Supercomputing Frontiers and Innovations

2026, Vol. 13, No. 2

Scope

- Future generation supercomputer architectures
- Exascale computing
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Novel approaches to computing targeted to solve intractable problems
- Convergence of high performance computing, machine learning and big data technologies
- Distributed operating systems and virtualization for highly scalable computing
- Management, administration, and monitoring of supercomputer systems
- Mass storage systems, protocols, and allocation
- Power consumption minimization for supercomputing systems
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Scientific visualization in supercomputing environments
- Education in high performance computing and computational science

Editorial Board

Guest Editor's Introduction for Special Issue on

Prospective Supercomputing Technologies

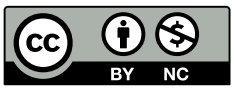
This special issue of Supercomputing Frontiers and Innovations was initiated by its editor-in-chief and founder, Vladimir Voevodin, who passed away prematurely on April 29, 2026. Vladimir was always a leading force in the field of supercomputing, both in the country and far beyond its borders. He had the ability to look far into the future, with a keen sense of the prospects for technological development, and for this reason, we dedicate this special issue of the journal to his memory.

This collection presents papers on promising supercomputing technologies and their application to solving pressing applied problems in science and engineering. It analyses the dynamics and trends in the development of the domestic supercomputing infrastructure and examines the challenges involved in the efficient numerical solution of boundary value problems for elliptic equations on graphics accelerators. A series of articles focuses on supercomputers with non-traditional architectures, specifically reconfigurable supercomputers and their system software, as well as methods for effectively solving machine learning and artificial intelligence problems using them. Generally speaking, supercomputing and artificial intelligence technologies are becoming increasingly intertwined at present; consequently, several articles in this collection are devoted to the application of AI technologies to enhance the efficiency of supercomputing. An important area of development in supercomputing technology is computing systems based on non-traditional methods of information processing. The collection therefore includes an article on photonic computers and an analysis of their effectiveness in modelling diffraction neural networks. The final article in the collection presents new methods of supercomputing for solving a key applied problem in remote sensing.

I hope that the papers in this collection will be of both academic and practical interest to researchers and specialists working in the field of supercomputing and big data processing.

Guest Editor
Academician of the RAS
Igor A. Kaliaev,
Southern Federal University,
Taganrog, Russian Federation

Contents



DOI: 10.14529/jsfi260201

An Approach to Analyzing the Entries of HPC Systems in Supercomputer Ratings as an Extension of the Functionality of the Top50 Russia Rating

Vitalii A. Kuleshov[1], *Dmitry A. Nikitenko*[1]

Supercomputer ratings such as Top500, HPCG, Graph500, IO500, and others reflect technology development trends such as performance, energy efficiency, and so on, and serve as a kind of "honor roll" for the global HPC community. Until now, there has been no convenient tool to track the dynamics of domestic systems' participation over time: when they appeared, how long they remained in the ratings, and how their positions have changed. This work fills this gap. To understand how HPC infrastructure is developing, we need to look not only at individual points reflecting the characteristics and position of systems at a given point in time, but also at their trajectories over time: when and why they appear in the ratings, how long they hold their positions, in which ratings they show greater stability, and in which they disappear. This new capability is offered by the proposed approach and implementation based on the Top50 Russian supercomputer rating: it combines data from all major global ratings for analysis. The solution is built as an analytical extension of the Top50 rating functionality. Visualizations are formed from simple charts of entry history to advanced tools using candlestick charts and transition matrices.

Keywords: hpc ratings, lifelines of supercomputers, top50 supercomputers of Russia supercomputers, evolution of computing systems, high-performance computing systems.

Introduction

The advancement of High-Performance Computing (HPC) remains a key driver of progress in fundamental scientific research, engineering applications, and industrial computing. Modern supercomputer platforms are employed in climate modeling, computational chemistry, deep neural network training, and numerous other tasks requiring scalable computational power. Architectural features and measured performance metrics directly influence the capability to solve new classes of applied problems and the competitiveness of national scientific and technological ecosystems.

Public supercomputer rankings, primarily the Top500 [18], serve not only as a catalog of current achievements but also as a mechanism for revealing and monitoring technological trends. They enable tracking of performance growth dynamics, the impact of energy efficiency, the proliferation of architectures with varying interconnect topologies, and more. Supplementary lists and benchmarks, such as Green500, IO500, Graph500, HPCG, MLPERF, and others, assess specialized system aspects, including energy efficiency, input/output operations, graph processing, and various workload classes. Collectively, they provide a comprehensive overview of the state of the global high-end computing resource pool.

However, each ranking release represents merely a snapshot: for each system, only the position and a set of specialized metrics are recorded at the time of submission. This approach fails to provide a complete picture of the "lifecycle" of physical systems. It does not reflect configuration history (upgrades, partial updates), overlooks naming ambiguities across releases, and hinders the assessment of positional stability over time. Consequently, managerial and research conclusions based solely on individual releases may be incomplete or misleading.

[1]Research Computing Center of Lomonosov Moscow State University, Moscow, Russian Federation

For monitoring and analyzing national representation in global rankings, a new approach is required that operates not on individual release data points, but on the trajectories of emergence and evolution of each physical system. Key questions include: when a system first enters the ranking and the duration of its presence; what configuration changes accompanied shifts in position; in which rankings the system demonstrates greater stability; and whether there are "traps" of rapid obsolescence for certain equipment categories. Answering these questions requires data formalization, validation procedures, and a set of normalized metrics enabling correct inter-release comparison.

Supercomputer rankings, and particularly the Top50 list, serve as fundamental indicators of the high-performance computing (HPC) field's development, enabling the tracking of both regional computational capacity progress and the evolution of architectural design solutions [5, 8]. This study proposes a comprehensive approach to analyzing the inclusion of systems from the national Top50 ranking, developed by the Research Computing Center of Lomonosov Moscow State University (RCC MSU), into international lists. The focus is placed on the Top500 as the primary data source, as it features the widest representation of domestic supercomputer complexes. Within this work, an analysis of the current state and structure of the national Top50 [19] supercomputer ranking website was conducted. Based on this analysis, a data model was developed for tracking the inclusion of Russian systems in international rankings, accounting for their performance characteristics. To ensure correct identification of physical systems and their configurations, an administrative panel was implemented for manual validation and entry of ranking inclusion data. This approach allows explicitly recording which specific system version and configuration is presented in a particular release.

Based on the formed data structure, temporal trajectories ("lifelines") were constructed for each system mentioned in the rankings, reflecting the history of its presence in international lists. To evaluate not only positional rank but also system quality, two normalized metrics were introduced: the share of achieved performance on the LINPACK benchmark relative to the total performance of ranking participants, and the share of theoretically achievable peak performance relative to the total theoretical peak.

Additionally, a statistical section was implemented dedicated to analyzing system entries into the Top50 and Top500 rankings. This includes an assessment of the system pool refresh rate and visualization of position and performance change trajectories. For extended analysis, 3D representations of lifelines and candlestick charts for aggregated display of metric changes were applied, as well as probabilistic models of transitions between rank categories, constructed based on Markov matrices. The latter are used to diagnose system position stability and identify categories with a higher probability of degradation and rapid obsolescence.

The subsequent sections of the paper develop the outlined ideas: the Literature Review (State of the Art) section examines existing rankings, metrics, and approaches to rank dynamics analysis; the Methodology section formalizes the data model and validation procedures; the Implementation section describes the software solution architecture and implemented visualization components; the Approbation section demonstrates the results of applying the approach to historical Top500 data and the analysis of Russian systems. The Conclusion summarizes the findings and outlines directions for future development.

1. State of the Art

1.1. Global Landscape and Methodological Foundation

The history of systematic ranking of high-performance systems began in 1993 with the emergence of the Top500 ranking, which over three decades has transformed from a simple "honor roll" into the most important tool for technological progress monitoring [11]. The fundamental performance metric in the Top500 is the LINPACK (HPL) test, which measures the speed of solving a dense system of linear equations [4]. Despite criticism for its one-sidedness, the project's editors emphasize its success as a stable metric for revealing long-term trends, such as the exponential growth of performance and the shift from homogeneous systems to complex hybrid architectures [15].

As the industry developed, the need arose to diversify evaluation criteria. In 2013 the Green500 [13] ranking, focused on energy efficiency (GFlops/W), received official status, which was a response to the environmental crisis and the rising cost of system ownership [1]. At the same time, the Graph500 [12] ranking for assessing graph processing and IO500 [14] for analyzing I/O subsystems appeared. However, as researchers note, the modern landscape remains fragmented: system data is often scattered, and rules for correlating positions across different lists are absent [1, 9].

As the longest-standing regional ranking, the Top50 list has undergone substantial modernization (vivification), adapting to innovations in supercomputer system design and introducing novel approaches to performance comparison across the CIS region [6].

1.2. Architectural Trends and the Problem of System Analysis

An analysis of the 27-year history of the Top500 conducted in 2021 [4] revealed an alarming trend: the HPC community tends to maximize FLOPS (processor performance) at the expense of the balance of other subsystems. The study highlighted the following points:

- Performance Efficiency (Rmax/Rpeak): The average efficiency is only 0.67, and it tends to decrease as architectures become more complex.
- Heterogeneity: Since 2011, the number of systems with accelerators (GPU) has been steadily increasing, and today they provide on average 84% of the peak performance (Rpeak) of leading supercomputers.
- Role of the Operating System: The transition to open-source software had a huge impact on the growth of computing power. Since the mid-2000s, Linux has become the dominant platform, providing flexibility for experiments with innovative system designs.

An important aspect of the analysis was the proposal of hierarchical models for describing architectures. Nikitenko D.A. [9] points out that standard descriptions in rankings often have a marketing character and lack details about the structure of nodes and interconnections. The model developed by him makes it possible to link the achieved performance with specific architectural features, which is critically important for the scientific comparison of systems from the Top50 and Top500 rankings.

1.3. Dynamics of the Life Cycle and the "Lifeline"

Researchers are particularly interested not in the static position of a system in the list, but in its evolution over time. D.G. Feitelson [3], in his works on interpreting Top500 data, identified

an empirical regularity: the rank of a given machine on average doubles every year. This means that a system is rapidly "washed out" of the list by new entrants.

Specialized tools have been developed for the visual investigation of these processes. One of the best known is the Top500Vis project by C. Steed [10], which provides interactive capabilities for analyzing distributions and characteristics of systems within individual releases. Figure 1 shows the rankings of countries in the TOP500, but the image is difficult to analyze, and there is no specific information about the systems installed in Russia. However, despite the effectiveness of such tools for analyzing statistical snapshots, methods in the scientific literature that focus on continuous trajectories "lifelines" of specific physical machines are still insufficiently developed. The "lifeline" approach proposed in this article enables the visualization of system ranking history by capturing the temporal intervals of their presence across lists. While these lines are fundamentally constructed from the fact of a system's inclusion, their interpretation is inextricably linked to research on system and component upgradability. According to recent 2025 data, it is precisely the capability for node-level modernization such as upgrading processors or interconnects that often determines a system's "lifeline" duration, allowing it to maintain competitiveness amid the rapid technological refresh of the installed base [2].

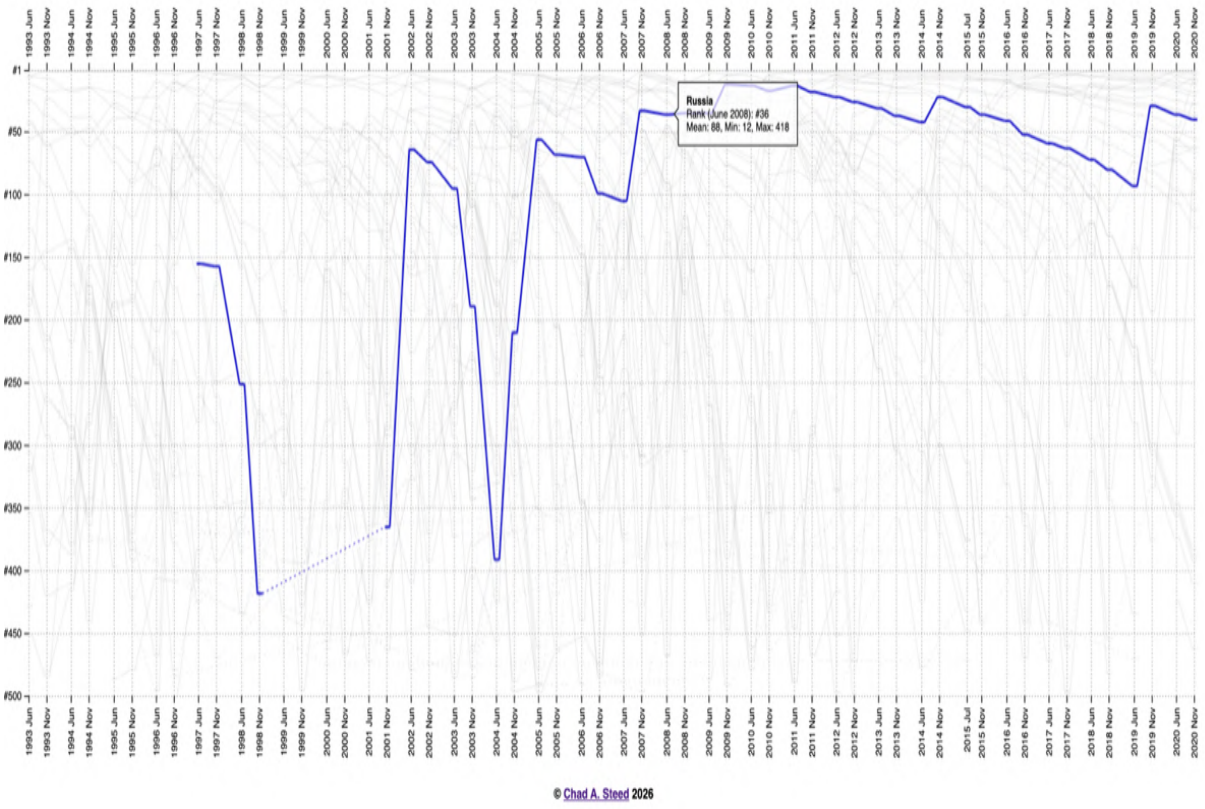


Figure 1. Russia's portrayal in the Top500Vis project (C. Steed et al.) [10]

Recent studies by Abdessalam Benhari [1] confirm a high system pool turnover rate: 80% of systems remain listed in the Top500 for fewer than 2 years, with an average presence duration of merely 1.4 years, Fig. 2. This dynamics necessitates novel visualization approaches capable of tracking individual physical system trajectories across numerous releases while accounting for hardware upgrades and configuration changes the primary focus of this paper.

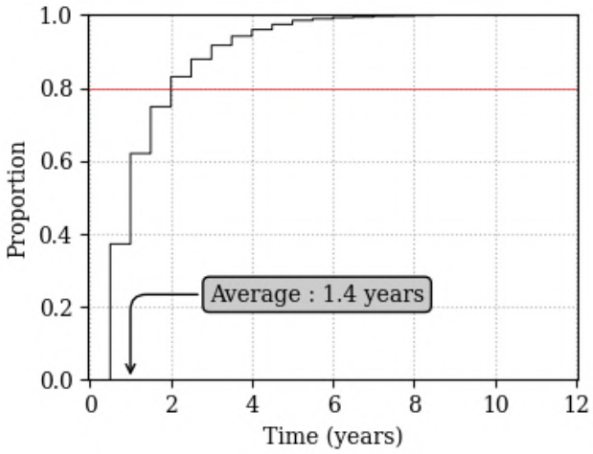


Figure 2. Lifespan of HPC systems in the Top500 (in years) from 1993.
The red horizontal line means that 80% of the systems stay less than 2 years

1.4. The Russian Context and the Shortage of Analytical Tools

In Russia, HPC resource monitoring has been conducted within the framework of the Top50 ranking since 2004, which is the most comprehensive source of data on national systems. However, a methodological gap exists: while global rankings record individual Russian systems, they do so in a fragmented manner, without unified rules for comparison. Current methods for analyzing Russian representation have a number of limitations:

- Publications of the rankings record data at the moment of submission and do not reflect the history of system upgrades.
- Problem of scale: The contribution of a single Russian system to the global computing power is statistically small, which requires the introduction of indicators relative to the national ranking in order to increase the sensitivity of the analysis.
- Lack of end-to-end integration: There is no tool capable of integrating data from Top50, Top500, Graph500, and other lists into a unified development trajectory.

It is important to note here that there are a number of HPC centers in Russia [21], which are potentially more interested in tools, demonstrating the current position and the trends of possessed machines.

1.5. Conclusions and Transition to the Proposed Methods

The literature review shows that despite the abundance of statistical data on supercomputers, the HPC community experiences an acute shortage of tools for in-depth analysis. Existing approaches are either too generalized (as in global rankings) or limited by regional frameworks.

For high-quality monitoring of national representation in the global ecosystem, it is necessary to move from the analysis of “release points” to the analysis of life-cycle trajectories. This requires the creation of a formalized data model, mechanisms for expert validation, and the implementation of advanced visualization methods (such as candlestick charts and transition matrices) capable of diagnosing the stability of positions and scenarios of equipment obsolescence. It is precisely these tasks that the method of analytical extension of the Top50 platform proposed in this work aims to address.

2. Proposed Method for Collecting and Analyzing Data on the Inclusion of Russian Supercomputer Systems in Global Rankings

The modern landscape of global supercomputer rankings represents a fragmented ecosystem in which national and international evaluation systems operate largely autonomously. The Russian domestic ranking Top50 serves as the most complete and detailed source of information on national computing systems, providing diverse data on the configurations, performance, and application domains of domestic supercomputers. At the same time, global lists, including Top500 and a number of other international rankings, have historically recorded individual Russian systems, but they do so in a scattered manner and without unified rules for comparing and analyzing the dynamics of their positions.

At present, there is no reproducible tool capable of integrating heterogeneous sources, harmonizing names and metadata, and then systematically tracking the inclusion of Russian systems in global rankings, constructing time series, and helping to identify stable trends. As a consequence of this fragmentation, there is a shortage of objective information about the international visibility and competitiveness of domestic developments.

The method described below is aimed at addressing this gap: it outlines approaches to the aggregation, normalization, and analysis of data on the inclusion of domestic supercomputer systems in global rankings, as well as tools for the quantitative assessment of their representation and dynamics.

2.1. Conceptual Data Model

The data model for describing inclusions extends the existing data model of the Top50 ranking [6]. The method is based on a conceptual model in which the following key entities and semantic relationships between them are identified:

- **Object**, a logical unit of the hardware/component structure (component, compute node, chassis, supercomputer as an aggregate).
- **Machine**, a record representing a specific computing system in the database. It refers to **Object** via a unique identifier.
- **Relation**, the semantics of relationships between objects. For the described method, the relationship type "precedes" is considered, which is used for versioning and end-to-end tracking of a system across ranking editions.

The extended data model, in turn, relies on the following entities:

- **Extratings_List**, an abstraction representing a specific international supercomputer ranking.
- **Extratings_Editions**, a specific edition of a ranking: issue number, publication date, and sequential release number.
- **Extratings_Entry**, the actual inclusion of a system in a specific edition. It includes the position and a reference to performance indicators.
- **Extratings_Units**, **Extratings_List_Units**, **Extratings_Score**, entities linking measurement units, priorities of units within a ranking, and specific values of performance indicators for a recorded inclusion of a system.

For each entity, mandatory attributes, data types, and integrity constraints are defined to ensure the correctness and consistency of the stored information. The constraints include

primary and foreign keys, uniqueness rules, as well as checks of permissible value ranges for quantitative indicators.

The conceptual data model was designed leveraging expertise from the Algo500 project [5], which established a unified repository of system architectural descriptions. This foundation enables the prospective development of flexible, user-defined rankings based on extensible entities.

2.2. Process of Data Collection and Validation

Manual input makes it possible to minimize the risk of erroneous aggregation and ensures quality control at each stage of observation formation. The function of data validation is assigned to the administrator of the Top50 ranking. This solution is practically justified both from the point of view of the workload and from the point of view of the methodological complexity of automation.

First, international rankings are published no more than twice a year, and the administrator, possessing expert knowledge about the positions of Russian systems in global lists, can efficiently perform verification with minimal labor costs.

Second, automated parsing is complicated by the fundamental heterogeneity of the rankings themselves: they are based on different tests and evaluation criteria. For example, the Top500 ranking is based on the standardized LINPACK benchmark, which ensures comparability of results. At the same time, the Graph500 and IO500 rankings use composite benchmarks that include both regulated components and parts that allow free configuration for a specific architecture in order to maximize performance. The Green500 ranking, in turn, does not impose strict requirements on the size of the problem being solved.

Such variability of evaluation methodologies, along with differences in system names, transliteration, and publication formats, makes automatic extraction and correct interpretation of data unreliable without expert verification.

2.3. Analytical Methods

Within the framework of the proposed method, the dynamics of the presence of Russian supercomputer systems in global rankings is studied. The analysis is aimed at identifying patterns in the change of their positions over time and at assessing the relative contribution of Russian systems to the total computing power of the corresponding national ranking datasets.

The method is based on the use of relative performance indicators in combination with tools of temporal and statistical analysis, which makes it possible to ensure comparability of results between different releases of rankings when their composition and scale change.

2.3.1. Lifelines of the system

One of the analysis methods is the construction of a sequence of inclusions of Russian system's in releases of global rankings. The construction of a life trajectory makes it possible to study the duration of a systems presence in an international ranking list, the dynamics of its positions, and changes in performance over time, taking into account technical upgrades.

2.3.2. Share of achieved productivity and theoretical peak value

Within the framework of the method, an indicator of the share of achieved and theoretically achievable performance is introduced. The share of achieved performance is defined as the ratio of the measured performance of a Russian system in the LINPACK test to the total performance of all systems in the national Top50 ranking. The share of the theoretical peak performance value is calculated in a similar way.

The choice of the national ranking as the normalization base is deliberate. The contribution of a single Russian system to the total computing power of the global ranking is statistically insignificant and does not allow meaningful patterns to be identified. At the same time, normalization relative to the national ranking provides higher sensitivity of the indicator and makes it possible to detect trends that remain invisible when analyzed at the global scale.

The proposed indicators reflect the relative contribution of a system to the total computing power and make it possible to analyze its relative hardware potential in the context of the distribution of computing resources.

2.3.3. Japanese candlesticks

To analyze the dynamics of the positions of Russian systems between releases of global rankings, an adapted representation in the format of "Japanese candlesticks" is used. Within a single release, the best and the worst positions of the represented Russian systems are recorded, which makes it possible to evaluate the amplitude of position changes and identify periods of significant shifts.

Such a representation provides a clear visualization of the direction of changes (improvement or deterioration of position), as well as their degree. The analysis of the sequence of "candles" in dynamics makes it possible to identify periods of relative stability, gradual decline of positions, or sharp changes caused by updates in the composition of the ranking or changes in the competitive environment.

2.3.4. Transition matrices

To formalize the dynamics, a transition matrix is used in which the movements of Russian systems between predefined position categories (1–10, 11–20, 21–30, 31–40, 41–50, 51–100, 101–200, 201–300, 301–400, 401–500) are recorded across consecutive releases of the ranking (quarters). For each system from the sample (for example, Top50 systems included in Top500, although the method is applicable to any ranking), the categories of its positions in adjacent periods are tracked. For each pair of categories $A \to B$, the empirical transition probability is calculated as the ratio of the number of recorded transitions from A to B to the total number of transitions from A.

Conceptually, this matrix represents a variation of the classical transition matrix of a Markov chain, constructed on the basis of frequency estimates of transitions between discrete states. Its application makes it possible to quantitatively assess the stability of the positions of Russian systems and to identify typical scenarios of their movement in the international ranking space.

Thus, the proposed method provides a comprehensive analysis of the dynamics of the presence of Russian supercomputer systems in global rankings through a combination of relative performance indicators and tools of temporal analysis. The use of shares total computing power makes it possible to take into account the changes in the scale and composition of rankings, while

the application of trajectory analysis, visual representations, and transition matrices makes it possible to formalize the nature and stability of the changes in system positions over time.

The integration of the specified elements within a unified methodological approach ensures the comparability of results between ranking releases and creates a foundation for identifying stable patterns of changes in the position of Russian systems in the global supercomputer space.

3. Implementation of the Proposed Method

The proposed method has been implemented as a server-side application written in Ruby [17], using the PostgreSQL [16] relational DBMS. The Top50 rating project is built on the Octoshell [7] system engine, developed at the Research Computing Center of Lomonosov Moscow State University (RCC MSU) and used for the management and support of computing infrastructure. The use of this engine allowed for the reuse of an already tested data storage architecture, user management mechanisms, and basic service components.

Within the framework of the present work, the system functionality was extended with modules for aggregation, normalization, and analysis of data on the inclusion of Russian supercomputer systems in global rankings. Additionally, an administrative panel for validation and confirmation of data was implemented, as well as a specialized statistics section providing visualization and analytical representation of results.

3.1. Technology stack

- **Programming language:** Ruby on Rails. Used for server logic, aggregation scripts, and business logic. Database extensions were implemented using Ruby migrations.
- **Data storage:** PostgreSQL, serving as the primary database.
- **Data schema management:** Ruby-based migrations, providing versioning of the database structure and reproducibility of deployment.
- **Visualization:** JavaScript and TSX (TypeScript + JSX) for implementing the user interface and interactive visualization components.

3.2. Data Model Implementation

The physical implementation of the data model is carried out in the PostgreSQL DBMS. The table structure corresponds to the conceptual model described in the Methodology section and includes the entities `Extratings_List`, `Extratings_Editions`, `Extratings_Entry`, `Extratings_Units`, `Extratings_Score` and `Extratings_List_Units`.

The database schema definition and its modifications are managed using Ruby-based migrations. This approach ensures reproducibility of the storage structure, schema version control, and consistency of changes when extending functionality.

At the DBMS level, mechanisms are implemented to maintain referential integrity and correctness of relationships between entities, which allows building complex aggregated queries and generating temporal selections without violating data consistency.

The semantics of measurement units are realized through two interrelated entities. `Extratings_Units` serves as a dictionary of units, while `Extratings_List_Units` acts as a junction table, ensuring unambiguous mapping of specific units to a particular rating and recording their priority through the `priority` field. This scheme is necessary because a single rating may

publish multiple performance metrics, and their contribution to the ranking is not always unambiguous. The `priority` field explicitly defines the preference of units during aggregation. Unique correspondence is ensured through the `extratings_list_id` and `extratings_unit_id` fields, which also allows reusing common units across multiple ratings without duplicating records. The `Extratings_List` entity formalizes the rating itself. The relationship between a specific edition and the rating is established via the `extratings_list_id` field in `Extratings_Editions`.

3.3. Implementation of the Administration Panel

The administration panel provides input of new entries regarding the inclusion of systems in international rankings. The main workflow involves the following steps:

- First, the administrator selects a system from the Top50 rating directory. Addition of new systems to the directory is handled within the core functionality of the rating: when a previously unregistered system appears, it is first registered in the project database and then becomes available for selection from the dropdown list.
- Second, the administrator selects the rating in which the entry should be recorded. For convenience, the interface automatically populates the fields for performance metrics in the units used by the selected rating. If necessary, measurement units can be changed via a dropdown menu to the units in which the benchmark result was recorded. An integrated conversion mechanism ensures that all values are brought to a unified scale.

The interface also allows extending the directory of existing ratings directly during operation. From the entry creation window, the administrator can create a new rating, specifying its characteristics and selecting the units used for system ranking. Similarly, a function is implemented for creating new measurement units with a link to a specific rating and setting their priority when determining the final system order.

