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

- Each processor element can send messages (microprograms) 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 communica- tion 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 comingfrom 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 processormemory 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 proces-

sor 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 any parallelism models taking into account the problem being solved. In support of this thesis, we can cite the theory of A-systems escribed 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 Asystems 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 processormemory;
  - 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 processormemory 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 semantics 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