified) in memory as a string consisting of
  - the code of the corresponding type of operation;
  - the sequence of addresses of memory fragments containing operands on which operations are performed – the source arguments and results. Any such fragment is specified by the address of the first byte and the number of bytes. The number of operands is <u>unambiguously</u> set by the operation type code.

  ]
- [A program running in memory is stored in memory as a sequence of specifications of particular operations (commands).]
- [Thus, both the processed data and the programs for processing this data are stored in the same memory (unlike, for example, the Harvard architecture) and are encoded in the same way.]

⟩

Let us consider in more detail the features of the logical organization of the traditional (von Neumann) architecture of computer systems, which significantly complicate the effective implementation of *ostis-systems* based on it:

- sequential processing that limits the efficiency of computers by the physical capabilities of the element base;
- low level of access to memory, i.e. complexity and awkwardness of performing the procedure of associative search for the necessary fragment of knowledge;
- linear memory organization and an extremely simple view of constructive objects directly stored in memory. This leads to the fact that in intelligent systems built on the basis of modern computers, the manipulation of knowledge is carried out with great difficulty. Firstly, it is necessary to operate not with the structures themselves but with their inconvenient linear representations (lists, adjacency matrices, incidence matrices); secondly, the linearization of complex structures destroys the locality of their transformations;
- the representation of information in the memory of modern computers has a level very far from the semantic one, which makes the processing of knowledge rather awkward, requiring consideration of a large number of details concerning not the meaning of the processed information but the way it is represented in memory;
- in modern computers, there is a very low level of hardware-implemented operations on non-numeric data and there is no hardware support for logical operations on knowledge fragments with a complex structure, which makes manipulating such fragments very difficult.

Attempts to overcome the limitations of traditional von Neumann computers have led to the appearance of many approaches associated with particular changes in the principles of logical organization of computers, primarily depending on the classes of problems and subject domains that a particular class of computers focuses on. All these tendencies, considered together, allow outlining some key principles of the logical organization of computers focused on the knowledge processing (knowledge processing machines – KPM). Let us list the main of these tendencies:

- transition to nonlinear memory organization (see [3], [4]) and hardware interpretation of complex data structures (see [5], [6], [7]);
- hardware implementation of associative access to information (see [3], [8], [9], [10], [11], [12], [13]);
- implementation of parallel asynchronous processes over memory (see [3], [14]), and, in particular, development of computing machines controlled by data flow (see [15], [16], [17], [18]);
- hardware interpretation of high-level languages (see [19], [20], [21]);
- development of database management hardware tools (database processors) (see [22], [23], [24]).

At the intersection of these tendencies, different classes of computing devices have appeared at different times. Let us list some of them:

- machines with hardware interpretation of complex data structures (see [6], [25], [26]);
- machines with developed associative memory (see [10], [27], [28]);
- associative parallel matrix processors (see [29]);
- homogeneous parallel structures for solving combinatorial logic problems on graphs and hypergraphs (see [30]);
- various graph processing devices (see [31], [32], [33], [34]), in particular, based on FPGA (see [35], [36], [37]) and vector processors (see [38]);
- systems that process information directly in memory by evenly distributing functional means in memory and, in particular, the processor-memory proposed by M. Weinzweig, focused on solving artificial intelligence problems (see [39], [40]);
- machines controlled by data flow (see [15], [18], [29]) and, in particular, processors, that are reconfigurable taking into account the semantics of the input data flow (see [41]);
- recursive computing machines (see [3]);
- relational database processors (see [9], [18], [22]);
- computers with restructurable memory (see [42], [43], [44], [45]);
- active semantic networks (M-networks) (see [46]);
- associative homogeneous environments (see [47]);
- neural-like structures (see [48], [49]). In recent years, the active development of the theory of artificial neural networks has led to the development of various approaches to the building-up of high-performance computers designed for training and interpretation of artificial neural networks (see [50], [51], [52]) and their implementation in various software and hardware complexes. In a separate direction, the so-called neuromorphic processors (see [53]) are distinguished by high performance and low power consumption.
- machines for interpreting logical rules (see [54]).

In addition, the development of graphics processors (graphics processing unit, GPU) has led to the possibility of organizing parallel computing directly on the GPU, for which specialized software and hardware architectures are being developed, for example, CUDA (see [55]) and OpenCL (see [56]). The advantage of the GPU in this case is the presence of a large number of cores within one GPU (compared to the central processor), which makes it possible to effectively solve problems with natural parallelism on such an architecture (for example, operations with matrices). Works dedicated to the principles of processing graph structures on the GPU are also being developed (see [57], [58], [59]).

In general, it can be said that due to the increase in the performance of modern computers, the number of developments of specialized hardware solutions has decreased in recent decades, since many complex computing problems can now be solved on traditional universal architectures in an acceptable time. As shown above, the exception is mainly specialized computers for processing artificial neural networks and other graph models, which is conditioned by the high demand for such models and their complexity.

At the same time, most of these approaches (even if they deviate far enough from the basic principles of computer organization proposed by von Neumann) implicitly retain the point of view of the computer as a large arithmometer and thereby retain its orientation to numerical problems. Works aimed at developing hardware architectures, designed to process information represented in more complex forms than in traditional architectures, have not been widely distributed and used due, firstly, to the specifics of the proposed solutions and secondly, due to the lack of a common universal and unified coding language for any information, in the role of which within the *OSTIS Technology*, the *SC-code* acts, as well as the appropriate proven technology for developing software systems for such hardware architectures. Thus, developers of such architectures often face the need to develop specialized software for these architectures, which ultimately leads to a strong limitation in the scope of application of such architectures, since their use turns out to be reasonable only if the complexity of developing specialized software proves itself, taking into account the low efficiency of solving the corresponding problems on more traditional architectures.

The *SC-code*, which is the formal basis of the *OSTIS Technology*, was originally developed as a language for encoding information in memory of *associative semantic computers*, so it originally contains principles such as universality (the ability to represent knowledge of any kind) and unification (uniformity) of representation, as well as minimization of the *Alphabet of the SC-code*, which, in turn, makes it easier to create a hardware platform that allows storing and processing texts of the *SC-code*.

The main methodological feature of the proposed approach to the development of hardware implementation tools for intelligent systems support is that such tools should be developed not before but after the main terms of the corresponding technology for the design and operation of intelligent systems will be tested on modern technical means. Moreover, within the *OSTIS Technology*, the methodology of transition to new hardware tools has been clearly thought out, which affects only the lowest level of the technology – the level of implementation of the basic semantic network processing machine (interpreter of the *SCP Language*).

The project of the *associative semantic computer* has a long history, the main stages of which are the following ones:

- 1984 – at the Moscow Institute of Electronic Technology, V. Golenkov defended the PhD dissertation on the topic “Structural organization and processing of information in electronic mathematical machines controlled by the flow of complex structured data”, in which the basic principles of semantic associative computers were formulated and considered (see [60]).
- 1993 – the Goskomprom Commission carried out successful tests for the prototype of the *associative semantic computer* developed on the basis of transputers within the research project “Parallel graph computing system focused on solving artificial intelligence problems” (see [61], [62]).
- 1996 – V. Golenkov defended the doctoral dissertation on the topic “Graphodynamic models and methods of parallel asynchronous processing of information in intelligent systems” (see [63]).
- 2000 – at the Institute of Management Problems of the Russian Academy of Sciences, P. Gaponov defended the PhD dissertation on the topic “Models and methods of parallel asynchronous processing of information in graphodynamic associative memory” (see [64]).
- 2000 – at the Institute of Software Systems of the Russian Academy of Sciences, V. Kuzmitsky defended the PhD dissertation on the topic “Principles of building a graphodynamic parallel computer focused on solving artificial intelligence problems” (see [65]).
- 2004 – at the Belarusian State University of Informatics and Radioelectronics, R. Serdyukov defended the PhD dissertation on the topic “Basic algorithms and tools for information processing in graphodynamic associative machines”, in which the basic software of semantic associative computers was considered (see [66]).

At the same time, despite the presence of a working prototype of the *associative semantic computer*, based on transputers, the main attention within the corresponding project and other listed works was paid to the principles of organizing distributed parallel processing of SC-code constructions, in particular, the SCD Language (Semantic Code Distributed) was developed for distributed storage of SC-code constructions and the SCPD Language for their distributed parallel processing. However, the general principles of information storage and the general architecture of each of the processor elements (of transputers) remained behind von Neumann. In particular, to encode SC-code constructions in traditional address memory, appropriate data structures have been developed, close to those described in [67].

Thus, we can say that the reasonableness and necessity of the development of the *associative semantic computer*, as well as the competence of the authors in this field is confirmed by more than 30 years of experience and a number of successful projects in this direction, however, at the same time, in the previous works, all the shortcomings of the von Neumann architecture discussed above have not been fully eliminated, and the development and implementation of the *associative semantic computer* project that eliminates these shortcomings remain relevant.

III. Analysis of existing computing system architectures

As shown in the previous paragraph, in order to overcome the shortcomings of existing computing system architectures, including the von Neumann one, many different approaches have been proposed. When developing new architectures and, in particular, the architecture of the *associative semantic computer*, it is advisable to identify the main features of classification and the corresponding classes (types) of computing system architectures in the form of an appropriate ontology. Let us consider a fragment of such an ontology developed on the basis of an analysis of existing solutions and the approaches identified by its results.

architecture of a computing system
⇒ *subdividing**:
  {• *architecture of a computing system with global RAM*
    ⇒ *subdividing**:
      {• *architecture of a computing system with global data RAM*
       • *architecture of a computing system with global program RAM*
       • *architecture of a computing system with global program and data RAM*
         ⇒ *note**:
           [An example of such an architecture is the von Neumann architecture.]
      }
   • *architecture of a computing system without global RAM*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with a single global internal memory*
   • *architecture of a computing system with multiple global internal memory*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with restructurable interprocessor connections*
   • *architecture of a computing system without restructurable interprocessor connections*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with structurally evolving memory*
   • *architecture of a computing system without structurally evolving memory*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with associative access to global (internal) memory*
    ⇒ *note**:
      [The associative type of access is important in systems focused on data storage with a complex structure and focused on scalable (including local) information processing mechanisms.]
   • *architecture of a computing system without associative access to global (internal) memory*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with addressable access to global memory with linear address space*
    ⇒ *note**:
      [Examples of such an architectures are ones that are most used at the moment, including the von Neumann architecture. The problems of device and data management for such architectures are considered in the work [68].]
   • *architecture of a computing system without addressable access to global memory with linear address space*
  }
⇒ *subdividing**:
  {• *architecture of a computing system with a system of register data processing commands*
    ⇒ *note**:
      [Most of the architectures currently in use are examples of architectures of this class, including the

von Neumann architecture. Architectures with a system of register data processing commands are convenient for data management problems both for image processing systems in user interface problems and for machine learning problems based on linear algebra apparatus.]