3.4. Implementation of the Lifelines of the Systems

On the page of each system that has been recorded at least once in international rankings, a chart is implemented displaying its lifelines across all considered rankings. By default, all available lists are visualized; if necessary, the user can restrict the display to only the trajectory in a selected rating. Each point on the chart corresponds to a specific entry of the system in a rating edition; hovering over a point shows a tooltip with information about the position, edition number and release date, as well as performance metrics recorded for that entry. Below the chart, a tabular summary of all entries is provided: the table lists the publication date, edition, position, and key metrics the same information as in the tooltip, but presented in a static and organized form for ease of verification and export.

The section containing aggregated representations is called “Statistics of system entries in Top50 and Top500” It reflects the current concentration of domestic data, while the structure of the section is designed so that, with future expansion of the dataset, its name or functionality can be modified without changing the storage or display logic of the modules.

The statistics section implements an aggregated display of lifelines for all systems over the entire considered period. In this view, each system is shown as a separate trajectory, which allows visual assessment of the density of position distribution, periods of strengthening or weakening of Russian systems presence in international rankings, and the nature of their simultaneous movement across different ranges of the ranking space. Additionally, a mode is provided for

simultaneous visualization of trajectories in an international ranking and in the Top50 rating, allowing comparison of the systems position dynamics in global and regional contexts. This combined view facilitates identification of consistency or divergence of trends between different ranking datasets.

3.5. Implementation of the Fractions of the Achieved and Theoretical Peak Value

The fractions of achieved and theoretical peak performance are displayed on the page of a specific machine as bar charts. The user can view the entire available time interval to assess the dynamics of metrics; bars are color-coded, and hovering displays a tooltip explaining the corresponding column (release date, metric value, etc.). Joint visualization of the fraction of achieved performance and the fraction of the theoretical peak allows a clear assessment of the divergence between practically realized and theoretically possible performance values and serves as a basis for conclusions about the practical efficiency of the systems software-hardware implementation.

3.6. Implementation of the "Japanese Candlesticks" Metric

The "Japanese candlesticks" visualization is implemented as an interactive client-side component (JS / TSX) within the statistics section. By default, the visual block is displayed in the same view as the lifelines, covering the entire available observation period. On the server side, time series are generated for each ranking publication: the boundary values of the position are computed and used as the "open" and "close" parameters of the candlestick. The resulting series are transmitted to the interface in a format suitable for direct rendering.

The visualization interface supports interactive controls, including selection of the time window (observation period) directly within the chart area, filtering by ranking, and adjustment of the aggregation step. Hovering over a candlestick element displays a tooltip with information about the best and worst position within the release, the edition number and publication date, and the corresponding performance metrics. This design allows the user to obtain both an overview for the entire period and a focused inspection of a specific time interval without navigating to other pages.

3.7. Implementation of Transition Matrices

The interactive implementation allows users to inspect numerical values and percentage estimates for specific category pairs and, if necessary, to restrict calculations to a defined time window or subset (e.g., only Russian systems or only Top50). The transition matrix serves as a tool for quantitatively assessing the stability of positions and identifying typical patterns of movement within the ranking space. The use of the transition matrix is meaningful within a well-defined temporal interval. While it is formally possible to compute the share of transitions from an arbitrary category A to category B relative to all transitions from A over the entire existence of the ranking, the substantive interpretation of such a value may be complicated by changes in the composition of participants and ranking conditions over time. To reduce this ambiguity, the interface by default displays transition probabilities calculated over the most recent two years of publications. This interval provides estimates that reflect current dynamics.

Additionally, the interface offers the option to switch to precomputed values over the last five years of publications, as well as a tool to select an arbitrary time interval within which

transition probabilities will be computed. This functionality allows users to define the temporal scale of analysis according to the specific research task.

The interface includes tabbed display modes: the matrix can be presented either in relative terms (probabilities of transition from category A to category B) or in absolute terms (number of recorded transitions). Although matrices over limited time windows are often sparse, such estimates are more representative and provide a more accurate reflection of the observed dynamics of system positions. Thus, the implemented software tool provides a complete workflow for ranking data: from entry and verification of observations to their analytical and visual interpretation. A centralized data storage model has been implemented, supporting the recording of positions, performance metrics, and their historical evolution, as well as mechanisms for administering reference directories and units of measurement. The interface provides a set of analytical views designed for the study of system dynamics: lifelines, aggregated trajectories, relative measures of achieved and peak performance, transition matrices with configurable time windows, and interactive 3D visualization. Implemented filtering mechanisms, time window selection, and display mode switching (absolute values versus probabilities) ensure analytical flexibility and allow the derivation of interpretable quantitative estimates that reflect the observed dynamics of ranking positions and system performance. Collectively, these components constitute an integrated environment for analyzing the evolution of computational systems over time, ensuring data consistency, reproducibility of calculations, and clarity of the resulting insights.

4. Approbation

The proposed method was validated using the implemented software system with data from the Top50 ranking as well as international supercomputer lists. Examples of the operation of key analytical components of the interface are presented below, demonstrating the practical applicability of the developed approach.

4.1. Data Collection and Administrative Dashboard

The validation of the data collection workflow demonstrates the end-to-end sequence of operations: starting with the entry of a system's inclusion via the administrative panel, followed by the verification of performance metrics against source documentation prior to final recording in the database. The interface streamlines this process by enabling the simultaneous selection of both the target system and the relevant ranking, while automatically populating performance unit fields in accordance with the measurement standards used in the selected list. Furthermore, the workflow supports the extensibility of core reference directories such as rankings and measurement units within a unified, coherent administrative environment.

To facilitate the addition of extensions, a dedicated management panel is provided, conveniently accessible directly beneath the ranking selection list. The introduction of new ranking lists also entails the definition of associated benchmarks used for system evaluation and ordering, including the option to register entirely new benchmark types as the field evolves. Similarly, a separate but logically linked panel is provided for the expansion of performance measurement units, ensuring that the system can adapt to emerging metrics without disrupting the core data model or requiring structural schema changes.

4.2. Approbation of the Lifelines of a Separate System

On the page of a specific system, the visualization of its lifeline has been validated, a graphical representation of the sequence of its entries in international rankings. The interface supports filtering trajectories by individual rankings, interactive tooltips displaying entry details (position, release date, performance metrics), and a tabular summary of all recorded observations for verification and export in CSV format. Figure 3 shows a portion of the table listing the Lomonosov-2 system's entries in the TOP500 ranking.

As an example, the Lomonosov-2 system is considered [20], which has an extensive history in the Top500 and Green500 rankings and is also referenced in other lists.

Rating	Position	Edition	Edition Publication Month	Performance Metrics
Top 500	130	44	2014-11-01	Rmax: 319.8 TFLOP/S, priority: 1, Rpeak: 423.42 TFLOP/S, priority: 2
Top 500	23	45	2015-06-01	Rmax: 1849.0 TFLOP/S, priority: 1, Rpeak: 2575.87 TFLOP/S, priority: 2

Figure 3. Lomonosov-2 System Entries in the Top500 Ranking (table excerpt)

Each ranking list is rendered as a distinct, self-contained table, ensuring modular data presentation and facilitating independent analysis of different datasets. Figure 4 shows the second table summarizing the Lomonosov-2 systems appearances in the GREEN500 ranking. The system implements a role-based access control mechanism, wherein authorized administrators are granted privileged operations over the data entries. Specifically, administrators can edit existing records to correct metadata or update performance metrics, as well as remove obsolete or erroneous entries when necessary. This administrative layer is designed to maintain data integrity and currency while preserving an audit trail of modifications, thereby supporting the long-term reliability of the ranking repository.

Rating	Position	Edition	Edition Publication Month	Performance Metrics
GREEN 500	128	9	2017-06-01	Energy Efficiency: 1.948 GFlops/watts, priority: 1
GREEN 500	130	10	2017-11-01	Energy Efficiency: 1.948 GFlops/watts, priority: 1

Figure 4. Lomonosov-2 System Entries in the GREEN500 Ranking (table excerpt)

Interactive tables provide a detailed representation of the dataset, offering access to historical records of system inclusions and their corresponding performance metrics across multiple ranking releases. This granular approach is essential because standard HPC rankings often represent merely the "tip of the iceberg", frequently lacking precise longitudinal data and comparative capabilities. To address the demand for novel data presentation formats and to revitalize traditional ranking lists, a lifecycle diagram is positioned directly above the tables, Fig. 5. This visualization serves as a critical instrument for tracking the evolution of the HPC fleet over time, enabling users to analyze system presence duration in rankings a factor increasingly associated with research on system upgradability and the long-term growth of regional computational capacity.

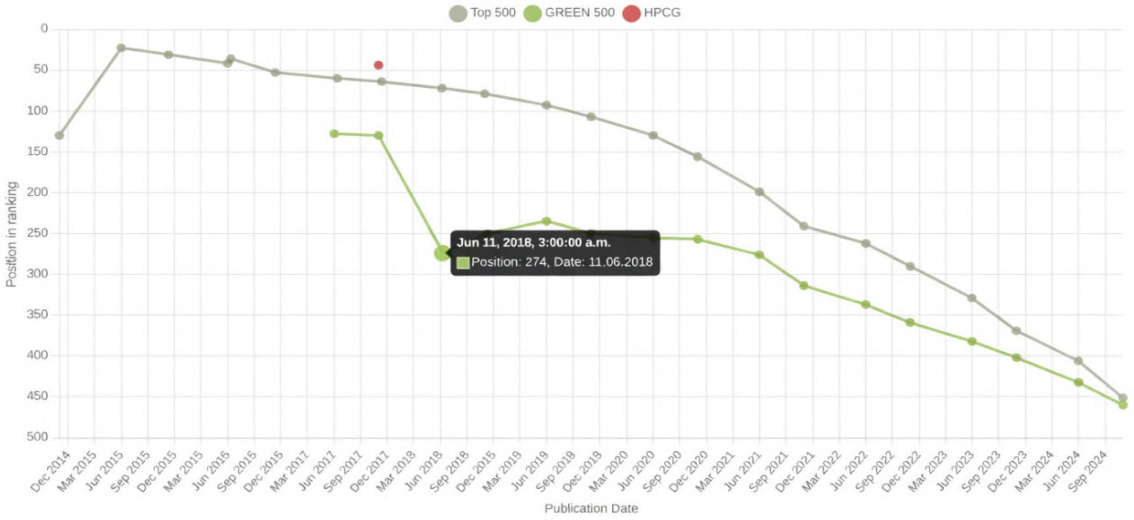


Figure 5. Lomonosov-2 system's lifeline on the system's page in the Top50 rating

4.3. Aggregated Display of Lifelines in the Statistics Section

In the Statistics section, validation of the aggregated representation of lifelines for all Russian systems across the entire observation period has been conducted. In practice, each system is displayed as an individual trajectory, which enabled assessment of position distribution density, identification of periods of strengthening or weakening presence of Russian systems in international rankings, as well as visual comparison of the synchronicity of their movement within the ranking space. The obtained graphical representations confirmed the clarity and informativeness of the proposed approach for analyzing the dynamics of Russian system entries into global rankings.

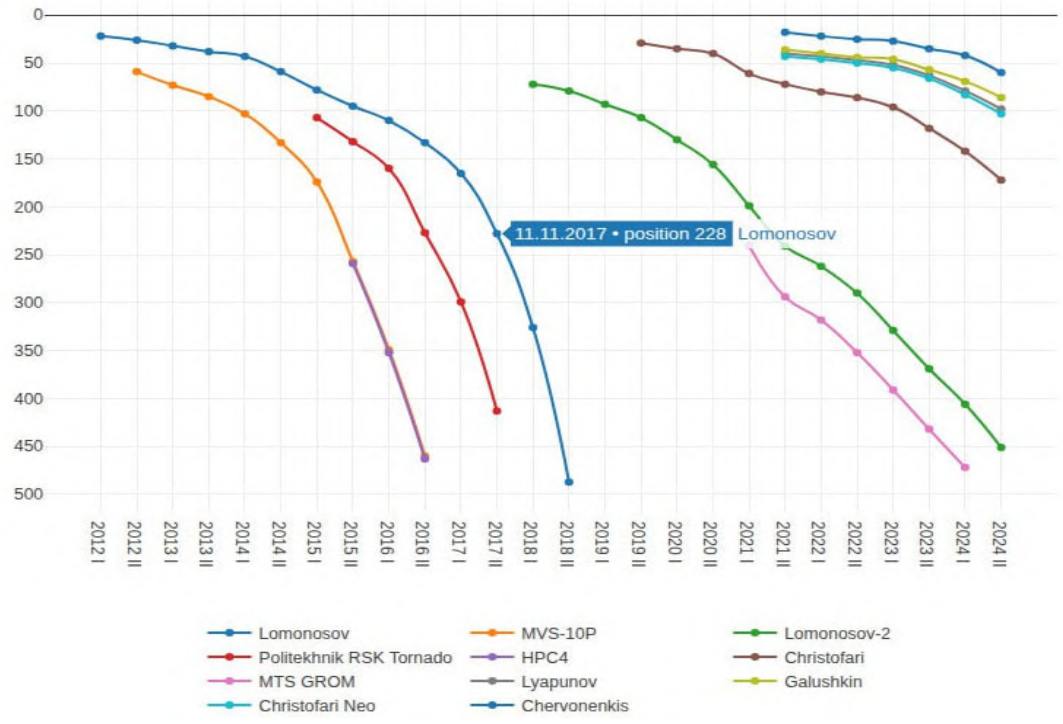


Figure 6. Lifelines of the Top50 participants mentioned in the Top500 in the statistics section

As part of this approbation, the mapping of the lifelines of Russian systems in the Top50 and Top500 ratings in a single visual space was also developed. This view made it possible to simultaneously trace the trajectories of the systems both on the international list and on the domestic rating field, identifying general trends in their advancement, periods of divergence and convergence of positions, as well as features of synchronicity or, conversely, asynchrony of dynamics in the two hierarchies. Figure 6 provides an aggregated view of the lifespans of all domestic systems listed in the TOP500. The practical value of this approach lies in the ability to more clearly assess the relationship between changes in the positions of systems in the global ranking and their status in the national context. The combined display of lifelines contributes to a deeper analysis of transitions between levels of rating significance, allows you to record stable and short-term changes in rank position and thereby increases the informative presentation of the results in the framework of comparative research. Figure 7 provides an aggregated view of the lifespans of all domestic systems participating in the TOP500, as well as a simultaneous display of the trajectories of the systems in the Top50.

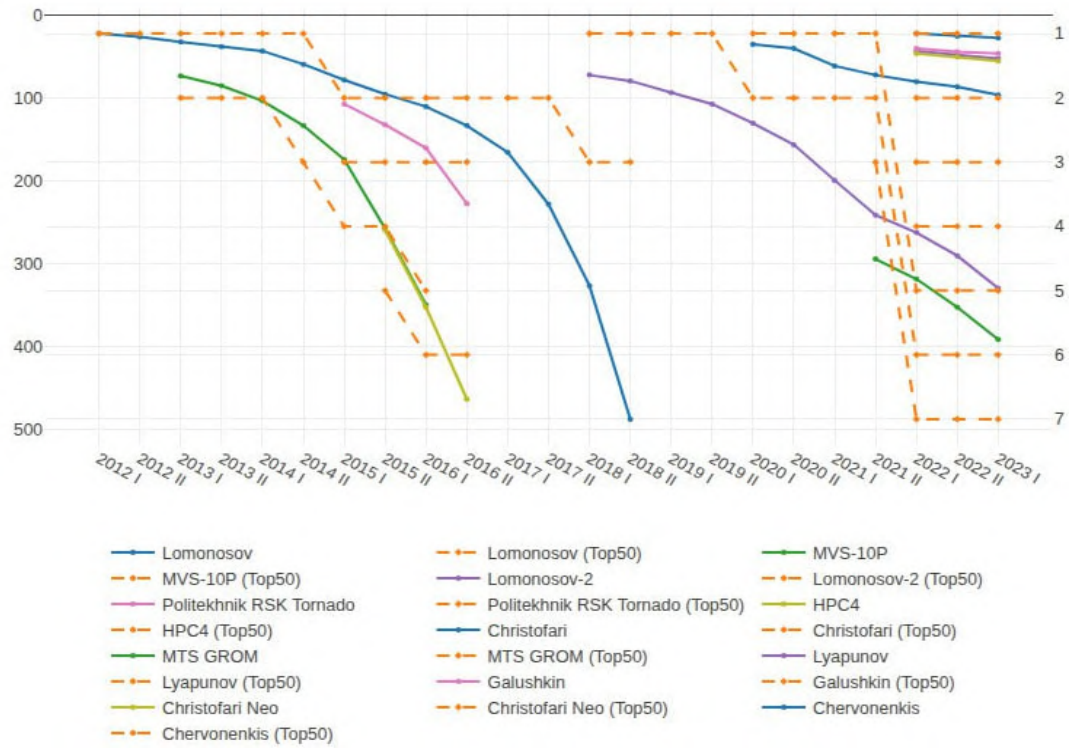


Figure 7. Joint mapping of the trajectories of the presence of Russian systems in the Top500 and Top50 rankings

4.4. Displaying Relative System Performance Indicators

The display of temporal trajectories for normalized indicators has been approved: the ratio of achieved LINPACK performance and the ratio of theoretical peak performance. Figure 8 shows the relative peak and theoretically achievable performance ratios for the Lomonosov-2 system. The graphical representation facilitates cross-release comparison of relative metric values and temporal dynamics analysis. Concurrent visualization enables assessment of the gap between actual and theoretical performance, supporting conclusions regarding computational system implementation efficiency throughout observation periods.

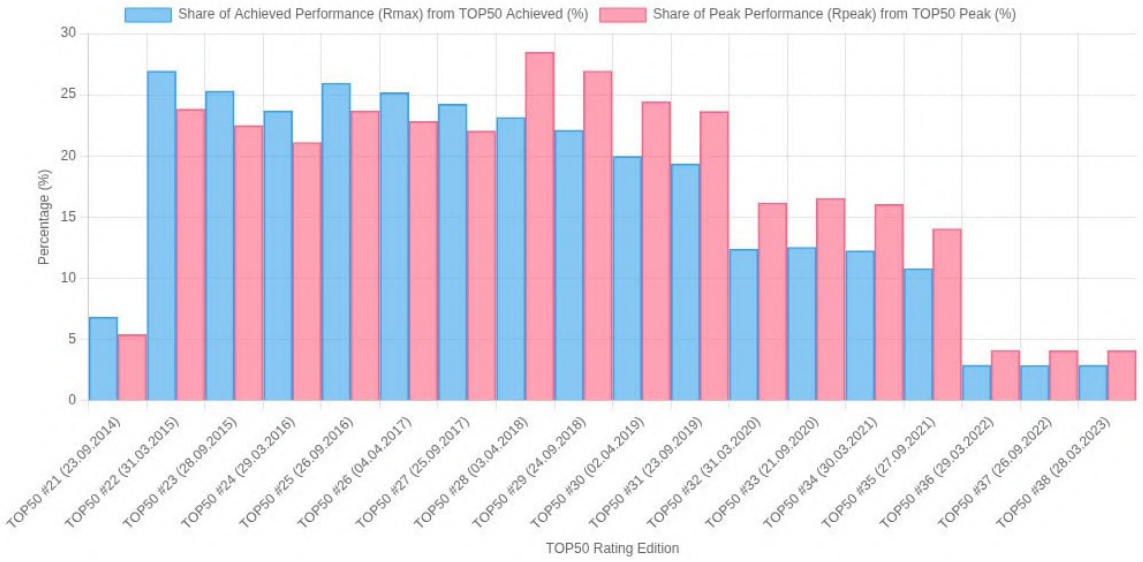


Figure 8. Displaying time series of normalized indicators: the percentage of LINPACK performance achieved and the percentage of the theoretical peak value for the Lomonosov-2 system

4.5. Approbation of the Display Technique in the "Japanese Candlestick's" Format

During validation, the visualization of position dynamics in the "candlestick" format was examined not only in terms of technical correctness but also from the perspective of its analytical usefulness. For each ranking release, a candle is constructed reflecting the range of positions of Russian systems, where the open value corresponds to the best observed position and the close value to the worst. This representation enables a compact yet informative depiction of positional variability within each release, allowing for quick identification of periods characterized by either concentration of systems in narrow ranking intervals or, conversely, significant dispersion across a wider range of positions. The visualization is presented as an interactive graph (Fig. 9), where each element corresponds to a separate observation period. In the default view, half-year intervals are displayed, which corresponds to the frequency of publication of the TOP500 ranking.

The correctness of boundary value calculations for the candles and the correspondence of displayed data to database records were thoroughly confirmed through comparison with the underlying dataset. In addition, particular attention was paid to the stability and consistency of the visualization when applied to different temporal segments, ensuring that the graphical representation remains accurate regardless of the selected observation window. Interactive presentation features were also verified, including the ability to obtain an initial overview across the entire observation period, switch to an arbitrary time interval directly within the graph area, and display detailed per-release information via tooltips. Taken together, these validation results demonstrate that the proposed visual tool is not only technically reliable but also effective for both rapid exploratory analysis and detailed verification of ranking position values.

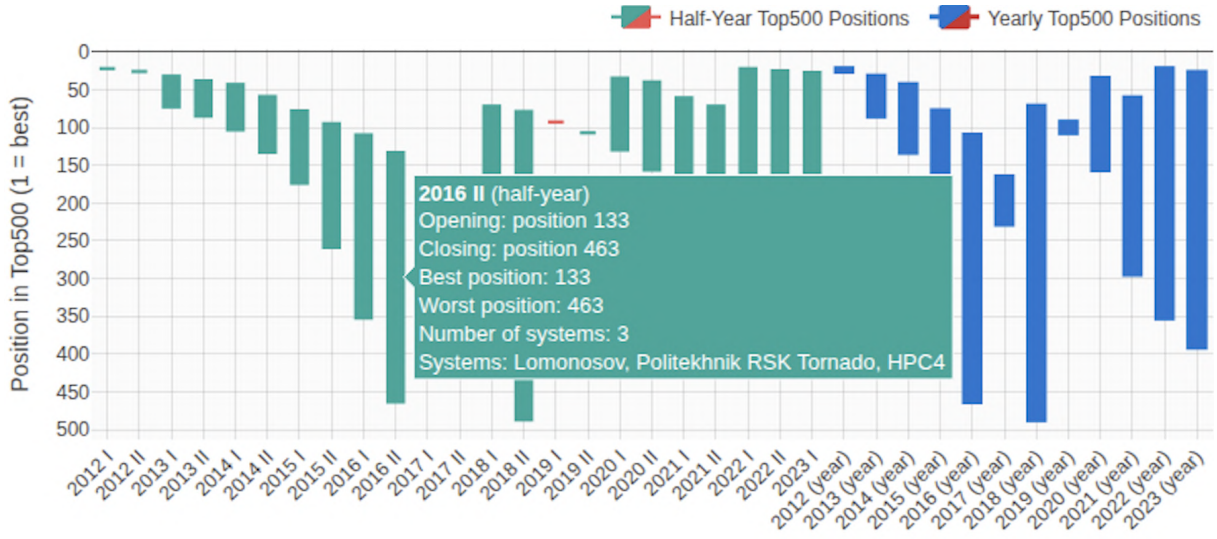


Figure 9. Approbation of the Japanese Candle Technique

4.6. Review of Transition Matrices between Rating Sectors

During validation, the calculation of system transition matrices between sectors of the ranking space (1–10, 11–20, etc.) was performed on the basis of actual observations collected from the integrated dataset. The empirical frequencies of transitions between categories were first computed and then transformed into relative transition probabilities in order to analyze the structural dynamics of the systems positions in the ranking space.

Special attention was paid to verifying the correctness and stability of the calculated probabilities. In particular, the results obtained for selected time intervals were examined to ensure that the normalization of transition frequencies was performed correctly and that the resulting probability distributions were consistent with the observed transitions. Additional verification procedures included cross-checking the aggregated transition counts with the underlying dataset and confirming that the computed probabilities accurately reflected the empirical structure of movements between ranking sectors.

Taken together, these capabilities enable a more comprehensive interpretation of the dynamics of supercomputer systems within the ranking space. The combination of empirical transition counts and normalized probabilities makes it possible to analyze not only the magnitude of movements but also the structural patterns underlying the redistribution of systems across different ranking sectors over time. Figure 10 displays the transition matrices in the users default view. Figure 11 illustrates the users ability to enter a dynamic date range to select the study interval for which the probabilistic transition matrices will be constructed.

The proposal to display numerical values of transitions from one group of the rating space to another is particularly important, since it allows us to move from a qualitative description of the dynamics to its more accurate quantitative interpretation. In contrast to simply recording the fact that systems move between rank intervals, numerical transition indicators make it possible to assess the intensity of such changes, identify dominant migration trends, and determine which transitions are stable and which are episodic. This provides a deeper understanding of the structural dynamics of the distribution of systems by rating sectors.

In addition, the representation of transitions in numerical form increases the visibility and comparability of results between different time periods. This is especially useful when analyzing

Figure 10. Matrix for analyzing the transition of supercomputers between sections of the list over a selected interval

Figure 11. Matrix for analyzing the transition of supercomputers between sections of the list over an arbitrary time interval

the evolution of the rating position of systems, as it allows you to track not only the fact of displacement, but also the scale of redistribution between groups. As a result, it becomes possible to identify periods of active restructuring of the composition of rating intervals, as well as to assess the degree of stability of the position of individual systems in dynamics. Figure 12 shows how users can view the number of system transitions between rating categories, as well as how they can select a date range spanning two years.

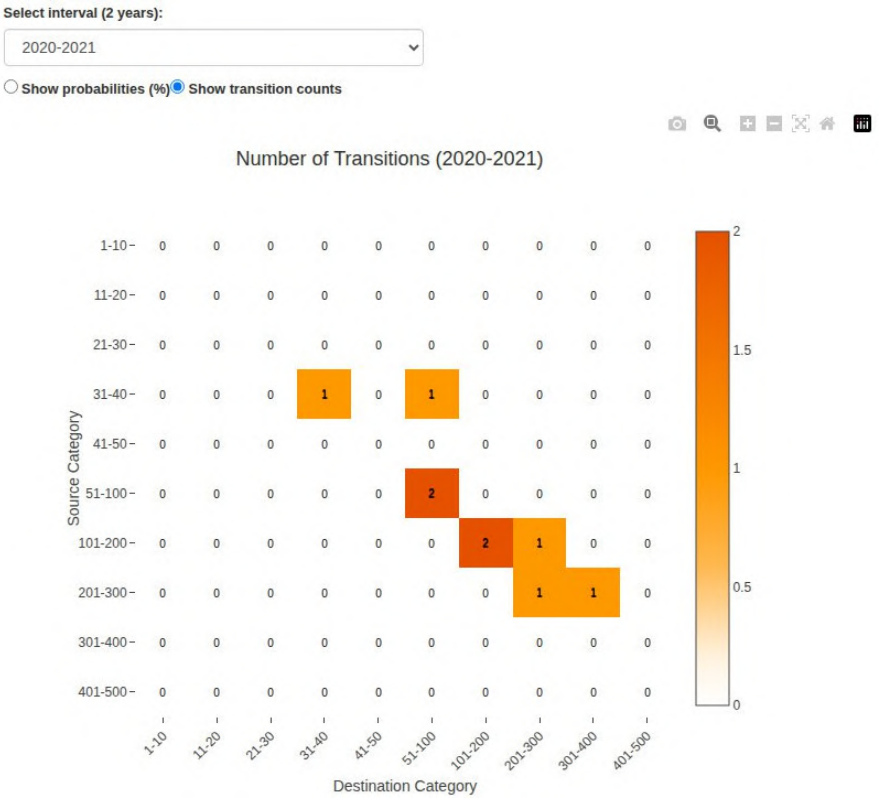


Figure 12. Displaying the number of transitions

Conclusions and Practical Significance

This article presents a new methodological approach and a set of software tools for an in-depth analysis of the dynamics of Russian supercomputer systems presence in global rankings. Users can explore the functionality on the Top50 ranking website, where it is available. Unlike traditional methods based on snapshots of publications, the proposed "lifelines" concept shifts the focus of the research to analyzing the life cycles of specific physical complexes based on their development trajectories. The principal results of the study are as follows:

- An extensible data model integrated into the *Top50* platform was developed, which ensures correct versioning of systems and enables comparison of their characteristics across different editions of international lists (e.g., *Top500*, *Graph500*, *Green500*, etc.).
- The rationale for using normalized indicators relative to the national ranking is substantiated. This normalization made it possible to reveal latent technological trends and to assess the actual competitiveness of domestic systems that are often statistically insignificant at the global scale.
- A suite of advanced visualization and analysis tools was implemented, including transition matrices and candlestick charts. Method validation demonstrated that these tools effec-

tively diagnose the stability of system positions and support forecasting of technological obsolescence scenarios.

The practical significance of the work is confirmed by the deployment of a functional analytical extension to the Top50 website, which serves as a convenient instrument for experts and decision-makers when evaluating the effectiveness of investments in national HPC infrastructure.

Upon analyzing the obtained images, distinct trends were identified and patterns were discovered:

- The lifelines of systems installed between 2020 and 2021 have a lower slope than those of systems installed between 2014 and 2016. This indicates a slower rate of degradation compared to previous-generation systems.
- The transition matrices show that the ranges 31–40 and 41–50 are zones of increased instability. Systems in these ranges rarely retain their original category in subsequent releases. Most of them begin a gradual shift toward lower ranges. These transitions occur smoothly over several releases rather than abruptly, indicating gradual technological obsolescence rather than a sudden loss of relevance. The observed transition probabilities in this range suggest that the 31–50 range is the area where decisions about the system's future are made, since maintaining its position will require significant modernization.
- The 101–200 ranking segment demonstrates the most stable pattern of transitions and can be viewed as a zone of relative stability for computing systems, where maintaining positions can be achieved without significant modernization, striking a balance between sufficient computing power and the cost of the upgrade.
- "Japanese candlesticks" and the percentage of updated systems metric reveal a three-tiered structure of the Top500 ranking: Positions 1–50 (high turnover/race), 101–200 (stagnation), 201–500 (rapid turnover/budget cycles), where the ownership strategy is dictated by the economics of system operation.

Acknowledgements

The results were obtained in Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement No. 25-11-00181).

References

1. Benhari, A., Denneulin, Y., Desprez, F., Dufossé, F., Trystram, D.: Green HPC: An analysis of the domain based on Top500. `https://hal.science/hal-04519645` (2024), accessed: 2026-04-05

2. Elishakov, D.G., Nikitenko, D.A.: Updateability of systems and their components in the Top50 supercomputer ranking. In: Parallel Computational Technologies – XIX All-Russian Conference with International Participation, PaVT'2025, Moscow, April 8–10, 2025. Short Papers and Poster Descriptions. pp. 335–335. South Ural State University Publishing, Chelyabinsk (2025). `https://doi.org/10.14529/pct2025`

3. Feitelson, D.G.: On the interpretation of Top500 data. Int. J. High Perform. Comput. Appl. 13(2), 146–153 (1999). https://doi.org/10.1177/109434209901300204

4. Khan, A., Sim, H., Vazhkudai, S.S., Butt, A.R., Kim, Y.: An analysis of system balance and architectural trends based on Top500 supercomputers. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia 2021). pp. 11–22. ACM (2021). https://doi.org/10.1145/3432261.3432263

5. Nikitenko, D., Antonov, A., Zheltkov, A., Voevodin, V.: Describing HPC system architecture for understanding its capabilities. In: Voevodin, V., Sobolev, S. (eds.) Supercomputing. RuSCDays 2020. Communications in Computer and Information Science, vol. 1331, pp. 425–435. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64616-5_37

6. Nikitenko, D., Zheltkov, A.: The Top50 list vivification in the evolution of HPC rankings. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. Communications in Computer and Information Science, vol. 753, pp. 14–26. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67035-5_2

7. Nikitenko, D., Zhumatiy, S., Paokin, A., et al.: Evolution of the Octoshell HPC center management system. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. Communications in Computer and Information Science, vol. 1063, pp. 19–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28163-2_2

8. Nikitenko, D.A.: Top 50: Ranking of the most powerful computing systems in the CIS. In: Voevodin, V.V., Tyrtyshnikov, E.E. (eds.) Numerical Methods, Parallel Computing and Information Technologies: Collection of Scientific Works, pp. 173–182. Moscow State University Publishing, Moscow (2008)

9. Nikitenko, D.A.: Hierarchical model of architecture of supercomputer systems for comparison and ranking. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering 11(4), 5–18 (2022). https://doi.org/10.14529/cmse220401

10. Steed, C.A.: Supercomputing Top500 visualizations. https://csteed.com/Top500Vis/ (2026), accessed: 2026-04-05

11. Strohmaier, E., Meuer, H.W., Dongarra, J., et al.: The Top500 list and progress in high-performance computing. Computer 48(11), 42–49 (2015). https://doi.org/10.1109/MC.2015.338

12. The GRAPH500 Organization: The GRAPH500 List. Official website. https://graph500.org/ (2026), accessed: 2026-04-05

13. The GREEN500 Organization: The GREEN500 List. Official website. https://green500.org/ (2026), accessed: 2026-04-05

14. The IO500 Organization: The IO500 List. Official website. https://io500.org/ (2026), accessed: 2026-04-05

15. The Linux Foundation: 20 years of Top500 data show Linux's role in supercomputing breakthroughs. https://www.linuxfoundation.org/blog/blog/

20-years-of-top500-data-show-linuxs-role-in-supercomputing-breakthroughs (2013), accessed: 2026-04-05

16. The PostgreSQL Global Development Group: PostgreSQL documentation. https://www.postgresql.org/docs/ (2026), accessed: 2026-04-05

17. The Ruby on Rails Team: Ruby on Rails guides. https://guides.rubyonrails.org/ (2026), accessed: 2026-04-05

18. The Top500 Organization: The Top500 List. Official website. https://top500.org/lists/top500/ (2026), accessed: 2026-04-05

19. Top50 Project: The Top50 Supercomputers List. Official website. https://top50.parallel.ru/list (2026), accessed: 2026-04-05

20. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., et al.: Supercomputer Lomonosov-2: Large scale, deep monitoring and fine analytics for the user community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). https://doi.org/10.14529/jsfi190201

21. Voevodin, V.V., Chulkevich, R.A., Kostenetskiy, P.S., et al.: Administration, monitoring and analysis of supercomputers in Russia: a survey of 10 HPC centers. Supercomputing Frontiers and Innovations 8(3), 82–103 (2021), https://doi.org/10.14529/jsfi210305

DOI: 10.14529/jsfi260202

Study of Graphics Accelerators Performance for Numerical Solution of the Poisson Equation Using the Chebyshev Method

Sergey V. Polyakov[1], ***Marina A. Kornilina***[1], ***Tatiana A. Kudryashova***[1]

The problem of the efficient numerical solution of boundary value problems for elliptic equations on high-performance computing systems is discussed. In the context of this problem, the use of a special Chebyshev iterative method to solve such problems by the multi-core central processors and graphics accelerators is analyzed. One of the objective functions of such analysis is to compare the time of solving an elliptic boundary value problem by one node with several central processors and one graphics accelerator. The motivation for this consideration is that in many practical applications, the numerical solution of an elliptic problem significantly slows down the overall algorithm. Taking this circumstance into account, as an example, a two-dimensional Dirichlet boundary value problem for the Poisson equation with constant and variable coefficients is considered. Its solution is based on the well-known "cross" scheme on the most detailed Cartesian grid. For this statement, a modified Chebyshev iterative algorithm is proposed. Such approach significantly reduces the requirements to the used RAM. A parallel software implementation of the algorithm, designed for both central processing units (CPUs) and graphics accelerators (GPUs), is examined. The study investigates the algorithm's convergence and the performance of the used computing systems. It analyzes their dependence on the dimension of the grid problem and other parameters. Numerical experiments are used to determine the limits of the algorithm's applicability and to discuss the efficiency of its implementation on CPUs and GPUs.

Keywords: Dirichlet boundary problem for Poisson equation, cross difference scheme, Chebyshev preconditioned method, parallel implementation for central and graphics processor units.

Introduction

The problem of efficient numerical solving boundary value problems for elliptic equations is encountered in many current applications. Fast and reliable numerical algorithms for solving elliptic problems are most in demand in computational fluid dynamics, electrodynamics, strength mechanics, and other fields. The general formulation of the differential problem is as follows:

$$Lu = f, \quad \mathbf{x} \in \Omega \subset E^m, \quad lu = g, \quad \mathbf{x} \in \partial\Omega. \tag{1}$$

Here L is the elliptic differential operator with respect to coordinates x_k of m-dimensional spatial vector $\mathbf{x} = (x_1, \ldots, x_m)$, defined in a compact region Ω of Euclidean space E^m, $u = u(\mathbf{x})$ is the desired function, $f = f(\mathbf{x})$ is specified in Ω function, l is boundary differential-algebraic operator, containing spatial derivatives of no higher than the 1st order, $\partial\Omega$ is border of the region Ω. It is assumed that the input data of problem (1), the boundary for the domain and the coefficients of the equations, have the required properties. Then a solution to problem (1) exists and is unique [1].

The general numerical procedure for solving problem (1) is as follows. First, the domain Ω is discretized. As a rule, a grid Ω_h is built for this purpose with a border $\partial\Omega_h$, based on a system of individual nodes or cells. Then, based on the chosen discretization, an approximation of equations (1) is constructed. Most often, either the finite volume method [2] or the finite element method [3] is used for this aim. The approximation results in a discrete formulation of

[1]Keldysh Institute of Applied Mathematics of RAS, Moscow, Russian Federation

the form:

$$\Lambda_h y_h = f_h, \quad \mathbf{x} \in \Omega_h, \quad l_h y_h = g_h, \quad \mathbf{x} \in \partial\Omega_h. \tag{2}$$

Here Λ_h, y_h, f_h, l_h and g_h are discrete analogues of the above-mentioned continuous objects defined on sets Ω_h and $\partial\Omega_h$. Equations (2) are a system of algebraic linear (quasilinear) equations of the form:

$$\mathrm{A}\mathbf{Y} = \mathbf{F}, \tag{3}$$

here $\mathbf{Y}$ and $\mathbf{F}$ are unknown and given vectors from the real vector space $\mathbb{R}^N$, A is a real matrix from the matrix space $\mathbb{R}^{N\times N}$, N is the number of unknowns which equal to either the number of nodes/cells in used grid for the finite volume method, or the number of basic functions for the finite element method. In the quasi-linear case, it is assumed that the elements of the matrix A and vector $\mathbf{F}$ depend on the components of the vector $\mathbf{Y}$.

In this paper, we take into account many years of experience in solving algebraic problems of the form (3), obtained on the basis of approximations of elliptic equations (1). Accumulated since the early 1950s, this experience has been systematized in extensive literature (see, for example, [4–33]) and summarized in [34]. In addition to many practical methods for accelerating the solving algebraic problems of the form (3), special iterative methods are currently being developed and used, oriented towards parallel computations using high-performance systems with CPUs and GPUs. At the same time, it is known that the use of a large number of computing devices does not yet guarantee a significant acceleration. The reasons for this are the limitations of the architectures of computing systems, including insufficient communication speed between computers, an imbalance in the performance of CPUs and GPUs, as well as different RAM bandwidths.

In the current situation, the following approach can be recommended for most specific applications. For moderately sized problems (within a billion unknowns in the solution vector), it is proposed to use a separate device for its numerical solution. This device can be a computing node equipped with either several multi-core, high-performance CPUs with sufficient shared RAM, or one or more GPUs with sufficient graphics memory. This approach eliminates the problem of network communications slowing down computations, but the relatively low data transfer rate over the buses within the node remains. The second problem can be partially solved by using specialized iterative algorithms focused on parallelization on shared memory.

This paper examines the practical implementation of the proposed approach using the example of solving a two-dimensional Dirichlet problem for the Poisson equation with constant and variable coefficients. This example is the most complex due to the strong sparseness of the matrix of the algebraic system (3). In the two-dimensional case, the convergence rate of the iterations is significantly lower than, for example, when solving the spatially three-dimensional problem (1). The goal of the study is to develop and test an iterative algorithm that would allow us to solve the problems (3) of maximum dimension using a single computing node equipped with a CPU and a GPU.

The article is organized as follows. Section 1 is devoted to the problem formulation and the numerical approach to its solution. In Section 2 the iterative schemes of the Chebyshev method are discussed. Section 3 contains a description of parallel algorithms for solving the discrete problem. Section 4 describes the test problems and analyzes the results of their solution. Conclusion summarizes the study and points directions for further work.

1. Mathematical Problem and Difference Scheme

Let us consider a linear Poisson equation in a finite simply connected rectangular area of the form:

$$\frac{\partial}{\partial x_1}\left(\kappa_1 \frac{\partial u}{\partial x_1}\right) + \frac{\partial}{\partial x_2}\left(\kappa_2 \frac{\partial u}{\partial x_2}\right) = -f\left(x_1, x_2\right), \quad (x_1, x_2) \in \Omega, \tag{4}$$

with Dirichlet boundary conditions:

$$u\left(x_1, x_2\right) = g\left(x_1, x_2\right), \quad (x_1, x_2) \in \partial\Omega. \tag{5}$$

We assume that the elliptic operator used in (4) satisfies the conditions of strict ellipticity at all points. The domain Ω, κ_α coefficients and the function f are quite smooth, the border of the region $\partial\Omega$ is piecewise smooth except for a finite number of points, the boundary function g is continuous. Then the solution of the problem (4), (5) exists and is unique [1].

We will solve the problem (4), (5) numerically using the finite difference method. To do this, we will assume that Ω coincides with an open square $(0,1) \times (0,1)$. Then a uniform Cartesian grid $\bar{\Omega}_h = \bar{\omega}_{x_1} \times \bar{\omega}_{x_2}$, here $\bar{\omega}_{x_\alpha} = \{x_{\alpha,i_\alpha} = i_\alpha h_\alpha = i_\alpha / N_\alpha, \quad i_\alpha = 0, \ldots, N_\alpha\}$, $\alpha = 1, 2$, can be introduced in its closure $\bar{\Omega} = \Omega \cup \partial\Omega$. On this grid, equation (4) is approximated by the well-known "cross" difference scheme supplemented by Dirichlet grid conditions:

$$\begin{gathered}
\frac{1}{\hbar_1}\left\{\kappa_{1,i_1+1/2,i_2}\frac{y_{i_1+1,i_2}-y_{i_1,i_2}}{h_1} - \kappa_{1,i_1-1/2,i_2}\frac{y_{i_1,i_2}-y_{i_1-1,i_2}}{h_1}\right\} + \\
\frac{1}{\hbar_2}\left\{\kappa_{2,i_1,i_2+1/2}\frac{y_{i_1,i_2+1}-y_{i_1,i_2}}{h_2} - \kappa_{2,i_1,i_2-1/2}\frac{y_{i_1,i_2}-y_{i_1,i_2-1}}{h_2}\right\} = f_{i_1,i_2}, \\
i_\alpha = 1, \ldots, N_\alpha - 1, \\
y_{i_1,i_2} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2.
\end{gathered} \tag{6}$$

In discrete equations (6), the grid function y_{i_1,i_2} is an approximation to the exact solution on the grid $u_{i_1,i_2} = u\left(x_{1,i_1}, x_{2,i_2}\right)$, $\kappa_{1,i_1\pm1/2,i_2}$ and $\kappa_{2,i_1,i_2\mp1/2}$ are expressions $0.5\left[\kappa_1\left(x_{1,i_1}, x_{2,i_2}\right) + \kappa_1\left(x_{1,i_1\pm1}, x_{2,i_2}\right)\right]$ and $0.5\left[\kappa_2\left(x_{1,i_1}, x_{2,i_2}\right) + \kappa_2\left(x_{1,i_1}, x_{2,i_2\pm1}\right)\right]$, $f_{i_1,i_2} = f\left(x_{1,i_1}, x_{2,i_2}\right)$, $g_{i_1,i_2} = g\left(x_{1,i_1}, x_{2,i_2}\right)$, $\hbar_1$ and $\hbar_2$ are the average steps of the grid, in this case equal to h_1 and h_2.

Before finding a solution of the difference scheme, a well-known transformation of equations (6) to a special form is performed:

$$
\begin{gathered}
d_{i_1,i_2} y_{i_1,i_2} - a_{1,i_1,i_2} y_{i_1-1,i_2} - b_{1,i_1,i_2} y_{i_1+1,i_2} - a_{2,i_1,i_2} y_{i_1,i_2-1} - b_{2,i_1,i_2} y_{i_1,i_2+1} = \varphi_{i_1,i_2}, \\
a_{1,i_1,i_2} = \frac{\hbar_2}{h_1} \kappa_{1,i_1-1/2,i_2}, \quad b_{1,i_1,i_2} = \frac{\hbar_2}{h_1} \kappa_{1,i_1+1/2,i_2}, \\
a_{2,i_1,i_2} = \frac{\hbar_1}{h_2} \kappa_{2,i_1,i_2-1/2}, \quad b_{2,i_1,i_2} = \frac{\hbar_1}{h_2} \kappa_{2,i_1,i_2-1/2}, \\
d_{i_1,i_2} = a_{1,i_1,i_2} + b_{1,i_1,i_2} + a_{2,i_1,i_2} + b_{2,i_1,i_2}, \\
\varphi_{i_1,i_2} = \hbar_1 \hbar_2 f_{i_1,i_2}, \quad i_\alpha = 1, \ldots, N_\alpha - 1, \\
y_{i_1,i_2} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2.
\end{gathered}
\tag{7}
$$

As a result of transformation (7), the matrix of system (3) becomes symmetric and positive definite. The portrait of the matrix depends on the ordering of the unknowns in the vector $\mathbf{Y}$ and the order of the equations. With a natural ordering of the view:

$$
\mathbf{Y} = \left(y_{0,0}, y_{1,0}, \ldots, y_{N_1,0},\ y_{0,1}, y_{1,1} \ldots, y_{N_1,1},\ \ldots,\ y_{0,N_2}, y_{1,N_2}, \ldots, y_{N_1,N_2}\right),
$$

the portrait of the matrix has a block tri-diagonal structure (see Fig. 1). This implies the preliminary exclusion of the function's boundary values y_{i_1,i_2} from consideration using the standard dimensionality reduction procedure. Below, we will use precisely this ordering of unknowns and equations.

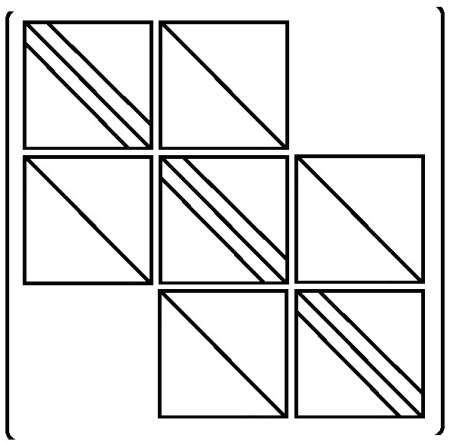


Figure 1. Portrait of the matrix of the system (3)

2. Iterative Algorithms

Let us now describe the proposed iterative approach. It is based on the Chebyshev iterative algorithm [6, 10]. We formulate it using the well-known two-layer iterative scheme [10]:

$$
\mathrm{B}_{s+1} \frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + \mathrm{A}\mathbf{Y}^s = \mathbf{F}, \quad s = 0, 1, \ldots. \tag{8}
$$

This scheme assumes setting some initial vector $\mathbf{Y}^0$ and calculating subsequent approximations to the solution of problem (3) using equations (8). The transition to the next approximation is carried out by inverting the matrices B_{s+1}. The convergence rate is adjusted by iterative parameters τ_{s+1}. A specific iterative algorithm is determined by the choice of a sequence of transition operators B_{s+1} and iterative parameters τ_{s+1}. It is known (see, for example, [10]) that scheme (8) converges to the solution of system (3) in a finite number of steps if the following conditions are met:

$$\mathrm{A} = \mathrm{A}^T > 0, \quad \mathrm{B}_{s+1} = \mathrm{B}_{s+1}^T > 0, \quad \tau_{s+1} > 0, \quad \mathrm{B}_{s+1} - 0.5\tau_{s+1}\mathrm{A} > 0, \quad s = 0, 1, \ldots. \tag{9}$$

Before solving system (3), a preconditioner matrix M is often chosen, which leads system (3) to a modified form:

$$\mathrm{M}^{-1}\mathrm{A}\mathbf{Y} \equiv \tilde{\mathrm{A}}\mathbf{Y} = \mathrm{M}^{-1}\mathbf{F} \equiv \tilde{\mathbf{F}}. \tag{10}$$

For example, in the iterative Jacobi method, the matrix M coincides with the main diagonal D_{A} of the matrix A. The purpose of such a transformation is to significantly reduce the number of iterations $n = n(\varepsilon)$ required to achieve accuracy $\varepsilon > 0$. Such a decrease occurs due to a change in the spectral properties of the matrix of the system (10). The criterion for convergence of iterations will be considered to be the achievement of a norm in the space $C(\Omega_h)$ of the discrepancy of a given value:

$$||R|| = ||\tilde{\mathbf{F}} - \tilde{\mathrm{A}}\mathbf{Y}|| \le \varepsilon. \tag{11}$$

The Chebyshev iterative method applied to the system (10) implies the use of unit transition matrices ($\mathrm{B}_{s+1} = \mathrm{I}$) and iterative parameters determined by the formulas [6, 10]:

$$\tau_{s+1} = \frac{\tau_0}{1 - \frac{\mu-1}{\mu+1}\cos\frac{\pi(2s+1)}{2n}}, \quad s = 0, \ldots, n-1, \tag{12}$$

here $\tau_0 = 2/\left(\lambda_{min}\left(\tilde{\mathrm{A}}\right) + \lambda_{max}\left(\tilde{\mathrm{A}}\right)\right)$ is the optimal parameter, $\mu = \lambda_{max}\left(\tilde{\mathrm{A}}\right)/\lambda_{min}\left(\tilde{\mathrm{A}}\right)$ is the spectral number of conditionality, $\lambda_{min}\left(\tilde{\mathrm{A}}\right)$ and $\lambda_{max}\left(\tilde{\mathrm{A}}\right)$ are minimum and maximum eigenvalues of the matrix $\tilde{\mathrm{A}}$. In this case, calculations are performed in a series of iterations $n = 2^m$, and the iterative parameters are ordered in a certain way [6].

The Chebyshev method can be combined with the Jacobi method. Such a combined method can be written either in operator form:

$$\frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + D_{\mathrm{A}}^{-1}\mathrm{A}\mathbf{Y}^s = D_{\mathrm{A}}^{-1}\mathbf{F}, \quad s = 0, 1, \ldots. \tag{13}$$

or in the form of a numerical scheme using the notation used in (7):

$$\begin{gathered}
y_{i_1,i_2}^{s+1} = (1 - \tau_{s+1})\, y_{i_1,i_2}^s + \tau_{s+1} d_{i_1,i_2}^{-1} r_{i_1,i_2}^{s+1}, \\
r_{i_1,i_2}^{s+1} = a_{1,i_1,i_2} y_{i_1-1,i_2}^s + b_{1,i_1,i_2} y_{i_1+1,i_2}^s + a_{2,i_1,i_2} y_{i_1,i_2-1}^s + b_{2,i_1,i_2} y_{i_1,i_2+1}^s + \varphi_{i_1,i_2}, \\
i_\alpha = 1, \ldots, N_\alpha - 1, \quad \alpha = 1, 2, \quad s = 0, 1, \ldots, \\
y_{i_1,i_2}^{s+1} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2, \quad s = 0, 1, \ldots.
\end{gathered} \tag{14}$$

Since one of the goals of the work is to apply the proposed iterative method in conditions of limited RAM, we will monitor the required volume for this. In particular, in the case of

problem (4) with the Laplace operator ($\kappa_\alpha \equiv 1$, $\alpha = 1, 2$), matrix storage A is not required when implementing the iterative scheme (13). RAM will be allocated only for vectors $\mathbf{Y}^{s+1}$, $\mathbf{Y}^s$, $\mathbf{F}$, which will amount to $3N$ cells, where $N = (N_1 + 1)(N_2 + 1)$. More generally, when $\kappa_\alpha \neq \text{const}$, $\alpha = 1, 2$, 4 more side diagonals of the matrix A are stored, which will require a total of $7N$ cells.

Then let us consider the issue of computational costs in the implementation of the iterative scheme (13). As is well known (see, for example, [10]), the general number of arithmetic operations for this scheme is $Q_1 = q_1 n(\varepsilon)$, where q_1 is the number of operations required to implement one step of the iterative scheme, $n(\varepsilon)$ is the number of such steps required to achieve accuracy ε. In the case of highly sparse matrices A value $q_1 = O(N)$, and it is not possible to lower it. Value $n(\varepsilon)$ depends on the scheme of the iterative algorithm and its parameters. In the general situation $n(\varepsilon) = O(N\ln(2/\varepsilon))$. This estimate is valid for two-layer schemes with a diagonal transition operator $\mathrm{B}_{s+1} = \mathrm{D}$. However, when using the Chebyshev parameter set or the three-layer Chebyshev method [10], the estimate of the number of iterations $n(\varepsilon)$ is reduced to $O\left(\sqrt{N}\ln(2/\varepsilon)\right)$. This fact explains the choice of this method as the basis for further analysis.

A variety of approaches are used to achieve a more significant acceleration of iteration convergence. The most popular of them are methods such as conjugate and biconjugate gradients [5–16, 20, 24] using Krylov subspaces of various dimensions. These methods are three-layered and automatically estimate the rate of convergence of iterations $O\left(\sqrt{N}\ln(2/\varepsilon)\right)$. When factorized matrices are used as transition operators B_{s+1} in them, for example, those close to Gaussian decomposition $\mathrm{A} = \mathrm{LU}$, an even higher convergence rate (of the order $O\left(N^{1/4}\ln(2/\varepsilon)\right)$ and lower) is achieved. However, the price for this acceleration is an increase in the required RAM and the use of additional information about the spectrum of the matrix A. In the most difficult-to-implement variants (see, for example, [20, 24]), not only additional RAM is required, but there is also an increase in labor intensity (increase q_1), which does not adapt well to the architectures of GPU systems.

It is also worth to mention the methods of variable directions [10], which have a maximum convergence rate (of the order), but are suitable only for solving problems in rectangular areas on Cartesian grids. Similar asymptotics can be obtained using multigrid iterative methods [31–33]. However, even here, complex logic and the re-sorting of unknowns and equations are difficult to implement on GPU systems.

Without disputing the advantages of the above methods, we will not consider them in this paper due to the available objective function – saving computing RAM when solving large-dimensional problems and simplicity of parallel implementation on hybrid nodes with CPU and GPU.

Another possibility to accelerate the convergence of iterations within the framework of the Chebyshev approach is the use of the alternating triangular method [4, 6, 10, 15, 28, 30, 35–40]. Its formulation is obtained if a factorized matrix of the form is used as the preconditioner matrix $\mathrm{M} = (\mathrm{D_A} + \mathrm{A_-})\,\mathrm{D_A^{-1}}\,(\mathrm{D_A} + \mathrm{A_+})$, here $\mathrm{A_-}$ and $\mathrm{A_+}$ are lower and upper triangular matrices, which together with $\mathrm{D_A}$ make up the original matrix $\mathrm{A} = \mathrm{D_A} + \mathrm{A_-} + \mathrm{A_+}$. Considering that the inversion of the matrix factors M to obtain M^{-1} does not increase the asymptotic value q_1 (it still amounts to $O(N)$), the scheme of the method can also be written as:

$$\left(\mathrm{I} + \mathrm{D_A^{-1}A_-}\right)\left(\mathrm{I} + \mathrm{D_A^{-1}A_+}\right)\frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + \mathrm{D_A^{-1}A}\mathbf{Y}^s = \mathrm{D_A^{-1}}\mathbf{F}, \quad s = 0, 1, \ldots. \tag{15}$$

Scheme (15) is transformed into a four-stage algorithm that requires additional storage of one vector (the residual vector $\mathbf{R}^s = \mathrm{D}_\mathrm{A}^{-1}\mathbf{F} - \mathrm{D}_\mathrm{A}^{-1}\mathrm{A}\mathbf{Y}^s$), compared with algorithm (13). Then the total required memory is either $4N$, or $8N$ cells. The convergence rate increases, including through the use of Chebyshev acceleration. Scheme (15) will be used mainly for the case of variable coefficients in equation (4).

3. Parallel Realization

The above-mentioned iterative schemes (13) and (15) have been adapted to parallel computing. In this context, scheme (13) has been discussed in many papers, scheme (15) has been studied in [35–40]. In this paper, the issue of software implementation of iterative schemes (13) and (15) on a node with multi-core processors and on a graphics accelerator was considered. As a result, the following solutions were implemented.

OpenMPI was chosen as the parallel programming standard for nodes with CPUs [41]. The motivation for this solution was that the most efficient calculations on shared memory are implemented when each core interacts with its individual RAM block. The OpenMPI utilities provide such work discipline due to their focus on distributed computing. In addition, the modern implementation of OpenMPI takes into account the hardware features of computing nodes, up to the automatic allocation of NUMA devices [42]. When using other computing standards (POSIX Threads [43], OpenMP [44]), these problems have to be solved independently and, as a rule, inefficiently.

The parallelization technique was based on the area separation method [45]. At the same time, the topology of the initial problem was taken into account, namely, the two-dimensionality and configuration of the area Ω. As a result, the difference grid was split into two spatial directions and compared with a two-dimensional grid of computers, which were CPU cores. The implementation of iterative schemes (13) and (15) did not require the convolution of the grid function into a linear vector. The final code was developed in the ANSI C language.

When developing parallel implementations of iterative schemes (13) and (15) for GPU computing, the OpenCL parallel programming standard was used [46]. This made it possible to use a single code for calculations on NVidia and AMD graphics accelerators. At the same time, the main program was developed in C++ and used thread separation based on the OpenMP standard to speed up work. In fact, within the framework of the general iterative scheme implemented on the CPU, time-consuming operations of matrix-vector multiplication and scalar product calculations were performed on the GPU. This approach has been proposed and successfully tested in many papers (see, for example, [47–55]). The parallelization technique in this case required convolution of the tabular grid function $y_h = \{y_{i_1,i_2}\}$ in a linear vector $\mathbf{Y}$. It was divided into adjacent blocks in accordance with the number of GPU multiprocessors. If there are several GPUs, you can distribute the implementation of each iteration between them. However, this will not be efficient enough when using the PCI Express bus.

4. Testing of Parallel Iterative Algorithms

Three tasks were used to test the proposed parallel iterative algorithms. The task data is shown in Tab. 1.

Table 1. The initial data of the test tasks

Task No.	View of coefficients κ_α	The exact solution formula
1	$\kappa_\alpha \equiv 1, \alpha = 1, 2$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2)$
2	$\kappa_\alpha \equiv 1, \alpha = 1, 2$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2) \cosh(a(x_1 - x_2))$
3	$\kappa_\alpha = 1 + b \cdot \kappa(x_1)\kappa(x_2),\ \alpha = 1, 2$ $\kappa(x) = \exp\left[-(x - 0.5)^4 / c^4\right]$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2) \cosh(a(x_1 - x_2))$

The right-hand side of equation (4) was calculated based on a given exact solution:

$$f(x_1, x_2) = -\frac{\partial}{\partial x_1}\left(\kappa_1 \frac{\partial u}{\partial x_1}\right) - \frac{\partial}{\partial x_2}\left(\kappa_2 \frac{\partial u}{\partial x_2}\right). \tag{16}$$

In tasks 1–3, the parameters are $k_1 = 2$, $k_2 = 2$; in tasks 2 and 3 $a = 2$, in task 3 parameter b is ranged from 1 to 10^3, parameter $c = 0.1$. The general view of the solutions to problem 1 is shown in Fig. 2, tasks 2 and 3 are shown in Fig. 3 and Fig. 4 shows the dependence of the coefficients on the coordinate at a fixed value of the coordinate $x_2 = 0.5$. The dependence κ_α on the coordinate x_2 was similar. The type of exact solutions determined their values at the boundary: $g(x_1, x_2) \equiv 0$.

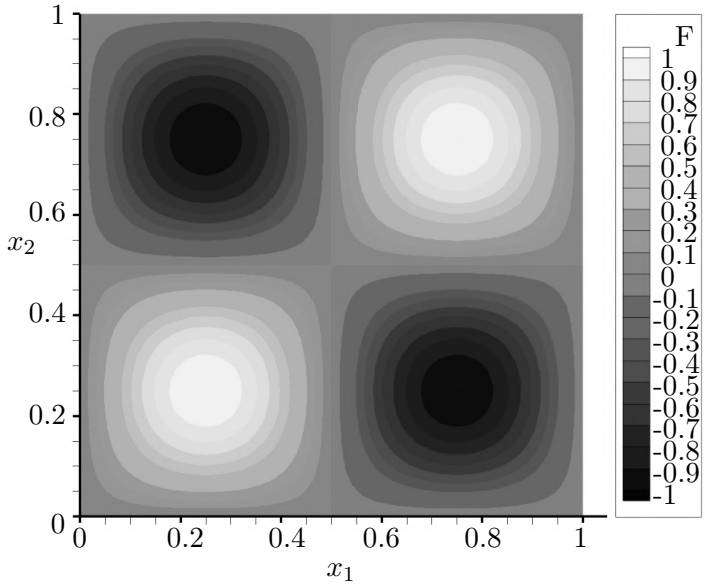


Figure 2. General view of the solution of task 1

Let us explain the choice of test tasks. The first two tasks use the Laplace operator as a differential operator. This is convenient because all the parameters of the spectrum at the differential and discrete levels are known for this operator. At the same time, task 1 corresponds to the case of separation of variables, in which the solution is represented by a finite Fourier series. In task 2, separation of variables is impossible, and the Fourier series becomes infinite. The task 3 differs from task 2 in the presence of dependencies of coefficients κ_α simulating an insert made of a material with significantly different properties. The latter allows us to study the influence of the number of conditionality of the corresponding matrix of the system (3) on the convergence of iterative algorithms. It should also be noted that even fast convergence of

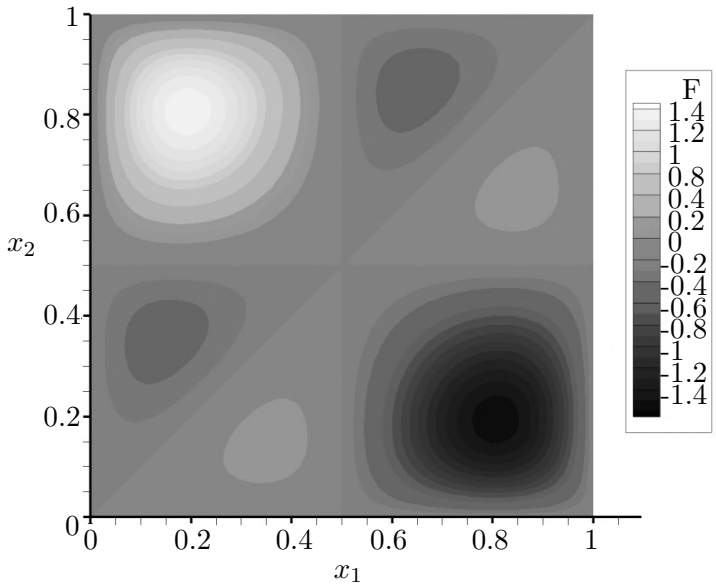


Figure 3. General view of solving tasks 2 and 3

iterations is not a guarantee of obtaining a solution to the numerical scheme with an accuracy corresponding to the order of approximation.

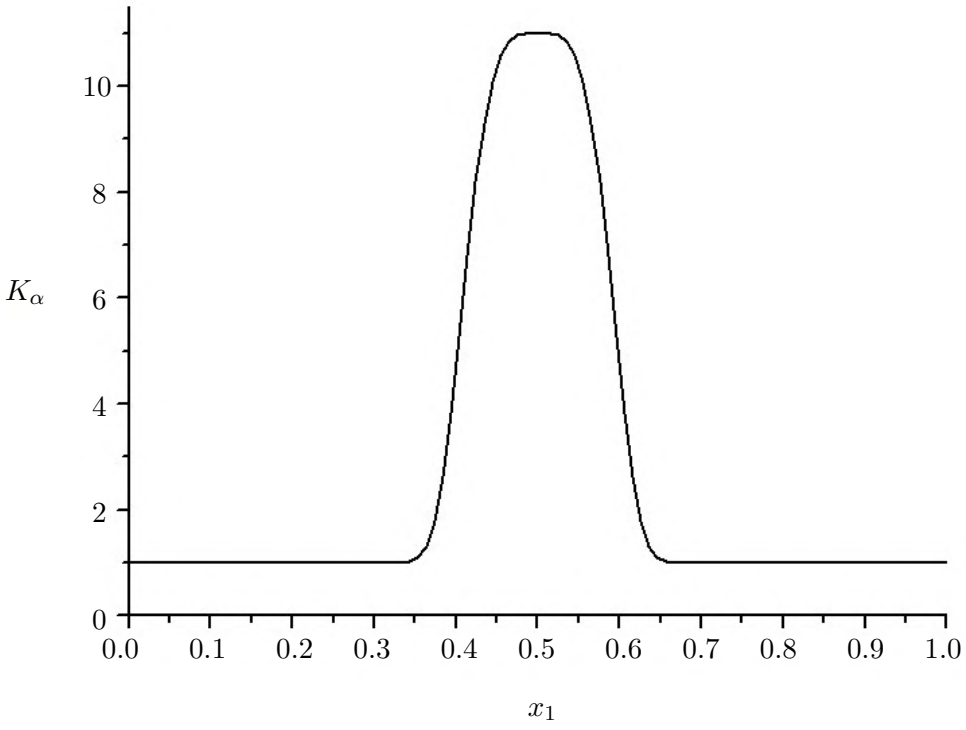


Figure 4. Dependence κ_α from x_1 for $x_2 = 0.5$ in task 3

During the test calculations, the IMM23, K60GPU, K120AMD, and K120NVI computing systems installed at the Keldysh IPM RAS Supercomputer Center for Collective Use were used. The characteristics of the systems used are shown in Tab. 2.

Table 2. Characteristics of the computing systems used

Name	Quantity CPU	Model and characteristics CPU	Quantity GPU	Model and characteristics GPU
Server IMM23	2	AMD EPYC 9124, 1500 MHz, 16 Cores, 32 Threads, Mem 128 Gb	1	gfx908 (AMD Instinct MI100 32Gb, PCIe Gen4), 1502 MHz, 120 CU, 7680 stream cores
Node K60GPU	2	Intel(R) Xeon(R) Gold 6142, 2600 MHz, 16 Cores, 32 Threads, Mem 384 Gb	4	GV100 (Nvidia V100 32Gb, PCIe Gen4), 1230–1380 MHz, 84 SM, TMU 640, CUDA cores 5120
Node K120AMD	2	Intel(R) Xeon(R) Gold 6444Y, 4000 MHz, 16 Cores, 32 Threads, Mem 512 Gb	4	gfx90a (AMD Instinct MI210 64Gb HBM2e, PCIe Gen4), 1700 MHz, 104 CU, 6656 stream cores
Node K120NVI	2	Intel(R) Xeon(R) Gold 6444Y, 4000 MHz, 16 Cores, 32 Threads, Mem 512 Gb	4	GH100 (NVidia H100 80GB, PCIe Gen5), 1095-1755 MHz, 114 SM, TMU 456, CUDA cores 14592

Let us consider the test results for solving task 1. Calculations were performed using both iterative schemes on the K60GPU cluster node and on the IMM23 server on central and graphics processors. The calculations used codes implemented in the ANSI C/C++ language using the OpenMPI and OpenCL parallel computing standards. The purpose of the calculations was to demonstrate Chebyshev acceleration of convergence of implemented iterative methods. Table 3 and Table 4 show the results of the calculations performed. For square grids, the number of nodes was chosen equal 2^k. When analyzing the above results, it is necessary to take into account that usually in Chebyshev methods, the accuracy of iteration is consistent with the approximation error of the difference scheme. In this case, we set the accuracy, which is about two decimal orders of magnitude higher than the expected accuracy of the difference scheme.

The data in Tab. 3 show that the applied variant of the conditional Chebyshev method (13) has a root dependence on the total number of grid elements, that is $n(\varepsilon) = O\left(\sqrt{n_1 \cdot n_2} \cdot \ln[2/\varepsilon]\right)$. This result is also illustrated in Fig. 5, where the dependence of the normalized number of iterations is shown $\bar{n}(\varepsilon) = n(\varepsilon)\left(\sqrt{n_1 \cdot n_2} \cdot \ln[2/\varepsilon]\right)^{-1}$.

When analyzing the results shown in Tab. 3, it can be seen that on the 2048×2048 and 4096×4096 grids, the numerical solution was obtained with approximately the same accuracy. Here the reason lies in reaching the threshold of machine accuracy in the second case. This leads to the fact that the right-hand side in equations (7) is calculated with large errors. In this case, it is possible to correct the situation by artificially multiplying the vector of the right side by a coefficient of the form 2^k. The resulting solution is then divided by this coefficient. This technique, let us call it "binary scaling", allows you to get a solution with the required accuracy even on detailed grids. Below, it is used in the calculations of tasks 1–3 for both iterative schemes.

Table 3. Test results for solving task 1 according to scheme (13)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Accuracy of the numerical solution, Δ
K60GPU, CPU, OpenMPI					
128×128	1 (1×1)	0.132	616	1e-6	3.937e-06
256×256	1 (1×1)	0.912	1408	5e-7	1.297e-04
512×512	4 (2×2)	1.926	3362	1e-7	8.117e-06
1024×1024	16 (4×4)	4.752	7681	5e-8	8.773e-06
2048×2048	32 (4×8)	28.437	17438	1e-8	7.152e-07
4096×4096	32 (4×8)	387.560	36862	5e-9	7.646e-07
K60GPU, GPU, OpenCL					
1024×1024	-	0.334	7681	5e-8	8.773e-06
2048×2048	-	2.159	17438	1e-8	7.152e-07
4096×4096	-	38.392	36862	5e-9	7.646e-07
IMM23, CPU, OpenMPI					
128×128	1 (1×1)	0.056	616	1e-6	3.937e-06
256×256	1 (1×1)	0.527	1408	5e-7	1.297e-04
512×512	4 (2×2)	1.297	3362	1e-7	8.117e-06
1024×1024	16 (4×4)	3.153	7681	5e-8	8.773e-06
2048×2048	32 (4×8)	20.921	17438	1e-8	7.152e-07
4096×4096	32 (4×8)	256.834	36862	5e-9	7.646e-07
IMM23, GPU, OpenCL					
1024×1024	-	0.518	7681	5e-8	8.773e-06
2048×2048	-	2.803	17438	1e-8	7.152e-07
4096×4096	-	47.639	36862	5e-9	7.646e-07

The data in Tab. 4 demonstrate a slower convergence of the iterative scheme (15). The reason is that this scheme used a series of Chebyshev parameters of length 2. In this situation, the root dependence of the number of iterations $n(\varepsilon)$ changes to a linear one as the dimension of problem (3) increases (see Fig. 5). For longer series, scheme (15) slows down even more. As a result, its variant with two iterative parameters is further used, which are calculated using formulas (12) with the selected parameter τ_0.

It should also be noted that when computing on processors from different manufacturers, the number of iterations may vary slightly due to different implementations of floating-point arithmetic. This is clearly seen from the data in Tab. 4. Since a more complex chain of calculations is implemented in the iterative scheme (15), rounding errors have a stronger effect on detailed grids when calculating task 1.

Let us now consider the results of solving task 2 using scheme (13) on large grids. The results were obtained on all the above-mentioned computers (see Tab. 5) using parallel technologies OpenMP for CPU and OpenCL for GPU. The calculations were performed with the parameter $\varepsilon = 10^{-6}$. The number of iterations was $n(\varepsilon) = 929, 1773, 3377, 4925, 5770, 7186, 8596, 10001, 11397$ for the grids specified in Tab. 5.

Table 4. Test results for solving task 1 according to scheme (15)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Parameter τ_0	Accuracy of the numerical solution, Δ
K60GPU, CPU, OpenMPI						
128×128	1 (1×1)	0.643	1092	1e-6	0.94	1.644e-04
256×256	1 (1×1)	7.622	4526	5e-7	0.97	2.761e-05
512×512	4 (2×2)	125.571	20646	1e-7	0.98	7.958e-06
1024×1024	16 (4×4)	629.533	94353	5e-8	0.98	2.180e-06
2048×2048	32 (4×8)	2536.435	416113	1e-8	0.99	5.904e-07
4096×4096	32 (4×8)	4980.037	1738673	5e-9	0.99	9.507e-08
K60GPU, GPU, OpenCL						
1024×1024	-	212.522	94353	5e-8	0.98	2.180e-06
2048×2048	-	609.667	416113	1e-8	0.99	5.904e-07
4096×4096	-	352.039	1738673	5e-9	0.99	9.507e-08
IMM23, CPU, OpenMPI						
128×128	1 (1×1)	0.375	1092	1e-6	0.94	1.644e-04
256×256	1 (1×1)	6.488	4526	5e-7	0.97	2.761e-05
512×512	4 (2×2)	30.236	20646	1e-7	0.98	7.958e-06
1024×1024	16 (4×4)	139.570	94352	5e-8	0.98	2.180e-06
2048×2048	32 (4×8)	2571.582	416119	1e-8	0.99	5.904e-07
4096×4096	32 (4×8)	5933.479	1733057	5e-9	0.99	9.572e-08
IMM23, GPU, OpenCL						
1024×1024	-	33.623	94352	5e-8	0.98	2.180e-06
2048×2048	-	453.943	416119	1e-8	0.99	5.904e-07
4096×4096	-	951.392	1733057	5e-9	0.99	9.572e-08

Analyzing the data in Tab. 5, we see that modern graphics accelerators can solve two-dimensional elliptic boundary value problems of sufficiently large dimensions, making them suitable for most current applications. Among the GPUs we used, the GH100 supports the maximum elliptic problem dimension. For variable coefficients, it reaches 32768×32768. Table 5 also shows that the main limiting factor in computational speed is RAM bandwidth. This applies to both CPUs and GPUs. The relative speedup achieved using a GPU versus a CPU node is shown in Fig. 6 for all four systems. They demonstrate that as the problem size increases, the GPU increasingly outperforms the CPU node.

Let us now discuss the application of scheme (15) for the case of variable coefficients of equation (4) using the example of solving task 3. The calculations were performed on the IMM23 server. Table 6 presents the data obtained on grids of different dimensions and for different values b. As expected, the convergence of scheme (15) when using large grids deteriorates with increasing b. This is again due to rounding errors associated with the calculations of matrix elements. Multigrid technology will likely help overcome this dependence.

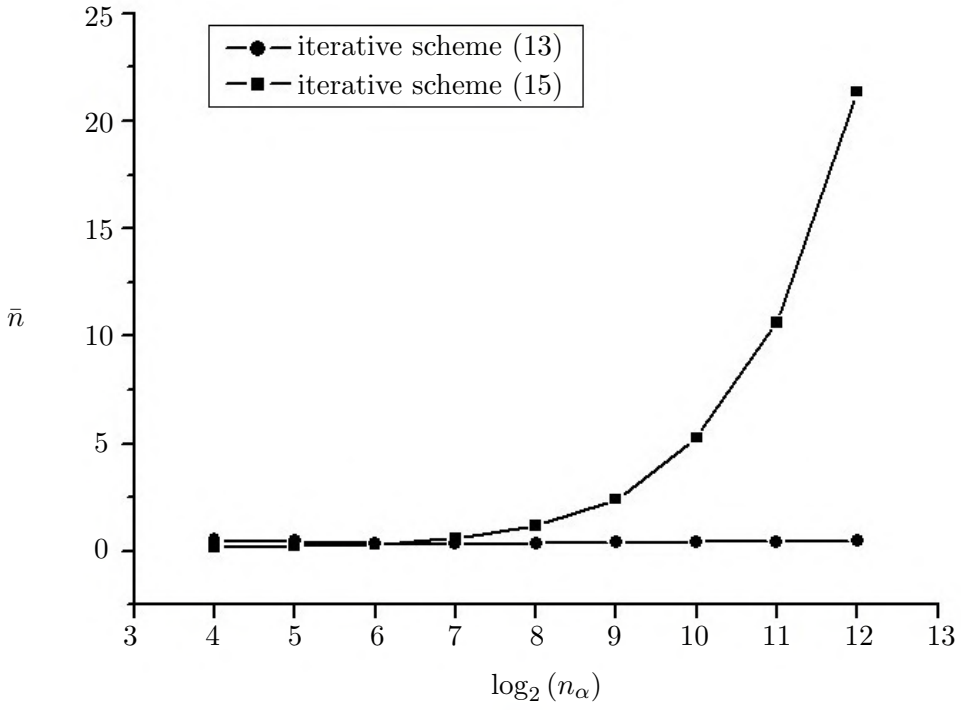


Figure 5. Dependence of the normalized number of iterations on the grid size when solving task 1

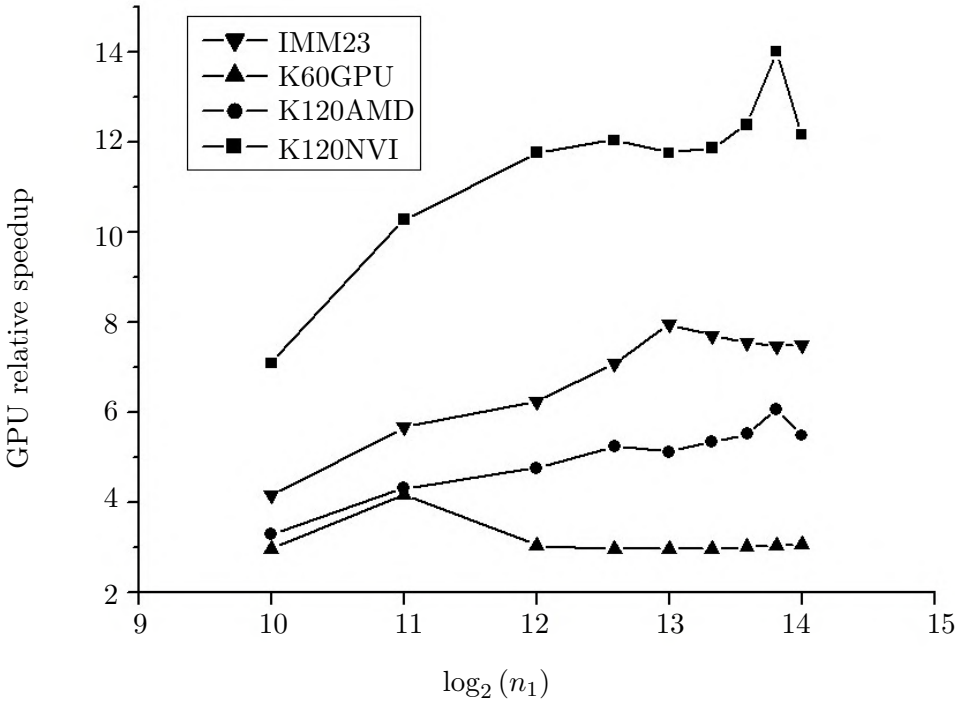


Figure 6. Relative acceleration ($S = GPU\_time/CPU\_time$) dependencies on grid size, obtained when solving task 2

Table 5. Test results for solving task 2 according to scheme (13)

Grid size $n_1 \times n_2$	Computer type and technology	Time (sec) Server IMM23	Time (sec) Node K60GPU	Time (sec) Node K120AMD	Time (sec) Node K120NVI
1024×1024	CPU, OpenMP	3.408	1.567	1.734	1.644
1024×1024	**GPU, OpenCL**	**0.821**	**0.529**	**0.528**	**0.232**
2048×2048	CPU, OpenMP	23.708	13.413	13.159	12.700
2048×2048	**GPU, OpenCL**	**4.185**	**3.224**	**3.055**	**1.236**
4096×4096	CPU, OpenMP	178.904	70.224	99.038	97.019
4096×4096	**GPU, OpenCL**	**28.686**	**23.118**	**20.797**	**8.246**
6144×6144	CPU, OpenMP	593.188	221.443	311.050	317.346
6144×6144	**GPU, OpenCL**	**83.707**	**74.891**	**59.440**	**26.355**
8192×8192	CPU, OpenMP	1298.320	460.028	655.477	641.901
8192×8192	**GPU, OpenCL**	**163.040**	**155.085**	**128.100**	**54.596**
10240×10240	CPU, OpenMP	2381.910	890.509	1253.330	1254.130
10240×10240	**GPU, OpenCL**	**309.563**	**300.987**	**234.493**	**105.671**
12288×12288	CPU, OpenMP	4188.890	1579.790	2215.260	2266.440
12288×12288	**GPU, OpenCL**	**554.968**	**524.636**	**402.219**	**182.974**
14336×14336	CPU, OpenMP	6604.910	2490.220	3800.570	4047.06
14336×14336	**GPU, OpenCL**	**884.013**	**819.417**	**626.890**	**288.752**
16384×16384	CPU, OpenMP	9494.430	3723.690	4986.160	5222.420
16384×16384	**GPU, OpenCL**	**1266.540**	**1218.240**	**909.693**	**429.445**

Table 6. Test results for solving task 3 according to scheme (15)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Parameter τ_0	Accuracy of the numerical solution, Δ
$b = 1$						
128×128	1 (1×1)	0.617	1763	1e-6	0.93	1.872e-04
256×256	4 (2×2)	2.710	7428	5e-7	0.96	4.493e-05
512×512	16 (4×4)	13.126	35564	1e-7	0.98	1.191e-05
1024×1024	32 (4×8)	114.466	147312	5e-8	0.99	2.871e-06
2048×2048	32 (4×8)	4022.841	650993	1e-8	0.99	7.285e-07
2048×2048	**GPU, OpenCL**	**710.624**	**650993**	**1e-8**	**0.99**	**7.285e-07**
$b = 8$						
128×128	1 (1×1)	0.454	1248	1e-6	0.88	3.939e-04
256×256	4 (2×2)	1.901	5210	5e-7	0.92	1.433e-04
512×512	16 (4×4)	9.349	25633	1e-7	0.97	2.264e-05
1024×1024	32 (4×8)	82.100	107287	5e-8	0.98	7.278e-06
2048×2048	32 (4×8)	2986.452	484223	1e-8	0.98	1.655e-06
2048×2048	**GPU, OpenCL**	**535.015**	**484223**	**1e-8**	**0.98**	**1.655e-06**
$b = 128$						
128×128	1 (1×1)	0.447	1278	1e-6	0.79	2.240e-03
256×256	4 (2×2)	1.838	5046	5e-7	0.74	6.291e-04
512×512	16 (4×4)	8.959	24638	1e-7	0.77	1.499e-04
1024×1024	32 (4×8)	80.111	104803	5e-8	0.77	4.052e-05
2048×2048	32 (4×8)	2958.324	478542	1e-8	0.77	9.888e-06
2048×2048	**GPU, OpenCL**	**529.501**	**478542**	**1e-8**	**0.77**	**9.888e-06**
$b = 1024$						
128×128	1 (1×1)	0.482	1340	1e-6	0.72	5.722e-03
256×256	4 (2×2)	1.833	5040	5e-7	0.71	1.515e-03
512×512	16 (4×4)	7.437	20228	1e-7	0.76	3.960e-04
1024×1024	32 (4×8)	65.711	85708	5e-8	0.77	1.066e-04
2048×2048	32 (4×8)	2461.501	398062	1e-8	0.77	2.594e-05
2048×2048	**GPU, OpenCL**	**430.032**	**398062**	**1e-8**	**0.77**	**2.594e-05**

Conclusion

The problem of effective numerical solution of boundary value problems for elliptic equations on central processors and graphics accelerators is considered. The two-dimensional Dirichlet boundary value problem for the Poisson equation with constant and variable coefficients was chosen as an example. For this formulation, it is proposed to use the classical "cross" difference scheme on a Cartesian grid and the modified Chebyshev iterative method. Based on the Chebyshev method, two iterative schemes using the Jacobi transition operator or alternating triangular factorization have been developed. For these schemes, parallel software implementations are presented, made for both central processors and graphics accelerators. The convergence of iterative schemes and the performance of computing systems have been studied depending on the problem dimension and other parameters. The limits of applicability of the developed approach have been determined in numerical experiments. The main conclusion of the work is that in many applications it is sufficient to allocate one calculator for the numerical solution of an elliptic boundary value problem on a grid of moderate volume. This computing unit can be a node with either multiple CPUs or multiple GPUs. The choice of a specific configuration is determined by the complexity of the task and the availability of the necessary resources to solve it.

References

1. Keldysh, M.V.: On the solvability and stability of the Dirichlet problem. Advances in Mathematical Sciences (8), 171–231 (1941). (in Russian)

2. Eymard, R., Gallouet, T.R., Herbin, R.: The finite volume method. Handbook of Numerical Analysis 7, 713–1020 (2000). https://doi.org/10.1016/S1570-8659(00)07005-8

3. Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z.: The finite element method: Its basis and fundamentals. Butterworth-Heinemann, Oxford (2013). https://doi.org/10.1016/C2009-0-24909-9

4. Axelsson, O.: A generalazed SSOR method. BIT 12, 443–467 (1972). https://doi.org/10.1007/BF01932955

5. Marchuk, G.I., Kuznetsov, Yu.A.: Iterative methods and quadratic functionals. Nauka. Sib. otdel., Novosibirsk (1972). (in Russian)

6. Bakhvalov, N.S.: Numerical methods: Analysis, algebra, ordinary differential equations. MIR Publishers, Moscow (1977).

7. Meijerink, J.A., van der Vorst, H.A.: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Math. Comp. 31(137), 148–162 (1977). https://doi.org/10.1090/S0025-5718-1977-0438681-4

8. Gustafsson, I.: A class of first order factorization methods. BIT 18, 142–156 (1978). https://doi.org/10.1007/BF01931691

9. Ortega, J.M.: Introduction to parallel and vector solution of linear systems. Plenum Press, New York (1988). https://doi.org/10.1007/978-1-4899-2112-3

10. Samarskii, A.A., Nikolaev, E.S.: Numerical methods for grid equations. Birkhauser Verlag, Basel-Boston-Berlin (1989). https://doi.org/10.1007/978-3-0348-9272-8

11. Il'in, V.P.: Iterative incomplete factorization methods. World Sci. Publ., Singapore (1992). https://doi.org/10.1142/1677

12. Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM Journal on Scientific Computing 13, 631–644 (1992). https://doi.org/10.1137/0913035

13. Axelsson, O.: Iterative solution methods. Cambridge Univ. Press, Cambridge (1994). https://doi.org/10.1017/CBO9780511624100

14. Kaporin, I.E.: New convergence results and preconditioning strategies for the conjugate gradient method. J. Numer. Linear. Algebra Applic. 1(2), 179–210 (1994). https://doi.org/10.1002/nla.1680010208

15. Kaporin, I.E.: High quality preconditioning of a general symmetric positive definite matrix based on its UTU+UTR+RTU-decomposition. J. Numer. Linear. Algebra Applic. 5(6), 483–509 (1998). https://doi.org/10.1002/(SICI)1099-1506(199811/12)5:6<483::AID-NLA156>3.0.CO;2-7

16. Axelsson, O., Kaporin, I.: On the sublinear and superlinear rate of convergence of conjugate gradient methods. Numerical Algorithms (25), 1–22 (2000). https://doi.org/10.1023/A:1016694031362

17. Milyukova, O.Yu.: Parallel approximate factorization method for solving discrete elliptic equations. Parallel Computing 27(10), 1365–1379 (2001). https://doi.org/10.1016/S0167-8191(01)00092-8

18. Higham, N.J.: Accuracy and stability of numerical algorithms. SIAM, Philadelphia (2002). http://doi.org/10.1137/1.9780898718027

19. Saad, Y.: Iterative methods for sparse linear systems, 2nd ed. SIAM, Philadelphia (2003). http://doi.org/10.1137/1.9780898718003

20. Van der Vorst, H.A.: Iterative Krylov methods for large linear systems. Cambridge Univ. Press, Cambridge (2003). https://doi.org/10.1017/CBO9780511615115.015

21. Il'in, V.P.: Finite element methods and technologies. ICM&MG SBRAS Publ., Novosibirsk, (2007). (in Russian)

22. Knight, P.: The Sinkhorn-Knopp algorithm: convergence and applications. SIAM J. Matrix. Anal. Appl. 30(1), 261–275 (2008). https://doi.org/10.1137/060659624

23. Walkert, H.F., Ni, P.: Anderson acceleration for fixed-point iterations. SIAM J. on Num. Analysis 49(4), 1715–1735 (2011). https://doi.org/10.1137/10078356X

24. Liesen, J., Strakos, Z.: Krylov subspace methods, principles and analysis. Oxford Univ. Press, Oxford (2013). https://doi.org/10.1093/acprof:oso/9780199655410.001.0001

25. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Numerical mathematics and scientific computation. 2nd ed. Oxford Univ. Press, Oxford (2014). https://doi.org/10.1093/acprof:oso/9780199678792.001.0001

26. Olshanskii, M.A., Tyrtyshnikov, E.E.: Iterative methods for linear systems theory and applications. SIAM, Philadelphia (2014). https://doi.org/10.1137/1.9781611973464

27. Pratara, P.P., Suryanarayana, P., Pask, J.E.: Anderson acceleration of the Jacobi iterative method. An efficient alternative to Krylov methods for large, sparse linear systems. J. Comput. Phys. 306, 43–54 (2016). https://doi.org/10.1016/j.jcp.2015.11.018

28. Vabishchevich, P.N., Zakharov, P.E.: Alternating triangular schemes for convection-diffusion problems. Comput. Math. Math. Phys. 56(4), 576–592 (2016). https://doi.org/10.7868/S0044466916040165

29. Anzt, H., Huckle, T.K., Brackle, J., Dongarra, J.: Incomplete sparse approximate inverses for parallel preconditioning. Parallel Computing 71, 1–22 (2018). https://doi.org/10.1016/j.parco.2017.10.003

30. Vabishchevich, P.N.: Alternating triangular schemes for second-order evolution equations. Comput. Math. Math. Phys. 59(2), 266–274 (2019). https://doi.org/10.1134/S0965542519020155

31. Fedorenko, R.P.: Relaxation method for solving difference elliptic equations. Zh. Vych. Matem. i Matem. Fiz. 1(5), 922–927 (1961). (in Russian)

32. Brandt, A.: Guide to multigrid development. In: Hackbusch, W., Trottenberg, U. (eds.) Multigrid Methods. Lect. Notes in Matem., vol. 960., pp. 220–312. Springer, Berlin, Heidelberg (1982). https://doi.org/10.1007/BFb0069930

33. Xu, J., Zikatanov, L.: Algebraic multigrid methods. Acta Numerica, 26, 591–721 (2017). https://doi.org/10.1017/S0962492917000083

34. Il'in, V.P.: Iterative preconditioned methods in Krylov spaces: trends of the 21st Century. Comput. Math. Math. Phys. 61(11), 1750–1775 (2021). https://doi.org/10.1134/S0965542521110099

35. Milyukova, O.Yu.: A parallel version of the generalized alternating triangular method for elliptic equations. Comput. Math. Math. Phys. 38(12), 1922–1932 (1998). (in Russian)

36. Milyukova, O.Yu.: Parallel versions of the alternating triangular method for solving three-dimensional elliptic equations. Comput. Math. Math. Phys. 42(10), 1472–1481 (2002). (in Russian)

37. Kaporin, I.E.: Using Chebyshev polynomials and approximate inverse triangular factorizations for preconditioning the conjugate gradient method. Comput. Math. Math. Phys. 52(2), 169–193 (2012). https://doi.org/10.1134/S0965542512020091

38. Milyukova, O.Yu.: Combination of numerical and structured approaches to the construction of a second-order incomplete triangular factorization in parallel preconditioning methods. Comput. Math. Math. Phys. 56(5), 699–716 (2016). https://doi.org/10.1134/S096554251605016X

39. Milyukova, O.Yu.: MPI+OpenMP parallel implementation of conjugate gradient method with factored implicit preconditioners. Math. Models Comput. Simul. 14(3), 367–380 (2022). https://doi.org/10.1134/S2070048222030103

40. Milyukova, O.Yu.: Some ways of parallel implementation of the conjugate gradient method with an implicit factorized preconditioner. Math. Models Comput. Simul. 16(4), 638–653 (2024). https://doi.org/10.1134/S2070048224700285

41. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/, accessed: 2026-05-20

42. Majo, Z., Gross, T.R.: Memory System Performance in a NUMA Multicore Multiprocessor. http://people.inf.ethz.ch/zmajo/publications/11-systor.pdf (2011), accessed: 2026-05-20

43. Barney, B.: POSIX Threads Programming. https://computing.llnl.gov/tutorials/pthreads/, accessed: 2026-05-20

44. Specifications – OpenMP. https://www.openmp.org/specifications, accessed: 2026-05-20

45. Domain Decompositon Methods (DDM). https://www.ddm.org/, accessed: 2026-05-20

46. OpenCL – The Open Standard for Parallel Programming of Heterogeneous Systems. https://www.khronos.org/opencl/, accessed: 2026-05-20

47. Kuo, F.-A., Smith, M.R., Hsieh, Ch.-W., *et al.*: GPU acceleration for general conservation equations and its application to several engineering problems. Computers & Fluids 45(1), 147–154 (2011). https://doi.org/10.1016/j.compfluid.2010.10.007

48. Niemeyer, K.E.: GPU Laplace solver using optimized red-black Gauss-Seidel with SOR solver. https://github.com/kyleniemeyer/laplace_gpu (2012), accessed: 2026-05-20

49. Dominguez, J.M., Crespo, A.J.C., Valdez-Balderas, D., *et al.*: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Computer Physics Communications 184(8), 1848–1860 (2013). https://doi.org/10.1016/j.cpc.2013.03.008

50. Phillips, E., Fatica, M.: A CUDA implementation of the high performance conjugate gradient benchmark. In: Jarvis, S., Wright, S., Hammond, S. (eds.) High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation. PMBS 2014. Lecture Notes in Computer Science, vol. 8966, pp. 68–84. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17248-4_4

51. Menshov, I., Pavlukhin, P.: Highly scalable implementation of an implicit matrix-free solver for gas dynamics on GPU-accelerated clusters. J. Supercomput. 73, 631–638 (2017). https://doi.org/10.1007/s11227-016-1800-1

52. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271, 108231 (2022). https://doi.org/10.1016/j.cpc.2021.108231

53. Gorobets, A.V.: Adapting a scientific CFD code to industrial applications on hybrid supercomputers. Supercomputing Frontiers and Innovations 9(4), 49–54 (2022). https://doi.org/10.14529/jsfi220405

54. Magomedov, A.R., Gorobets, A.V.: Heterogeneous implementation of preconditioners based on Gauss–Seidel method for sparse block matrices. Computational Mathematics and Modeling 33(4), 438–442 (2022). https://doi.org/10.1007/s10598-023-09585-2

55. Khrapov, S.S., Agafonnikova, E.O., Khoperskov, A.V.: Prospects for improving computational efficiency of hydrodynamic simulations on supercomputers by increasing the number of GPUs per compute node. Supercomputing Frontiers and Innovations 12(2), 43–59 (2025). https://doi.org/10.14529/jsfi250204

DOI: 10.14529/jsfi260203

Multi-Agent Task Flow Dispatching in a Heterogeneous Shared Supercomputer Center

Igor A. Kaliaev[1], *Anatoly I. Kaliaev*[1], *Sergei A. Semenistyi*[1]

The slowdown in performance growth of general-purpose processors is driving the adoption of specialized accelerators such as GPUs and FPGAs in shared supercomputer centers (SSCs). FPGAs are attractive due to hardware-level parallelism and energy efficiency, but their integration significantly increases system heterogeneity: even with similar hardware, differences in bitstreams make nodes functionally nonequivalent. For streams of short jobs, frequent reconfiguration causes substantial overhead, while limited configuration-memory endurance constrains the number of configuration cycles. Classical HPC schedulers usually ignore current FPGA configurations and reconfiguration costs, leading to suboptimal task placement and reduced throughput. This paper presents a multi-agent task dispatcher for a heterogeneous SSC that explicitly accounts for FPGA configuration state and reconfiguration latency. Software agents on each FPGA node make local decisions on task admission and configuration changes, coordinating via bulletin-board queues. The model incorporates computation time, data transfer, and reconfiguration overhead into the scheduling objective. A prototype was implemented on a cluster with up to 18 Tertius2T reconfigurable units and two server nodes. Experiments with queues of up to 3500 tasks show that the multi-agent dispatcher reduces average task completion time and keeps placement and reconfiguration overheads within 40–1200 ms despite individual FPGA configuration times of at least 13 s, demonstrating resilience to heterogeneity and improved resource utilization.

Keywords: high performance computing, hybrid computing systems, multi-agent scheduler, reconfigurable systems, DAG scheduling.

Introduction

High-performance computing clusters (HPCs) are a key tool for solving problems characterized by high computational complexity, the need for accurate numerical modeling, and large volumes of floating-point operations. Traditional HPCs are dominated by architectures based on CPUs and GPUs. However, increasing demands for performance, energy efficiency, and data processing flexibility are leading to the proliferation of scalable heterogeneous architectures incorporating specialized hardware accelerators, including FPGAs [12]. FPGAs enable the implementation of computing cores directly in hardware, providing massive parallelism, low latency, and the ability to adapt the architecture to a specific task. This makes them attractive for high-performance computing, data stream processing, and the implementation of specialized machine learning algorithms. At the same time, the inclusion of FPGAs in large shared supercomputer centers (SSCs) raises new challenges in resource management and dispatching.

A distinctive feature of FPGA operation in HPCs is the presence of a heterogeneous set of configurations used by various users and applications. Switching between configurations requires a configuration procedure, which can be significant and comparable in time to task execution [15]. For a stream of short tasks, frequent reconfigurations lead to significant overhead and reduced utilization of the computing infrastructure. Additionally, the limited resource of configuration cycles when using non-volatile memory must be taken into account.

Classical task management systems in HPCs are typically focused on CPU/GPU nodes and do not take into account the dynamics of FPGA configurations or the resource of the number

[1]Scientific Research Institute of Multiprocessor Computing Systems of the Southern Federal University, Taganrog, Russian Federation

of reconfigurations. This leads to inefficient task distribution, increased latency, and reduced system throughput.

Team management systems in HPCs are traditionally based on centralized or hierarchical schedulers implemented within software packages such as SLURM, PBS, LSF, and others. Such schedulers take into account priorities, resource constraints, and allocation policies, but are primarily designed for homogeneous or slightly heterogeneous CPU/GPU node configurations. Support for FPGA specificity is typically limited to a rough estimate of the presence or absence of the corresponding resource on a node [16].

In recent years, approaches to scheduling in heterogeneous clusters have been actively developing using heuristic and metaheuristic algorithms, as well as machine learning methods. A number of studies consider models that account for the heterogeneity of computing devices and performance differences for different types of tasks [1]. However, the problem of explicitly accounting for FPGA reconfiguration time and resources during stream processing remains less well-developed.

A separate area of research is related to the application of multi-agent technologies to the management of distributed computing systems. Agent-based approaches enable decentralized decision making, increased fault tolerance, and reduced load on the central dispatcher. In the context of heterogeneous FPGA systems, this opens the possibility of locally accounting for the current state of configurations and predicting future reconfigurations at the level of individual nodes. The approach proposed in this paper combines the ideas of multi-agent control and a specialized model of reconfigurable nodes, which allows not only to distribute tasks between nodes, but also to optimize the sequence of FPGA configurations in order to reduce overhead costs.

The article is organized as follows. Section 1 introduces the problem of task dispatching in heterogeneous shared supercomputer centers with FPGA-based nodes and outlines the motivation for adopting a multi-agent approach. Section 2 presents the computing environment model and formulates the task dispatching problem with explicit consideration of computation, communication, and reconfiguration costs. Section 3 describes the proposed multi-agent dispatching system for a cluster based on Tertius-2T reconfigurable computing units, including its architecture, agent functionality, and coordination mechanisms. Section 4 reports the results of experimental studies and evaluates the effectiveness of the proposed approach. Section 5 discusses the results and the limitations of the proposed method. Conclusion summarizes the study and points directions for further work.

1. Review of Approaches to Task Dispatching in Heterogeneous High-Performance Computing Clusters with Reconfigurable Accelerators

1.1. Problem Statement

The task of job dispatching in high-performance computing clusters (HPCs) involves distributing a set of jobs across multiple computing nodes to minimize certain target metrics: make-span, average latency, network infrastructure load, and energy consumption. In the case of heterogeneous clusters incorporating FPGA-based reconfigurable computing units (RCUs), the task becomes significantly more complex: the functional equivalence of nodes is violated

due to configuration differences, and the job execution time includes not only the computational component but also the reconfiguration cost.

1.2. Traditional Job Management Systems in HPCs

Job management in HPCs has historically developed within the framework of centralized batch schedulers. The most common of these are SLURM (Simple Linux Utility for Resource Management), PBS/Torque, LSF, and the Microsoft HPC platform. These systems implement a rich set of queuing, prioritization, resource reservation, and fault-tolerance policies. In particular, SLURM supports backfill scheduling and heterogeneous jobs (hetjobs), which allow for the allocation of different types of resources to individual task components [13].

However, the standard capabilities of such schedulers when applied to FPGA accelerators are significantly limited. In its basic configuration, SLURM does not support the selection of nodes with FPGA resources; this requires the development of specialized extensions. A more fundamental limitation is that classic schedulers abstract from the internal state of accelerators: they do not track the current FPGA configuration, do not consider the cost of configuration switching when assigning tasks, and do not limit the number of configuration cycles based on the resource of non-volatile memory. In conditions where the configuration time of a single FPGA node can be tens of seconds, ignoring this factor leads to significant performance losses when processing queues of short, heterogeneous tasks.

Among domestic developments, the TaskMaster HPC system, developed at the National Research University Higher School of Economics for the cHARISMa supercomputer, is noteworthy [8]. The system monitors task performance, identifies incorrectly launched calculations, and recommends optimizations. According to the developers, the implementation of the system increased the supercomputer's effective performance by 20.5% in the first half of 2023. However, this system is focused on a CPU/GPU cluster and does not cover the specifics of managing reconfigurable resources.

1.3. Task Scheduling Algorithms in Heterogeneous Systems

1.3.1. Classical heuristics

Among the task scheduling algorithms for heterogeneous systems, the HEFT (Heterogeneous Earliest Finish Time) algorithm occupies a special place. This algorithm ranks tasks from bottom to top based on computational and data transfer cost estimates, and then assigns each task to the resource that provides the earliest completion time. HEFT has low computational complexity and produces good results in practice; however, it does not take resource dynamics into account and does not implement load balancing. Subsequently, improvements to HEFT were proposed, including the lookahead-HEFT algorithm, which takes into account the impact of locally made decisions on the descendants of a task in the graph, and experiential HEFT for cloud environments, which takes into account historical data on the effectiveness of assignments [3].

1.3.2. Accounting for FPGA reconfiguration overhead

In the context of systems with reconfigurable accelerators, a key challenge is optimizing configuration switches. Complete FPGA reconfiguration involves loading a new bitstream for the entire circuit, which is associated with a significant time cost. An alternative is dynamic

partial reconfiguration (DPR), which allows changing only selected regions of the die while the rest of the circuit is running. DPR provides significant acceleration of configuration switches and opens up opportunities for multitasking on a single FPGA chip, but requires significant engineering effort during design and is not applicable to all classes of tasks [11].

In the work of Rodriguez-Canal et al. an approach to preemptive task scheduling on FPGAs using DPR is proposed, allowing for the simultaneous execution of multiple cores in different reconfigurable regions and the preemption of lower-priority tasks. The authors show that the worst-case overhead of their approach does not exceed 10%, while the performance gain compared to full reconfiguration is at least 24%. In the broader context of real-time FPGA systems, the FRED architecture is proposed, which uses ticket queues to control access to reconfigurable regions and eliminate task contention.

The problem of integrated optimization of task scheduling, partitioning, and placement on FPGAs with DPR is considered as a joint problem in a number of studies. The proposed methods typically rely on Integer Linear Programming (ILP) or heuristic algorithms and are aimed at minimizing the total execution time, taking into account the reconfiguration overhead. In this paper, a specialized task scheduling algorithm is proposed for reconfigurable computing systems, taking into account both communication dependencies and the cost of switching configurations [2, 5].

It is important to note that most of the listed approaches consider a single FPGA resource or a small group of them, while the problem of scheduling task flows across a large heterogeneous cluster of dozens of FPGA nodes with different configuration states and limited configuration memory resources remains significantly less studied.

1.3.3. Configuration memory lifespan

Non-volatile memory (NVM) used to store FPGA configuration bitstreams has a limited lifespan in terms of write/erase cycles. For flash memory, this figure is typically around 100,000 Program/Erase cycles, while for some NVM types (PCM, RRAM) it ranges from 10^6 to 10^9 cycles. With intensive use of various configurations in a large data center, the memory lifespan can be exhausted significantly faster than the design lifespan without special measures. The problem of wear leveling has been well studied in relation to data memory in file systems and storage systems, but in the context of managing FPGA configuration memory in data centers, specialized solutions are significantly fewer. This is an important argument in favor of scheduling algorithms that specifically reduce the number of reconfigurations [9, 17].

1.4. Multi-Agent Approaches to Computing Resource Management

1.4.1. Conceptual foundations of multi-agent dispatching

Multi-agent systems (MAS) represent a paradigm in which distributed system control is implemented by multiple autonomous software agents interacting with each other and the environment. Each agent has a local state, goals, and behavioral strategy; decision making is decentralized, ensuring scalability and fault tolerance [10]. Applied to HPC, the multi-agent approach eliminates the need for a single central scheduler as a bottleneck and implements dynamic, self-organizing resource management.

Theoretical and methodological foundations of multi-agent scheduling of distributed systems were developed at the Scientific Research Institute of Multiprocessor Computing Systems

of the Southern Federal University (Taganrog), including for GRID computing and heterogeneous distributed systems. The paper [7] formulates formal models of multi-agent dispatching for homogeneous and heterogeneous distributed systems of the first and second types, and develops strategies and algorithms for the behavior of resource agents under conditions of limited inventory, uncertainty, and a competitive environment.

1.4.2. Contract Net Protocol and its extensions

The most widely used mechanism for coordinating agents in task allocation is the Contract Net Protocol (CNP), proposed by Smith in 1980. Within the CNP framework, a manager agent announces a task, contractor agents submit proposals with cost estimates, and the manager selects the winner and concludes the "contract". The protocol allows for a hierarchical subcontracting structure and is similar to a sealed auction mechanism. The use of CNP in MAS enables efficient task allocation without global knowledge of the system state, relying solely on locally available information and consistent messages.

A number of studies aim to improve the efficiency of CNP by modifying bid evaluation strategies, introducing pheromone mechanisms, taking into account the current workload, and varying the number of bidders. Thus, an improved CNP with a limited number of bidders allows for a simultaneous reduction in message traffic and task allocation time while maintaining a high percentage of successfully assigned tasks. A formal analysis of CNP and the conditions for its correct application are discussed in [4, 18].

1.4.3. Blackboard architecture in MAS

A blackboard architecture is an alternative and complementary coordination mechanism in MAS. In this architecture, agents do not exchange messages directly, but read and write data to a shared structure – a blackboard – containing the current state of tasks, resources, and results. This ensures loose coupling between agents, transparency of the system state, and simplifies the addition of new participants. Blackboard architecture is actively used in real-time systems and, more recently, in multi-agent systems [6].

In the context of HPC, the use of a blackboard as a centralized task queue fits well with the decentralized logic of agents: the queue acts as a task "market" in which agents compete or cooperate for assignments while maintaining full control over local resources and configurations.

1.4.4. MAS for HPC: current state

The paper [14] proposes a hierarchical MAS for distributed task scheduling in HPC, combining a global coordinator, cluster brokers, and node-level executor agents. Interaction is based on a hybrid of CNP and auction mechanisms. The authors demonstrate that this scheme provides significantly better resilience to queue instability and resource load fluctuations than traditional centralized schedulers.

A promising approach is also the use of dynamic MAS with adaptive task reassignment when the composition of agents or task parameters change. The DRAMA system offers a unified abstraction of agents and tasks as resource objects with explicit attributes, enabling scheduling based on the affinity of a task to a specific agent – a concept directly applicable to the compatibility of a task with the current FPGA configuration.

1.4.5. Comparative analysis of approaches

Table 1 provides a comparison of the approaches considered based on key characteristics relevant to the dispatching task in a heterogeneous HPC with FPGA.

Table 1. Considered approaches

Approach	Centralization	FPGA Configuration Awareness	Reconfiguration Time Awareness	NVM Resource Awareness	Scalability
SLURM/PBS (basic)	Yes	No	No	No	High
SLURM + FPGA extension	Yes	Partial	No	No	High
HEFT and variations	Centralized offline	No [3]	No	No	Medium
DPR schedulers (FRED, etc.)	Local on-chip	Yes (partial)	Partial [11]	No	Low (one FPGA)
Hierarchical MAS (CNP)	Decentralized	No [14]	No	No	High
Proposed MAS (present work)	Decentralized	Yes	Yes	Yes	High

An analysis of the table reveals that existing approaches either fail to consider the specifics of FPGA nodes at all (traditional schedulers, classic algorithms like HEFT), or only consider configuration at the individual FPGA chip level (DPR schedulers), or fail to explicitly limit the configuration memory resource. The scheduling MAS proposed in this paper is designed to simultaneously consider all of these factors across a cluster of dozens of RDBs.

2. System Models

2.1. Computing Environment Model

HPC can be represented as a set of computing nodes $N = \{n_1, n_2, \ldots, n_m\}$ of various configurations, providing various resources for task execution. The nodes are connected to the communication network via communication channels B_i with varying bandwidths.

The parameters of the computing node n_j include:

- r_j^{CPU} – the number of available processor cores.
- r_j^{RAM} – the amount of available RAM.
- r_j^{SCONF} – static configuration;
- r_j^{DCONF} – dynamic configuration;
- W_j – the node's computing power allocated to all cores.

The static configuration of the compuitation unit (CU) describes parameters that do not change during operation. For example, the presence of a specific GPU, NPU, or FPGA model, connected peripherals, etc. The dynamic configuration of a CU describes the parameters that can be influenced by the corresponding software available on the CU: installed software libraries, loaded data files, FPGA firmware, and settings of specialized hardware used during task execution (e.g., the position of controlled surveillance cameras, power supply mode for autonomous CUs). For each pair of dynamic configurations a and b, the reconfiguration time $T_j^{RECONF}(a, b)$ from state a to state b is known.

For the dispatching system to operate, a number of task-related information is required: the received input data, the task structure (operation graph), the task type, time constraints for its execution, and its metadata – information that allows for task classification when assessing its complexity. Next, we consider the HPC task model.

2.2. Task Model

The vast majority of known works in the field of task scheduling in distributed systems are devoted to optimizing the placement of batch tasks based on their structure, the operations they comprise, and the parameters of the computing system. Given the above considerations, a task for HPC can be described by a tuple $z_i = \langle D_i^{in}, G_i, D_i^{out}, T_i^{MAX}, m_i \rangle$, where D_i^{in} is the input data descriptor, G_i is the directed acyclic graph of the task, D_i^{out} is the output data descriptor, T_i^{MAX} is the required execution time, and m_i is the task metadata (user ID, task ID, etc.).

The task graph $G = (V, E)$ consists of a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ representing the operations that must be performed by HPC to complete the task and a set of edges $e_{i,j} \in E \subseteq V \times V$ describing the dependencies between the operations.

Each edge $e_{i,j} \in E$ represents a constraint that operation v_i is a source of data for operation v_j and must be executed before it. Each edge $e_{i,j}$ is associated with a data transfer volume D_{ij} (in bytes). An example of a task graph is shown in Fig. 1.

For each operation v_k, there is a set of operations $pred(v_k)$ on which v_k depends and a set of operations $succ(v_k)$ that depend on v_k. Operations $v_i \in pred(v_k)$ are connected to v_k by edges $e_{i,k}^{pred} \in E_k^{pred}$. Operations $v_j \in succ(v_k)$ are connected to v_k by edges $e_{k,j}^{succ} \in E_k^{succ}$.

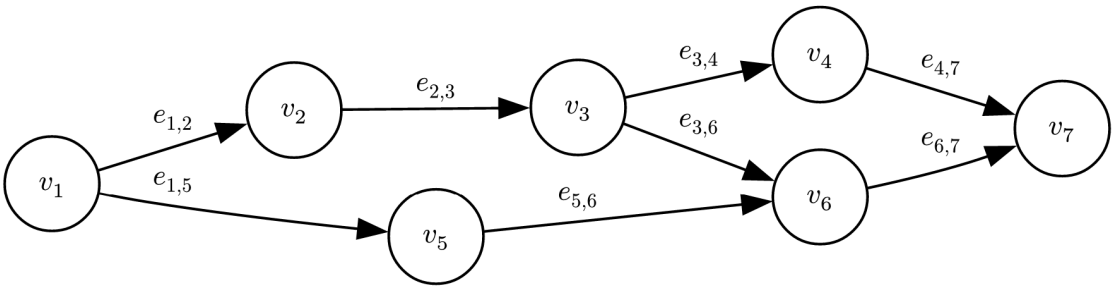


Figure 1. Example of a task graph

Regardless of the composition of HPC, the task execution process can be simplified as the following sequence:

- placing the task in the HPC queue;
- assigning nodes to execute the task by the HPC dispatcher;
- preparing the selected nodes for the task: loading data, deploying and configuring the necessary software components, reconfiguring hardware accelerators (loading FPGA configurations, ANN weights), and
- configuring other hardware;
- launching the appropriate software modules to execute the task;
- transmitting the output data to the recipient (user).

Thus, the actual execution of any computational operations takes only a fraction of the time. The total execution time of an individual task z can be simplified by the formula:

$$\tau(z) = T_{wait}(z) + T_{sched}(z) + T_{conf}(z) + T_{calc}(z), \tag{1}$$

where $T_{wait}(z)$ is the wait time for task z in the queue, $T_{sched}(z)$ is the time it takes to assign the task to the HPC nodes, $T_{conf}(z)$ is the time it takes to configure the corresponding HPC resources, and $T_{calc}(z)$ is the calculation time.

Next, let us consider the factors that influence these variables.

The wait time for a task in the queue, T_{wait}, is determined by how busy the dispatching system is processing other tasks. In fact, this indicator determines the throughput of the dispatching system as a whole.

The time it takes to assign the task to the HPC nodes, T_{sched}, is determined by the performance of the scheduling algorithm used by the dispatching system. Different algorithms have different computational complexity depending on the number of scheduled operations, the connections between them, the number of HPC nodes, and other parameters.

The time it takes to configure the HPC T_{conf} resource is determined by the complexity of bringing the HPC nodes selected during scheduling to a state of readiness to perform the assigned operations. It is worth noting that incoming tasks may require different dynamic configurations of computing nodes, and with a high influx of such tasks, frequent reconfiguration of the computing nodes will lead to unnecessary time expenditures. Therefore, when implementing task dispatching in HPC, it is necessary to consider the compatibility of different types of operations when assigning them to a single node.

Each node n_i can execute a certain subset of operation types $O(n_j) \equiv O^j \subseteq O$ according to its static and dynamic configuration, and each operation corresponds to one of the operation types $O(v_i) \in O$. Moreover, the task graph may contain several operations of the same type ($\bigcup O(v_i) \subseteq O$, $\bigcup O(n_j) = O$).

The parameters of operation v_k include:

- d_k^{CPU} – the required number of processor cores;
- d_k^{RAM} – the required memory size;
- d_k^{SCONF} – the required static configuration;
- d_k^{DCONF} – the required dynamic configuration.

We define a function $\mathcal{A} : V \to N$ that determines the assignment of each operation $v_i \in V$ to node $n_j \in N$. Operation v_k can be executed on node n_j if the compatibility conditions are met:

$$d_k^{CPU} \le r_j^{CPU}, \quad d_k^{RAM} \le r_j^{RAM}, \quad d_k^{SCONF} = r_j^{SCONF}, \quad d_k^{DCONF} = r_j^{DCONF}. \tag{2}$$

To run multiple operations on node n_j simultaneously, the total values of the d_k^{CPU} and d_k^{RAM} parameters must be less than or equal to the corresponding node parameters, and d_k^{SCONF} and d_k^{DCONF} must match for all operations. That is, the compatibility conditions must be met:

$$\begin{cases} \sum_{\mathcal{A}(v_k)=n_j} d_k^{CPU} \le r_j^{CPU}, \\ \sum_{\mathcal{A}(v_k)=n_j} d_k^{RAM} \le r_j^{RAM}, \\ d_k^{SCONF} = r_j^{SCONF}, \forall k : \mathcal{A}(v_k) = n_j, \\ d_k^{DCONF} = r_j^{DCONF}, \forall k : \mathcal{A}(v_k) = n_j, \end{cases}, \tag{3}$$

where $\mathcal{A}(v_k)$ is the node on which operation v_k is located.

2.3. Task Execution Time Estimation

As previously noted, each change of the dynamic configuration of node n_j from state a to state b takes a certain time $T_j^{RECONF}(a,b)$. Thus, the preparation time can be expressed as follows:

$$T_{conf} = \max_{v_i \in V} \left(T_{\mathcal{A}(v_i)}^{RECONF} \left(r_{\mathcal{A}(v_i)}^{DCONF}, d_i^{DCONF} \right) \right). \tag{4}$$

The computation time T_{calc} generally depends on the quality of the placement proposed by the scheduler, that is, on how well the selected HPC nodes match the assigned operations.

Consider a task described by a graph $G(V,E)$ and distributed across a set of nodes $\mathcal{N}\prime \subseteq \mathcal{N}$ belonging to the HPC. Let us estimate the execution time of such a task.

If operations v_i and v_j are executed on different nodes, then the data transfer time between them is:

$$T_{transfer}(v_i, v_j) = \frac{D_{ij}}{\min\left(B\left(\mathcal{A}(v_i)\right), B(\mathcal{A}(v_j))\right)} \quad , \tag{5}$$

where D_{ij} is the volume of data transferred between operations v_i and v_j, $B(n)$ is the throughput of the channel connecting node n to the communication medium.

Let $start(v_i)$ and $finish(v_i)$ denote the start and end moments of the execution of operation v_i, and $T(v_i, n_j)$ be the execution time of operation v_i on node n_j. Then:

$$finish(v_i) = start(v_i) + T(v_i, \mathcal{A}(v_i)). \tag{6}$$

For operation v_i depending on the completion of operation v_p:

$$start(v_i) = \max_{v_p \in pred(v_i)} (finish(v_p) + T_{transfer}(v_i, v_p)\,[\mathcal{A}(v_i) \neq \mathcal{A}(v_p)]), \tag{7}$$

where $[\mathcal{A}(v_i) \neq \mathcal{A}(v_p)]$ is a condition indicating that the operations are performed on different nodes and that data transfer time must be taken into account.

Thus, the total task completion time is calculated recursively using the formula:

$$T_{calc} = finish(v_{exit}) = T(v_{exit}, \mathcal{A}(v_{exit})) + \max_{v_p \in pred(v_{exit})} (finish(v_p) + T_{transfer}(v_{exit}, v_p)\,[\mathcal{A}(v_{exit}) \neq \mathcal{A}(v_p)]),\ v_{exit} : succ(v_{exit}) = \emptyset. \tag{8}$$

3. Resource Dispatching for a High-Performance Computing Cluster Based on the Tertius-2 RCU

When using traditional job scheduling methods in high-performance computing clusters (HPCs) that include reconfigurable computing units, the execution time of the corresponding operations $T(v_i, n_j)$ includes not only the actual computation time T_{calc} (v_i, n_j) but also the resource reconfiguration time $T_{n_j}^{RECONF}\left(r_{n_j}^{DCONF},\ d_i^{DCONF}\right)$. This does not have a significant impact when performing long calculations based on a single reconfiguration (when $T_{calc} \gg T^{RECONF}$), but when intensively processing jobs with short operations, when the computation time is comparable to the reconfiguration time, it will lead to a significant loss of time and delays in job processing.

To overcome the problem described above, a multi-agent dispatching system monitors the state of reconfigurable resources and reduces reconfiguration latency by rationally distributing task operations across HPC nodes, taking into account their current and planned configurations.

As part of the research on "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence Methods", experimental studies of a multi-agent cluster dispatching system using the Tertius-2T RCU manufactured by "SRC SC & NC" Co. Ltd were conducted at the Polytechnic Scientific and Technical Center.

To experimentally study the proposed method for coordinating data flow processing task assignments in HPC, a prototype software system implementing the corresponding algorithms was developed.

The system structure is shown in Fig. 2.

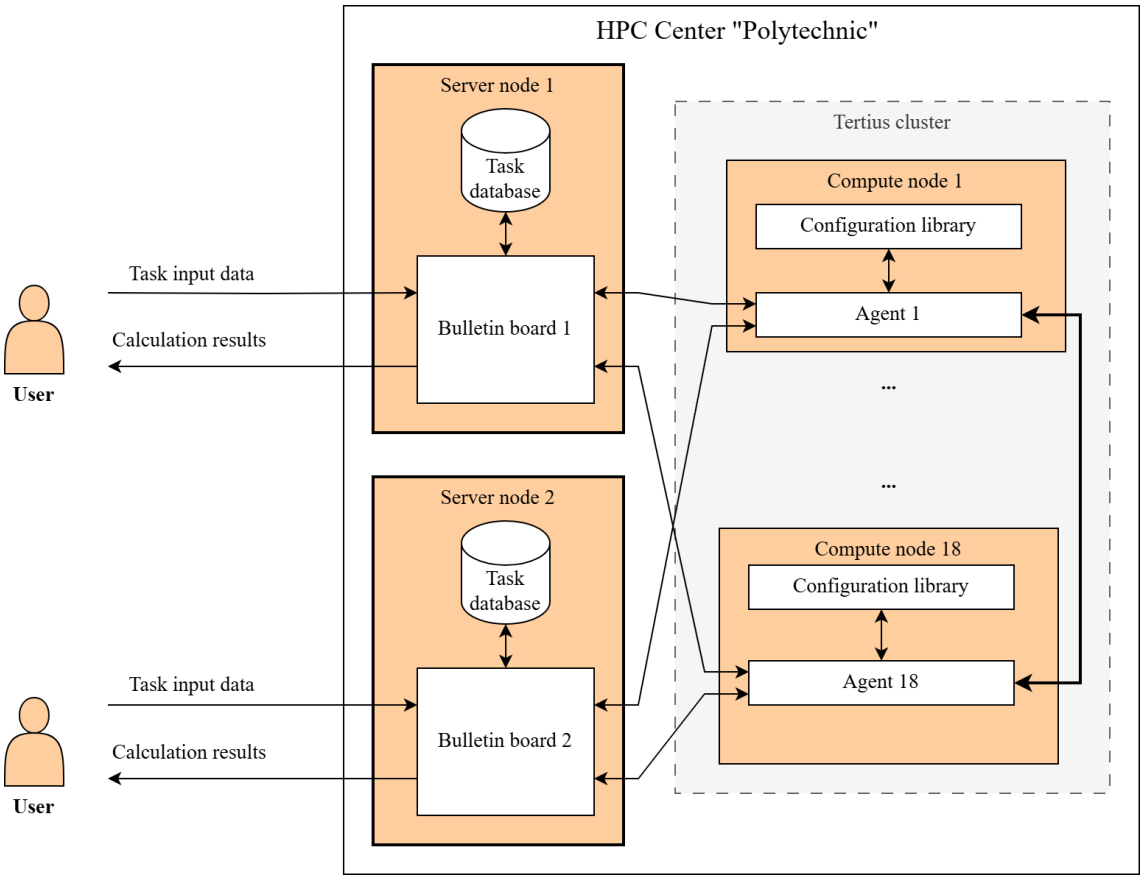


Figure 2. General system structure

The system includes:

- 18 Tertius-2T reconfigurable computation units (RCUs);
- 2 server nodes;
- 18 agents deployed on RCUs;
- 2 bulletin boards serving as job queues on the server nodes.

The software (SW) of the Dispatching MAS agent is designed to organize the execution of user jobs on RCUs within the supercomputer center (SCC).

The agent implements the following functions:

- monitoring available jobs on RCUs;

- receiving job information for RCUs from the bulletin board (BB);
- receiving job input data for RCUs from BB;
- receiving configuration module files for RCU required to execute tasks from the high-performance reconfigurable computing cluster (HPRCC) configuration module library;
- configuring RCU to execute user tasks according to the type of each task by selecting the required configuration module of HPRCC;
- initiating the process of executing user tasks on RCU;
- transmitting the current status of tasks to RCU during their execution;
- receiving the results of user tasks and transmitting these results to RCU.

RCU implements the following functions:

- storing task configuration data and associated application files and data;
- receiving task information for RCU from the user, as well as the task data itself;
- storing task information for RCU, the task data itself, and the task calculation results before transferring them to the user.
- transferring output data (execution results) of tasks to RCU to the user.

The structure of BB is shown in Fig. 3.

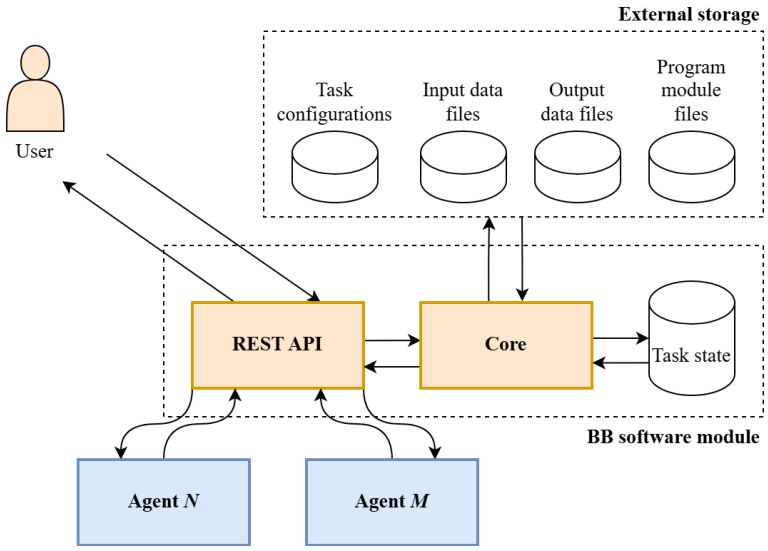


Figure 3. Bulletin board structure

The agent structure is shown in Fig. 4.

The MAS software was developed in C++ using the Qt 5.15 library and is designed to run on GNU/Linux OS.

4. Results of Experimental Studies of the Dispatch MAS

To evaluate the effectiveness of the proposed dispatch MAS, experimental studies were conducted on a cluster consisting of up to 18 Tertius-2T RCUs and two server nodes. The experiments simulated the processing of task queues ranging from 200 to 3,500 tasks with varying numbers of nodes and in two modes:

- without reconfiguration optimization (basic mode);
- with reconfiguration optimization and partial agent consent.

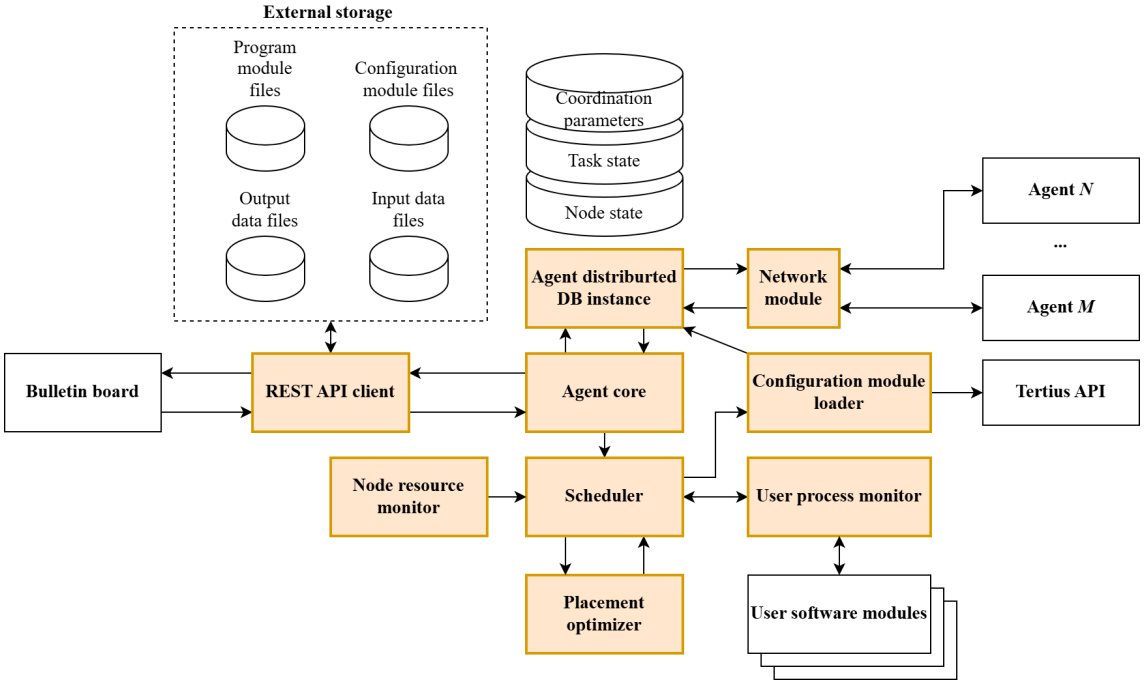


Figure 4. Agent structure

The tasks represented a set of typical operations reflecting target scenarios for using FPGAs in a HPC. For each task, the solution time was measured, including possible FPGA reconfiguration, and the net computation time with a suitable configuration. For each series of experiments, the following were recorded:

- queue size and total number of tasks;
- number of nodes involved;
- total solution time for all tasks in the queue;
- average solution time for one task;
- minimum, maximum, and average net computation time.

The experimental results are presented in Tab. 2.

Table 2. Experimental results

Exp. No	Total number of tasks	Number of nodes	Time to solve all tasks in the queue, sec	Average time to solve 1 task, sec	Net solution time, sec		
					min	max	avg
without reconfiguration optimization							
1	200	17	105	9.0	0.253	3.932	1.602
2	5000	17	3294	11.2	0.247	3.951	1.597
3	3500	6	8458	14.5	0.269	7.851	1.533
4	3500	17	2841	13.8	0.249	3.949	1.567
with reconfiguration optimization and partial agent consent							
5	1000	9	700	6.300	0.251	3.922	1.582
6	1000	10	413	4.130	0.248	3.930	1.613
7	1000	16	762	12.192	0.246	4.019	1.591
8	3500	6	5263	9.022	0.268	7.848	1.531

In the mode without reconfiguration optimization, a significant increase in the average task execution time is observed with increasing queue size and/or decreasing the number of nodes. This is due to the fact that tasks of different types are assigned to nodes without regard for current and future configurations, which leads to frequent FPGA reconfigurations and high overhead.

Enabling the partial agreement algorithm and reconfiguration optimization results in a significant reduction in the average task execution time. For queues of up to 3,500 tasks, the reduction can be multiple compared to the basic mode, while the net computational time remains comparable. This indicates that the gain is achieved precisely due to the reduction in reconfiguration time and more efficient use of already configured nodes.

The dynamics of task execution time during queue processing is illustrated in Fig. 5. In the optimization mode, a more even distribution of solution time is observed, the absence of pronounced peaks associated with large-scale reconfigurations, and faster queue burnout.

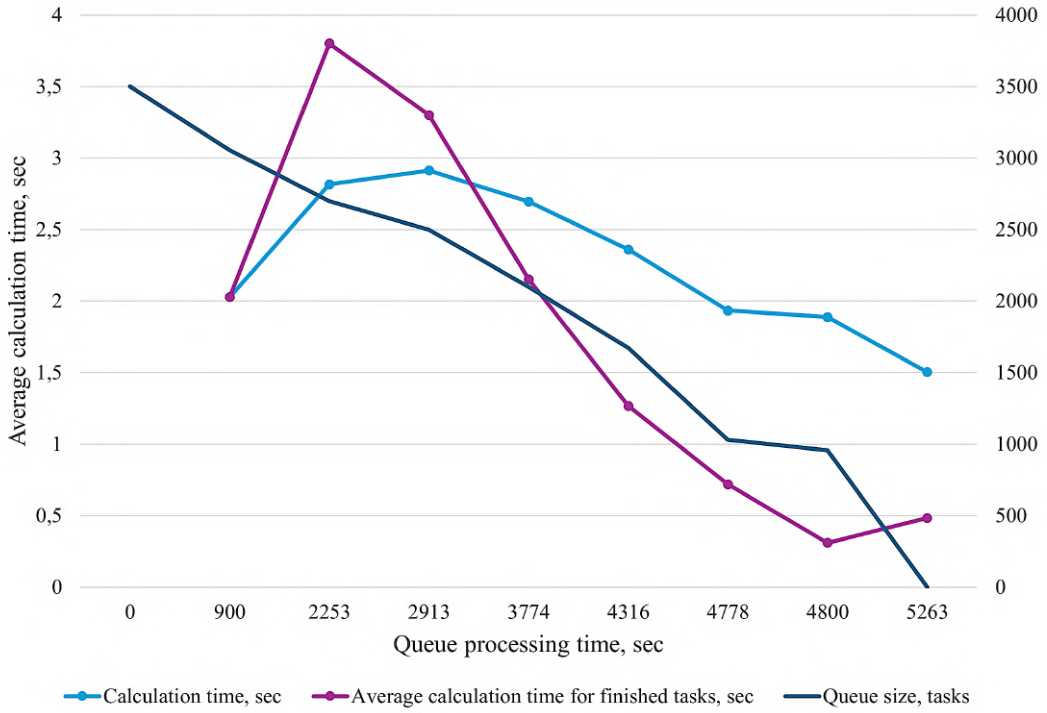


Figure 5. Dynamics of task resolution time changes during queue processing

The study demonstrated a significant reduction in average task resolution time, CPU load, and network infrastructure utilization when using the partial agreement algorithm for dispatching task queues up to 3,500 in size.

5. Discussion of Results and Limitations

The experimental results demonstrate that accounting for FPGA reconfiguration time and the use of a multi-agent approach can significantly improve the efficiency of task flow processing in a heterogeneous cluster. The greatest gains are achieved in modes where:

- tasks are relatively short compared to configuration time;
- the task flow contains several common types that regularly recur in the queue;
- the number of available nodes is sufficient to separate responsibilities by task type.

However, this approach has several limitations. It is necessary to support agent deployment on each node, which increases software complexity and places demands on its reliability. The decentralized nature of the algorithm generates communication overhead between agents and bulletin boards, although experiments show that this overhead is significantly lower than the time savings from reconfigurations.

Another limitation is the dependence of efficiency on the accuracy of computational and configuration time estimates. Without adequate models or statistics, agents may make suboptimal decisions. Furthermore, the current implementation does not take into account such factors as power consumption, FPGA resource degradation, and possible task priorities, which represents a promising area for further research.

Conclusion

This paper examines the problem of task flow dispatching in a heterogeneous shared supercomputer center incorporating reconfigurable FPGA-based computing units. It is shown that reconfiguration overhead can account for a significant portion of the time spent processing short tasks, and that the resource allocated to the number of configuration cycles is an important limiting factor.

A multi-agent dispatching system is proposed in which software agents running on each RCU make decisions on task assignment and FPGA reconfiguration based on local information and partial consensus mechanisms. A system model that takes into account computational and reconfiguration time is described, and optimization criteria are formulated.

Experimental studies on a cluster using the Tertius-2T RCU showed that the proposed methods can significantly reduce the average task resolution time and reconfiguration overhead for queue sizes of up to 3,500 tasks. Simultaneously, the load on the RCU CPU and network infrastructure is reduced, confirming the effectiveness of the multi-agent approach.

Future development plans include enhancing the system to integrate various types of computing devices into a single computing supercluster. This will enable end-to-end task dispatching across CPUs, GPUs, and FPGAs with a unified set of quality-of-service criteria, taking into account their architectural features. This will also enable adaptive load redistribution between heterogeneous nodes based on online performance and workload assessments for individual computing types. Additional constraints related to energy efficiency and the service life of the FPGA's non-volatile configuration memory will be incorporated into the model, allowing for schedule optimization based on these factors.

A separate area of development is the transfer and adaptation of the proposed multi-agent system for industrial use at existing SCCs with heterogeneous configurations, as well as expanding the range of supported applications, including complex pipelined and graph computing workloads.

References

1. Adimora, K., Gundla, S.R., Sun, H.: Machine learning approaches for optimizing high-performance computing scheduling: a comprehensive survey and analysis. Cluster Computing 28(13), 831 (2025). https://doi.org/10.1007/s10586-025-05521-8

2. Chen, S., Huang, J., Xu, X., *et al.*: Integrated optimization of partitioning, scheduling, and floorplanning for partially dynamically reconfigurable systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39(1), 199–212 (2018). https://doi.org/10.1109/tcad.2018.2883982

3. Houssein, E.H., Gad, A.G., Wazery, Y.M., Suganthan, P.N.: Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation 62, 100841 (2021). https://doi.org/10.1016/j.swevo.2021.100841

4. Hsieh, F.S.: Analysis of contract net in multi-agent systems. Automatica 42(5), 733–740 (2006). https://doi.org/10.1016/j.automatica.2005.12.002

5. Huang, M., Narayana, V.K., Simmler, H., *et al.*: Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 3(4), 1–25 (2010). https://doi.org/10.1145/1862648.1862650

6. Jiang, Y., Yi, P., Zhang, S., Zhong, Y.: Constructing agents blackboard communication architecture based on graph theory. Computer Standards & Interfaces 27(3), 285–301 (2005). https://doi.org/10.1016/j.csi.2004.09.003

7. Kaliaev, A.: Multiagent approach for building distributed adaptive computing system. Procedia Computer Science 18, 2193–2202 (2013). https://doi.org/10.1016/j.procs.2013.05.390

8. Kostenetskiy, P., Shamsutdinov, A., Chulkevich, R., *et al.*: HPC TaskMaster – Task Efficiency Monitoring System for the Supercomputer Center. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. pp. 17–29. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-11623-0_2

9. Macronix International Co., L.: Wear Leveling in NAND Flash Memory. https://www.mxic.com.tw/Lists/ApplicationNote/Attachments/1913/AN0289V1-Wear%20Leveling%20in%20NAND%20Flash%20Memory.pdf (2014), accessed: 2016-04-08

10. Parasumanna Gokulan, B., Srinivasan, D.: An Introduction to Multi-Agent Systems, vol. 310, pp. 1–27 (07 2010). https://doi.org/10.1007/978-3-642-14435-6_1

11. Rodriguez-Canal, G., Brown, N., Torres, Y., Gonzalez-Escribano, A.: Task-based preemptive scheduling on FPGAs leveraging partial reconfiguration. Concurrency and Computation: Practice and Experience 35(25), e7867 (2023). https://doi.org/10.1002/cpe.7867

12. Samayoa, W.F., Crespo, M.L., Cicuttin, A., Carrato, S.: A survey on FPGA-based heterogeneous clusters architectures. IEEE Access 11, 67679–67706 (2023). https://doi.org/10.1109/ACCESS.2023.3288431

13. Shehata, A., Groszkowski, P., Naughton, T., *et al.*: Bridging paradigms: Designing for HPC-Quantum convergence. Future Generation Computer Systems 174, 107980 (2026). https://doi.org/10.1016/j.future.2025.107980

14. Thiyagaraj, B.: Multi-Agent Systems for Distributed Task Scheduling in High-Performance Computing. American International Journal of Computer Science and Technology 3(6), 11–22 (2021). https://doi.org/10.63282/3117-5481/AIJCST-V3I6P102

15. Vipin, K., Fahmy, S.A.: FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. ACM Comput. Surv. 51(4) (Jul 2018). https://doi.org/10.1145/3193827

16. Wang, B., Chen, Z., Xiao, N.: A Survey of System Scheduling for HPC and Big Data. In: Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications. pp. 178–183. HP3C 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3407947.3407977

17. Zhang, J., Wang, C., Zhu, Z., *et al.*: Realizing extreme endurance through fault-aware wear leveling and improved tolerance. In: 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 964–976. IEEE (2023). https://doi.org/10.1109/HPCA56546.2023.10071093

18. Zhang, J., Wang, G., Song, Y.: Task assignment of the improved contract net protocol under a multi-agent system. Algorithms 12(4), 70 (2019). https://doi.org/10.3390/a12040070

DOI: 10.14529/jsfi260204

Rapid Training of Neural Networks Using the Arctur Reconfigurable Computer System

Ilya I. Levin[1], *Dmitriy A. Sorokin*[1], *Vasiliy B. Kovalenko*[1]

Linear scaling of hardware resources in widely used systems based on graphics processing units (GPUs) and central processing units (CPUs) for neural network training problems provides only logarithmic scaling of real performance. This is due to the presence of dataflow discontinuities in the computational graphs of neural network training problems, which lead to a significant decrease in computational intensity and may even lead to a complete stop between execution stages. These limitations are unacceptable if neural network training must be performed within a limited time regardless of the amount of processed data. Reconfigurable computer systems (RCS) based on field-programmable gate arrays (FPGAs), unlike traditional systems, demonstrate the potential to address these limitations through the structural organization of calculations, as well as by adapting to the specifics of solving the problem. Methods for minimizing the impact of dataflow discontinuities have been formulated and theoretically justified for RCS. Based on these methods, a methodology has been developed to construct efficient computing structures to solve neural network training problems, ensuring real RCS performance that exceeds 70% of peak performance. For GPU-based systems, achieving this level of computational efficiency is generally impossible. For the first time, neural network-based data processing problems have been implemented using a parallel-pipeline approach on the Arctur RCS, which contains 96 XCVU37P FPGAs. Experimental results have shown that for these problems, the energy efficiency of a single Arctur RCS block is 56% higher than the energy efficiency of 16 NVIDIA DGX A100 blocks. Increasing the number of Arctur RCS nodes enables near-linear scaling of real performance. For example, the RCS rack consisting of 16 Arctur nodes is capable of providing the real performance of about 7 PFLOPS. For neural network problems such as ResNet-50v1.5 training, the performance of this rack is comparable to that of 12 NVIDIA DGX A100 racks with 2.2× lower power consumption.

Keywords: reconfigurable computer systems, FPGA, neural network training, dataflow discontinuities.

Introduction

The progressive development of national economies is impossible without the widespread application of artificial intelligence technologies for analysis of visual,audio and textual information. Neural network technologies are increasingly used to solve such weakly formalized problems, characterized by high computational complexity [1]. Neural networks are becoming a universal computational tool, and further expansion of their application domains is expected in the coming years.

The practical value of neural network technologies is determined not only by the achieved accuracy in solving the problem but also by the ability to adapt them to changing data environments. However, the increasing diversity of analyzed data is not only accompanied by an increase in the amount of processed data and computational complexity, but also by the degradation of previously trained models, thereby reducing their effectiveness. It is necessary to fine-tune or retrain them.

In real-world applications, when new objects of interest appear, retraining of modern neural networks under conventional conditions can take from several days to several weeks, and in some cases months. However, in rapidly changing environments, it is often necessary to allow for fast

[1] "Scientific Research Center of Supercomputers and Neurocomputers" Co Ltd ("SRC SC & NC" Co Ltd), Taganrog, Russian Federation

training (within several hours), regardless of the amount of processed data. In these conditions, rapid adaptation of neural network systems becomes a key requirement for maintaining the quality of information analysis.

To solve the problem promptly, it is required to linearly increase the computing power of training systems with corresponding scaling of hardware resources. However, most existing systems used for high-accuracy neural network training are based on tensor processing systems: NVIDIA GPUs [2], as well as similar TPU systems by Google [3] and Ascend systems by Huawei [4]. The architecture of tensor processors is adapted for matrix-vector multiplication and data-parallel execution. While they provide efficient pipelined execution within individual neural network layers, inter-layer execution remains constrained by strong data dependencies inherent in the computational graph of training problem.

As a result, execution across layers is segmented into synchronization-bound stages defined by the runtime execution model. Inter-layer data exchange is handled through host-managed runtime scheduling and synchronization in CPU-accelerator coordinated execution systems, introducing additional overhead that reduces continuity of pipeline execution across the full computational graph. These effects directly introduce dataflow discontinuities and reduce computational intensity, as execution phases become bounded by synchronization and resource idling occurs between dependent stages.

Another limitation arises from normalization operations, including post-normalization [5] and pre-normalization [6] of activations, which are required to stabilize training dynamics. These methods improve numerical stability and reduce the risk of gradient fading or explosion. However, commonly used normalization schemes (batch, layer, etc.) require computation of statistical properties over the entire set of normalized elements. Before these statistics are calculated, full buffering of intermediate activations is required, which introduces dataflow discontinuities. These combined effects, under synchronization-dominated execution regimes, result in a logarithmic performance scaling trend for representative neural network training workloads. This behavior is a direct consequence of the inability to sustain continuous pipeline execution across the full computational graph under inter-layer synchronization constraints. Therefore, achieving fast neural network training independent of data volume requires a corresponding exponential increase in hardware resources, resulting in increased system cost and energy consumption.

In contrast, the RCS based on FPGAs demonstrate the potential for near-linear performance scaling in time-consuming, strongly coupled problems across domains such as mathematical physics, geophysics, and evolutionary modeling [7–9]. Several attempts have been made to apply FPGA systems to solve machine-learning problems. However, existing approaches typically use FPGAs as coprocessors within GPU-like execution models [10]. More efficient problem solving on the RCS requires the use of a structural organization of calculations [11]. In this paradigm, the input data, target output data, and the chosen solution method define a computational graph that is mapped to the RCS architecture without a semantic gap. In CPU- and GPU-based systems, this mapping is realized through control-flow-driven or data-parallel execution models with runtime-managed scheduling and synchronization. In RCS, the computational graph vertices corresponding to computational operations and edges representing data dependencies are implemented directly in hardware, enabling pipeline-oriented execution across multiple stages of the graph. This approach enables parallel-pipeline processing of multiple stages of the computational graph and mitigates dataflow discontinuities, defined as breaks in data propagation between dependent operations that lead to reduced computational intensity

and underutilization of computing resources. Calculations are parallelized across both data and iterations using linear or nested pipeline structures, thereby also reducing memory overhead.

The idea of applying structural organization of calculations to neural network training is not new. In [12], Academician A.V. Kalyaev proposed a structural implementation of deep neural network training using static neuroprocessors and inter-layer hardware switching networks. The proposed approach explicitly considers simultaneous execution of forward pass, backpropagation, and weight update operations within a unified hardware structure. However, such a tight coupling of training phases introduces fundamental execution constraints in modern backpropagation-based training systems. Weight update operations depend on globally accumulated gradient information and therefore require completion of preceding computational stages. As a result, update phases introduce synchronization points that interrupt continuous execution flow. Given that the number of weight parameters in modern neural networks is comparable to the amount of intermediate data processed between updates, these synchronization points lead to significant dataflow discontinuities and reduced utilization of computing resources.

It should be noted that the earlier RCS (from the early 2000s until recently) were unable to efficiently solve neural network training problems. Structural implementation of real neural network problems on RCS, especially learning problems, requires integration of multiple FPGAs into a unified computing field, with inter-device communication implemented via high-speed hardware interconnects. Given the data dependency patterns in neural network workloads, such interconnects can currently be implemented only using multi-gigabit transmission technologies [13]. In addition, structural training implementations require multiple independent memory access channels per FPGA, which can be addressed using stacked multi-channel DRAM technologies [14].

The architectural design of the Arctur RCS, based on Xilinx XCVU37P FPGAs, satisfies these requirements. The system provides the high computational density (7.11 TFLOPS/L), a high-bandwidth inter-FPGA interconnect subsystem based on MGT transceivers with aggregate throughput up to 200 Tbps, and distributed HBM memory with a total capacity of 768 GB. This paper covers the methods for minimizing dataflow discontinuities in structural implementation of neural network training problems on the Arctur RCS and evaluates their performance and energy efficiency in comparison with NVIDIA tensor processing systems. The remainder of the paper is organized as follows. Section I describes the Arctur RCS used in the experiments. Section II analyzes dataflow discontinuities in neural network training and proposes mitigation methods. Section III presents hardware implementations of computing structures for neural network training, including a representative convolutional layer in training mode. Section IV presents performance and energy efficiency results for the ResNet-50v1.5 training problem, compared to the NVIDIA DGX A100 system. The conclusion summarizes the main results of the research, highlights the most significant findings, and discusses their practical applicability and potential for future use.

1. Arctur Reconfigurable Computer System

The Arctur computer system implements a set of architectural solutions that realize the proposed computing paradigm. The computing and communication subsystems of the Arctur RCS enable the implementation of large-scale computational graphs within a unified computing field. The system provides the required power delivery and cooling characteristics [15]. The Arctur RCS block is shown in Fig. 1.

Figure 1. The Arctur RCS block

The Arctur RCS node is implemented in a 3U 19" form factor and contains 16 vertically mounted computing modules interconnected via a backplane. Each module integrates six XCVU37P FPGAs from the Xilinx UltraScale+ family. A photograph of the computing module is shown in Fig. 2.

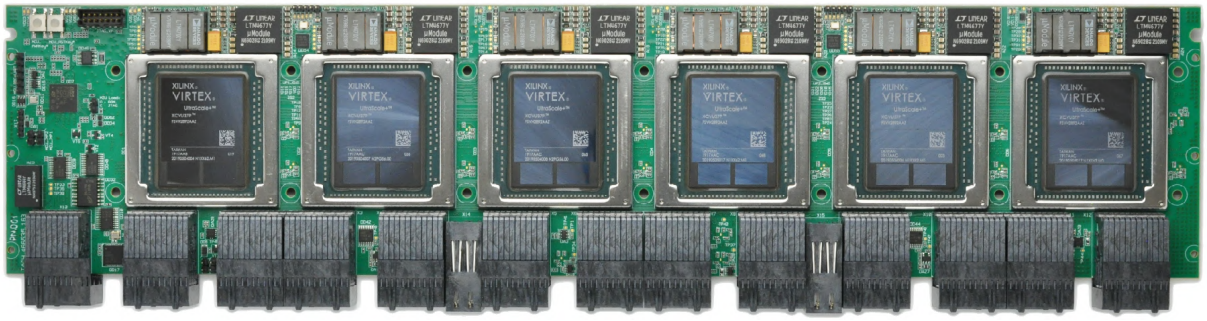


Figure 2. The Arctur RCS computing module

Therefore, a single Arctur RCS block provides high-density integration of computing resources, comprising 96 high-capacity FPGAs with a total of more than 270 million logic cells. The cooling of all components is implemented using immersion technology based on a dielectric fluid with high electrical insulation strength, thermal conductivity, and heat capacity at low viscosity.

The Arctur RCS features an extensive communication subsystem. FPGAs are interconnected via both horizontal and vertical communication links. Horizontal interconnects within a computing module are implemented using 24 MGT transceivers with data rates up to 24 Gbps per link, providing a full-duplex bandwidth of up to 576 Gbps between adjacent FPGAs. In addition, a ring interconnect between the first and last FPGA in the module is implemented using 16 MGT transceivers with a bandwidth of up to 384 Gbps. An auxiliary full-duplex communication channel based on 24 LVDS transceivers provides additional bandwidth of up to 28.8 Gbps. The structure of horizontal interconnects is shown in Fig. 3.

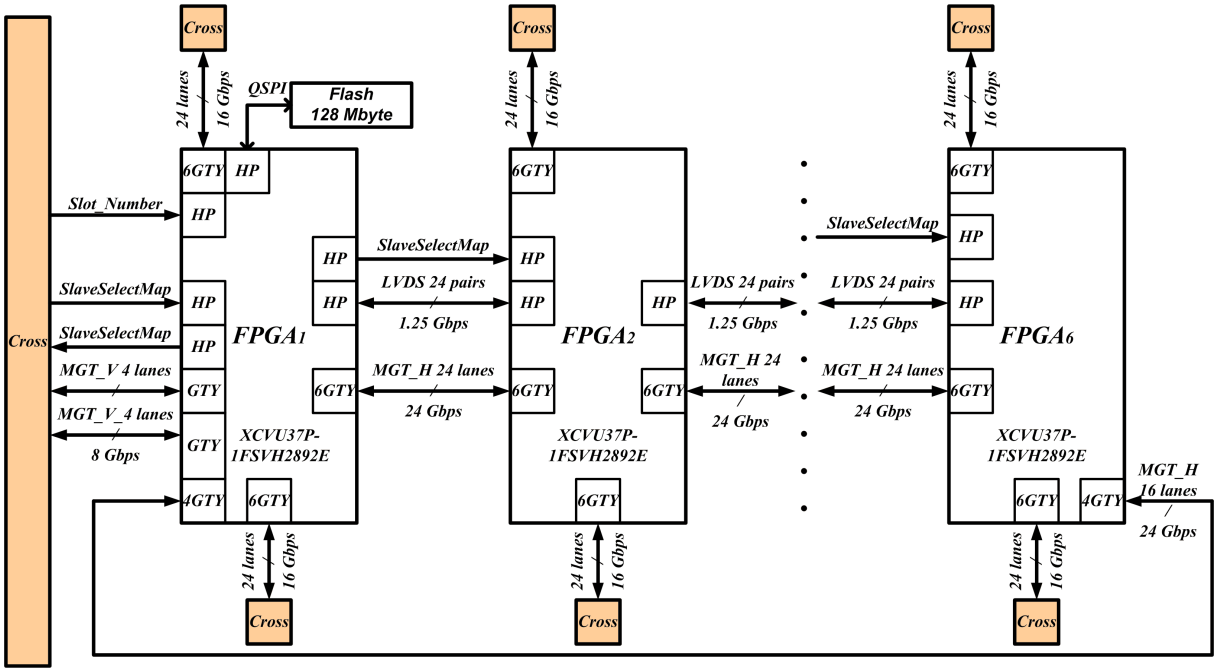


Figure 3. Horizontal interconnect structure of the Arctur RCS computing module

Vertical interconnects between computing modules are implemented using 144 MGT transceivers with data rates up to 16 Gbps per link, providing a full-duplex bandwidth of up to 2.3 Tbps between modules. In addition, dedicated service channels are provided for FPGA configuration, command transmission, and data exchange between the host processor and any computing module, with a bandwidth of 32 Gbps. The Arctur RCS block can also be interconnected with other blocks via optical transceivers, forming full-duplex links with bandwidth up to 1.8 Tbps. The structure of vertical interconnects is shown in Fig. 4.

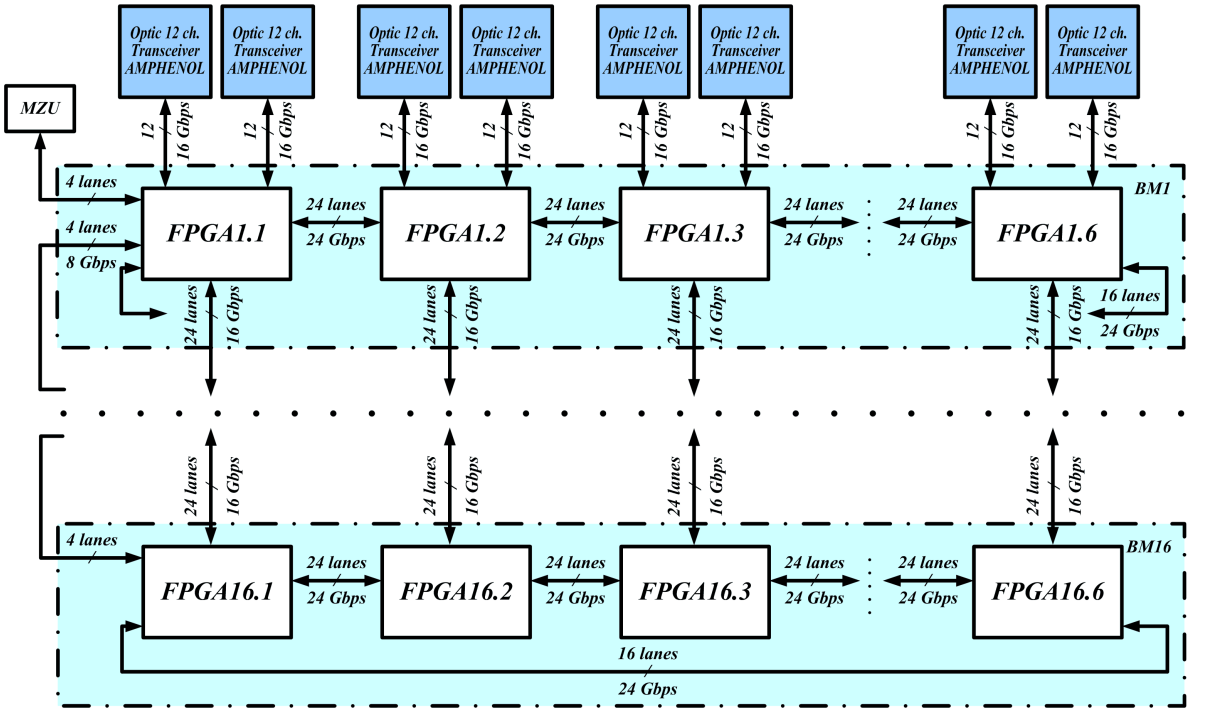


Figure 4. Vertical interconnect structure of the Arctur RCS block

The main characteristics of the Arctur RCS block are presented in Tab. 1.

Table 1. Technical characteristics of the Arctur RCS

Parameter	Value
Number of computing modules	16
Number of XCVU37P FPGAs	96
Peak performance, TFLOPS	610
HBM memory capacity, GB	768
Optical transceivers (12-channel, 12 Gbps)	12
Interfaces	2×1Gb Eth, USB, DisplayPort
Constant power supply voltage, V	380 ± 20
Dimensions, mm	$485 \times 131.5 \times 1344$
Maximum power consumption, kW	25
Limit power consumption (floating-point workloads), kW	16

The theoretical energy efficiency of the Arctur RCS block, defined as the ratio of peak performance to maximum power consumption, is 38 GFLOPS/W. For comparison, a single NVIDIA DGX A100 provides a peak performance of 2496 TFLOPS with the power consumption of 6.5 kW [16], corresponding to a theoretical energy efficiency of 384 GFLOPS/W. Therefore, under peak theoretical estimates, the Arctur RCS block demonstrates lower energy efficiency compared to the DGX A100 block. However, experimental research shows that the computational efficiency of a single Arctur RCS block is approximately five times higher than that of the DGX A100 block for real neural network training problems. Such loss of computational efficiency in tensor systems is caused by low specific performance in solving practical problems. The primary reasons for this include high inter-layer data dependencies and the presence of dataflow discontinuities.

Next, we consider the types of dataflow discontinuities in detail, as well as the methods for their mitigation, enabling significantly higher real performance on the Arctur RCS block compared to DGX-based systems.

2. Methods for Minimizing Dataflow Discontinuities

The efficiency of solving problems both on RCS and tensor (and any other) data flow processing systems is significantly reduced if there are such g-th and $(g+1)$-th fragments of the information graph between which the intensity of data flows Ig of processed data [17] drops to zero. Such a situation is defined as a dataflow discontinuity. In RCS, dataflow discontinuities also affect the entire computational graph, similarly to tensor processing systems, since the structure of the main data dependencies in implementations of problems of neural network training remains unchanged. Therefore, linear scaling of hardware resources does not lead to a proportional increase in achieved performance. These discontinuities must be eliminated or at least minimized. Suppose that for solving a problem F, containing subtasks f_{g-1},f_g, and f_{g+1}, it is necessary to organize pipeline processing of the input vector X_i.

Without loss of generality, we assume that $X_i = \{x_1, x_2, \ldots, x_J\}$ with intensity I_1 is fed to subtask f_{g-1}, and at its output a vector $X'_i = \{x'_1, x'_2, \ldots, x'_J\}$ is formed with intensity I_1 and is passed to f_g and f_{g+1}. At the output of f_g, a variable m_i is formed with intensity I_2 and

is also passed to subtask f_{g+1}, so that the output vector $Y_i = \{y_1, y_2, \ldots, y_J\}$ is formed with intensity I_3. A simplified computing structure of problem F is shown in Fig. 5.

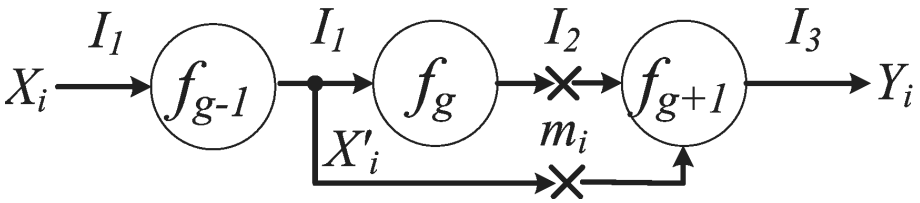


Figure 5. Simplified structure of the problem F

Within the structural organization of calculation paradigm, the computational graph of any problem F is directly mapped onto the RCS hardware resources. In this context, representations of the problem F, its subproblem f_g, or even an individual function, as well as their corresponding computing structures implemented at the system level in the RCS, are considered as mutually consistent levels of abstraction.

Let subtask f_g implement a function of the form $m_i = \frac{1}{J}\sum_{j=1}^{J} x'_j$ and subtask f_{g+1} implement a function of the form $Y_i = \frac{X_i}{m_i}$. Obviously, until all J elements of vector X'_i are processed in f_g, the result m_i cannot be formed; therefore, at this time $I_2 = I_3 = 0$. A dataflow discontinuity arises, and subtask f_{g+1} cannot process vector X'_i during formation of m_i.

The processing time T_{proc} of a set of vectors X_i, for $i = 1, ..., N$, in problem F can be approximated by the following empirical model capturing the effect of dataflow discontinuities:

$$T_{proc} = \sum_{i=1}^{N} \left[(1 + \frac{I_{in} - I_{out}}{I_{in}}) \cdot t(X_i) \right], \tag{1}$$

where $t(X_i)$ is the time for reading X_i,

I_{in} is the intensity of input data entering the pipeline,

I_{out} is the intensity of output data formation from the pipeline.

Since subtask f_{g+1} starts execution only after completion of f_g, and both process vectors of equal dimensions $\|X'\| = \|X_i\| = J$, then for problem F at the moment of reading X_i, the output intensity is equal to $I_{out} = I_3 = 0$. Then, the expression (1) transforms into:

$$T_{proc}(I_3) = 2 \cdot \sum_{i=1}^{N} t(X_i). \tag{2}$$

Therefore, in the presence of a dataflow discontinuity, the processing time of problem F is doubled. Moreover, during calculation of X'_i and m_i, the hardware resources assigned to f_{g+1} are idle, and during calculation of Y_i, the resources assigned to f_{g-1} and f_g are idle. To overcome the dataflow discontinuity between the g-th and $(g+1)$-th fragments of the information graph of the problem F, the transformation shown in Fig. 6 can be performed.

If there exists a function f_g^*, such that the domain of its acceptable values $D(f_g^*)$ is close to the domain $D(f_g)$, then for $Y_i^* = f_{g+1}(f_g^*(f_{g-1}(X_i)))$ and $Y_i = f_{g+1}(f_g(f_{g-1}(X_i)))$ there exists $D(f_g^*) \cap D(f_g)$, where $Y_i^* \approx Y_i$. If $f_{g+1}(f_g^*)$ produces output vector Y_i^* with intensity $I_3^* \approx I_2^* > 0$, the dataflow discontinuity is eliminated. Such cases are referred to as first-type

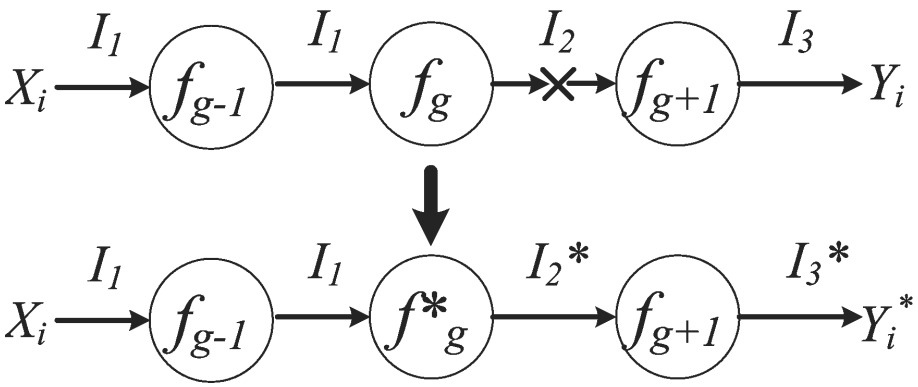


Figure 6. Overcoming dataflow discontinuity in the problem F

dataflow discontinuities. In this case, the pipeline execution with output intensity $I_{out} = I_3^*$ can be implemented.

The transformation $f_g \xrightarrow[D(f_g^*) \cap D(f_g)]{} f_g^*$ in this case enables pipeline data processing in the problem F, reducing the solution time proportional to $\frac{T_{proc}(I_3)}{T_{proc}(I_3^*)}$.

In practice, such substitution is not always possible, since the existence of $f_g^* \approx f_g$ depends on the algorithm of the problem, required accuracy, and data processing velocity. Therefore, in the general case, the influence of dataflow discontinuities can only be reduced.

Suppose that for solving a problem Z, containing subtasks z_{g-1}, z_g, and z_{g+1}, it is necessary to organize the pipeline processing of the input vector X_i. Without loss of generality, assume that X_i is processed sequentially through z_{g-1}, z_g and z_{g+1}, producing intermediate vectors X_i', X_i'', and output vector Y_i with intensities I_1 and I_2, respectively. $X_i' = z_{g-1}(X_i, Y_{i-1})$, $X_i'' = z_g(X_i', Y_{i-1})$, and $Y_i = z_{g+1}(z_g(z_{g-1}(X_i, Y_{i-1}), Y_{i-1}), Y_{i-1})$.

A simplified structure of the problem Z is shown in Fig. 7.

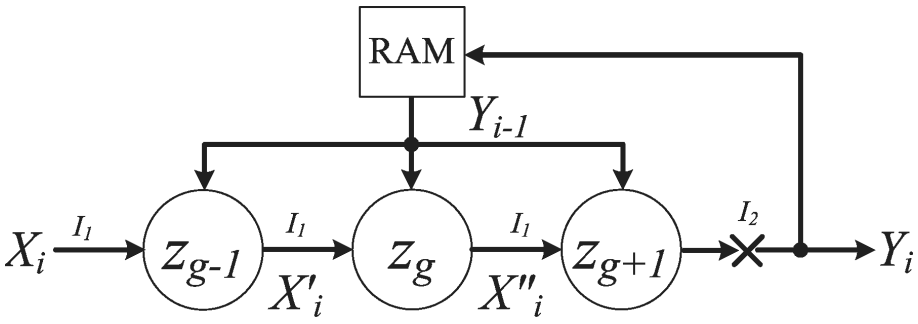


Figure 7. Simplified structure of the problem Z

A dataflow discontinuity arises ($I_2 = 0$), since z_{g+1} cannot update the vector Y_{i-1} until the entire vector X_i has been processed. The processing time T_{proc} for a set of vectors X_i, where $i = 1, \ldots, N$, in the problem Z can be estimated by expression (1).

At $I_2 = 0$, the updating of vector Y_i will begin only after the completion of processing of X_i. In the case where $\|X_i\| \gg \|Y_i\|$, the update time can be neglected and pipeline execution remains efficient, $T_{proc} \to t(X_i)$.

However, when $\|X_i\| \approx \|Y_i\|$, $T_{proc} \to 2 \cdot N \cdot t(X_i)$, meaning that half of the computing resources remain idle during updates of Y_{i-1}.

For time-consuming neural network training problems, the substitution $Z \xrightarrow[D(Z^*)\cap D(Z)]{} Z^*$ leads in practice to a change in architecture and training algorithm, which is unacceptable. Therefore, the effect of this dataflow discontinuity can only be reduced by partitioning RAM into p independent pages and organizing independent p-channel access to write Y_i, as shown in Fig. 8. Such cases are referred to as second-type dataflow discontinuities.

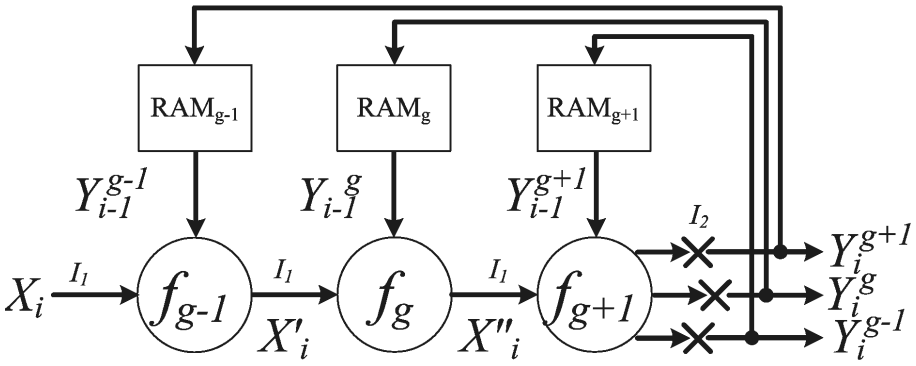


Figure 8. Structure of the problem Z with p-channel RAM access

The presence of a large number of high-speed horizontal and vertical links in the Arctur RCS switching subsystem enables independent multi-channel access to distributed memory in each FPGA. This reduces the write time of Y_i by a factor of p. In this case $T_{proc} \to N \cdot \left[t(X_i) + \frac{t(Y_i)}{p}\right]$. If p is such that $\frac{t(Y_i)}{p} \ll t(X_i)$, then $T_{proc} \to N \cdot t(X_i)$.

The described methods for minimizing first- and second-type dataflow discontinuities under structural organization of calculations improve the RCS performance by reducing the overhead costs of organizing the computing process. However, the efficient implementation of neural data processing on RCS is not possible using existing synthesis frameworks for neural network functional blocks. Existing FPGA frameworks such as FPGA AI Suite (Altera) [18] and Vitis AI (AMD Xilinx) [19] are intended for adaptation and deployment of pretrained neural network models developed in TensorFlow, PyTorch, and ONNX. FINN (AMD Xilinx) is oriented toward automated generation of specialized computing structures for execution of quantized neural networks on FPGAs in low-bit integer representations [20]. Universal frameworks are significantly less developed for the full training cycle of neural network models on FPGAs. Implementation of training, including forward pass, backpropagation, gradient calculation, and parameter updates, requires dedicated RTL-level hardware architecture. Existing solutions such as F-CNN are research-oriented and typically limited to small convolutional neural networks (up to five layers) in integer representation [21]. To improve the efficiency of structural implementation of neural calculations, the authors developed a library of neural functional blocks and a prototype framework for synthesis on RCS of convolutional neural network computing structures, including training structures. Using these developments, the authors synthesized the computing structure for ResNet-50v1.5 neural network for the first time on multi-chip RCS with support for various floating-point formats.

3. CNN Framework for the RCS

Convolutional neural networks (CNNs) are intended for data processing with spatial or locally structured organization, in which feature extraction is performed through convolution

operations using a set of local filters. Due to their ability to form hierarchical feature representations, this class of neural networks is widely used in image analysis problems, including classification, object detection, and segmentation. In addition to image processing, CNNs are used to detect anomalies, identify characteristic patterns, and predict text, sound, and time series processing. The developed library of neural network functional blocks contains all necessary components for designing CNNs on RCS, including: a linear convolution layer, normalization layers (batch, layer, and group normalization), activation functions (ReLU, sigmoid, Softmax), max-pooling layer, global average pooling layer, and fully connected layer. For neural network training on RCS, the library implements the components, required for calculating errors based on cross-entropy, calculating gradients of layers of all types, calculating gradients of normalization and activation, and correcting the weighting coefficients. In addition, the library includes synchronization and dataflow management blocks required for neural network construction on RCS.

All library components are universal for RCS systems with different hardware configurations and resource capacities. Therefore, to ensure the required RCS performance level, the prototype framework enables synthesis of computing structures using different methods for organization of calculations: procedural, pipeline, parallel-pipeline, nested pipeline, and macro-pipeline. Parallelization of calculations or structural reduction can be performed across different dimensions, including number of neurons, convolution kernels, data channels, operations, and other parameters, defined in the reduction transformation methodology [22].

Calculations in the developed functional blocks can be implemented using different numeric formats: FP32 (IEEE754 standard), TF32 (NVIDIA format), NicFP (18-bit format developed by SRC SC & NC), and BF16 (Google format).

Figure 9 shows a typical computing structure of a convolutional layer in training mode, which can be synthesized in the prototype framework using the neural network functional block library.

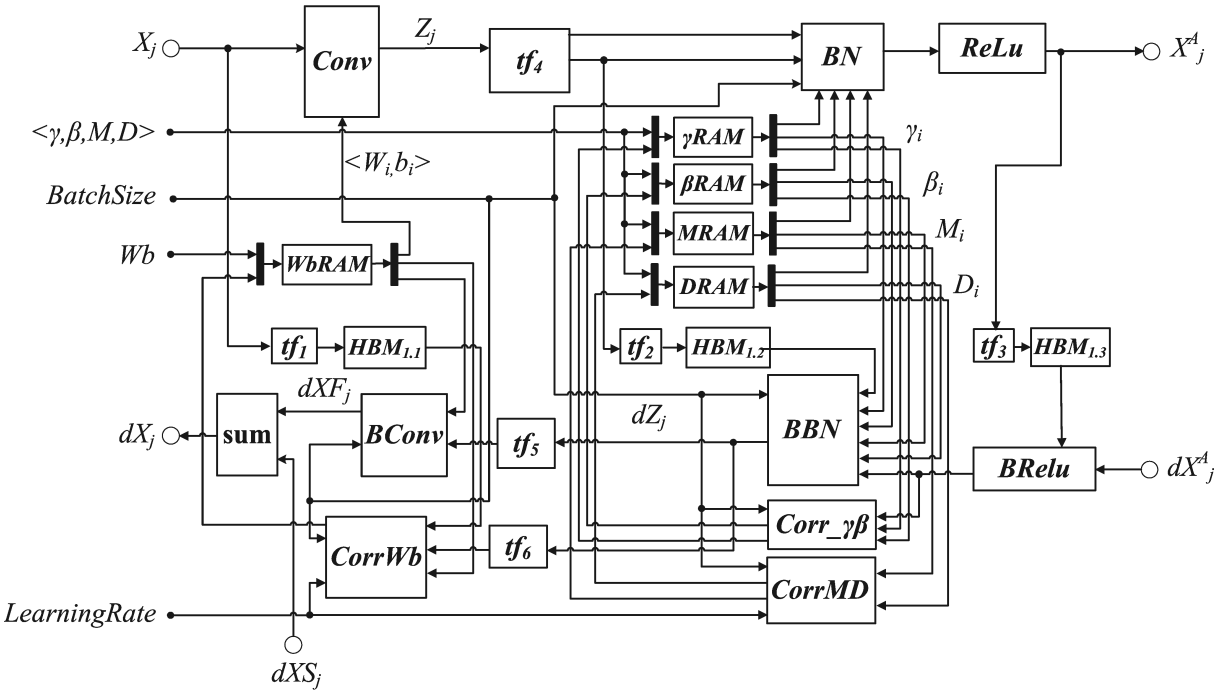


Figure 9. Computing structure of a convolutional layer in training mode

The presented training computing structure includes the following functional blocks for the forward pass: convolution layer ($Conv$), batch normalization block (BN), activation block ($ReLU$), weight memory block W_i and bias memory block b_i ($WbRAM$), scaling coefficient memory blocks γ_i and β_i ($\gamma RAM, \beta RAM$), and mean and variance memory blocks M_i and D_i ($MRAM, DRAM$).

For backpropagation, the computing structure includes: input operand memory block X_i ($HBM_{1.1}$); pre-activation memory block Z_i ($HBM_{1.2}$); activation memory block X_i^A ($HBM_{1.3}$); dataflow transformation blocks tf_1–tf_6 to reorder data flow between connected functional blocks according to the execution order defined by the computing structure; gradient calculation block for activations ($BReLU$); gradient calculation block for pre-activations dZ_i (BBN); scaling coefficient update block ($Corr\_\gamma\beta$); mean/variance update block ($CorrMD$); weight and bias update block ($CorrWb$); input gradient calculation block dXF_j ($BConv$); dXF_j accumulation block with gradients of the dXS_j traversal branch (sum).

Similarly, convolutional layer computing structures for both inference and training on RCS can be implemented with different kernel types (1×1, 1×1/2, 3×3, 3×3/2, 7×7/2, etc.), different numbers of neurons per layer, kernel counts per neuron, and memory access channel configurations. Based on this, CNNs of different architectures can be synthesized, including Sequential networks, Inception networks (parallel concatenation of Sequential branches), Residual networks (accumulation of Sequential branches with shortcut connections), and Dense networks (concatenation of outputs of all previous layers).

The authors' research is established a strict functional relationship between available RCS hardware resources, parallelization and reduction coefficients, the chosen calculation method, data representation format, and the parameters with which convolutional layers and networks can be implemented. For the developed framework prototype, this enabled the creation of a methodology for automated calculation of neural network computing structure parameters.

This enabled a transition from native RTL design of FPGA computing structures to structural-parametric RTL design [23], which significantly accelerates the development of multi-FPGA neural network architectures and reduces synthesis time for a 12-FPGA RCS "Arctur" implementation of ResNet-50v1.5 training from one and a half years to three months.

The developed library and prototype framework enabled synthesis of a computing structure for ResNet-50v1.5 training with pipelined data processing and support for multiple floating-point formats for the first time on a multi-chip RCS.

4. Analysis of the Arctur RCS Efficiency in Neural Network Training Problems

The selection of the ResNet-50v1.5 neural network from the MLPerf benchmark for implementation the training problem on the Arctur RCS hardware platform is based on the fact that ResNet-50v1.5 is a widely used reference model in image classification problems. Its training results are used to compare different computing platforms in terms of performance and energy consumption during the CNN training. In addition, a large number of scientific publications are available that provide experimental data obtained during ResNet-50v1.5 training on tensor computing systems. This makes it possible to perform a reliable analysis of the efficiency of the proposed solutions. The ResNet-50v1.5 training task consists of two subtasks: *Forward pass* and *Backward propagation*. Its computing structure is shown in Fig. 10.

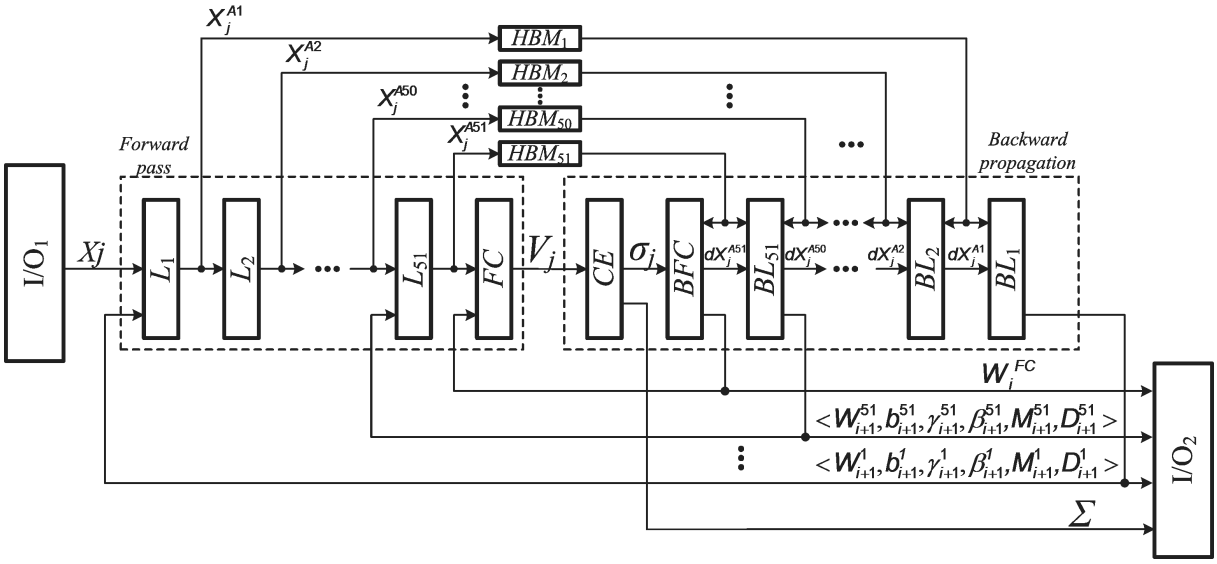


Figure 10. Computing structure of the ResNet-50v1.5 training task

The *Forward Pass* computing structure consists of the functional blocks L_1–L_{51} and FC, implementing the linear convolution layer $conv7\times7/2$, local maximum pooling layer $max\_pool3\times3/2$, linear convolution layers $conv1\times1$, $conv1\times1/2$, $conv3\times3$,$conv3\times3/2$, global average pooling layer $avg_pool7\times7$, as well as the fully connected classification layer $full\_connect1000$. The L_1, L_3–L_{50} linear layer blocks also implement the $BatchNormalization$ and $ReLu$ activation functions. The FC block also implements $soft\_max$ activation function.

The *Backward Propagation* computing structure consists of the functional blocks CE, BFC, BL_{51}–BL_1, implementing the calculation error based on cross-entropy $calc\_\sigma$, gradient calculation with respect to $soft\_max$ input, and gradient calculation with respect to the input and parameters of $full\_connect1000$ layer, as well as gradient calculation for $avg\_pool7\times7$ and $max\_pool3\times3/2$ layers, and gradient calculation for convolution layers.

Before the training process of ResNet-50v1.5, initial values of the packet counter $i = 0$, layer parameters $W_i, b_i, \gamma_i, \beta_i, M_i, D_i$ are initialized in the computing structure. The image counter $j = 0$ is also initialized for images within a batch, and the $BatchSize$ and $LearningRate$ are set.

In *Forward Pass* computing structure, pixel values of image X_j from the batch $Batch_i = \{X_0, X_1, \ldots, X_{BatchSize}\}$ are fed into block L_1. At the output of L_1, an activation stream X_j^{A1} is formed, which is passed to L_2 and simultaneously stored in HBM_1. From L_2, the activation stream X_j^{A2} is passed to L_3 and stored in HBM_2. For all other blocks L_3–L_{51} and FC, data streams are formed similarly and written to HBM_3–HBM_{51}.

In *Backward Propagation* computing structure, the vector of class probability predictions V_j from FC is passed to block CE, where the error vector σ_j is formed. The input of BFC receives the activation stream X_j^{A51} together with vector σ_j. At the output of BFC, the gradient stream dX_j^{A51} is formed, which is passed to BL_{50} together with X_j^{A51} and X_j^{A50}. From BL_{50}, the gradient stream dX_j^{A50} together with X_j^{A50} and X_j^{A49} is passed to BL_{49} and so on for all BL_{48}–BL_1. Simultaneously, during gradient calculation for inputs dX_j^{A51}–dX_j^{A1}, in blocks BFC, BL_{50}–BL_3 and BL_1 for all images in the i batch ($j = 1 \ldots BatchSize$), gradients with respect to parameters are computed: weight coefficients dW_i^{FC}, dW_i^{50}–dW_i^3, dW_i^1, bias coeffi-

cients db_i^{FC}, db_i^{50}–db_i^3, db_i^1, scaling coefficients $d\gamma_i^{50}$–$d\gamma_i^3$, $d\gamma_i^1$, shift coefficients $d\beta_i^{50}$–$d\beta_i^3$, $d\beta_i^1$, as well as moving averages dM_i^{50}–dM_i^3, dM_i^1 and variances dD_i^{50}–dD_i^3, dD_i^1.

After processing $Batch_i$, new values W_{i+1}, B_{i+1}, γ_{i+1}, β_{i+1}, M_{i+1}, D_{i+1} are calculated and overwritten in the parameter memory $WbRAM$, γRAM, βRAM, $MRAM$, $DRAM$ for each layer L_1, L_3–L_{50}. At this stage, $LearningRate$ may also be updated. After that, the images from $Batch_{i+1}$ are fed into L_1, and the process repeats. In CE block the total loss Σ is also calculated, based on which training termination can be decided. Once training is complete, the parameters obtained in *Backward propagation* can be used for image classification.

Experimental research studies of the proposed approach were performed on two Arctur RCS computing modules (12 Xilinx XCVU37P FPGAs). The computing structure of ResNet-50v1.5 training was synthesized using Vivado 2022.1 in NicFP processing format. The clock frequency of the computing structure was 500 MHz. Training of ResNet-50v1.5 was performed on the ImageNet dataset with RGB-image resolution 224×224, $BatchSize = 256$, $LearningRate = 0.01$. The processing velocity of the computing structure was 2200 img/s. This corresponds to an achieved performance of 55.44 TFLOPS on two computing modules or 27.72 TFLOPS per computing module. The peak performance of a computing module in NicFP format is 38.16 TFLOPS. Therefore, the computational efficiency of the computing module in the ResNet-50v1.5 training task is 72.6%.

For comparison, the processing velocity of ResNet-50v1.5 training on a single NVIDIA Tesla A100 80GB GPU in TF32+XLA processing mode is 938 img/s, with the real performance of 23.63 TFLOPS [24]. At the same time, the peak performance of A100 in TF32 mode is 156 TFLOPS [25], which corresponds to a computational efficiency of 15.14% during ResNet-50v1.5 training.

The NicFP (18-bit) and TF32 (19-bit) data representation formats differ only in mantissa length – 9 bits and 10 bits, respectively. The numerical analysis of the accuracy shows an error of about 0.07%. Therefore, it can be argued that the computational efficiency of the Arctur RCS exceeds the computational efficiency of the NVIDIA A100 GPU by almost 4.8 times.

Further scaling of the computing structure of the ResNet-50v1.5 training task on the Arctur RCS block (16×computing module) enabled, due to the structural organization of calculations, a linear increase at image processing velocity to 17600 img/s. The real performance of about 443.5 TFLOPS was obtained with the computational efficiency of 72%.

Based on experimental energy consumption data, obtained during the research, an evaluation of the energy efficiency of the Arctur RCS block was performed in comparison with a single NVIDIA DGX A100 block (8×Tesla A100 80GB GPUs + 2 AMD EPYC 7742 CPUs). The DGX A100 block achieves a performance of 182.6 TFLOPS [24] with the estimated power consumption of about 4.9 kW based on NVIDIA experimental data for the DGX A100 (4×Tesla A100 40GB) node [26]. The energy efficiency of DGX A100 node in this case is estimated at 37.27 GFLOPS/W. The Arctur RCS block achieves a performance of 443.5 TFLOPS with a power consumption of about 11 kW. Therefore, its energy efficiency is 40.3 GFLOPS/W, which is about 8% higher than that of DGX A100 block.

At the same time, NVIDIA A100 GPUs, focused on solving AI problems, do not have linear scalability. Published research results, conducted by NVIDIA [26] and the results of the MLPerf benchmark [27], show that at training the ResNet-50v1.5 neural network an eight-fold increase in the number of A100 GPUs in the computing system leads to a 7.7-fold increase in system performance, while a 128-fold increase results in increase of only 79 times, and a 4000-fold

increase results an increase of only 600 times. This is due to the fact that with an increase in the scaling factor, the impact of the time required to exchange calculation results between system nodes on the overall time to solve the problem increases. Unlike tensor systems, information links between blocks of the computing structure (Fig. 9) on RCS are implemented in hardware, which helps minimize the overhead costs of data exchange.

Theoretical researches show that the computing structure in NicFP format scaled to four The Arctur RCS blocks provides the real performance of about 1.77 PFLOPS. Such performance levels are only achievable on 16 DGX A100 nodes. At the same time, power consumption of four The Arctur RCS nodes is approximately 50 kW, while 16 DGX A100 nodes consume about 78.4 kW.

A single RCS rack, consisting of 16 Arctur RCS blocks, can potentially provide the real performance of 7 PFLOPS (NicFP). For neural network problems such as image classification or training of ResNet-50v1.5, the performance of this rack is approximately equivalent to 12 NVIDIA DGX A100 racks. The power consumption of the Arctur RCS rack does not exceed 0.21 MW, which is 2.2 times lower than that of 12 DGX A100 racks (approximately 0.46 MW).

Conclusion

The main advantage of the structural organization of calculations over the procedural approach is the ability to adapt the RCS architecture to the information graph of the solving problem. To exploit this advantage on the Arctur RCS for neural network data analysis and training problems, methods for minimizing dataflow discontinuities have been developed and experimentally validated. These methods enable the synthesis of pipelined neural network computing structures on RCS with parallelization across both data and iterations.

A library of functional blocks has been developed to create and train multilayer CNNs of various architectures (Sequential, Inception, Residual, and Dense) on RCS. Pipelined data processing is supported for widely used floating-point formats (FP32, TF32, BF16), as well as for the proprietary NicFP format.

A prototype framework has also been developed, enabling the transition from native to structural-parametric RTL design, accelerating the development process of neural network computing structures for RCS by up to six times.

Therefore, a hardware-software foundation has been formed, which can be the basis for the development of domestic reconfigurable high-performance computing complexes that provide high quality and operational solutions to neural network problems. With linear scaling of computing resources, these systems provide near-linear performance scaling. At the same time, the energy efficiency of computing systems based on the Arctur RCS, consisting of dozens of racks, is expected to be significantly higher than that of systems based on tensor accelerators comprising hundreds of racks.

References

1. Lavrentyev, A.S., Solovyev, V.D.: Artificial intelligence: theory and practice. Nauka Publ., Moscow (2020).

2. NVIDIA DGX A100. `https://www.nvidia.com/ru-ru/data-center/dgx-a100/`, accessed: 2026-04-06

3. Cloud TPU system architecture. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm, accessed: 2026-04-06.

4. Huawei Ascend 910 (512 TFLOPS). https://www.ixbt.com/news/2019/08/23/huawei-ascend-910-512-tflops-310.html, accessed: 2026-04-06

5. Xiong, R., Yang, Y., He, D., *et al.*: On layer normalization in the transformer architecture. Proceedings of the 37th International Conference on Machine Learning, 2020, vol. 119, pp. 10524–10533. https://proceedings.mlr.press/v119/xiong20b.html, accessed: 2026-04-08

6. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. Computer Vision – ECCV 2016, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0-8

7. Kasarkin, A.V., Levin, I.I., Sorokin, D.A.: New iteration parallel-based method for solving graph NP-complete problems with reconfigurable computer systems. IOP Conference Series: Materials Science and Engineering 919(1), 052007 (2020). https://doi.org/10.1088/1757-899X/919/5/052007

8. Sorokin, D.A., Matrosov, A.Y., *et al.*: Real-time implementation of the problem of surface-related multiple prediction on RCS. Parallel Computational Technologies, PCT 2019, