Information systems. Computer sciences. Issues of information security

Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.9:681.3
https://doi.org/10.32362/2500-316X-2025-13-6-7-24
EDN WGZAHH

RESEARCH ARTICLE

Organization and study of cluster computing systems with functional architecture determined by executable models. Automata executable models of information processing

Grigory V. Petushkov @

MIREA – Russian Technological University, Moscow, 119454 Russia
@ *Corresponding author, e-mail: petushkov@mirea.ru*

• **Submitted:** 04.08.2025 • **Revised:** 12.09.2025 • **Accepted:** 06.10.2025

Abstract

Objectives. An urgent task is to improve the functional architecture of cluster computing systems by introducing methodologies for creating software at the applied and intermediate levels based on formalized specifications. One such methodology is based on the use of automatic specifications for computer systems software. The complexity of resolving the problem is caused by the branching of the algorithms built, as well as the presence of cyclic sections. The execution time of the branched sections of the program and the number of cycles run depends on the type of conditions entered. In practice it can be determined using a detailed simulation model and analysis of the control program created on its basis. The aim of the work is to find approaches to the definition of functional architecture which can be applied practically at the main levels of the subject orientation of cluster computing systems.

Methods. The methods proposed and used are based on the concept of organization and research of cluster-type computing systems with a functional architecture as defined by executable automatic models.

Results. The paper proposes methods of constructing automatic and logical-probabilistic models of cluster computing systems and creating software tools based on them. The concept of the logical-probabilistic model "temporal probabilistic system of canonical equations (CES)" is introduced. This enables a visual formalization to be obtained, as well as implementation of automatic models and work programs typical for cluster and other applications. It also significantly reduced the number of "incremental" additions when enumerating discrete time moments. The main feature of the new logical-probabilistic model is the preservation of the original CES in its basis.

Conclusions. The work concludes that the choice of the system and functional architecture of a computing cluster should be determined not so much by the peak characteristics of the communication equipment specified by the manufacturer, as by the actual indicators achieved at the level of user applications and cluster usage modes. It is also shown that executable automatic models can be applied at almost all levels of cluster computing systems subject orientation.

Keywords: cluster computing system, intermediate level application, functional architecture, finite automaton models, logical-probabilistic model, logical-algebraic model, query processing modes, simulation results

For citation: Petushkov G.V. Organization and study of cluster computing systems with functional architecture determined by executable models. Automata executable models of information processing. *Russian Technological Journal.* 2025;13(6):7–24. https://doi.org/10.32362/2500-316X-2025-13-6-7-24, https://www.elibrary.ru/WGZAHH

Financial disclosure: The author has no financial or proprietary interest in any material or method mentioned.

The author declares no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Организация и исследование кластерных вычислительных систем с функциональной архитектурой, определяемой исполнимыми моделями. Автоматные исполнимые модели обработки информации

Г.В. Петушков [@]

МИРЭА – Российский технологический университет, Москва, 119454 Россия
[@] *Автор для переписки, e-mail: petushkov@mirea.ru*

• **Поступила:** 04.08.2025 • **Доработана:** 12.09.2025 • **Принята к опубликованию:** 06.10.2025

Резюме

Цели. Актуальной является задача совершенствования функциональной архитектуры кластерных вычислительных систем за счет внедрения методологий создания программного обеспечения прикладного и промежуточного уровней на основе формализованных спецификаций. Одна из таких методологий основана на использовании автоматных спецификаций программного обеспечения вычислительных систем. Сложность решения задачи вызвана разветвленностью построенных алгоритмов, а также наличием циклических участков. Время выполнения разветвленных участков программы и число проходимых циклов зависят от вида вводимых условий и на практике могут быть определены при помощи детальной имитационной модели и анализа созданной на ее основе управляющей программы. Цель работы – нахождение подходов к определению функциональной архитектуры, которые возможно применять практически на основных уровнях предметной ориентации кластерных вычислительных систем.

Методы. Предлагаемые и использованные методы основаны на концепции организации и исследования вычислительных систем кластерного типа с функциональной архитектурой, определяемой исполнимыми автоматными моделями.

Результаты. Предложены методы построения автоматных и логико-вероятностных моделей кластерных вычислительных систем и создания на этой основе программных средств. Вводится понятие логико-вероятностной модели «темпоральная вероятностная система канонических уравнений», которая позволит получить наглядную формализацию и реализацию автоматных моделей и рабочих программ, характерных для кластерных и других приложений, и в существенной степени сократить число «инкрементных» сложений при перечислении моментов дискретного времени. Главной особенностью новой логико-вероятностной модели является сохранение в ее основе исходной системы канонических уравнений.

Выводы. Сделан вывод о том, что выбор системной и функциональной архитектуры вычислительного кластера должен определяться не столько указанными производителем пиковыми характеристиками коммуникационной аппаратуры, сколько реальными показателями, достигаемыми на уровне приложений пользователей и режимов использования кластера. Показано, что исполнимые автоматные модели могут применяться практически на всех уровнях предметной ориентации кластерных вычислительных систем.

Ключевые слова: вычислительная система кластерного типа, приложение промежуточного уровня, функциональная архитектура, логико-вероятностная модель, логико-алгебраическая модель, режимы обработки запросов, результаты моделирования

Для цитирования: Петушков Г.В. Организация и исследование кластерных вычислительных систем с функциональной архитектурой, определяемой исполнимыми моделями. Автоматные исполнимые модели обработки информации. *Russian Technological Journal*. 2025;13(6):7–24. https://doi.org/10.32362/2500-316X-2025-13-6-7-24, https://www.elibrary.ru/WGZAHH

Прозрачность финансовой деятельности: Автор не имеет финансовой заинтересованности в представленных материалах или методах.

Автор заявляет об отсутствии конфликта интересов.

INTRODUCTION

Clustering is one of the most modern trends in the field of computing systems development. The emergence of cluster computing systems is due to advances in network technologies, most often local ones. When connecting machines into a cluster, computers are combined using network technologies based on bus architecture or a switch. This has led to an increase in the number of computing clusters purchased or leased as cloud services [1]. According to forecasts by a number of marketing companies, the cluster computing market is expected to grow to USD102.4 bn by 2032.[1]

The scope of application of clusters in the organization of information and subject-oriented systems used for the collection, processing, and subsequent analysis of information is constantly expanding. At the same time, the limitations of simple homogeneous cluster systems complicate the creation of systems which provide a high level of structural and functional dynamics and effective problem orientation based on the development of the *middleware* level software.

The rapid development of applications based on machine learning and artificial intelligence has created a need to train a large number of models. At the current time, one of the most powerful supercomputers in the world is the Colossus supercomputer cluster based on Nvidia graphics processing units (Nvidia Corporation, USA). This cluster can theoretically achieve a performance of about 497.9 exaflops (497900000 teraflops), setting new standards in supercomputing power. The goal of xAI (USA) is to increase the number of graphics processing units (GPUs) in Colossus to 1 million in the coming years.[2] Currently, the xAI supercluster has begun training a large language model (LLM) artificial intelligence system using more than 200000 Nvidia H100, H200, and GB200 graphics processing units optimized for deep learning neural network tasks. The cluster network is based on a high-speed Nvidia Spectrum-X Ethernet switch with a bandwidth of up to 800 Gb/s.[3]

The functional architecture of computing clusters is based on the coordinated operation of the following components: workflow management system; cluster monitoring system; libraries for parallel processing; cluster management tools; global process space connecting all cluster nodes; resource management system; network (possibly parallel) file system; and network services, including cloud services, providing access to the cluster for many users [1]. It is assumed that current issues in the field of high-performance computing will remain relevant in the future: the need for further significant increases in parallelism and data transfer speeds; the development of high-performance computing architecture and technology; the trend towards workflows and use cases extending beyond data centers; the existence of many powerful scientific and industrial drivers; and the transition from high-performance computing as isolated systems to high-performance infrastructures [2].

An important step in the development of science and industry is linked to the development and

[1] Cluster Computing Market Overview. https://www.marketresearchfuture.com/reports/cluster-computing-market-1746. Accessed June 02, 2025.

[2] Tyson M. Elon Musk fires up ‘the most powerful AI cluster in the world’ to create the ‘world’s most powerful AI’ by December – system uses 100000 Nvidia H100 GPUs on a single fabric. Published July 22, 2024. https://www.tomshardware.com/pc-components/gpus/elon-musk-fires-up-the-most-powerful-ai-training-cluster-in-the-world-uses-100000-nvidia-h100-gpus-on-a-single-fabric. Accessed June 02, 2025.

[3] Half a million GPUs in four months: how Musk is building the world’s most powerful cluster. https://www.braintools.ru/article/18041. Accessed June 02, 2025 (in Russ.).

implementation of the ELBJUWEL supercomputer with artificial intelligence (AI) capabilities.[4] The efforts of the developers are focused on creating a unique innovative platform which will combine expertise in the field of AI and high-performance computing. The works [3–5] are devoted to describing the needs for high-performance computing when solving machine learning problems.

The next problem faced by supercomputing centers is the inefficient use of resources for high-performance computing when resolving certain computational tasks. Such tasks can block valuable computing resources and slow down calculations for other users. In order to address this issue, the National Research University Higher School of Economics has developed a task monitoring system for the cHARISMa high-performance computing cluster which automatically generates conclusions about their performance [6]. This university has accumulated extensive experience in using the supercomputer complex based on the cHARISMa cluster to resolve tasks for various categories of users. These tasks include: searching, analyzing, and forecasting data on social networks [7]; researching machine learning models for predicting the risks of major cardiovascular events in patients with myocardial infarction and different genotypes [8]; and many others.

Additional information on existing software packages in cluster systems is provided in [9–11].

Russian cluster projects include the MVS-100K supercomputer installed at the Interdepartmental Supercomputer Center of the Russian Academy of Sciences and the Lomonosov supercomputer installed at the Research Computing Center of Lomonosov Moscow State University as part of the SKIF project[5]. The "Chervonenkis," "Galushkin," and "Lyapunov" supercomputers, created by Yandex, also have a cluster architecture[6]. They run on Nvidia A100 graphics accelerators (Nvidia A100 GPUs with tensor cores) with an InfiniBand communication system based on Mellanox switches (Israel)[7].

Many issues related to the computing resources required by ordinary users and organizations arise in connection with the organization and use of computing clusters. Therefore, the review of literature must be supplemented with an analysis of some characteristic foreign sources. Articles [12, 13] note the shortcomings of cluster computing systems. Some of these shortcomings contradict the advantages which can be explained by the specifics of enterprises and users. Clusters are difficult to manage without experience and given a large cluster size, it will be difficult to detect a malfunction.

The problem with troubleshooting arises because the user is dealing with a single entity, and when a malfunction is detected, it is unclear which component is causing the problem.

