OSTIS Research Papers Collection

BSUIR's editorial board

2023

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 Distributed, 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 microprograms 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

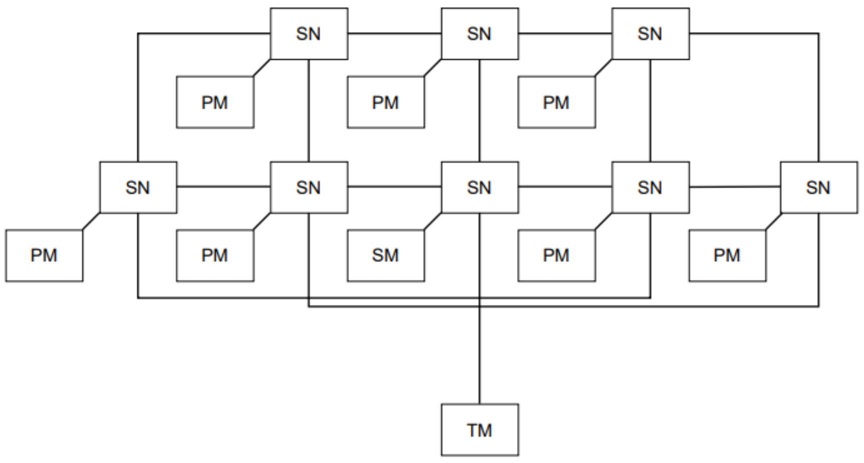

Figure 1: Example of implementing the coarse-grained architecture

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 microprograms, 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