- *architecture of a computing system without a system of register data processing commands*

}

⇒ *subdividing**:

{● *architecture of a computing system with a command system for stack data processing*

⇒ *note**:

[Examples of the application of such an architecture are LISP machines (see [69], [70], [71]), other examples can also be found in the works [72], [73].]

- *architecture of a computing system without a command system for stack data processing*

}

⇒ *subdividing**:

{● *architecture of a computing system with support for a command system for processing generalized strings*

⇒ *note**:

[An example of such an architecture is the architecture that supports the operations of the generalized strings and lists processing model proposed in the work [74]. The underlying model makes it possible to efficiently perform operations not only on strings and lists but also to work with key-value relations in order to integrate them into knowledge-driven systems. The software implementation of this model uses B-trees.]

- *architecture of a computing system without support for a command system for processing generalized strings*

}

⇒ *subdividing**:

{● *architecture of a computing system with support for a command system for processing graph structures*

⇒ *note**:

[An example of such an architecture is the architecture of the Leonhard computer (see [75]). This computer is focused on processing graph and hypergraph structures of various types, including hierarchical graphs (see [76]). The representation is supported in the form of strings and a list of adjacent vertices, ordered local lists of incident edges, and a global ordered list of incident edges.]

- *architecture of a computing system without a support system for processing graph structures*

}

⇒ *subdividing**:

{● *architecture of a computer system with a command system for (hardware) knowledge processing*

⇒ *note**:

[An example of such an architecture is the architecture of the Leonhard computer (see [75]). The Leonhard computer supports the DISC command systems (Discrete Instruction Set Computing) (see [76]). The DISC command system supports the following commands: creating an integer relation with a schema that is a set of objects of a formal context (the first domain of a binary relation), whereas the corresponding set of images is a set of non-negative integers (the second domain of a binary relation); adding a pair to a formal context containing an object (key) to be added, which is added as a tuple by adding elements of this tuple, together with an integer image (value) for this object; getting the next or previous object in a linearly (lexicographically) ordered list of objects; getting the next larger or previous smaller object in a linearly (lexicographically) ordered list of objects; getting the minimum or maximum object in a linearly (lexicographically) ordered list of objects; getting the number (cardinality of the set) of images for a given object (key tuple); searching pairs by key; deleting pairs; deleting all pairs of

formal context, including objects (keys) and images (values); a slice (subset) of formal contexts; combination, intersection, and complementation of formal contexts. B+ trees are used to represent the processed data. Other architectures consider the implementation of knowledge processing operations using a logical model of knowledge representation (see [77]), LISP structures (see [69], [70]), generalized formal languages (see [74], [78]). In the last case, for the development of a system of knowledge processing commands, the transition from knowledge processing to meta-knowledge processing (based on the semantics of becoming relevant and irrelevant) is considered, the result of which is a system of meta-operations (see [78]).]
- *architecture of a computer system without a command system for (hardware) knowledge processing*

}

⇒ *subdividing**:

{• *architecture of a computing system with adaptive data distribution*

⇒ *note**:

[Adaptive data distribution (including, as a special case, virtual address space) is important for data (and knowledge) management and virtualization problems for multitasking and multi-user systems, as well as also closely related to the scalability capabilities of the system.]
- *architecture of a computing system without adaptive data distribution*

}

⇒ *subdividing**:

{• *architecture of a computing system with a command system for non-local information processing*

⇒ *note**:

[An example of such an architecture is a cellular automaton. Elementary cellular (binary) automata are divided into: ones rapidly transitioning into a homogeneous state (a state consisting only of zeros or units); rapidly transitioning into a stable or cyclic state; remaining in a chaotic (random) state; forming both areas with a stable or cyclic state and areas in which complex interactions of elements of states manifest themselves, up to Turing-complete ones.

Information processing using cellular automata allows building computing systems, including ones with a tunable (including fractal-like) structure based on local parallel (competitively) performed simple rules. There are varieties of cellular automata that support irreversible, reversible, deterministic, non-deterministic, specialized, universal (including Turing-complete) computations. The work of cellular automata resembles wave processes propagating in the environment of processor-memory elements of the *associative semantic computer*, considered below.]
- *architecture of a computing system without a command system for non-local information processing*

}

⇒ *subdividing**:

{• *architecture of a computing system with strictly binary data representation in RAM*

⇒ *note**:

[Most modern architectures of digital computing systems, including implementations of the von Neumann architecture, use binary representation]
- *architecture of a computing system with not strictly binary data representation in RAM*

}

⇒ *subdividing**:

{• *architecture of a computing system with strictly discrete data representation*

⇒ *note**:

[An example of such an architecture is the von Neumann architecture.]
- *architecture of a computing system without strictly discrete data representation*

}

⇒ *subdividing**:

{• *architecture of a computing system with discrete data representation*
    ⊂ *architecture of a computing system with strictly discrete data representation*
• *architecture of a computing system without discrete data representation*
}
⇒ *subdividing**:
{• *architecture of a computing system with data flow control*
    ⇒ *note**:
        [Architectures of computing systems with data flow control are seen as more natural when solving many artificial intelligence problems. Variants of such architectures are considered in the works []. The architecture of cellular automata can be considered as the architecture of a computing system controlled by a data flow.]
• *architecture of a computing system without data flow control*
}
⇒ *subdividing**:
{• *architecture of a computing system with data flow control*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computing system without command flow control*
}
⇒ *subdividing**:
{• *architecture of a computing system with a processor with an arithmetic-logical unit*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computing system without a processor with an arithmetic-logical unit*
}
⇒ *subdividing**:
{• *architecture of a computer system with a control unit with an instruction counter*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computer system without a control unit with an instruction counter*
}
⇒ *subdividing**:
{• *architecture of a computer system with a control unit with a command register*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computer system without a control unit with a command register*
}
⇒ *subdividing**:
{• *architecture of a computing system with an input-output device*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computing system without an input-output device*
}
⇒ *subdividing**:
{• *architecture of a computing system with access to an external (operational) storage device*
    ⇒ *note**:
        [An example of such an architecture is the von Neumann architecture.]
• *architecture of a computing system without access to an external (operational) storage device*
}
⇒ *subdividing**:
{• *architecture of a computing system with scalable (modular) global memory*
    ⇒ *note**:
        [Scalability as a feature of an architecture is important for systems focused on training (self-training) in order to solve a wide class of problems. Such architectures can be focused on the processing of knowledge structures integrated into a single semantic space.]
• *architecture of a computing system without scalable global memory*
}
⇒ *subdividing**:
{• *architecture of a computing system with support for an active graph memory model*
    ⇒ *note**:
        [An active graph memory model in the architectures of computa-

tional systems is important for the efficient and consistent (convergent) implementation of parallel knowledge processing operations, including mechanisms of excitation and inhibition of knowledge processing operations.]
- *architecture of a computing system without support for an active graph memory model*

}

⇒ *subdividing**:

{• *architecture of a computing system with support for parallel information processing*

⇒ *note**:

[Architectures of computing systems with support for parallel knowledge processing are important for the effective implementation of knowledge processing processes (see [78]), improving the performance and scalability of knowledge processing systems, including multi-agent systems in the form of intelligent computer systems and collectives of intelligent computer systems. Various models of architectures of computing systems with support for parallel knowledge processing are considered in the work [65].]
- *architecture of a computer system with sequential information processing*

}

⇒ *subdividing**:

{• *architecture of a computing system with support for a sequential consistency model of global RAM*

⇒ *note**:

[Architectures with support for consistency models are focused on solving the problem of controlling interacting processes, including their synchronization and synchronous and asynchronous mechanisms for running knowledge processing algorithms. The purpose of supporting a sequential consistency model (see [78]) is to ensure the existence of global states of the knowledge base as structures of a single semantic space in intelligent computer systems. To ensure a particular consistency model, various mechanisms can be used, considered in the works [64], [66], as well as [67].]
- *architecture of a computing system without support of a sequential consistency model of global RAM*

}

⇒ *subdividing**:

{• *architecture with support for causal memory consistency model*

⇒ *note**:

[The purpose of supporting the causal consistency model (see [78]) is to ensure interoperability and convergence in a single semantic space of knowledge structures of agents in collectives of intelligent computer systems. To ensure a particular consistency model, various mechanisms can be used, considered in the works [64], [66], as well as [67].]
- *architecture without support for causal memory consistency model*

}

⇒ *subdividing**:

{• *architecture of the computing system asymmetric*

⇒ *note**:

[The architectures of computing systems with asymmetry are important for the evolution of multi-agent systems, intelligent computer systems, and their collectives, in which asymmetry is considered in a broad sense, including as the heterogeneity of such systems or collectives. A special case of heterogeneity is the heterogeneity of the architecture, which allows implementing both integrated and hybrid models of knowledge processing within intelligent computer systems and their collectives.]
- *architecture of the computing system symmetric*

}

To determine the architecture of *associative semantic computers*, in accordance with the identified classes and features, as well as the general principles underlying such architectures, it is necessary to consider specific sets of architectures within the appropriate feature space and conduct a comparative analysis of the elements of these sets in order to justify the choice of (optimal) architecture variants for *associative semantic computers*.

At this stage of the work, several main variants for the architecture of *associative semantic computers* have been specified, which are considered in more detail below.

IV. General principles underlying associative semantic computers for ostis-systems

The proposed approach to the development of the *associative semantic computer* is based on the ideas considered in the works of V. Golenkov (see [63]) and developed in the work of V. Kuzmitsky [65].

When formalizing subject domains that have a rather complex semantic organization, the processed data is naturally grouped into some complex structures. The efficiency of solving problems related to the processing of complex structured data on multiprocessor computing systems increases significantly when the structure of the connections between the processor elements of the computing system solving this problem coincides with the structure of the data processed during its solution (or, more generally, is displayed in the structure of the processed data in a simple and natural way). With the transition to data processing of an increasingly complex structural and semantic organization (and then to knowledge processing), the maintenance of high efficiency of the computing system is ensured mainly by increasing the number of processor elements working simultaneously and complicating the structure of connections between them (see [65]).

We will consider such a tendency in the development of computer hardware as the main line of evolution that creates prerequisites for the appearance of *associative semantic computers*. It includes parallel regular special processors (vector, matrix ones), special calculators for solving problems on graphs, and hardware support for semantic and neural networks. This line is also joined by associative processors (in which associative memory cells act as processor elements), database processors, and computing systems that effectively solve certain classes of problems due to the coincidence of the structure of connections between processor elements with the structure of the information graph of the algorithm (systolic calculators, data flow machines) (see [65]).

A natural result of the development of computing systems is the transition to systems that change the structure of connections between processor elements in the process of functioning. Such systems adjust their internal structure to the structure of the processed data and the information graphs of the algorithms of the problems being solved and can solve different classes of problems while maintaining high efficiency.

Thus, a developed computer focused on the knowledge processing should generally be a collective of special processors focused on the most effective solution of certain classes of problems and have the following features:

- Special processors are a multiprocessor computing system.
- The structure of the connections between the processor elements of special processors coincides with the data structure or (less often) with the structure of the information graph of the algorithm for solving the problem.
- The connections between the processor elements of special processors have a reconfigurable structure.
- The set and functions of special processors are determined for each knowledge processing machine specifically depending on the set of subject domains that this machine is focused on and the specifics of the problems solved in these domains.
- The set of knowledge processing mechanisms determined for some semantic processor should be "immersed" in the language of knowledge representation and processing. At the same time, the languages of semantic networks seem to be the most convenient for this purpose.
- Processor elements correspond to vertices or fragments of a semantic network.
- Information processing is reduced to a change in the structure of connections between processor elements, corresponding to a change in the configuration of the semantic network.

As a semantic special processor, we can propose a nonlinear (graph) restructurable (dynamic) processor-memory that implements some kind of semantic network processing language in hardware, and the computer of this kind itself can thus be called a graphodynamic parallel associative computer, or an *associative semantic computer*.

Taking into account the above, as well as the general principles of information processing in ostis-systems described in [67], let us consider more specifically the principles underlying the implementation of *associative semantic computers*:

- Nonlinear memory – each elementary fragment of a text stored in memory can be logically incident to an unlimited number of other elementary fragments of this text. Thus, memory cells, unlike ordinary memory, are connected not by fixed conditional connections that specify a fixed sequence (order) of cells in memory but by a logically or even physically (using technical means of switching) conducted connections of accidental configuration. These connections correspond to arcs, edges, hyper-edges of the graph (sc-text) recorded in memory.
- Restructurable (reconfigurable) memory – the operation of processing the information stored in memory is reduced not only to changing the state of the elements but also to reconfiguring the connections between them. That is, during the processing of information in restructurable memory, not only and

not even so much the states of memory cells change, as is the case of ordinary memory, but the configuration of connections between these cells. That is, in restructurable memory, during the information processing, not only the labels on the vertices of the graph recorded in memory are redistributed, but also the structure of this graph itself changes.
- As an internal way of encoding knowledge stored in the memory of the *associative semantic computer*, a universal (!) method of nonlinear (graph-like) semantic representation of knowledge – SC-code – is used.
- Information processing is carried out by a collective of agents working on shared memory. Each of them reacts to the corresponding situation or event in memory (a computer controlled by stored knowledge).
- There are software-implemented agents whose behavior is described by agent-oriented programs stored in memory, which are interpreted by the corresponding collectives of agents.
- There are basic agents that cannot be software implemented (in particular, these are agents of interpretation of agent programs, basic receptor agents-sensors, basic effector agents).
- All agents work on shared memory at the same time. Moreover, if several conditions of its application arise for an agent at some point in time in different parts of memory, different information processes corresponding to the specified agent in different parts of memory can be run simultaneously.
- In order for information processes of agents running in parallel in shared memory not to "interfere" with each other, its current state is recorded and constantly updated in memory for each information process. That is, each information process informs others about its intentions and wishes, which other information processes should not interfere with. The implementation of such an approach can be carried out, for example, on the basis of the mechanism for locking elements of semantic memory, considered in [67].
- The processor and the memory of the *associative semantic computer* are deeply integrated and form a single processor-memory. The processor of the *associative semantic computer* is evenly "distributed" over its memory so that processor elements are simultaneously computer memory elements. That is, each cell is supplemented by a functional (processor) element, and tunable connections between cells become switched communication channels between processor elements. At the same time, each processor element has its own special internal register memory, reflecting aspects of the current state of performing elementary operations of the micro-program language that provide interpretation of a higher-level language (SCP Language) that are important for this processor element.
  Information processing in the *associative semantic computer* is reduced to reconfiguration of communication channels between processor elements, therefore the memory of such a computer is nothing but a switchboard (!) of the specified communication channels. Thus, the current state of the configuration of these communication channels is the current state of the information being processed. This principle provides a significant acceleration of information processing by eliminating the stages of transferring information from memory to the processor and back, but it is paid for at the cost of a large redundancy of processor (functional) means evenly distributed over memory.

V. Architecture of associative semantic computers for ostis-systems

associative semantic computer
⊂ *computer with graphodynamic associative memory*
:= [associative semantic computer]
:= [sc-computer]
:= [hardware implemented basic interpreter of semantic models (sc-models) of computer systems]
:= [hardware implemented ostis-platform]
:= [hardware variant of the ostis-platform]
:= [associative semantic computer controlled by knowledge]
:= [computer with a nonlinear restructurable (graphodynamic) associative memory, the processing of information in which is reduced not to a change in the state of memory elements but to a change in the configuration of the connections between them]
:= [universal computer of a new generation, specially designed for the implementation of semantically compatible hybrid intelligent computer systems]
:= [universal computer of a new generation, focused on hardware interpretation of logical and semantic models of intelligent computer systems]
:= [universal computer of a new generation, focused on hardware interpretation of ostis-systems]
:= [ostis-computer]
:= [computer for the implementation of ostis-systems]
:= [computer controlled by the knowledge represented in the SC-code]
:= [computer focused on processing SC-code texts]
:= [computer whose internal language is an SC-code]
:= [computer that implements sc-memory and interprets scp-programs]
:= [our proposed new generation computer focused on the implementation of intelligent computer

systems and using SC-code as an internal language]

⇒ *subdividing**:

{● *scp-computer*

:= [sc-computer that provides interpretation of scp-programs]

⇒ *generalized model**:
basic ostis-platform

● *sc-computer with an extended set of hardware-implemented sc-agents*

:= [sc-computer that provides interpretation of scp-programs]

⇒ *generalized model**:
specialized ostis-platform

}

scp-computer

:= [minimum configuration of a hardware-implemented ostis-platform, within which the interpretation of scp-programs is provided]

:= [minimum configuration of a hardware-implemented ostis-platform, within which only basic sc-agents are implemented in hardware]

⇒ *explanation**:
[Within the scp-computer, (1) sc-memory, (2) basic sc-agents providing interpretation of scp-programs, (3) elementary receptor sc-agents, (4) elementary effector sc-agents are implemented in hardware.]

To refine the architecture of *associative semantic computers*, it is necessary to clarify:

- the basic structure of the computer and, in particular, its processor-memory;
- the alphabet of elements stored in the processor-memory of the computer;
- the system of commands interpreted by a computer;
- principles of controlling the process of interpreting these commands;
- the system of micro-programs that ensure the implementation of the principles for controlling the specified process.

Since the internal language for encoding information of the *associative semantic computer* is the SC-code, the alphabet of elements stored in the processor-memory of the computer coincides with the *Alphabet of the SC-code*, considered in [67]. At the same time, the alphabet of physically encoded syntactic labels can be expanded, for example, for performance reasons, by analogy with how it is done in the software implementation of the *ostis-platform* [67].

As a command system for the *associative semantic computer*, the *SCP Language*, discussed in detail in [67], is proposed. Thus, as already mentioned in the specified paragraph, the *SCP Language* is an assembler for the *associative semantic computer*.

To determine the basic structure of the *associative semantic computer*, let us clarify the variants of such a structure proposed in the works of V. Golenkov and V. Kuzmitsky (see [63], [65]). In particular, in the work of V. Kuzmitsky, a transition from coarse-grained architectures of graphodynamic machines to fine-grained ones is proposed (see [65]).

Models of coarse-grained architectures have parallel functioning modules with the following features:

- each module has a strictly fixed functional purpose within the architecture of the graphodynamic machine as a whole (the so-called global functional purpose);
- each module has a relatively large amount of memory (the number of memory elements is much larger than the total number of modules);
- the memory of each type of module has its own non-elementary set of operations that perform some completed transformations on sufficiently large fragments of memory.

The main formal difference for models of fine-grained architectures is a different ratio between the total number of modules and the number of memory elements of each module (module memory capacity), which tends to unity, and the level of complexity of model operations. Accordingly, the features of models of fine-grained architectures can be considered as the following (see [65]):

- For each module separately, its functional purpose may not be viewed within the graphodynamic machine as a whole. At the same time, each separate module at a particular time can have some so-called local functional purpose, the corresponding set of which can already be considered as having a certain so-called global functional purpose within the graphodynamic machine as a whole.
- The amount (number of elements) of memory of each module is minimal and tends to unity. As a result, the total number of modules is comparable to the total number of memory elements of all modules.
- For each module (in general), the set of operations performed on its memory is elementary and limited (finite), since it affects only one element (or only a few elements) of memory and is determined by the obvious limitation (finiteness) in the semantics of interpreting the contents of the graph memory element in a graphodynamic machine.

The reasonableness of the transition from coarse-grained to fine-grained architectures is conditioned by a corresponding increase in the degree of potential parallelism in knowledge processing procedures.

Taking into account the above, we can talk about several options for clarifying the basic structure of the *associative semantic computer*, each of which has certain advantages and disadvantages. Let us consider these options in more detail.

A. Architecture of an associative semantic computer based on the von Neumann architecture

One of the most logical and architecturally simplest options for the hardware implementation of the ostis-platform is the implementation of means for storing SC-code constructions and interpreting scp-programs at the hardware level, similar to how it is done in software versions of the ostis-platform based on modern computers [67]. In this case, the overall architecture of the computer remains behind von Neumann (with the explicit allocation of a separate processor module and a separate memory module). The main features of this implementation option include the following ones:

- A memory module (implementation of *sc-memory*) is a set of cells, each of which can store some sc-element or can be empty. Each cell of such memory has some unique internal address, similar to the address of the von Neumann memory cells. At the same time, when processing information stored in such memory at the level of the command language (SCP Language), unlike von Neumann memory, cell addresses are not taken into account; access to sc-elements is carried out strictly by incidence relations between them. The exception is some key sc-elements, the set of which is specified separately and accessed by some other identifier, for example, the system sc-identifier or the main sc-identifier for some external language but not by address. It is assumed that if some sc-element is stored in a cell, it stores information characterizing this sc-element, namely:
  - a syntactic label specifying the type of the corresponding sc-element;
  - contents of the sc-file or a link to an external file system (if the sc-file is stored);
  - a list of the incidence relations of this sc-element with other sc-elements, which actually means storing a set of memory cell addresses corresponding to the sc-elements incident to this sc-element. The specific list of types of stored relations can be specified depending on the implementation. For example, by analogy with how it is done in the software implementation of the ostis-platform [67], it is advisable to store in a memory cell the address of the cell corresponding to the first sc-connector incident to the corresponding sc-element with the corresponding incidence type, and within the cell corresponding to this sc-connector, to store the address of the cell corresponding to the next sc-connector, incident to the same sc-element with the same incidence type, etc. With this approach, the size of each memory cell can be fixed.
  - a label of the sc-element lock indicating the label of the corresponding process;
  - an access level label and any other labels, if necessary.
- The processor module implements a set of commands corresponding to *atomic types of scp-operators*.
- To connect with the external environment, a *terminal module* is introduced (see [63], [65]), which in general can be implemented in different ways and whose tasks are:
  - to prepare (generate) information coming from the external environment for its subsequent loading into the processor module and the module memory;
  - to transfer (use, implement) information prepared (received, represented) in the processor module and memory module to the external environment.
- To store the contents of large sc-files, it may be advisable to have a separate file memory built according to von Neumann principles. Then the semantic memory cells corresponding to such sc-files will store not their contents directly but the address of this file on the file memory.
- To implement the principles of multi-agent information processing proposed within the OSTIS Technology [67], it is necessary to implement (for example, within the terminal module) an event registration and processing subsystem that will allow the initiation of sc-agents when the corresponding events occur in memory.

The advantages of this implementation option include:

- Relative simplicity and low labor intensity of implementation compared to the development of the full-fledged processor-memory variant discussed below. In particular, with the availability of a stable version of the ostis-platform software implementation, in which at least the lower level is implemented in sufficiently low-level languages, such as C, to simplify the process of developing hardware architecture, it is possible to use automation tools for the transition from C programs to descriptions in hardware description languages (HDL, for example, VSDL and Verilog), also known as “C to HDL”. Popular tools and languages of this class include LegUp (see [79]), VHDPlus (see [80]), SystemC (see [81]), MyHDL for Python (see [82]), and many others.
- Simplicity of integration with modern computer systems, in particular, a hybrid variant can be considered, in which the *associative semantic computer* is implemented as a separate plug-in module for a modern computer designed to increase the efficiency of processing sc-constructions.

The obvious key disadvantage of this option is its orientation to the von Neumann architecture with all its disadvantages listed above. In addition, in this option, by default, parallel processing of sc-constructions is not provided at the hardware level. This disadvantage is partially eliminated in the next version of the coarse-grained architecture of *associative semantic computers*.

B. Variant of the coarse-grained architecture of associative semantic computers

The goal of the transition to the coarse-grained architecture of *associative semantic computers* is to implement parallel processing of sc-constructions at the hardware level.

The main features of this implementation option include the following ones:

- The *associative semantic computer* is divided into several modules of the same type, arranged in a similar way as the implementation option for the associative semantic computer considered in the previous paragraph, built on the basis of the von Neumann architecture. Such modules will be called "combined modules", since such a module has its own processor module and its own memory module (storage module); there are no separately allocated common processor modules. There may be a separate shared memory module, into which, if necessary, information that does not fit into the memory of a particular combined module will be recorded.
- The terminal module that provides the connection of the system of combined modules with the external environment is still allocated.
- Separately, a file memory module can be allocated.
- The number of combined modules is relatively small (2 – 16), each module is a sufficiently powerful device (in fact, it is a separate *associative semantic computer*), and, accordingly, one combined module may be enough to solve problems of some classes.
- At the same time, in general, it is necessary to use several combined modules to solve the problem. In this case, the processed sc-construction is distributed among several modules, for which sc-nodes-copies are created, allowing for semantic communication between fragments of the sc-construction stored in different combined modules. To record such constructions, an extension of the SC-code was developed, called SCD-code (Semantic Code Distibuted, see [63], [65]), respectively, the constructions of such a language were called scd-constructions, their elements – scd-elements (scd-nodes, scd-arcs).
- Similarly, an extension of the SCP Language, called the SCPD Language, was developed for processing scd-constructions, taking into account the fact that different fragments of the processed construction can be physically stored in different combined modules. At the same time, it is assumed that all elements of the scd-construction representing an scpd-program (a program of the SCPD Language) should be located in the memory of one combined module, but each scpd-program can have several complete copies in different combined modules.
- To synchronize parallel information processing operations, combined modules exchange messages that can contain both fragments of processed scd-constructions and commands of the SCDP Language. Accordingly, the SCPD Language, in comparison with the SCP Language, has additional tools that support distributed processing of graph constructions (see [63], [65]):
  - The SCPD Language has built-in tools that allow recognizing "your" and "foreign" combined module; for this purpose, operators are introduced to work with module identifiers.
  - It is possible to create a copy of the scd-element in the memory of another module. For this purpose, a group of operators is introduced to work with copies: creating a copy of the scd-element in the specified module, transferring the connections of the original element to the copy, gluing copies of the element together, searching for a copy of this element in a given module, etc.
  - It is possible to explicitly call the scpd-program remotely in the specified processor module. To run the same processes performing in parallel in different processor modules, operators are specified, which run the program in modules from the specified list.
  - There are means of inter-process and intra-process synchronization: message generation operators, message waiting operators, process transfer operators in the waiting mode for completing execution of distributed executing operators, waiting operators for completing execution of all distributed executing operators.
- For message exchange, each combined module has corresponding submodules that allow sending and receiving messages, as well as a message buffer for storing a queue of received messages waiting to be processed and messages waiting to be sent.
- To interpret SCPD-programs, a family of micro-programs is being developed in a language that generally depends on the selected hardware components from which the combined modules are built.

The described implementation option for *associative semantic computers* with coarse-grained architecture also includes the previously mentioned multi-transputer implementation (see [62], [63]). This implementation is based on IBM PC 386 (486, Pentium) and 8 T805 transputers. In Figure 1, this implementation option is schematically shown on 8 transputers, where each transputer simultaneously performs the role of a switching node ("SN") and a processor module ("PM") or a storage module ("SM"). The entire system interacts with the external environment through a terminal module ("TM").

The main advantage of the coarse-grained architecture of *associative semantic computers* is the orientation to hardware support for parallel processing of SC-code constructions. At the same time, this implementation option has a number of disadvantages:

- Each combined module is built according to the principles of the *von Neumann machine*, accordingly, its disadvantages are not fully eliminated.
- Despite the preservation of the general principles of the SC-code and SCP Language, distributed storage and processing of sc-constructions requires the development of separate language tools, such as an SCD-code and SCPD Language, and their support based on the selected hardware architecture. In addition, as can be seen from the principles of the SCPD Language discussed above, when developing scpd-programs, it is necessary to explicitly take into account the fact that processing is performed distributed.

The next step from the point of view of the hierarchy in the architectures of *associative semantic computers* is the fine-grained architecture of associative semantic computers.

C. Variant of the fine-grained architecture of associative semantic computers

As already mentioned, the reasonableness of the transition from coarse-grained to fine-grained architectures is conditioned by a corresponding increase in the degree of potential parallelism in knowledge processing procedures. At the same time, the maximum possible parallelism will obviously take place with the maximum implementation of fine-grained architectures in which one structural module of processor-memory will correspond to one memory element, that is, in our case, one sc-element.

Let us consider in more detail the principles underlying the fine-grained architecture of the *associative semantic computer*:

- The processor-memory of the *associative semantic computer* consists of modules of the same type, which will be called processor elements of sc-memory, or simply *processor elements*. Each *processor element* corresponds to one sc-element (stores one sc-element). At the same time, at any given moment, each *processor element* can be empty (not store any sc-element) or filled, that is, have a mutually unambiguously corresponding stored sc-element. At the physical level, an appropriate attribute with two meanings is introduced to describe this fact. Thus, each *processor element* is "responsible" for only one sc-element and, unlike the coarse-grained version of the *associative semantic computer* architecture, the problem cannot be solved by one *processor element*, and the number of such *processor elements* is quite large (corresponds to the maximum possible number of sc-elements stored in the knowledge base of some ostis-system). Experience in the development of applied ostis-systems shows that, on average, the number of sc-elements in the knowledge base of such an ostis-system ranges from several hundred thousand to several million. The situation when it is necessary to represent an sc-construction within the processor-memory, the number of elements of which is greater than the number of *processor elements*, is not currently being considered and requires additional research.
- Each *processor element* (by analogy with a memory cell in the case of the implementation of the *associative semantic computer* on the von Neumann architecture) has some unique internal identifier – an *address of the processor element*. Addresses of *processor elements*, unlike addresses of von Neumann memory cells, do not provide direct access to *processor elements* but allow unambiguously identify the processor element when exchanging messages according to the principles discussed below.
- Each processor element has a memory which stores:
  - a syntactic label specifying the type of the corresponding sc-element;
  - the contents of the sc-file or a link to an external file system (if this processor element corresponds to the sc-file);
  - a list of logical connections of this processor element with others, that is, a list of addresses of processor elements associated with this processor element by *logical communication channels*, indicating the type of communication (for more information about *logical communication channels*, see below);
  - a label of blocking sc-elements, indicating the label of the corresponding process;
  - other labels, if necessary (for example, labels of the access level to the stored sc-element);
  - wave micro-programs run by this processor element at the moment (for more information about *wave micro-programs*, see below) and temporal data for these micro-programs, as well as a queue of micro-programs, if necessary.
- Processor elements are interconnected by two types of communication channels – *physical communication channels* and *logical communication channels*:
  - In general, the number of *physical communication channels* for each *processor element* can be arbitrary, in addition, theoretically, *physical communication channels* between processor elements can be rebuilt (reconnected) over time, for example, in order to optimize the time of message transmission

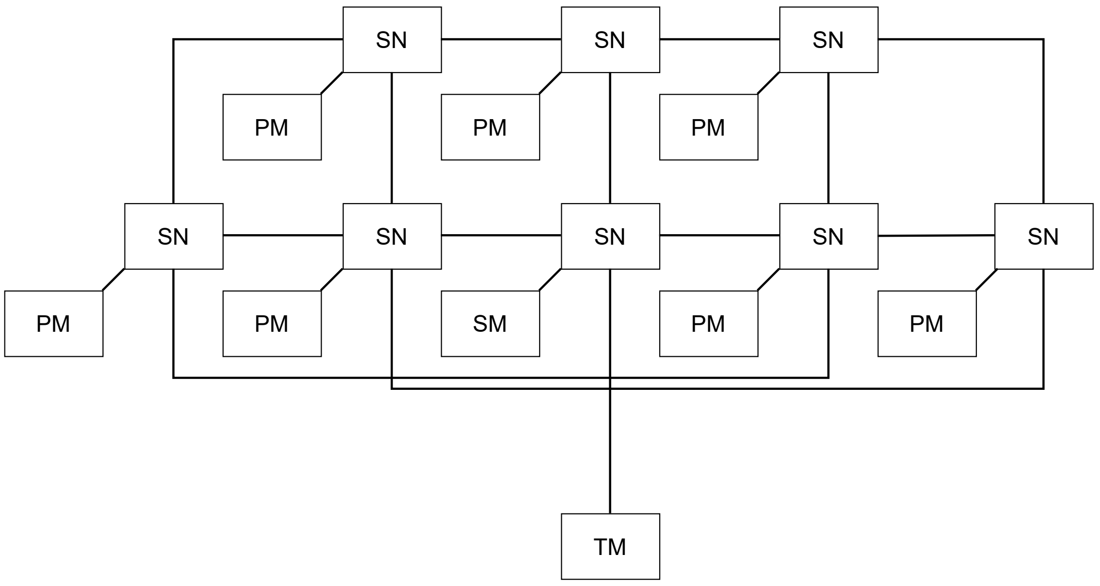

Figure 1. Example of implementing the coarse-grained architecture

between processor elements. The configuration of *physical communication channels* is not taken into account at the level of logical knowledge processing, both at the level of the SCP Language and at the level of the language of the micro-programs, providing interpretation of commands for the SCP Language, that is, *scp-operators*. For simplification, within this work, we will consider an option of the physical implementation of sc-memory, in which each processor element has a fixed and the same number of *physical communication channels* (N) for all processor elements, while the configuration of such communication channels does not change over time. Obviously, the minimum value of N is 2, in this case we will get a linear chain of *processor elements*. With N equal to 4, we will get a two-dimensional "matrix" of *processor elements*, with N equal to 6 – a three-dimensional "matrix" of *processor elements*, etc. As "adjacent" *processor elements* we will call ones that are directly connected by a *physical communication channel*.

- In this case, we can say that each processor element has its own "address" (unique identifier) in some multidimensional space, the number of dimensions (features) of which is determined by number N of *physical communication channels* associated with one *processor element*. In the examples above, the dimensionality of such a space is N/2, which suggests that it is advisable to make number N even-numbered.
- Each *physical communication channel* and each *logical communication channel* are thus defined by a pair of *addresses of processor elements*.
- *Logical communication channels* between processor elements are formed dynamically and correspond to *incidence relations* between sc-elements. Thus, *logical communication channels* can describe two types of incidence relations – *incidence of sc-pair designations with their components* and *incidence of oriented sc-pair designations with their second components* [67]. At the same time, the configuration of *logical communication channels* in general is not related in any way to the configuration of *physical communication channels*: incident sc-elements can be physically stored in processor elements that are not adjacent. At the same time, it is obvious that, in general, some *physical communication channels* may correspond to logical ones.
- In addition to the incidence relations, *logical communication channels* can correspond to other types of connections between sc-elements, by analogy with how it is done in the software implementation of the ostis-platform [67]. For example, to simplify the implementation of search algorithms in the knowledge base and reduce the amount of memory that each *processor element* should have, it is advisable to store in the memory of the processor element the address of only the first sc-connector incident to the corresponding sc-element with the corresponding incidence type, and within the processor element corresponding to this sc-connector, the address of the next sc-connector incident to the same sc-element with the same incidence type, etc. With this approach, the

amount of processor element memory that stores logical connections between processor elements can be fixed.

- Each processor element can send messages (micro-programs) to other processor elements and receive messages from other processor elements via *logical communication channels* and has corresponding receptor-effector submodules. At the physical level, messages are transmitted, in turn, via *physical communication channels*, the configuration of which, as mentioned above, is fixed and generally does not depend on the configuration of logical communication channels.
- Thus, processor elements form a homogeneous processor-memory, in which there are no separately allocated modules designed only for storing information and separately allocated modules designed only for its processing.
- To connect such a processor-memory with the external environment, a *terminal module* is introduced, which in general can be implemented in different ways and whose tasks are:
  - preparation (generation) of information coming from the external environment for its subsequent loading into processor modules;
  - transfer (use, implementation) of information prepared (received, represented) in processor modules to the external environment.
- To store the contents of large sc-files, it may be advisable to have a separate file memory associated with processor-memory and built according to traditional von Neumann principles. This is conditioned by the fact that the main purpose of building-up processor-memory is to ensure as much parallelism as possible when processing SC-code constructions, while in the case of storing and processing the contents of sc-files, which by definition are information constructions external to the SC-code, it is advisable to use modern traditional approaches.

These principles allow formulating a key feature of processing information stored within such a processor-memory. Unlike the von Neumann architecture (and other architectures developed around the same time, for example, the Harvard architecture) and even from the *ostis-platform software version*, the proposed processor-memory architecture has no shared memory available for all modules that process information. Due to this, parallel processing of information is greatly simplified, but the implementation of a set of micro-programs for interpreting information processing commands in such memory becomes more complicated, since each processor element becomes very "short-sighted" and "sees" only those processor elements that are connected to it by *logical communication channels*.

Thus, the language for describing the micro-programs for interpreting commands of the *associative semantic computer* cannot be built as a traditional programming language, for example, of a procedural type, since all such languages assume the possibility of direct address or associative access to random memory elements. The proposed micro-program description language is proposed to be built according to the principles of *wave programming languages* (see [83], [84]) and insertion programming (see [85], [86]).

Within such a micro-programming language, two types of waves are distinguished:

- waves transmitted only via *logical communication channels* (for example, when searching for incident sc-elements);
- waves transmitted over all communication channels (for example, when creating new logical communication channels, that is, when generating new sc-elements).

Let us consider in more detail the principles for interpreting commands (of scp-operators) within the processor-memory considered above:

- Each *processor element* can interpret some limited set of micro-programs. Taking into account the fact that one processor element corresponds to one sc-element, the set of operations associated with the transformation of this sc-element is very limited (generate an sc-element of the specified type, delete an sc-element, change the contents of the sc-file, set or remove the lock label, etc.). Thus, an important task of the processor element is to generate messages for other processor elements and send them.
- Each processor element can generate and store temporary data for micro-programs in memory. It is assumed that the amount of memory available to the processor element is sufficient to represent all the necessary data for a possible set of micro-programs, since such micro-programs are quite simple (see the previous principle). In case, for some reason, overflow still occurs, then various approaches can be used, for example, described in the work [65].
- Each processor element can form a micro-program and send it as a wave message for running by other processor elements. Messages are transmitted via physical communication channels. Since the configuration of physical communication channels is generally not related to the configuration of logical communication channels, each processor element independently decides whether to run the micro-program and transfer it further. Here we can draw an analogy with the wave algorithm for finding a path in a graph (a variant of the breadth search).
- Frequently, processor elements will not run the micro-program but transmit it further, thus, the processor elements themselves also perform the role of switching elements, while, in general, each

processor element can enter an arbitrary number of routes when transmitting messages through logical communication channels between processor elements.
- As in the case of the coarse-grained architecture, each processor element has a queue of micro-programs to be run (incoming messages) and a queue of micro-programs to be sent (outgoing messages). At the same time, within each processor element, it is also possible to talk about the possibility of performing any operations in parallel (for example, generating outgoing messages and processing the current stored sc-element).

Accordingly, a good case can be made about the existence of a hierarchy of micro-programs:

- Micro-programs for changing the stored sc-element:
  - perform the specified transformation of the contents of this sc-node;
  - change the label of the sc-element type (if such a change does not contradict the *Syntax of the SC-code*);
  - replace the lock of this sc-element for the specified process (including removing the label);
  - delete the sc-element.
- Micro-programs for processing sc-elements stored in others (not necessarily adjacent processor elements):
  - generate an incident sc-connector (and a new *logical communication channel*), possibly together with an adjacent sc-element;
  - generate both or one sc-element connected by this sc-connector;
  - find all sc-connectors (that is, the addresses of their corresponding processor elements) of the specified type, incident to this sc-element by the specified incidence type;
  - find sc-nodes incident to this sc-connector.
- Micro-programs for managing the running processes of other micro-programs:
  - forward the specified micro-program for running from this processor element through all specified channels (incident sc-connectors of the specified type) to all adjacent sc-elements of the specified type;
  - wait for the running of the specified type of micro-programs generated by the specified processor element and transmit the result of their running to the processor element that requested the appropriate information.
- And others.

Obviously, when solving a specific problem, these micro-programs can be combined into more complex micro-programs. The above hierarchy is not complete at the moment and requires further clarification.

Based on the principles represented, a hierarchy of programming languages is formed for the proposed fine-grained architecture of *associative semantic computers*:

- The SCP Language, independent of the implementation of the ostis-platform, on which the programs of sc-agents of knowledge processing are written. The SCP Language is a “watershed” between the platform-dependent part and the platform-independent part of the ostis-system, so it is the lowest-level language among all possible platform-independent languages and at the same time a high-level language from the point of view of the ostis-platform.
- The language of the micro-programs that the processor elements exchange with each other and which are run by these processor elements. In fact, an interpreter of the SCP Language is being developed in this language. It is important to note that the micro-program language is focused on the transmission of messages via *logical communication channels* and does not take into account the configuration of *physical communication channels*. For this, another lower-level language is introduced.
- A language for writing programs for managing processes of exchanging messages (micro-programs). The introduction of such a language is necessary because, as it was said, the micro-programming language itself does not take into account:
  - Configuration of physical communication channels. Thus, when sending a message via a logical communication channel, it is necessary to generate the necessary number of messages depending on the number of available physical communication channels, encode the transmitted message for transmission over a physical communication channel, transmit a message taking into account that the same physical communication channel can generally be included in an arbitrary number of routes between processor elements, decode the message on the receiving processor element. All these problems require the development of appropriate programs.
  - Queuing incoming and outgoing messages inside the *processor element*, adding messages to the queue, extracting messages from the queue for execution, etc.

Advantages of the proposed fine-grained architecture variant of *associative semantic computers*:

- Within the proposed fine-grained architecture, unlike coarse-grained one, there is no need to create copies of sc-elements and to develop special coding languages for the resulting constructions, such as the SCD-code, since each processor element stores one atomic fragment of the entire stored sc-construction, and the number of logical connections with other processor elements is unlimited.

- The above clearly distinguished hierarchy of programming languages makes it possible to exclude at the level of development of user programs (in the SCP Language and higher-level languages based on it) the need to take into account the fact of distributed storage of sc-constructions and the general principles of organizing the ostis-platform. In other words, the development of languages such as the SCPD Language is not required.
- The extensibility of the architecture makes it easy to increase the number of processor elements without significantly reducing performance, since there are no explicitly allocated processor modules and storage modules in the proposed architecture, respectively, the need to transfer information between such modules is eliminated; in addition, the processor module ceases to be a shared resource for a large number of simultaneously run processes. All of the above will eventually solve the problem known as the "bottleneck" problem of the von Neumann architecture (see [87]).
- The key advantage of the proposed fine-grained architecture is its orientation to the maximum possible support for parallel information processing at the hardware level and, ultimately, the possibility of implementing <u>any</u> parallelism models taking into account the problem being solved. In support of this thesis, we can cite the theory of A-systems described in the work of V. Kotov and A. Narinyani [88]. According to the authors, this concept should be interpreted as a universal model for a certain class of parallel systems, which requires clarification in the case of specific implementations. In particular, within this theory, processor elements are distinguished, activation/deactivation of which is carried out by means of the so-called trigger function, which takes the values 0 and 1. It is clear that in a particular implementation, any attribute with the values "true" and "false" can be used as such a function, indicating that a particular processor element should be activated at the next moment in time. The authors show the possibility of formalizing any parallel algorithms based on this model, consider the possibility of reducing such algorithms to sequential ones, synchronization options within such a model. An obvious parallel can be drawn between A-systems and the proposed fine-grained architecture of *associative semantic computers*, taking into account the presence of a wave programming language:
  - processor elements from the theory of A-systems correspond to *processor elements* of the processor-memory;
  - in the role of trigger functions for processor elements, the micro-programs act, transmitted by waves from one processor element to another and, accordingly, activating the activity of processor elements.

  It is worth noting that despite the fact that the considered work on the theory of A-systems has been known for more than half a century, the authors of this work failed to implement the ideas of this theory in hardware. In our opinion, this is conditioned by the fact that the level of development of microelectronics at that time did not meet the requirements necessary for the implementation of the theory of A-systems.

Together with the listed advantages, we can highlight the key disadvantage for the proposed fine-grained version of the architecture of *associative semantic computers*, which consists in a strong dependence of the processor-memory performance on the time of transmission of wave micro-programs from one processor element to another. At the same time, since at the logical level messages are transmitted via *logical communication channels* and in reality – via *physical communication channels*, the processor-memory performance will depend on how closely the configuration of *logical communication channels* corresponds to the configuration of *physical communication channels*. Obviously, in the general case, one-to-one correspondence of these configurations is impossible, since the number of *physical communication channels* incident to a given processor element is limited, unlike the number of *logical communication channels*. Nevertheless, there are several options for optimizing the placement of sc-constructions in the processor-memory:

- When recording ("stacking") sc-constructions into processor-memory (especially in the case of sufficiently large sc-constructions), it is possible to take into account the <u>semantics</u> of the fragments being recorded and write them in such a way that those sc-elements, to which the message will be transmitted from this sc-element most likely, were physically closer to this sc-element. So, for example, it is possible to take into account the denotational semantics of searching scp-operators, which are focused on processing *three-element sc-constructions* and *five-element sc-constructions*, as well as store sc-elements incident to a given sc-connector as close to it as possible.
- If the number of logical connections between the elements of the sc-construction does not exceed the number of available physical communication channels of the processor element and the sc-graph is planar (although the sc-graph is not a classical graph, we can talk about its planarity by analogy with the planarity of classical graphs), then it is possible to write the sc-construction to the processor-memory in such a way that the configuration of *logical communication channels* mutually uniquely corresponds to some subset of physical communication channels. Thus, it is relevant to develop algorithms

for optimal “stacking” of sc-graphs into processor-memory to ensure the subsequent efficiency of message transmission between processor elements.
- Since the configuration of *logical communication channels* changes during the processing of sc-constructions, it is also advisable to talk about the development of algorithms for repositioning (“defragmentation”) of the sc-construction already recorded in the processor-memory in order to ensure the subsequent efficiency of message transmission. Such reallocation can be performed, for example, according to a schedule during a period when the processor-memory is not used for solving other problems.
- In addition, if there is a hardware capability, the *physical communication channels* can also be re-switched in order to approximate their configuration to the configuration of *logical communication channels*.

Let us consider an example of the optimal variant of writing the simplest *five-element sc-construction* into the proposed processor-memory within the fine-grained architecture of *associative semantic computers*.

In Figure 2, the record of some *five-element sc-construction* in the SCg-code is shown.

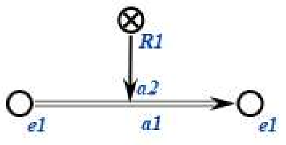

Figure 2. SCg-text. Example of a *five-element sc-construction*

In Figure 3, an incidence graph for the same *five-element sc-construction* is shown, which allows reducing the sc-construction to a classical graph with two types of connections. For clarity, the syntactic types of the corresponding sc-elements are not shown in the Figure.

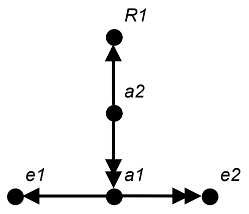

Figure 3. Incidence graph for a *five-element sc-construction*

In Figure 4, one of the possible optimal options for recording the resulting incidence graph into processor-memory is shown. Dotted lines show *physical communication channels* between processor elements, solid lines show *physical communication channels* corresponding to *logical communication channels*. Note that it is advisable to record element ***R1*** in the *processor element* adjacent to the *processor element* storing element ***e1*** or element ***e2***, as shown in the Figure. Due to this, the processor elements storing the specified sc-elements are directly connected by a *physical communication channel*, which simplifies communication in the case of sending messages via *physical communication channels* without taking into account *logical communication channels*.

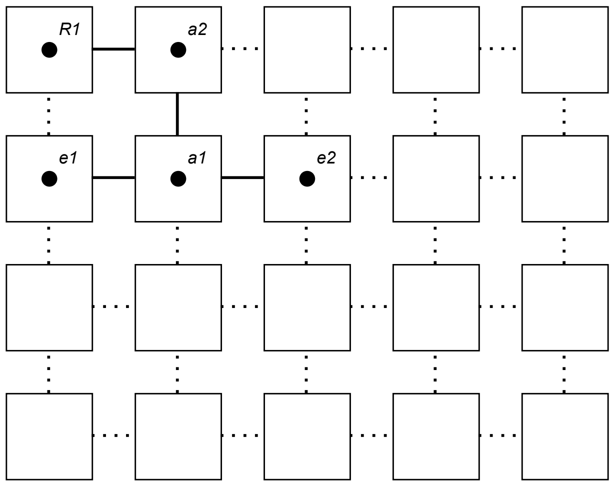

Figure 4. Example of stacking an sc-construction into a processor-memory

VI. Conclusion

In the article, the disadvantages of the currently dominant von Neumann architecture of computer systems as a basis for building-up intelligent computer systems of a new generation are considered, the analysis of modern approaches to the development of hardware architectures that eliminate some of these disadvantages is carried out, the need for the development of fundamentally new hardware architectures representing a hardware implementation of ostis-platforms – *associative semantic computers* – is demonstrated.

The general principles underlying *associative semantic computers* are proposed, three possible variants of the architecture of such computers are considered, their advantages and disadvantages are represented.

Further development of the approaches proposed in the work requires solving a number of problems, both technical and organizational ones:
- development of a wave language for recording micro-programs, that are exchanged between processor elements and run by these processor elements;
- development of a language for writing programs for controlling the exchange of micro-programs and managing the queue of micro-programs;
- organization of active participation of specialists in the field of microelectronics in clarifying the principles of implementation of processor elements

and processor-memory in general, clarifying the element base and lower-level architectural features of *associative semantic computers*;
- development of algorithms for optimizing the ways of recording sc-constructions to processor-memory and repositioning already recorded sc-constructions in order to ensure the subsequent efficiency of message transmission between processor elements;
- clarification of the typology of information processes in the processor-memory, their features, and the corresponding typology of labels;
- clarification of the principles of implementing multi-agent knowledge processing within the processor-memory, in particular, the development of principles for implementing event-based information processing in such memory.

Acknowledgment

The author would like to thank the research groups of the Departments of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics and the Brest State Technical University for their help in the work and valuable comments.

References

[1] J. von Neumann, "First draft of a report on the EDVAC," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[2] M. D. Godfrey and D. F. Hendry, "The computer as von Neumann planned it," *IEEE Annals of the History of Computing*, vol. 15, no. 1, pp. 11–21, 1993. [Online]. Available: https://doi.org/10.1109/85.194088

[3] V. Glushkov [et al.], *Rekursivnye mashiny i vychislitel'naya tekhnika [Recursive machines and computer engineering]*. IC of the Academy of Sciences of the Ukrainian SSR, 1974.

[4] J. Aylif, *Principy postroeniya bazovoj mashiny [Principles of building the basic machine]*. Mir, 1973.

[5] D. Moldovan, W. Lee, C. Lin, and M. Chung, "SNAP: Parallel Processing Applied to AI," *Computer*, pp. 39–49, 1992.

[6] Y. Chu, "Evolution of computer memory structure," *Proc. National Computer Conf. AFIPS Press*, pp. 733–748, 1976.

[7] L. Kalinichenko and V. Ryvkin, *Mashiny baz dannyh i znanij [Database and knowledge machines]*. M. : Nauka, 1990, (In Russ.).

[8] J. Martin, *Organizaciya baz dannyh v vychislitel'nyh sistemah: Per. s angl. [Organization of databases in computing systems: Trans. from English]*. Mir, 1980.

[9] E. Ozkarahan, *Mashiny baz dannyh i upravlenie bazami dannyh [Database machines and database control]*. Mir, 1989.

[10] T. Kohonen, *Associativnaya pamyat': Per. s angl. [Associative memory: Trans. from English]*. Mir, 1980, (In Russ.).

[11] V. Ignatushchenko, "K postanovke zadachi povysheniya effektivnosti vychislitel'nyh sistem na osnove associativnyh metodov obrabotki informacii [To the formulation of the problem of increasing the efficiency of computing systems based on associative methods of information processing]," *Voprosy kibernetiki. Mnogoprocessornye vychislitel'nye sistemy s perestraivaemoj strukturoj (Arhitektura. Struktura. Primeneniya) [Questions of cybernetics. Multiprocessor computing systems with a tunable structure (Architecture. Structure. Applications)]*, pp. 14–21, 1981.

[12] S. Berkovich, Yu. Kochin, and V. Molchanov, "Ob effektivnosti primeneniya associativnoj pamyati v vychislitel'nyh procedurah [On the effectiveness of using associative memory in computational procedures]," *Vychislitel'nye sistemy. Vyp. 62. [Computing systems. Issue 62.]*, pp. 97–105, 1975.

[13] M. Aizerman, L. Gusev, S. Petrov, and I. Smirnova, "Dinamicheskij podhod k analizu struktur, opisyvaemyh grafami (osnovy grafodinamiki) [Dynamic approach to the analysis of structures described by graphs (fundamentals of graph dynamics)]," *Avtomatika i telemekhanika [Automation and telemechanics]*, no. 7/8, pp. 135–151/123–136, 1977.

[14] G. Marchuk [et al.], *Modul'naya asinhronnaya razvivayushchayasya sistema: V 2-h ch. [Modular asynchronous developing system: In 2 parts]*. Academy of Sciences of the USSR. Sib.department. Computing Center, 1978.

[15] I. Prangishvili and G. Stetsyura, "Sovremennoe sostoyanie problemy sozdaniya EVM s netradicionnoj strukturoj i arhitekturoj, upravlyaemyh potokom dannyh [Current state of the problem of creating computers with an unconventional structure and architecture controlled by the data flow]," *Izmerenie, kontrol', avtomatizaciya [Measurement, control, automation]*, no. 1, pp. 36–48, 1981.

[16] Y. Zatuliver and I. Medvedev, "Voprosy postroeniya i mnogoprocessornoj realizacii yazyka strukturno-parallel'nogo programmirovaniya s upravleniem potokami dannyh [Issues of building and multiprocessor implementation of a structurally parallel programming language with data flow control]," *Voprosy kibernetiki. Mnogoprocessornye vychislitel'nye sistemy s perestraivaemoj strukturoj (Arhitektura. Struktura. Primeneniya) [Issues of cybernetics. Multiprocessor computing systems with a tunable structure (Architecture. Structure. Applications)]*, pp. 123–166, 1981.

[17] W.B. Ackerman, "Data flow language," *Proc. National Computer Conf. AFIPS Press*, pp. 1087–1095, 1979.

[18] G. Myers, *Arhitektura sovremennyh EVM: V 2-h kn. Kn. 2. / Per. s angl. [Architecture of modern computers: In 2 books. Book 2. / Transl. from English.]*. Mir, 1985.

[19] V. Glushkov, "Fundamental'nye issledovaniya i tekhnologiya programmirovaniya [Fundamental research and programming technology]," *Programmirovanie [Programming]*, no. 2, pp. 3–13, 1980.

[20] V. Glushkov, S. Pogrebinsky, and Z. Rabinovich, "O razvitii struktur mul'tiprocessornyh EVM s interpretaciej yazykov vysokogo urovnya [On the development of multiprocessor computer structures with interpretation of high-level languages]," *Upravlyayushchie mashiny i sistemy [Control machines and systems]*, no. 6, pp. 61–66, 1978.

[21] Z. Rabinovich, "O koncepcii mashinnogo intellekta [About the concept of machine intelligence]," *Kibernetika i sistemnyj analiz [Cybernetics and system analysis]*, no. 2, pp. 163–173, 1995.

[22] I. Zadykhailo [et al.], *Proekt associativnogo parallel'nogo processora na CMD, orientirovannogo na podderzhku relyacionnyh baz dannyh [Project of an associative parallel processor on the CMD, focused on the support of relational databases]*. Academy of Sciences of the USSR. Institute of applied mathematics, 1979.

[23] S. Schuster, H. Nguyen, E. Oskarachan, K. Smith, "RAP.2 – an associative processor for databases and its applications," *IEEE Trans. on Computers*, no. 6, pp. 446–458, 1979.

[24] E. Suvorov and Ya. Fet, "Processory baz dannyh [Database processors]," *Iss. of the USSR Academy of Sciences. Tech. Cybernet.*, no. 6, pp. 63–75, 1985.

[25] H.J. Brukle, "High level language oriented hardware and post – von Neumann era," *Proc. 5-th Symp Computer Architecture*, pp. 60–65, 1978.

[26] Y. Chu, "Architecture of a hardware data interpreter," *Proc. 4-th IEEE Symp. on Computer Architecture*, pp. 1–9, 1977.

[27] T. Kohonen, *Associativnye zapominayushchie ustrojstva: Per. s angl. [Associative storage devices: Trans. from English.]*. Mir, 1982.

[28] K. Foster, *Associativnye parallel'nye processory [Associative parallel processors]*. Energoizdat, 1981.

[29] A. Ershov, *Algoritmy, matematicheskoe obespechenie i arhitektura mnogoprocessornyh vychislitel'nyh sistem [Algorithms, mathematical support, and architecture of multiprocessor computing systems]*. Nauka, 1982.

[30] L. Berstein, V. Lisyak, and V.Rabinovich, "Odnorodnaya programmiruemaya struktura dlya resheniya kombinatorno-logicheskih zadach na grafah i gipergrafah [Homogeneous programmable

structure for solving combinatorial logic problems on graphs and hypergraphs]," *Metody rascheta i avtomatizaciya proektirovaniya ustrojstv v mikroelektronnyh CVM [Calculation methods and automation of device design in microelectronic digital computers]*, pp. 39–52, 1975.

[31] V. Vasiliev and E. Raldugin, *Elektronnye modeli zadach na grafah [Electronic models of graph problems]*. Naukova dumka [Scientific thought], 1987.

[32] P. Sapaty, "Aktivnoe informacionnoe pole kak model' strukturnogo resheniya zadach na grafah i setyah [Active information field as a model of structural problem solving on graphs and networks]," *Iss. of the USSR Academy of Sciences. Tech. Cybernet.*, no. 5, pp. 184–208, 1984.

[33] A. Popov, "Primenenie geterogennoj vychislitel'noj sistemy s naborom komand diskretnoj matematiki dlya resheniya zadach na grafah [Application of a heterogeneous computing system with a set of discrete mathematics commands for solving graph problems]," *Informacionnye tekhnologii [Information technologies]*, vol. 25, no. 11, pp. 682–690, 2019, (In Russ.).

[34] A. Popov, "Principy organizacii geterogennoj vychislitel'noj sistemy s naborom komand diskretnoj matematiki [Principles of organization of a heterogeneous computing system with a set of discrete mathematics commands]," *Informacionnye tekhnologii [Information technologies]*, vol. 26, no. 2, pp. 67–79, 2020, (In Russ.).

[35] J. Zhang, S. Khoram, and J. J. Li, "Boosting the Performance of FPGA-based Graph Processor using Hybrid Memory Cube: A Case for Breadth First Search," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[36] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2021.

[37] W. S. Song, V. Gleyzer, A. Lomakin, and J. Kepner, "Novel graph processor architecture, prototype system, and results," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Sep. 2016. [Online]. Available: https://doi.org/10.1109/hpec.2016.7761635

[38] I. V. Afanasyev, V. V. Voevodin, K. Komatsu, and H. Kobayashi, "VGL: a high-performance graph processing framework for the NEC SX-aurora TSUBASA vector architecture," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 8694–8715, Jan. 2021. [Online]. Available: https://doi.org/10.1007/s11227-020-03564-9

[39] M. Weinzweig, *Obuchayushchayasya sistema iskusstvennogo intellekta s associativnoj pamyat'yu-processorom [Learning Artificial Intelligence system with associative memory processor]*. USSR Academy of Sciences, Scientific Council. according to the complex. probl. "Cybernetics", 1980.

[40] M. Weinzweig and M. Polyakova, "Mekhanizm myshleniya i modelirovanie ego raboty v real'nom vremeni [Mechanism of thinking and modeling its work in real time]," *Intellektual'nye processy i ih modelirovanie [Intelligent processes and their modeling]*, pp. 208 – 229, 1987.

[41] S. Somsubhra, "Reconfigurable semantic processor," Oct 2006.

[42] Z. Rabinovich, "Nekotoryj bionicheskij podhod k strukturnomu modelirovaniyu celenapravlennogo myshleniya [Some bionic approach to structural modeling of purposeful thinking]," *Kibernetika [Cybernetics]*, no. 2, pp. 115–118, 1979.

[43] ——, "Razvitie struktur universal'nyh EVM v svyazi s problemami avtomatizacii nauchnyh issledovanij [Development of universal computer structures in connection with the problems of automation of scientific research]," *Avtomatika [Automation]*, no. 5, pp. 63–72, 1979.

[44] V. Gladun, *Evristicheskij poisk v slozhnyh sredah [Heuristic search in complex environments]*. Naukova dumka [Scientific thought], 1977.

[45] ——, *Planirovanie reshenij [Planning solutions]*. Naukova dumka [Scientific thought], 1987.

[46] N. Amosov, A. Kasatkin, L. Kasatkina, and S. Talaev, *Avtomaty i razumnoe povedenie [Automata and reasonable behavior]*. Naukova dumka [Scientific thought], 1973.

[47] E. Zolotov and I. Kuznetsov, *Rasshiryayushchiesya sistemy aktivnogo dialoga [Expanding active dialogue systems]*. Nauka, 1982.

[48] A. Galushkin, "Sovremennye napravleniya razvitiya nejrokomp'yuternyh tekhnologij v Rossii [Modern directions of development of neurocomputer technologies in Russia]," *Otkrytye sistemy [Open systems]*, no. 4, pp. 25–28, 1997, (In Russ.).

[49] R. Hecht-Nielsen, "Nejrokomp'yuting: istoriya, sostoyanie, perspektivy [Neurocomputing: history, state, prospects]," *Otkrytye sistemy [Open systems]*, no. 4, pp. 23–28, 1998.

[50] L. Komartsova and A. Maksimov, *Nejrokomp'yutery. - 2-e izd. [Neurocomputers. – 2nd ed.]*, ser. Informatika v tekhnicheskom universitete [Computer Science at the Technical University]. Moscow: Bauman Moscow State Technical University, 2004.

[51] (2022, Dec) USB Accelerator | Coral. [Online]. Available: https://coral.ai/products/accelerator/

[52] M. Moussa, A. Savich, and S. Areibi, "Architecture, system and method for artificial neural network implementation," Jun 2013.

[53] "Nejromorfnyj processor "Altaj" Neuromorphic processor "Altai"," mode of access: https://motivnt.ru/neurochip-altai/. — Date of access: 29.03.2023.

[54] J. D. Allen, J. Philip, and L. Butler, "Parallel machine architecture for production rule systems," Jun 1989.

[55] "CUDA Toolkit," mode of access: https://developer.nvidia.com/cuda-toolkit. — Date of access: 29.03.2023.

[56] "OpenCL," mode of access: https://www.khronos.org/opencl/. — Date of access: 29.03.2023.

[57] H.-N. Tran and E. Cambria, "A survey of graph processing on graphics processing units," *The Journal of Supercomputing*, vol. 74, no. 5, pp. 2086–2115, Jan. 2018. [Online]. Available: https://doi.org/10.1007/s11227-017-2225-1

[58] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua, "Graph processing on GPUs," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–35, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3128571

[59] Y. Lü, H. Guo, L. Huang, Q. Yu, L. Shen, N. Xiao, and Z. Wang, "GraphPEG," *ACM Transactions on Architecture and Code Optimization*, vol. 18, no. 3, pp. 1–24, Sep. 2021. [Online]. Available: https://doi.org/10.1145/3450440

[60] V. Golenkov, "Strukturnaya organizaciya i pererabotka informacii v elektronnyh matematicheskih mashinah, upravlyaemyh potokom slozhnostrukturirovannyh dannyh [Structural organization and processing of information in electronic mathematical machines controlled by the flow of complex structured data]," PhD diss.: 05.13.01 ; 05.13.13, Minsk, 1996.

[61] ——, *Parallel'nyj grafovyj komp'yuter (PGC), orientirovannyj na reshenie zadach iskusstvennogo intellekta, i ego primenenie (Preprint; No 2) [Parallel graph computer (PGC), focused on solving artificial intelligence problems, and its application (Preprint; No. 2)]*. Institute of Technical Cybernetics, 1994.

[62] V. Golenkov [et al.], *Terminal'nyj modul' parallel'nogo grafovogo komp'yutera (PGC): Interfejs s pol'zovatelem i processornymi modulyami, struktura, logicheskaya organizaciya: materialy po matematicheskomu obespecheniyu EVM [Terminal module of a parallel graph computer (PGC): User interface and processor modules, structure, logical organization: materials on computer mathematical support]*. Institute of Technical Cybernetics, 1994.

[63] V. Golenkov, "Grafodinamicheskie modeli i metody parallel'noj asinhronnoj pererabotki informacii v intellektual'nyh sistemah [Graphodynamic models and methods of parallel asynchronous processing of information in intelligent systems]," Doct. diss.: 05.13.11 ; 05.13.17, Minsk, 1996, 396 p.

[64] P. Gaponov, "Modeli i metody parallel'noj asinhronnoj pererabotki informacii v grafodinamicheskoj associativnoj pamyati [Models and methods of parallel asynchronous processing of information in graphodynamic associative memory]," PhD diss.: 05.13.11, Minsk, 2000, 114 p.

[65] V. Kuzmitsky, "Principy postroeniya grafodinamicheskogo parallel'nogo komp'yutera, orientirovannogo na reshenie zadach iskusstvennogo intellekta [Principles of building a graphodynamic parallel computer focused on solving artificial intelligence problems]," PhD diss.: 05.13.11, Minsk, 2000, 236 p.

[66] R. Serdyukov, “Bazovye algoritmy i instrumental’nye sredstva obrabotki informacii v grafodinamicheskih associativnyh mashinah [Basic algorithms and tools for information processing in graphodynamic associative machines],” PhD diss.: 05.13.11, Minsk, 2004, 114 p.
[67] V. Golenkov, N. Gulyakina, and D. Shunkevich, *Standart otkrytoj tekhnologii ontologicheskogo proektirovaniya, proizvodstva i ekspluatacii semanticheski sovmestimyh gibridnyh intellektual’nyh komp’yuternyh sistem [Standard of the open technology for ontological design, production, and operation of semantically compatible hybrid intelligent computer systems]*, V. Golenkov, Ed. Minsk: Bestprint, 2021.
[68] J. von Neumann, *Teoriya samovosproizvodyashchihsya avtomatov [Theory of self-reproducing automata]*. M.: Mir, 1971, (In Russ.).
[69] D.A. Moon, “Symbolics architecture,” *Computer*, vol. 20, no. 1, pp. 43–52, 1987. [Online]. Available: doi:10.1109/MC.1987.1663356
[70] S. Smith, “The LMI Lambda Technical Summary. Technical report, LMI Inc.” Los Angeles, CA, 1984.
[71] G. L. Steele and W. D. Hillis, “Connection Machine Lisp: fine-grained parallel symbolic processing,” in *Proceedings of the 1986 ACM conference on LISP and functional programming (LFP ’86)*. New York: ACM, 1986, pp. 279–297.
[72] P. McJones. (2018, Dec.) Parallel Lisps: Connection Machine Lisp (StarLisp). Computer History Museum. Mode of access: https://www.softwarepreservation.org/projects/LISP/parallel# Connection_Machine_\protect\discretionary{\protect\protect\leavevmode@ifvmode\kern+.1667em\relax\OMS/cmsy/m/n/8\char2}{}{}Lisp_(StarLisp). — Date of access: 29.12.2018.
[73] V. van der Leun, *Introduction to JVM Languages*. Packt Publishing, Jun. 2017.
[74] V. Ivashenko, “String processing model for knowledge-driven systems,” *Doklady BGUIR*, vol. 18, pp. 33–40, 10 2020.
[75] B. Rasheed and A. Popov, “Network Graph Datastore Using DISC Processor,” in *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, Jan. 2019, pp. 1582–1587.
[76] E. Dubrovin and A. Popov, “Graph representation methods for the discrete mathematics instructions set computer,” in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, Jan. 2020, pp. 1925–1930.
[77] C. Hewitt. Middle History of Logic Programming: Resolution, Planner, Prolog and the Japanese Fifth Generation Project. Mode of access: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=07A4074D9BBEC56C92085D21A10A6B4D?doi=10.1.1.363.8517&rep=rep1&type=pdf. — Date of access: accessed 20.03.2020.
[78] V. Ivashenko, *Modeli resheniya zadach v intellektual’nyh sistemah. V 2 ch. CH. 1 : Formal’nye modeli obrabotki informacii i parallel’nye modeli resheniya zadach : ucheb.-metod. posobie [Problem-solving models in intelligent systems. In 2 p. P. 1 : Formal models of information processing and parallel problem-solving models: study guide]*. Minsk: BSUIR, 2020, (In Russ.).
[79] “LegUp High-Level Synthesis,” mode of access: http://legup.eecg.utoronto.ca/. — Date of access: 29.03.2023.
[80] “VHDPlus,” mode of access: https://vhdplus.com/. — Date of access: 29.03.2023.
[81] “SystemC Community Portal,” mode of access: https://systemc.org/. — Date of access: 29.03.2023.
[82] “MyHDL,” mode of access: https://www.myhdl.org/. — Date of access: 29.03.2023.
[83] P. Sapaty, “Yazyk VOLNA-0 kak osnova navigacionnyh struktur dlya baz znanij na osnove semanticheskih setej [language VOLNA-0 as the basis of navigation structures for knowledge bases based on semantic networks],” *Iss. of the USSR Academy of Sciences. Technical cybernetics*, no. 5, pp. 198–210, 1986.
[84] D. I. Moldovan and Y.-W. Tung, “SNAP: A VLSI architecture for artificial intelligence processing,” *Journal of Parallel and Distributed Computing*, vol. 2, no. 2, pp. 109–131, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0743731585900310
[85] A. Letichevsky, Yu. Kapitonova, V. Volkov, V. Vyshemirsky, and A. Letichevsky (j.), “Insercionnoe programmirovanie [Insertion programming],” *Kibernetika i sistemnyj analiz [Cybernetics and system analysis]*, no. 1, pp. 19–32, 2003.
[86] A. Letichevsky, “Insercionnoe modelirovanie [Insertion modeling],” *Upravlyayushchie sistemy i mashiny [Control systems and machines]*, no. 6, pp. 3–14, 2012.
[87] J. Backus, “Can programming be liberated from the von Neumann style?” *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978. [Online]. Available: https://doi.org/10.1145/359576.359579
[88] V. Kotov and A. Nariniani, “Asinhronnye vychislitel’nye processy nad obshchej pamyat’yu [Asynchronous computing processes over shared memory],” *Kibernetika [Cybernetics]*, no. 3, pp. 64–71, 1966.

Ассоциативные семантические компьютеры для интеллектуальных компьютерных систем нового поколения

Голенков В. В., Шункевич Д. В.,
Гулякина Н. А., Ивашенко В. П.,
Захарьев В. А.

В работе рассмотрены недостатки доминирующей в настоящее время фон-Неймановской архитектуры компьютерных систем в качестве основы для построения интеллектуальных компьютерных систем нового поколения, проведен анализ современных подходов к разработке аппаратных архитектур, устраняющих некоторые из указанных недостатков, обоснована необходимость разработки принципиально новых аппаратных архитектур, представляющих собой аппаратный вариант реализации платформы интерпретации систем, построенных на базе Технологии OSTIS, — *ассоциативных семантических компьютеров*.

Предложены общие принципы, лежащие в основе *ассоциативных семантических компьютеров*, рассмотрены три возможных варианта архитектуры таких компьютеров, представлены их достоинства и недостатки.

Received 13.03.2023