The following circumstance can also be attributed to the disadvantages of cluster computing systems [14]. Clusters are not suitable for commercial and business use by all consumers, as they require special programming skills, knowledge of systems and programming languages that are not widely used for commercial purposes. Personnel are required to have special technical skills for operation and administration.

A large number of the medium-cost and low-cost computing clusters considered are based on various types of switches, including Infiniband and Ethernet switches. In the example of the computing cluster and its infrastructure as shown in Fig. 1, traffic from different local networks can intersect if this does not interfere with the main function of the cluster nodes. Cluster nodes N_1–N_{16} process user load; U_1 and U_2 are control nodes which monitor the status of the cluster's hardware and software and take action to reconfigure it in response to any event occurring in the cluster; M_1 and M_2 are shared backup storage devices. They store information accessible to all cluster nodes and used by them to access shared data, including data about a failed node, which can be used by a backup node. S_1 and S_2 are servers accessible via public and client networks. The private network L_2 level switch exchanges data between cluster nodes using hardware MAC[8] addresses. Command messages used by nodes to check the cluster's operability, reconfigure it, and synchronize it are transmitted over the private network.

The L_3 level switch of the public network exchanges data using IP[9] or hardware MAC addresses. At the public network level, access to the cluster is virtualized as a single system. A local network built on the basis of an L_{2+} level switch with added features provides client access to the cluster. The presence of several network switches in the computing cluster infrastructure enables the use of three main types of networks: communication, transport, and service [15].

[4] *ParTec AG*: *A More Efficient Supercomputer for the AI Revolution*. Frankfurt, Bloomberg; 2024. 43 p.

[5] Center for Collective Use of Ultra-High-Performance Computing Resources at Lomonosov Moscow State University. https://parallel.ru/cluster. Accessed June 02, 2025 (in Russ.).

[6] Chernyavtseva V. *Yandex has created three of Russia's most powerful supercomputers*. https://nplus1.ru/news/2021/11/15/chervonenkis. Accessed June 02, 2025 (in Russ.).

[7] Russia suddenly burst into the world's top most powerful supercomputers. https://www.cnews.ru/news/top/2021-11-16_rossijskie_superkompyutery. Accessed June 02, 2025 (in Russ.).

[8] Media Access Control.

[9] Internet Protocol Address is a unique numerical device identifier.

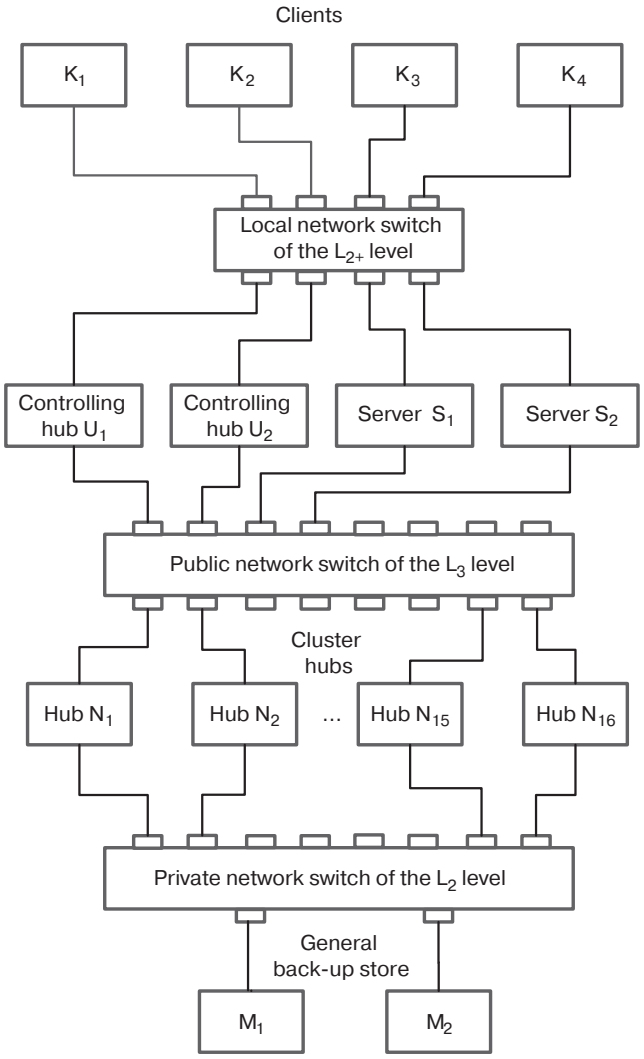


Fig. 1. Option for organizing a computing cluster and its infrastructure

In order to solve the pressing tasks set out in this paper—organizing an effective functional architecture for clusters by creating new application-class and *middleware*-class software—it is important to focus on existing, well-developed software: message processing services—*message-oriented middleware*, services that provide big data analytics and connection to data storage—*data warehousing and big data analytics* data warehousing and big data analytics, and protocols and products that provide *interprocess communications* [15].

1. AUTOMATIC MODELS OF INTERMEDIATE-LEVEL CLUSTER APPLICATIONS

A computing system in operation [1] is defined at an abstract level as a set of functional devices operating in time. When assessing the quality of operation, it is proposed that the content of the operations performed be abstracted and the operation of functional devices in the time reference system be considered. Therefore, it will be useful to construct formal models for analyzing the functioning of computing clusters. In addition, as follows from the "Computing and Cluster Systems" course [15], "in practice, the peak characteristics of communication equipment specified by the manufacturer are not as important as the actual performance achieved at the user application level." This statement implies that the choice of the system and functional architecture of a computing cluster should be determined mainly by the applications and modes of use of the cluster, including those implemented at the *middleware* level. Therefore, part of the application software and middleware can be conditionally considered as system software that determines the functionality of the entire computer cluster.

The main effect of interpreting the models proposed is the possibility that they can be used as formalized specifications when describing parallel processes in cluster computing systems and networks at the level of tasks, data, algorithms, and machine instructions, i.e., at the basic levels of abstraction—from conceptual representation to implementation details. The selection of the following model examples based on program diagrams is based on compliance with a high level of generality. The algorithms must contain all basic algorithmic constructs which enable the implementation of sequences, branches, and cycles. It must be possible to reinterpret types of parallelization—at the task level, at the data level, at the algorithm level, and at the machine-level command level, with the possibility of alternating sequential single-threaded parts of the program with multi-threaded parallel sections.

However, it is only possible to investigate the actual operation of applications on a working cluster. The problem can be resolved at the preliminary stages with less effort and expense by using executable formal models, on the basis of which simulation models of the cluster's operation should be constructed. These models may include characteristic or simplified fragments of real applications.

At this stage of model construction, the semantics of data and operations are not considered, i.e., preserving the generality of the models, the meanings of variables and operation symbols are not interpreted. It is assumed that the methods for creating and interpreting models can be further used in the creation of working interpreted applications when the cluster is put into operation. In this case, formal models can play the role of formalized specifications.

Convenient models for subsequent use for these purposes are: graph-scheme algorithm language (GSA), finite automata, and logic-algebraic models based on

first-order predicate logic. This subsection proposes using Moore's finite partial automaton model[10] [16]. This model is also well known from works in the field of microprogramming [17, 18]. Figures 2 and 3 show examples of GSA selected to illustrate the creation of application models: GSA_1 and GSA_2. The main criteria for selection are the usual requirements for GSA correctness and the presence of sequences of operators and branches. GSA_1 (Fig. 2) contains operator vertices (hereinafter simply operators) $A_0, A_1, A_2, \ldots, A_{16}, A_{27}, A_K$. In addition, GSA_1 contains parallel fragments represented by structured operators C_1, C_2, …, C_8, each of which corresponds to an "internal" copy of GSA_2 (Fig. 3); each copy, or clone, contains local operators $A_{17}, A_{18}, A_{19}, \ldots, A_{25}, A_{26}$.

Both GSAs contain conditional vertices (hereinafter referred to as logical conditions) $x_1, x_2, \ldots, x_5$ (GSA_1) and $x_6, x_7, \ldots, x_{10}$ (GSA_2). Condition symbols are treated as names of unary predicates. The values of logical conditions—0 (true) or 1 (false)—are calculated after the execution of operators, including operators for entering input conditions (input signals, input symbols, or partial automata).

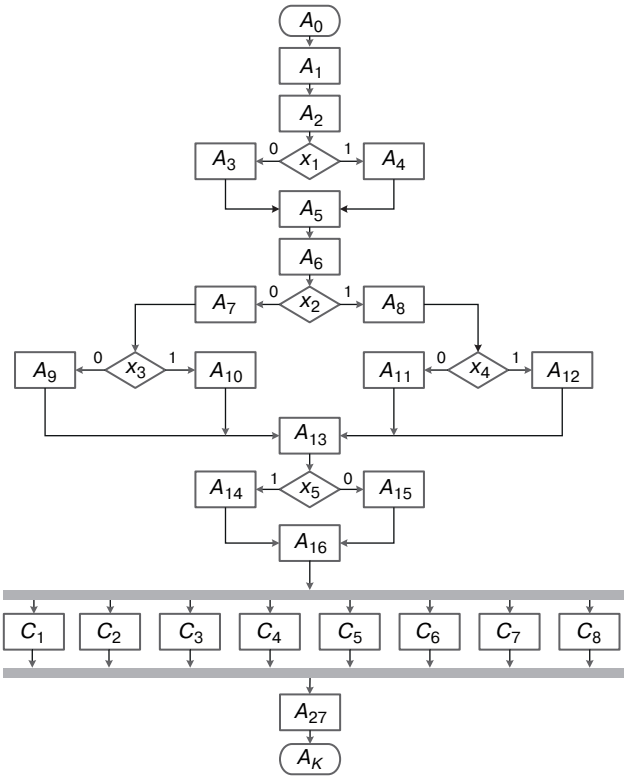


Fig. 2. Flowchart of the GSA_1 algorithm for cluster application operation

The construction and study of automata models will be carried out for the case of SPMD (Single Program, Multiple Data) methods. In the following subsections, other executable models will be constructed based on a logical-algebraic approach: MPMD (Multiple Programs, Multiple Data) and MPSD (Multiple Programs, Single Data) [1]. The latter method is most suitable for pipeline data processing. All these methods are used to achieve parallelism. There are a number of implementation options for these methods used in computing clusters.

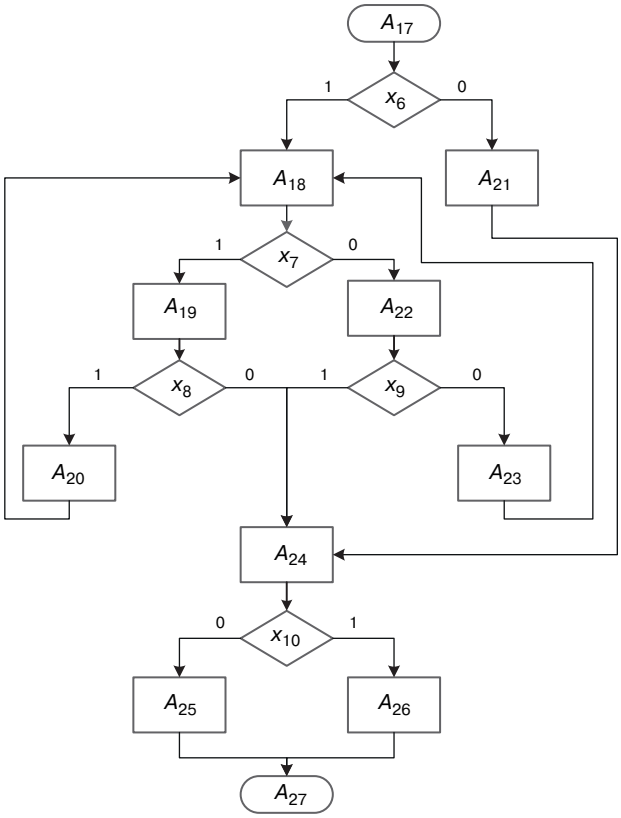


Fig. 3. Block diagram of the GSA_2 algorithm for a single copy of a parallel application section for a cluster

Canonical equation systems (CES) [18, 19], which describe transitions from one state to another, were chosen as the initial language for specifying partial automata. Permissible parallel transitions correspond, for example, to the representation of parallel sections in modified logical diagrams of algorithms, known from works on microprogramming [20]. In these diagrams, parallel sections are considered to be private logical diagrams of algorithms and enable for simple reinterpretation into the graphical form of GSA. Parallel GSA languages were also used in works [21, 22]. The structuring of hierarchical automata states was proposed earlier in a number of works [23–25].

The following concepts are used in the proposed CES models. Operators are assigned a one-to-one

[10] Gurenko V.V. *Introduction to Automata Theory*. Moscow: Bauman Moscow State Technical University; 2013. https://rusist.info/book/10028635?ysclid=mf5p2z07v616437010. Accessed June 02, 2025 (in Russ.).

correspondence with so-called private events, represented by unary predicates of the form $A_i(t)$, defined on the set of discrete time values t. Partial input variables, or input conditions, are represented by unary predicates of the form $x_j(t)$, also defined on the set of discrete time values t. Unary predicates of the form $z_k(t)$, are also introduced to take true values only after the corresponding events of the form $A_k(t)$ have already occurred. This corresponds to the fact that the operator A_k has completed its work. Thus, when $z_k(t) = 0$ (false), the event $A_k(t)$ is preserved, and when $z_k(t) = 1$ (true), it is not preserved. The first condition for the event $A_k(t)$ enables its execution to be extended, and when the second, opposite condition is fulfilled, the event $A_k(t)$ is completed. The condition for the onset of an event corresponds to the transition from the preceding event. The remaining features of the CES construction can be conveniently explained using examples of the transition from GSA to CES.

Figure 4 shows the state transition graph of the sequential application's automaton model, constructed by transitioning from GSA_1 and GSA_2. Section C_0 is highlighted, intended for subsequent cloning when transitioning to the application simulation corresponding to the SPMD cluster operating mode. This graph, as will be required later, can also be viewed as a sequential composition of two partial automata: the first automaton corresponds to states 1–16, and the second to states 17–27.

Figures 2 and 3 use standard GSA notation for logical conditions: $x_1, x_2, \ldots, x_{10}$, which are also considered in the CES entry-level language for specifying partial finite automata as names of unary predicates. In Fig. 4 and further in Fig. 5, other names are used for the three input variables. These are also convenient for further testing of applications using partial automaton analysis: x (value x = true after the end of any operator without calculation or without entering a condition value); nx (value nx = true if, after the end of the operator, a false condition value is calculated or entered, otherwise nx = false); and yx (value yx = true if, after the end of the operator, a true condition value is calculated or entered, otherwise yx = false). The locations for calculating and checking these variables are uniquely determined by the location of the operator. If necessary, this can be used for the usual numbering of logical conditions $x_1, x_2, \ldots, x_{10}$ and used further in the compilation of CES and logic-algebraic expressions. In Fig. 4, the first state is designated as the initial, starting (Start) state, and the 28th as the final, ending (End).

A system of canonical equations can be considered as a system of production rules designed to represent knowledge in automata models. Productions can be used to represent knowledge that can take the form of rules such as "*premise* → *conclusion*, *condition* → *action*." The left side of the rule is called the antecedent, and

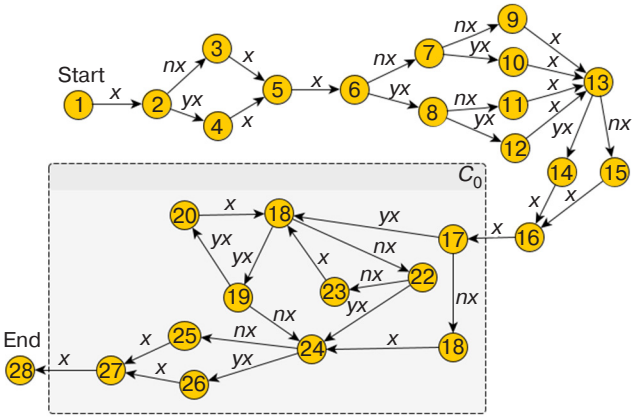


Fig. 4. State transition graph of the sequential application automaton model; section C_0, is highlighted, intended for subsequent cloning when transitioning to SPMD mode

the right side is called the consequent. The antecedent is the premise of the rule (the conditional part) and consists of elementary statements using the logical symbols AND, OR, NOT. The consequent (conclusion) includes one or more statements which express either a certain fact or an indication of a specific action to be performed [26–28]. A set of productions forms a production system for which special procedures are specified for selecting productions and executing one or another production from among those selected.

A distinctive feature of CES, considered as a type of production system, is the placement of the conditional part (antecedent) on the right, and the action or conclusion (consequent) on the left. Therefore, the direction of the conclusion is from right to left. This is largely due to the theory and practice of microprogrammed automata synthesis and the representation, along with canonical equations, of excitation functions of elementary automata (D-triggers, or delay elements) in the unitary encoding of the states of a finite partial automaton [17, 18, 29]. In the future, the concept of "product" will also be used in the construction of logical-algebraic models of cluster-type systems.

2. AUTOMATIC CES MODEL OF THE SEQUENTIAL PART OF THE APPLICATION

Figure 4 shows the state transition graph of the sequential application's automaton model. Some of the states $a_1, a_2, \ldots, a_{16}, a_{28}$ were obtained by marking the states of the Moore automaton on GSA_1, shown in Fig. 2. The rest of the states $a_{17}, a_{18}, \ldots, a_{27}$ belong to the subgraph C_0, which was constructed by marking the states of the Moore automaton on GSA_2.

At this stage, the recurrent predicate equations of the $\boldsymbol{SP}_{\mathbf{Seq}}$ CES for GSA_1 are compiled without taking

into account the structured operators C_1, C_2, …, C_8 and, accordingly, without the structured states a_{17}, a_{18}, …, a_{27} of the automaton, i.e., the model currently covers only operators A_0, A_1, …, A_{16}:

$$A_0(t+1) = A_0(t) \,\&\, \neg x_0(t) \vee x_{\text{begin}}(t);$$
$$A_1(t+1) = A_0(t) \,\&\, x_0(t) \vee A_1(t) \,\&\, \neg z_1(t);$$
$$A_2(t+1) = A_1(t) \,\&\, z_1(t) \vee A_2(t) \,\&\, \neg z_2(t);$$
$$A_3(t+1) = A_2(t) \,\&\, z_2(t) \,\&\, \neg x_1(t) \vee A_3(t) \,\&\, \neg z_3(t);$$
$$A_4(t+1) = A_2(t) \,\&\, z_2(t) \,\&\, x_1(t) \vee A_4(t) \,\&\, \neg z_4(t);$$
$$A_5(t+1) = A_3(t) \,\&\, z_3(t) \vee A_4(t) \,\&\, z_4(t) \vee A_5(t) \,\&\, \neg z_5(t);$$
$$A_6(t+1) = A_5(t) \,\&\, z_4(t) \vee A_6(t) \,\&\, \neg z_6(t);$$
$$A_7(t+1) = A_6(t) \,\&\, z_6(t) \,\&\, \neg x_2(t) \vee A_7(t) \,\&\, \neg z_7(t);$$
$$A_8(t+1) = A_6(t) \,\&\, z_6(t) \,\&\, x_2(t) \vee A_8(t) \,\&\, \neg z_8(t);$$
$$A_9(t+1) = A_7(t) \,\&\, z_7(t) \,\&\, \neg x_3(t) \vee A_9(t) \,\&\, \neg z_9(t);$$
$$A_{10}(t+1) = A_7(t) \,\&\, z_7(t) \,\&\, x_3(t) \vee A_{10}(t) \,\&\, \neg z_{10}(t);$$
$$A_{11}(t+1) = A_8(t) \,\&\, z_8(t) \,\&\, \neg x_4(t) \vee A_{11}(t) \,\&\, \neg z_{11}(t);$$
$$A_{12}(t+1) = A_8(t) \,\&\, z_8(t) \,\&\, x_4(t) \vee A_{12}(t) \,\&\, \neg z_{12}(t);$$
$$A_{13}(t+1) = A_9(t) \,\&\, z_9(t) \vee A_{10}(t) \,\&\, z_{10}(t) \vee A_{11}(t) \,\&\, z_{11}(t) \vee A_{12}(t) \,\&\, z_{12}(t) \vee A_{13}(t) \,\&\, \neg z_{13}(t);$$
$$A_{14}(t+1) = A_{13}(t) \,\&\, z_{13}(t) \,\&\, x_5(t) \vee A_{14}(t) \,\&\, \neg z_{14}(t);$$
$$A_{15}(t+1) = A_{13}(t) \,\&\, z_{13}(t) \,\&\, \neg x_5(t) \vee A_{15}(t) \,\&\, \neg z_{15}(t);$$
$$A_{16}(t+1) = A_{14}(t) \,\&\, z_{14}(t) \vee A_{15}(t) \,\&\, z_{15}(t) \vee A_{16}(t) \,\&\, \neg z_{16}(t),$$

where $x_{\text{begin}}(t)$ is the input variable ("signal").

Copies (or clones) of the application module compiled according to the $\boldsymbol{SP}_{\text{Seq}}$ CES for GSA_1 are loaded onto all computing nodes of the cluster and executed in parallel mode, processing the same data or entering data of the same type. The automatic model assumes different execution times for events corresponding to application operators. In the further description of the CES equations, for the sake of brevity the terms "event" and "state" will be considered synonymous.

Below are descriptions of some key equations from the given CES. The initial equation has the following form:

$$A_0(t+1) = A_0(t) \,\&\, \neg x_0(t) \vee x_{\text{begin}}(t).$$

According to this equation, when the true value of the input variable $x_{\text{begin}}(t)$ = true appears in the automaton, the initial event $A_0(t+1)$ = true is set in the next cycle, which corresponds to its inception. This event is retained as long as the condition for its retention $A_0(t) \,\&\, \neg x_0(t)$ at $A_0(t+1)$ = true and $x_0(t)$ = false. Further, when the input signal $x_0(t)$ = true is received, the true condition $A_0(t) \,\&\, x_0(t)$ for the initiation of a new event $A_1(t+1)$ is formed in the automaton:

$$A_1(t+1) = A_0(t) \,\&\, x_0(t) \vee A_1(t) \,\&\, \neg z_1(t).$$

This event persists as long as the condition $A_1(t) \,\&\, \neg z_1(t)$ for its persistence is true. It will end with ($A_1(t+1)$ = false, i.e., this statement will become false) when operator A_1 generates the sign $z_1(t)$ = true indicating the end of its work. As can be seen from the recursive predicate equations of this $\boldsymbol{SP}_{\text{Seq}}$ CES, the event of establishing the truth of the antecedent (right statement) occurs at a fixed moment in time t, and the event of establishing the truth of the consequent (left statement) occurs at the next moment in time ($t+1$).

3. AUTOMATIC CES MODEL OF ONE OF THE PARALLEL SECTIONS (CLONES) OF THE APPLICATION

Figure 4 shows the state transition graph of the sequential application's automaton model; section C_0 is highlighted. This is intended for subsequent cloning when transitioning to SPMD mode.

Recursive predicate equations of CES $\boldsymbol{MD}_{\text{Clon}}$ for GSA_2:

$$A_{17}(t+1) = A_{16}(t) \,\&\, z_{16}(t) \vee A_{17}(t) \,\&\, \neg z_{17}(t);$$
$$A_{18}(t+1) = A_{17}(t) \,\&\, z_{17}(t) \,\&\, x_6(t) \vee A_{20}(t) \,\&\, z_{20}(t) \vee \vee A_{23}(t) \,\&\, z_{23}(t) \vee A_{18}(t) \,\&\, \neg z_{18}(t);$$
$$A_{19}(t+1) = A_{18}(t) \,\&\, z_{18}(t) \,\&\, x_7(t) \vee A_{19}(t) \,\&\, \neg z_{19}(t);$$
$$A_{20}(t+1) = A_{19}(t) \,\&\, z_{19}(t) \,\&\, x_8(t) \vee A_{20}(t) \,\&\, \neg z_{20}(t);$$
$$A_{21}(t+1) = A_{17}(t) \,\&\, z_{17}(t) \,\&\, \neg x_6(t) \vee A_{21}(t) \,\&\, \neg z_{21}(t);$$
$$A_{22}(t+1) = A_{18}(t) \,\&\, z_{18}(t) \,\&\, \neg x_7(t) \vee A_{22}(t) \,\&\, \neg z_{22}(t);$$
$$A_{23}(t+1) = A_{22}(t) \,\&\, z_{22}(t) \,\&\, \neg x_9(t) \vee A_{23}(t) \,\&\, \neg z_{23}(t);$$
$$A_{24}(t+1) = A_{19}(t) \,\&\, z_{19}(t) \,\&\, \neg x_8(t) \vee A_{22}(t) \,\&\, z_{22}(t) \,\&\, x_9(t) \vee A_{21}(t) \,\&\, \neg z_{21}(t) \vee A_{24}(t) \,\&\, \neg z_{24}(t);$$
$$A_{25}(t+1) = A_{24}(t) \,\&\, z_{24}(t) \,\&\, \neg x_{10}(t) \vee A_{25}(t) \,\&\, \neg z_{25}(t);$$
$$A_{26}(t+1) = A_{24}(t) \,\&\, z_{24}(t) \,\&\, x_{10}(t) \vee A_{26}(t) \,\&\, \neg z_{26}(t);$$
$$A_{27}(t+1) = A_{25}(t) \,\&\, z_{25}(t) \vee A_{26}(t) \,\&\, z_{26}(t) \vee A_{27}(t) \,\&\, \neg z_{27}(t).$$

The shared $\boldsymbol{SP}_{\text{Seq}}$ CES model and a single copy of the $\boldsymbol{MD}_{\text{Clon}}$, CES model, taken together, define a single CES model designated $\boldsymbol{SP}_{\text{Seq}} * \boldsymbol{MD}_{\text{Clon}}$, where the symbol "*" denotes the operation of combining two CESs into one common CES. The state transition graph for this model is shown in Fig. 4. Another interpretation, as mentioned earlier, allows the state transition graph $\boldsymbol{SP}_{\text{Seq}} * \boldsymbol{MD}_{\text{Clon}}$ to be considered as a sequential composition of partial automata.

4. SEQUENTIALLY-PARALLEL COMPOSITION OF AUTOMATONS DETERMINING THE OPERATION OF CLUSTER COMPUTERS

The flowchart of the GSA_1 algorithm in Fig. 2 contains parallel sections, abbreviated as structured operators C_1, C_2, …, C_8. In a complete

single-level representation, each of these operators is replaced by GSA_2 in Fig. 3 while, in the program implementation, it is executed independently of the others on its "own" cluster computer. The complete network CES model of a sequential-parallel network of automata is represented by the following expression:

$$\boldsymbol{SP}_{\mathbf{Seq}} * (\boldsymbol{ParReplicate}(\mathbf{1..8})\boldsymbol{MD}_{\mathbf{Clon}}),$$

wherein ***Replicate*(1..8)** is the inclusion of the CES into the general system 8 times in a row. ***Par*** is the indication that these replicas must be executed in parallel.

The state transition graph for this network of partial automata with structured states C_1–C_8 is shown in Fig. 5. When constructing a network CES model, equations need to be compiled to include structured events. Each structured event represents a nested partial automaton. The use of hierarchical finite automata is a fundamentally important method of software design, and corresponds to the concept of a "subroutine".

The advantage of the method of formalizing algorithms using CESs is the compact logical description of transition functions [16, 17]. Structured events with the same names are introduced in a manner analogous to structured states C_1–C_8. Each structured event is "embedded" in already compiled $\boldsymbol{MD}_{\mathbf{Clon}}$.

The initiation and continuation of events C_1–C_8 (initiation and parallel operation of independent software modules) are described by the following CES $\boldsymbol{S}_{\mathbf{C}}$:

$$\begin{aligned}
C_1(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_1(t)\ \&\ \neg w_1(t);\\
C_2(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_2(t)\ \&\ \neg w_2(t);\\
C_3(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_3(t)\ \&\ \neg w_3(t);\\
C_4(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_4(t)\ \&\ \neg w_4(t);\\
C_5(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_5(t)\ \&\ \neg w_5(t);\\
C_6(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_6(t)\ \&\ \neg w_6(t);\\
C_7(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_7(t)\ \&\ \neg w_7(t);\\
C_8(t+1) &= A_{16}(t)\ \&\ z_{16}(t) \vee C_8(\mathrm{t})\ \&\ \neg w_8(t).
\end{aligned}$$

Each of the events C_1–C_8 originates when the compound statement $A_{16}(t)$ & $z_{16}(t)$ is true, i.e., it is a consequence of the successful completion of the event A_{16}. Each of these events C_i persists until the termination condition $w_i(t)$, i = 1, 2, …, 8 is satisfied. Events C_1–C_8 start simultaneously, but do not necessarily end simultaneously, since the termination conditions may not depend on each other. However, the transition to event A_{28} should only occur after all events C_1–C_8 have been completed. Therefore, the CES model should be followed by events which determine barrier synchronization and consist of waiting for the completion of the events C_1–C_8 in each of the branches, as well as the subsequent initiation and retention of indicator events D_1–D_8 for the termination of all branches.

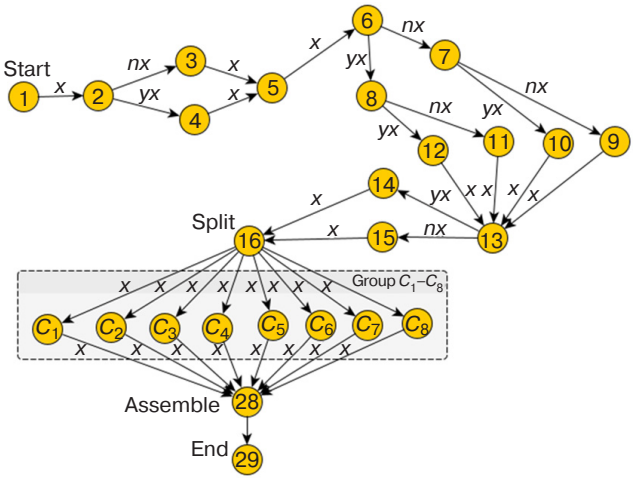


Fig. 5. Sequential-parallel network of automata with nested states of application execution in a cluster in SPMD mode

The system of canonical $\boldsymbol{S}_{\mathbf{D}}$ equations describing the origin and conservation of these events has the following form:

$$\begin{aligned}
D_1(t+1) &= C_1(t)\ \&\ w_1(t) \vee D_1(t)\ \&\ \neg D_9(t);\\
D_2(t+1) &= C_2(t)\ \&\ w_2(t) \vee D_2(t)\ \&\ \neg D_9(t);\\
D_3(t+1) &= C_3(t)\ \&\ w_3(t) \vee D_3(t)\ \&\ \neg D_9(t);\\
D_4(t+1) &= C_4(t)\ \&\ w_4(t) \vee D_4(t)\ \&\ \neg D_9(t);\\
D_5(t+1) &= C_5(t)\ \&\ w_5(t) \vee D_5(t)\ \&\ \neg D_9(t);\\
D_6(t+1) &= C_6(t)\ \&\ w_6(t) \vee D_6(t)\ \&\ \neg D_9(t);\\
D_7(t+1) &= C_7(t)\ \&\ w_7(t) \vee D_7(t)\ \&\ \neg D_9(t);\\
D_8(t+1) &= C_8(t)\ \&\ w_8(t) \vee D_8(t)\ \&\ \neg D_9(t).
\end{aligned}$$

The following single equation, additionally designated as the CES $\boldsymbol{D}_{\mathbf{9}}$, system, describes the expectation of the occurrence of all indicator events D_1–D_8 for the completion of parallel work on all branches:

$$\begin{aligned}
D_9(t+1) = D_1(t)\ \&\ D_2(t)\ \&\ D_3(t)\ \&\ D_4(t)\ \&\ D_5(t)\ \&\\
\&\ D_6(t)\ \&\ D_7(t)\ \&\ D_8(t) \vee D_9(t)\ \&\ \neg v_9(t).
\end{aligned}$$

The equations describing the transition to the final events A_{28}, A_{29} and further to event A_1 of the control module are as follows:

$$\begin{aligned}
A_{28}(t+1) &= D_9(t)\ \&\ v_9(t) \vee A_{28}(t)\ \&\ \neg z_{28}(t);\\
A_{29}(t+1) &= A_{28}(t)\ \&\ z_{28}(t) \vee A_{29}(t)\ \&\ \neg z_{29}(t);\\
A_1(t+1) &= A_{29}(t)\ \&\ z_{29}(t) \vee A_1(t)\ \&\ \neg z_1(t).
\end{aligned}$$

In order to use the equations for the events A_{28} and A_{29} in a combined expression for a network of partial automata, they are designated as separate CESs $\boldsymbol{A_{28}}$ and $\boldsymbol{A_{29}}$, respectively.

In order to implement a detailed network CES model for executing an application in a cluster in sequential-parallel processing (SPMD) mode, all equations must be combined in the following sequence:

$$\text{SPMD: } \boldsymbol{SP_{Seq}} * \boldsymbol{S_C} * \boldsymbol{S_D} * \boldsymbol{D_9} * \boldsymbol{A_{28}} * \boldsymbol{A_{29}},$$

moreover, the initiation of the previously described parallel replicas $\boldsymbol{ParReplicate(1..8)MD_{Clon}}$ is performed upon the occurrence of the events C_1–C_8, determined by the CES $\boldsymbol{S_C}$ subsystem. The completion of these replicas occurs upon the occurrence of the events D_1–D_8, determined by the CES $\boldsymbol{S_D}$ subsystem. When the replicas are running, the range of variation of the variable t, which counts system time, is expanded.

The resulting general CES model of the network of application execution machines in a cluster in SPMD sequential-parallel processing mode belongs to the class of executable models. It is easily programmable in algorithmic languages containing message passing operators, as well as in assembly language for microcontrollers and microprogramming language. A simulation model is built on its basis, enabling the functioning of the cluster system to be studied at the micro level.

The networks of partial finite automata considered in this work consist of automata which simulate application of software modules of a computing cluster connected by a message passing interface at the inputs and outputs. Each module can receive a message at the input which transmits control with data, process it, and transmit a control message with data to the next module. Other types of inter-module interaction in the cluster are not considered. Therefore, issues of automata composition and other methods of constructing complex automata from simple ones are not taken into account here [30].

Automatic programming is currently considered one of the technologies to significantly reduce the time required to write programs and simplify their testing [31]. Systems of canonical equations also enable the creation of application, intermediate, and system-level programs based on them.

As is well known, mutual blocking, ambiguity, and deadlock and other configurations lead to violations of GSA correctness. Therefore, the work proposes to resolve the problem of checking the graph scheme for correctness on an abstract model of algorithm interactions—on Petri nets [32, 33]. Methods for transitioning from a parallel GSA to a Petri net are given, for example, in [21, 22].

5. FORMALIZATION OF LOGICAL-PROBABILISTIC MODELS OF PARTIAL AUTOMATON NETWORKS CREATED BASED ON THE CES LANGUAGE

The concept of a logical-probabilistic model such as "temporal probabilistic CES" (TPCES) will enable the visual formalization and implementation of automatic models and working programs characteristic of cluster and other applications, for example, with pipeline parallelism. This will also significantly reduce the number of "incremental" additions when listing discrete time moments:

$$\text{TPCES} = (\text{CES}_0, \text{S}, \text{X}, \text{T}_\text{X}, \text{W}_\text{TX}, \text{T}_\text{Z}, \text{W}_\text{TZ}),$$

wherein network CES_0 is the initial language adopted for describing the CES finite state machine network, limited by the description of partial machines and characterized by cluster and pipeline computing systems, mainly by the sequential execution of events in time. Furthermore, only simple parallelism of events is enabled without interaction between copies of CES_0 branches, ending with barrier synchronization of branches. S is a finite set of events $\{S_0(t), S_1(t), \ldots, S_n(t)\}$ specified by unary predicates. X is a finite set of input events specified by unary predicates $\{X_0(t), X_1(t), \ldots, X_m(t)\}$. T_X is a finite set of random time intervals $\{t_{x0}, t_{x1}, \ldots, t_{xm}\}$ from the current moments to the moments of occurrence of input events X. W_TX is a finite set of probability distribution functions of the form $P\{t_{xk} = i\} = p_{ki}$, $i = 0, 1, \ldots, i_k$, of random time intervals from the set T_X. T_Z is a finite set of random time intervals $\{t_{z0}, t_{z1}, \ldots, t_{zn}\}$ of preservation. Thus, the occurrence of events from the set S; W_TZ is a finite set of probability distribution functions of the form $P\{t_{zr} = j\} = p_{rj}, j = 0, 1, \ldots, j_r$, of random time intervals from the set T_Z.

Random variables (pseudorandom variables in software implementations) t_x and t_z only take non-negative integer values. Only finite probability distributions of integer random variables are considered. It is also possible to use integer constants as values for t_{Xk}, $k = 0, 1, \ldots, m$ and t_{zr}, $r = 0, 1, \ldots, n$.

The structure of an application containing sequential and independent parallel sections forming a network of automata may enable for a deeper level of nesting, which is characteristic of most cluster computing systems.

The initial non-interpretability of the TPCES model enables it to be applied at the level of programs, program modules, operators, down to the level of machine commands and microprograms.

The general approach to developing a statistically executable model of a cluster application is as follows. The method of organizing a sequence of events is

used, in which periods of event generation alternate with periods of event preservation. Let us assume, for example, that for a sequential section of an application formalized, for example, by the CES $\boldsymbol{SP}_{\mathbf{Seq}}$ system, the actions specified by the following three equations are performed:

$$A_2(t+1) = A_1(t)\ \&\ z_1(t) \vee A_2(t)\ \&\ \neg z_2(t);$$
$$A_3(t+1) = A_2(t)\ \&\ z_2(t)\ \&\ \neg x_1(t) \vee A_3(t)\ \&\ \neg z_3(t);$$
$$A_4(t+1) = A_2(t)\ \&\ z_2(t)\ \&\ x_1(t) \vee A_4(t)\ \&\ \neg z_4(t).$$

As determined when constructing any equation, the onset of the next event, for example, event $A_2(t + 1)$, occurs at time $(t + 1)$. For this event to occur at the specified moment in time then at the previous moment in time t the statements $A_1(t)$ and $z_1(t)$ need to be true—i.e., event $A_1(t)$ would have occurred for the last time at this moment, which would be indicated by the appearance of the true value of the statement $z_1(t)$—the end of the action of the event $A_1(t)$. From the moment of time $(t + 1)$, event $A_2(t + 1)$ begins and continues until statement $z_2(t)$ is false. The appearance of the true value of the statement $z_2(t)$ will lead the compound statement $A_2(t)\ \&\ \neg z_2(t)$ being become false, and the condition for the continuation of event $A_2(t)$ not to be fulfilled. At the next moment in time, the event $A_2(t)$ will occur for the last time, and if the statement $A_2(t)\ \&\ z_2(t)\ \&\ \neg x_1(t)$ is true, then the event $A_2(t)$ will begin to occur (will originate) at the next moment in time $(t + 1$". The preservation of the event $A_2(t)$ can be "extended" by moving to the time mark $t = t + t_{z2}$ of its end, where the value of the time interval t_{z2} is determined using a pseudo-random number generator with a given distribution law. Acting in a similar way, the inception of the event $A_3(t + 1)$ can be delayed by delaying the action of input variable $x_1(t)$ by the amount of the time interval t_{x1}. This is achieved by moving to the time mark of the event $t = t + t_{x1}$ corresponding to the activation of the variable $x_1(t)$. The value of the variable t_{x1} is set by a pseudo-random number generator. The compound statement $A_2(t)\ \&\ z_2(t)\ \&\ \neg x_1(t)$ will become true, and then event $A_3(t + 1)$ will occur at the next moment in time $(t + 1)$. The preservation of the event A_3 and the initiation and preservation of the event A_4 occur in a similar manner. Thus, by transitioning from event to event, the logical-probabilistic model of the cluster application is implemented.

Figure 5 shows a network automaton model of application execution in a cluster in SPMD sequential-parallel mode. With the accepted mode of operation of independent parallel programs in the parallel section of the SPMD mode, independent $\boldsymbol{MD}_{\mathbf{Clon}}$-type CES models can be used to represent structured events C_1–C_8.

6. RESULTS OF STATISTICAL EXPERIMENTS WITH CLUSTER SYSTEM MODELS IN PARALLEL-SERIAL SPMD

Simulation statistical models of application execution were constructed on the basis of automatic probabilistic CES models. The table shows the numerical values of the acceleration coefficient of the application executed in SPMD mode on a computing cluster, obtained from the constructed simulation model. Following [1], acceleration is taken as the ratio of the application execution time on one node $(t_{seq} + Nt_N)$ to the sum $(t_{seq} + t_N)$ of the execution time of the same application in parallel mode on all N nodes of the cluster t_N, and the time t_{seq} of a single execution of a sequential section of the application:

$$k = (t_{seq} + Nt_N) / (t_{seq} + t_N).$$

It was assumed that parallel sections are executed independently of each other. Since the execution times of sequential and parallelized sections are unknown in advance, their statistical characteristics are determined by performing a statistical experiment. Therefore, in the given formula, t_{seq} and t_N are estimates of the mathematical expectations of these time intervals. In the simulation, it was assumed that the execution time of each operator was uniformly distributed from 1 to 9 ms. Transitions between conditions are equally probable (0.5 each).

The results are summarized in the table. The entries in the column headers indicate that the acceleration coefficient k is calculated at the specified time value $T = t_{seq}$, expressed in units of model time (here in milliseconds, ms).

Table. Estimates of the calculation acceleration coefficient k determined using statistical models

N	$T = 0$	$T = 10$	$T = 20$	$T = 30$	$T = 40$	$T = 50$
1	1	1	1	1	1	1
2	2	1.98	1.96	1.94	1.92	1.9
4	4	3.88	3.77	3.67	3.57	3.48
8	8	7.46	7.0	6.6	6.25	5.93
16	16	13.8	12.25	11.0	10.0	9.25
32	32	24.2	19.6	16.5	14.3	12.6

Examples of dependencies of the acceleration coefficient k on the number of nodes in the cluster are illustrated in Fig. 6.

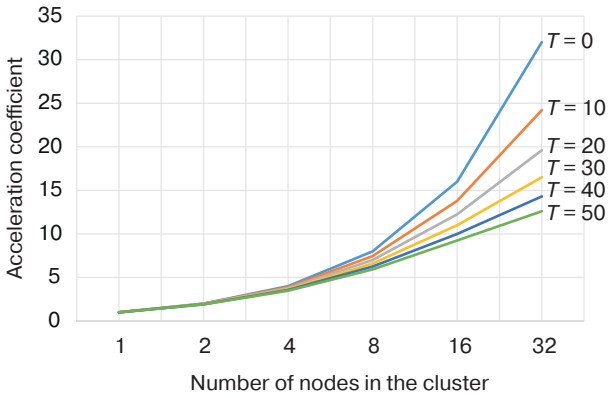


Fig. 6. Results of statistical experiments with cluster models running a sequential-parallel application $L_{Seq} * ParReplicate(i = 1..8)C_i$

At $T = 0$, there is no sequential section, so the acceleration factor values are equal to the number of nodes involved. As values of T increase, the effect of parallelization becomes less pronounced, since the execution time of the sequential section has a greater impact on the result of the acceleration factor calculation.

The complexity of the problem is caused by the branching nature of the selected algorithm, as well as the presence of cycles. The execution time of the branched sections of the program and the number of cycles passed depend on the type of conditions entered and, in practice, can be determined using a detailed simulation model. Both automaton CES models and models based on logical algebraic expressions are executable models, since in order to study the properties and dynamic behavior of the modeled object, the model must be "executed," i.e., run on a computer and the processes of event change must be studied.

The acceleration coefficient k was calculated based on the results of statistical modeling. For example, Fig. 7 shows a histogram of the distribution of the execution time $t_{ex} = t_{seq} + Nt_N$ of the application without parallelization at $N = 1$.

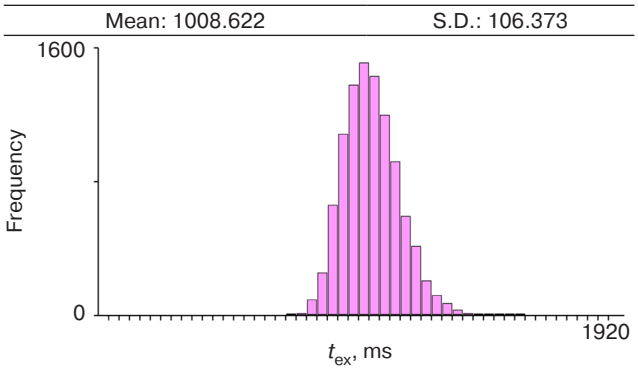


Fig. 7. Histogram of the execution time distribution $t_{ex} = t_{seq} + 32t_{32}$ of the application at $N = 1$ without parallelization; the abscissa axis shows the frequency class boundaries for the histogram, the histogram step is 40 ms; the ordinate axis shows the number of hits in each frequency class with a sample size of 10000

Here, $Mean1 = t_{seq} + 32t_{32} = 1008.622$ ms is the estimate of the mathematical expectation of the application execution time without parallelization.

Figure 8 shows a histogram of the application execution time distribution for $N = 32$ with a sequential section and parallelization. Here, $Mean2 = t_{seq} + t_{32} = 80.266$ ms is the estimate of the mathematical expectation of the application execution time with a sequential section and parallelization of the rest. The execution time of the sequential section was determined in the same experiment and is equal to $t_{seq} = 49.991$ ms.

Then $k = (t_{seq} + 32t_{32})/(t_{seq} + t_{32}) = 12.58$. The same result is obtained by calculating the acceleration coefficient using the known formula for Amdahl's second law [1]:

$$k = N / [\beta N + (1 - \beta)] = 12.61$$

at $N = 32$ with a proportion of sequential calculations

$$\beta = t_{seq} / (t_{seq} + 32t_{32}) = 49.991 / 1008.622 = 0.04957.$$

A small error is caused by the use of the static modeling method when evaluating time parameters in the computing cluster model.

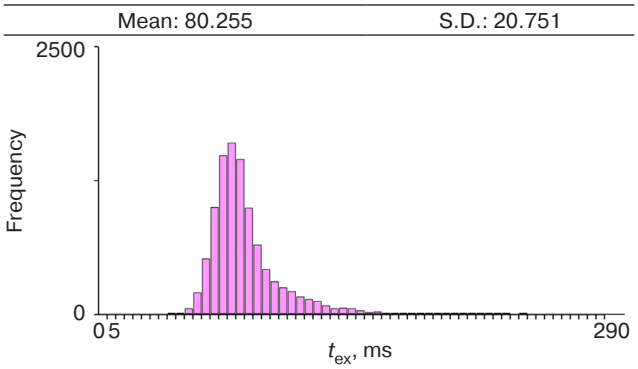


Fig. 8. Histogram of the execution time distribution $t_{ex} = t_{seq} + t_{32}$ of the application at $N = 32$, with a sequential section and parallelization of the rest; the abscissa axis shows the frequency class boundaries for the histogram, the histogram step is 5 ms; the *y*-axis shows the number of hits in each frequency class for a sample size of 10000

7. TRANSITION FROM AUTOMATIC MODELS TO ASYNCHRONOUS LOGICAL-ALGEBRAIC MODELS OF CLUSTER COMPUTING SYSTEMS AT THE *MIDDLEWARE* LEVEL

Logical-algebraic models are based on logical calculi and algebraic systems. Logical calculi include propositional calculus and predicate calculus.

The logical-algebraic operational expression (LAOE) apparatus, based on the integration of a number of models, is described from the perspective of various

applications. It is substantiated in works [34, 35]. However, in the final stages of testing the models by means of studying their dynamics, the direct use of Petri nets is recommended, in this way replacing or removing "innovations" related to the additional use of first-order predicate logic. In order to maintain continuity in the names of the cluster application modules, the names of the operator vertices in GSA_1 (Fig. 2) and GSA_2 (Fig. 3) were chosen for the names of the positions. Double indices were chosen for the indexing of the transitions of the Petri net.

Figure 9 shows examples which illustrate transitions from GSA to the CES model of a partial automaton and further to the initial logical algebraic expressions for a Petri net. Canonical equations have been compiled for the fragments in Figs. 9a and 9b, and logical algebraic expressions have been compiled for the fragment in Fig. 9c.

The system of canonical equations for the examples in Figs. 9a and 9b is as follows:

$$A_3(t+1) = A_2(t)\ \&\ z_2(t) \vee A_3(t)\ \&\ \neg z_3(t);$$
$$A_4(t+1) = A_3(t)\ \&\ z_3(t)\ \&\ \neg x_1(t) \vee A_4(t)\ \&\ \neg z_4(t);$$
$$A_5(t+1) = A_3(t)\ \&\ z_3(t)\ \&\ x_1(t) \vee A_4(t)\ \&\ \neg z_4(t).$$

The CES model is synchronous, and during simulation, the current time of event execution must be counted. This slows down the simulation program. According to the previously introduced designations for input variables in partial automata in Fig. 4 and Fig. 9b, x = true, $nx = \neg x_1$ and $yx = x_1$.

The asynchronous LAOE system shown in Fig. 9c is represented as follows:

$$T_{2,3}: [M(A_2)\&\neg M(A_3)](\{M(A_2) \neg \text{false}, M(A_3) \neg \text{true}\} \vee \vee Ret);$$

$$T_{3,4}: [M(A_3)\ \&\ \neg M(A_4)\ \&\ \neg X(A_3)](\{X(A_3) \neg \text{undef}, M(A_3) \neg \text{false}, M(A_4) \neg \text{true}\} \vee Ret);$$

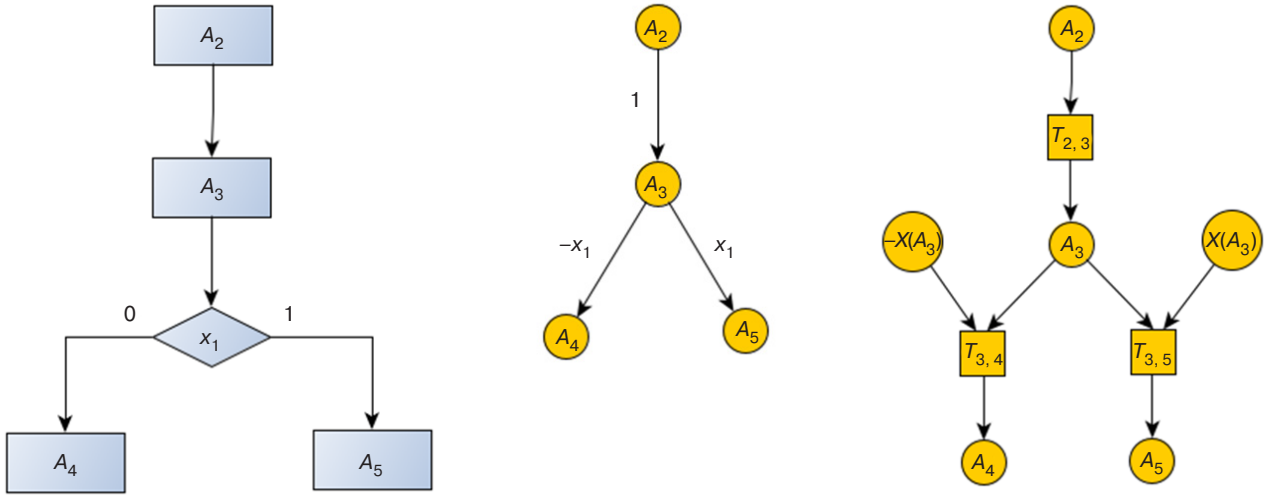
$$T_{3,5}: [M(A_3)\ \&\ \neg M(A_5)\ \&\ X(A_3)](\{X(A_3) \neg \text{undef}, M(A_3) \neg \text{false}, M(A_5) \neg \text{true}\} \vee Ret),$$

wherein undef is an undefined value of a logical condition.

The LAOE expressions given in this case are interpreted as rules for triggering transitions in a Petri net. Here, M is a unary predicate, or a function for marking positions, with the same name as the operators of the original GSA; $M(A_i)$ is a statement, the truth of which corresponds to the presence of one label in position A_i, while falsity corresponds to the absence of a label. X is a unary predicate which defines the conditions in the original GSA. $X(A_i)$ is a statement which takes true, false, or undefined values, determined by the result of executing the operator A_i. The *Ret* operator enhances the procedural component of the LAOE and transitions to its repeated execution when the condition enclosed in square brackets is false.

A logical-algebraic operational model can obviously be constructed using a state transition graph (Fig. 9b), which served as the basis for constructing a Petri net (Fig. 9c). For this purpose, compliance is required with the rule of forming conditions by operators, including condition input operators.

The rules for triggering transitions can be further modified or supplemented in accordance with the requirements of the subject area. Additional events—message transmission, message reception, transmission acknowledgment, event duration determination, represented by binary or ternary predicate modification operations—may not correspond to the generally accepted concepts of Petri nets. Therefore, transition



(a) (b) (c)

Fig. 9. Fragments of the GSA (a), the transition graph of a partial automaton (b), and a Petri net (c)

rules can take the form of more general LAOEs, for which other apparatus is used: apparatus of algorithmic algebra systems [36, 37]; abstract machine networks [38]; relational calculus and algebras [39].

CONCLUSIONS

1. The relevance of the tasks addressed in this article arises from the limitations of simple homogeneous cluster systems which complicate the creation of systems providing a high level of structural and functional dynamics. New approaches to designing the system and functional architecture of computing clusters can be based on organizing the effective use and management of cluster operations, by means of enhanced problem orientation by creating *middleware* applications and software.
2. The method proposed and used in this work is based on the concept of architecture design defined by executable models. This is a type of object-oriented design.
3. A distinctive feature of the methods proposed in this work is the use of automatic, network automata. In the future, logical-algebraic approaches may be implemented, in order to define the system and functional architecture, applied at virtually all levels of subject orientation of cluster computing systems. These will ensure the implementation of the architectural concept formed when creating a cluster system model at various levels of abstraction—from conceptual representation to implementation details.
4. The work shows that the main effect of interpreting the proposed automata models and methods is the possibility of their use as formalized specifications when describing parallel processes in cluster computing systems at the level of tasks, data, algorithms, and machine instructions.
5. The results of statistical experiments show the correctness of constructing probabilistic-automata CES models and logical-probabilistic models, as well as the possibility of using them as formalized specifications.

REFERENCES

1. Voevodin V.V., Voevodin Vl.V. *Parallel'nye vychisleniya* (*Parallel Computing)*. St. Petersburg: BHV-Petersburg; 2002. 608 p. (in Russ.).
2. Pleiter D. Supercomputer Architectures: Current State and Future Trends. *The AQTIVATE Project, European Union's HORIZON MSCA Doctoral Networks Programmer, Grant Agreement No. 101072344.* September 2023. 38 p.
3. Boldyrev A., Ratnikov F., Shevelev A. Approach to Finding a Robust Deep Learning Model. *IEEE Access*. 2025;13: 102390–102406. https://doi.org/10.1109/ACCESS.2025.3578926, https://doi.org/10.48550/arXiv.2505.17254
4. Mishenin R.M., Kostenetskii P.S. Modeling the task flow of the HSE computing cluster using SLURM Simulator. In: *Parallel Computing Technologies* (*PCT'2025*): Proceedings of the 19th All-Russian Scientific Conference with International Participation. Moscow; 2025. P. 324 (in Russ.).
5. Promyslov G., Efremov A., Ilyasov Y., Pisarev V., Timofeev A. Efficiency of Machine Learning Tasks on HPC Devices. In: *Parallel Computing Technologies* (*PCT'2025*): Proceedings of the 19th All-Russian Scientific Conference with International Participation. Moscow; 2025. P. 56–81 (in Russ.).
6. Kostenetskiy P.S., Kozyrev V.I., Chulkevich R.A., Raimova A.A. Enhancement of the Data Analysis Subsystem in the Task-Efficiency Monitoring System HPC TaskMaster for the cHARISMa Supercomputer Complex at HSE University. In: Sokolinsky L., Zymbler M., Voevodin V., Dongarra J. (Eds.). *Parallel Computational Technologies* (*PCT'2024*). *Communications in Computer and Information Science*. Springer; 2024. V. 2241. P. 49–64. https://doi.org/10.1007/978-3-031-73372-7_4
7. Slastnikov S.A., Zhukova L.F., Semichasnov I.V. Application for data retrieval, analysis, and forecasting in social networks. *Informatsionnye tekhnologii i vychislitel'nye sistemy = Journal of Information Technologies and Computing Systems.* 2024;1:97–108 (in Russ.). https://doi.org/10.14357/20718632240110
8. Kirdeev A., Burkin K., Vorobev A., Zbirovskaya E., Lifshits G., Nikolaev K., Zelenskaya E., Donnikov M., Kovalenko L., Urvantseva I., Poptsova M. Machine learning models for predicting risks of MACEs for myocardial infarction patients with different VEGFR2 genotypes. *Front. Med.* 2024;11:1452239. https://doi.org/10.3389/fmed.2024.1452239
9. Al-Khulaidi A., Sadovoy N. Analysis of existing software packages in the cluster systems. *Vestnik Donskogo gosudarstvennogo tekhnicheskogo universiteta = Vestnik of Don State Technical University*. 2010;10(3):303–310 (in Russ.). https://elibrary.ru/mvsqql
10. Ladygin I.I., Loginov A.V., Filatov A.V., Yankov S.G. *Klastery na mnogoyadernykh protsessorakh* (*Clusters on Multi-Core Processors*). Moscow: MPEI Publ.; 2008. 112 p. (in Russ.). ISBN 978-5-383-00142-4. https://elibrary.ru/qmsnap
11. Kokots A.V. Development of a software model of an effective cluster computing system *Vychislitel'nye seti. Teoriya i praktika = Network Journal. Theory and Practice.* 2016;2(29):6.1. Available from URL: https://network-journal.mpei.ac.ru/ (in Russ.). Accessed June 02, 2025.

12. Kaur K., Rai A.K. A Comparative Analysis: Grid, Cluster and Cloud Computing. *Int. J. Adv. Res. Computer Commun. Eng*. 2014;3(3):5730–5734.
13. Omer S.M.I., Mustafa A.B.A., Alghali F.A.E. Comparative study between Cluster, Grid, Utility, Cloud and Autonomic computing. *IOSR Journal of Electrical and Electronics Engineering* (*IOSR-JEEE*). 2014;9(6):61–67. http://doi.org/10.9790/1676-09636167
14. Kumar R. Comparison between Cloud Computing, Grid Computing, Cluster Computing and Virtualization. *Int. J. Mod. Computer Sci. Appl*. (*IJMCSA*). 2015;3(1):42–47. http://doi.org/10.13140/2.1.1759.7765
15. Voevodin Vl.V., Zhumatii S.A. *Vychislitel'noe delo i klasternye sistemy* (*Computing and Cluster Systems*). Moscow: MSU Press; 2007. 150 p. (in Russ.). ISBN 978-5-211-05440-0
16. Hopcroft J.D., Motwani R., Ulman J.D. *Vvedenie v teoriyu avtomatov, yazykov i vychislenii* (*Introduction to Automata Theory, Languages, and Computations*): transl. from Engl. Moscow: Vil'yams; 2015. 528 p. (in Russ.). ISBN 978-5-8459-1969-4
[Hopcroft J.E., Motwani R., Ulman J.D. *Introduction to Automata Theory, Languages and Computation*. Boston, etc.: Addison-Wesley Publ. Comp.; 2001. 521 p.]
17. Baranov S.I. *Sintez mikroprogrammnykh avtomatov* (*Graf-skhemy i avtomaty*) (*Synthesis of Microprogrammed Automata* (*Graph-Schemes and Automata*)). Leningrad: Energiya; 1979. 231 p. (in Russ.).
18. Vashkevich N.P. *Sintez mikroprogrammnykh upravlyayushchikh avtomatov* (*Synthesis of Microprogram Control Automata*). Penza; 1990. 115 c. (in Russ.).
19. Vashkevich N.P., Sibiryakov M.A. The formal automatic models of algorithms of processing of cached data. *Sovremennye naukoemkie tekhnologii = Modern High Technologies*. 2016;(8-2):205–213 (in Russ.). https://elibrary.ru/whksst
20. Lazarev V.G., Piil' E.I., Turuta E.N. *Postroenie programmiruemykh upravlyayushchikh ustroistv* (*Construction of Programmable Control Devices*). Moscow: Energoatomizdat; 1984. 264 p. (in Russ.).
21. Anishev P.A., Achasova S.M., Bandman O.L. *Metody parallel'nogo programmirovaniya* (*Methods of Parallel Programming*). Novosibirsk: Nauka; 1981. 180 p. (in Russ.).
22. Yuditskii S.A., Magergut V.Z. *Logicheskoe upravlenie diskretnymi protsessami. Modeli, analiz, sintez* (*Logical Control of Discrete Processes. Models, Analysis, Synthesis*). Moscow: Mashinostroenie; 1987. 176 p. (in Russ.).
23. Girault A., Lee E.A. Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*. 1999;18(6):742–760. https://doi.org/10.1109/43.766725
24. Stefansson E., Johansson K.H. Hierarchical finite state machines for efficient optimal planning in large-scale systems. In: *2023 European Control Conference* (*ECC*). June, Bucharest, Romania. IEEE; 2023. https://doi.org/10.23919/ECC57647.2023.10178139
25. Alur R., Yannakakis M. Model checking of hierarchical state machines. *ACM SIGSOFT Software Engineering Notes*. 1998;23(6):175–188. http://doi.org/10.1145/503502.503503
26. Bolotova L.S. *Sistemy iskusstvennogo intellekta. Modeli i tekhnologii, osnovannye na znaniyakh* (*Artificial Intelligence Systems. Models and Technologies Based on Knowledge*). Moscow: Finansy i statistika; 2012. 664 p. (in Russ.). ISBN 978-5-279-03530-4
27. Thayse A., Gribomont P., Louis J. *Logicheskii podkhod k iskusstvennomu intellektu: ot klassicheskoi logiki k logicheskomu programmirovaniyu* (*Logical Approach to Artificial Intelligence: From Classical Logic to Logical Programming*): transl. from French. Moscow: Mir; 1990. 429 p. (in Russ.). ISBN 5-03-001636-8
28. Ueno H., Ishizuka M. (Eds.). *Predstavlenie i ispol'zovanie znanii* (*Representation and Use of Knowledge*): transl. from. Japan. Moscow: Mir; 1989. 220 p. (in Russ.). ISBN 5-03-000685-0
29. Evreinov E.V., Kosarev Yu.G. *Odnorodnye universal'nye sistemy vysokoi proizvoditel'nosti* (*Homogeneous Universal Systems of High Productivity*). Novosibirsk: Nauka; 1966. 308 p. (in Russ.).
30. Belov V.V., Vorob'ev E.M., Shatalov V.E. *Teoriya grafov* (*Graph Theory*). Moscow: Vysshaya shkola; 1976. 392 p. (in Russ.).
31. Polikarpova N.I., Shalyto A.A. *Avtomatnoe programmirovanie* (*Automata Programming*): 2nd ed. St. Petersburg: Piter; 2011. 176 p. (in Russ.). ISBN 987-5-4237-0075-1
32. Peterson J. *Teoriya setei Petri i modelirovanie system* (*Petri Net Theory and the Modeling of Systems*): transl. from Engl. Moscow: Mir; 1984. 368 p. (in Russ.).
[Peterson J.L. *Petri Net Theory and the Modeling of Systems.* NY: Prentice-Hall; 1981. 310 p.]
33. Kotov V.E. *Seti Petri* (*Petri Nets*). Moscow: Nauka; 1984. 160 p. (in Russ.).
34. Volchikhin V.I., Zinkin S.A. Logic and algebraic models and methods in designing functional architecture of distributed data storage and processing systems. *Izvestiya vuzov. Povolzhskii region. Tekhnicheskie nauki = University Proceedings. Volga Region. Technical Sciences*. 2012;2:3–16 (in Russ.). https://elibrary.ru/pfpgml
35. Zinkin S.A. Elements of a new object-oriented technology for modeling and implementing systems and networks for storing and processing data. *Informatsionnye tekhnologii = Information Technologies*. 2008;10:20–27 (in Russ.).
36. Andon F.I., Doroshenko A.E., Tseitlin G.E., Yatsenko E.A. *Algebroalgoritmicheskie modeli i metody parallel'nogo programmirovaniya* (*Algebraic Algorithmic Models and Methods of Parallel Programming*). Kiev: Akademperiodika; 2007. 634 p. (in Russ.).
37. Yushchenko E.L., Tseitlin G.E, Gritsai V.P., Terzyan T.K. *Mnogourovnevoe strukturnoe proektirovanie programm. Teoreticheskie osnovy, instrumentarii* (*Multilevel structural design of programs. Theoretical foundations, tools*). Moscow: Finansy i statistika; 1989. 208 p. (in Russ.). ISBN 5-279-00233-X
38. Gurevich Y. Abstract State Machines: An Overview of the Project. In: Seipel D., Turull-Torres J.M. (Eds.). *Foundations of Information and Knowledge Systems. Lecture Notes in Computer Science.* Springer; 2004. V. 2942. P. 6–13. https://doi.org/10.1007/978-3-540-24627-5_2

39. Maier D. *Teoriya relyatsionnykh baz dannykh* (*The Theory of Relational databases*): transl. from Engl. Moscow: Mir; 1987. 608 p. (in Russ.).
[Maier D. *The Theory of Relational databases:* 1st ed. Computer Sci. Press; 1983. 656 p.]

СПИСОК ЛИТЕРАТУРЫ

1. Воеводин В.В., Воеводин Вл.В. *Параллельные вычисления*. СПб.: БХВ-Петербург; 2002. 608 с.
2. Pleiter D. Supercomputer Architectures: Current State and Future Trends. *The AQTIVATE Project, European Union's HORIZON MSCA Doctoral Networks Programmer, Grant Agreement No. 101072344.* September 2023. 38 p.
3. Boldyrev A., Ratnikov F., Shevelev A. Approach to Finding a Robust Deep Learning Model. *IEEE Access*. 2025;13: 102390–102406. https://doi.org/10.1109/ACCESS.2025.3578926, https://doi.org/10.48550/arXiv.2505.17254
4. Мишенин Р.М., Костенецкий П.С. Моделирование потока задач вычислительного кластера НИУ ВШЭ с использованием SLURM Simulator. В сб.: *Параллельные вычислительные технологии* (*ПаВТ'2025*): сборник трудов XIX Всероссийской научной конференции с международным участием. М.: 2025. С. 324.
5. Promyslov G., Efremov A., Ilyasov Y., Pisarev V., Timofeev A. Efficiency of Machine Learning Tasks on HPC Devices. В сб.: *Параллельные вычислительные технологии* (*ПаВТ'2025*): сборник трудов XIX Всероссийской научной конференции с международным участием. М.; 2025. С. 56–81.
6. Kostenetskiy P.S., Kozyrev V.I., Chulkevich R.A., Raimova A.A. Enhancement of the Data Analysis Subsystem in the Task-Efficiency Monitoring System HPC TaskMaster for the cHARISMa Supercomputer Complex at HSE University. In: Sokolinsky L., Zymbler M., Voevodin V., Dongarra J. (Eds.). *Parallel Computational Technologies* (*PCT'2024*). *Communications in Computer and Information Science*. Springer; 2024. V. 2241. P. 49–64. https://doi.org/10.1007/978-3-031-73372-7_4
7. Сластников С.А., Жукова Л.Ф., Семичаснов И.В. Приложение поиска, анализа и прогнозирования данных в социальных сетях. *Информационные технологии и вычислительные системы.* 2024;1:97–108. https://doi.org/10.14357/20718632240110
8. Kirdeev A., Burkin K., Vorobev A., Zbirovskaya E., Lifshits G., Nikolaev K., Zelenskaya E., Donnikov M., Kovalenko L., Urvantseva I., Poptsova M. Machine learning models for predicting risks of MACEs for myocardial infarction patients with different VEGFR2 genotypes. *Front. Med.* 2024;11:1452239. https://doi.org/10.3389/fmed.2024.1452239
9. Аль-Хулайди А.А., Садовой Н.Н. Анализ существующих программных пакетов в кластерных системах. *Вестник Донского государственного технического университета* (*Вестник ДГТУ*). 2010;10(3-46):303–310. https://elibrary.ru/mvsqql
10. Ладыгин И.И., Логинов А.В., Филатов А.В., Яньков С.Г. *Кластеры на многоядерных процессорах.* М.: Издательский дом МЭИ; 2008. 112 с. ISBN 978-5-383-00142-4. https://elibrary.ru/qmsnap
11. Кокоц А.В. Разработка программной модели функционирования кластерной вычислительной системы. *Вычислительные сети. Теория и практика*. 2016;2(29):6.1. URL: https://network-journal.mpei.ac.ru/. Дата обращения 02.06.2025.
12. Kaur K., Rai A.K. A Comparative Analysis: Grid, Cluster and Cloud Computing. *Int. J. Adv. Res. Computer Commun. Eng*. 2014;3(3):5730–5734.
13. Omer S.M.I., Mustafa A.B.A., Alghali F.A.E. Comparative study between Cluster, Grid, Utility, Cloud and Autonomic computing. *IOSR Journal of Electrical and Electronics Engineering* (*IOSR-JEEE*). 2014;9(6):61–67. http://doi.org/10.9790/1676-09636167
14. Kumar R. Comparison between Cloud Computing, Grid Computing, Cluster Computing and Virtualization. *Int. J. Mod. Computer Sci. Appl*. (*IJMCSA*). 2015;3(1):42–47. http://doi.org/10.13140/2.1.1759.7765
15. Воеводин Вл.В., Жуматий С.А. *Вычислительное дело и кластерные системы*. М.: Изд-во МГУ; 2007. 150 с. ISBN 978-5-211-05440-0
16. Хопкрофт Д., Мотвани Р., Ульман Д. *Введение в теорию автоматов, языков и вычислений*: пер. с англ. М.: Вильямс; 2015. 528 с. ISBN 978-5-8459-1969-4
17. Баранов С.И. *Синтез микропрограммных автоматов* (*Граф-схемы и автоматы*). Л.: Энергия; 1979. 231 с.
18. Вашкевич Н.П. *Синтез микропрограммных управляющих автоматов*. Пенза: Изд-во Пенз. политехн. ин-та; 1990. 115 с.
19. Вашкевич Н.П., Сибиряков М.А. Формальные автоматные модели алгоритмов обработки кэшируемой информации. *Современные наукоемкие технологии*. 2016;(8-2):205–213. https://elibrary.ru/whksst
20. Лазарев В.Г., Пийль Е.И., Турута Е.Н. *Построение программируемых управляющих устройств*. М.: Энергоатомиздат; 1984. 264 с.
21. Анишев П.А., Ачасова С.М., Бандман О.Л. *Методы параллельного программирования*. Новосибирск: Наука; 1981. 180 с.
22. Юдицкий С.А., Магергут В.З. *Логическое управление дискретными процессами. Модели, анализ, синтез*. М.: Машиностроение; 1987. 176 с.
23. Girault A., Lee E.A. Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*. 1999;18(6):742–760. https://doi.org/10.1109/43.766725
24. Stefansson E., Johansson K.H. Hierarchical finite state machines for efficient optimal planning in large-scale systems. In: *2023 European Control Conference* (*ECC*). June, Bucharest, Romania. IEEE; 2023. https://doi.org/10.23919/ECC57647.2023.10178139

25. Alur R., Yannakakis M. Model checking of hierarchical state machines. *ACM SIGSOFT Software Engineering Notes*. 1998;23(6):175–188. http://doi.org/10.1145/503502.503503
26. Болотова Л.С. *Системы искусственного интеллекта. Модели и технологии, основанные на знаниях*. М.: Финансы и статистика; 2012. 664 с. ISBN 978-5-279-03530-4
27. Тейз А., Грибомон П., Луи Ж. *Логический подход к искусственному интеллекту: от классической логики к логическому программированию*: пер. с франц. М.: Мир; 1990. 429 с. ISBN 5-03-001636-8
28. *Представление и использование знаний*; под ред. Х. Уэно, М. Исидзука. М.: Мир; 1989. 220 с. ISBN 5-03-000685-0
29. Евреинов Э.В., Косарев Ю.Г. *Однородные универсальные системы высокой производительности*. Новосибирск: Наука, Сибирское отд.; 1966. 308 с.
30. Белов В.В., Воробьев Е.М., Шаталов В.Е. *Теория графов*. М.: Высшая школа; 1976. 392 с.
31. Поликарпова Н.И., Шалыто А.А. *Автоматное программирование*: 2-е изд. СПб.: Питер; 2011. 176 с. ISBN 987-5-4237-0075-1
32. Питерсон Дж. *Теория сетей Петри и моделирование систем*: пер. с англ. М.: Мир; 1984. 368 с.
33. Котов В.Е. *Сети Петри*. М.: Наука; 1984. 160 с.
34. Волчихин В.И., Зинкин С.А. Логико-алгебраические модели и методы в проектировании функциональной архитектуры распределенных систем хранения и обработки данных. *Известия вузов. Поволжский регион. Технические науки.* 2012;2:3–16. https://elibrary.ru/pfpgml
35. Зинкин С.А. Элементы новой объектно-ориентированной технологии для моделирования и реализации систем и сетей хранения и обработки данных. *Информационные технологии*. 2008;10:20–27.
36. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. *Алгеброалгоритмические модели и методы параллельного программирования*. Киев: Академпериодика; 2007. 634 с.
37. Ющенко Е.Л., Цейтлин Г.Е, Грицай В.П., Терзян Т.К. *Многоуровневое структурное проектирование программ. Теоретические основы, инструментарий*. М.: Финансы и статистика; 1989. 208 с. ISBN 5-279-00233-X
38. Gurevich Y. Abstract State Machines: An Overview of the Project. In: Seipel D., Turull-Torres J.M. (Eds.). *Foundations of Information and Knowledge Systems. Lecture Notes in Computer Science.* Springer; 2004. V. 2942. P. 6–13. https://doi.org/10.1007/978-3-540-24627-5_2
39. Мейер Д. *Теория реляционных баз данных*: пер. с англ. М.: Мир; 1987. 608 с.

About the Author

Grigory V. Petushkov, Vice-Rector, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: petushkov@mirea.ru. RSCI SPIN-code 4985-4344, https://orcid.org/0009-0006-0801-429X

Об авторе

Петушков Григорий Валерьевич, проректор, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: petushkov@mirea.ru. SPIN-код РИНЦ 4985-4344, https://orcid.org/0009-0006-0801-429X

Translated from Russian into English by Lyudmila O. Bychkova
Edited for English language and spelling by Dr. David Mossop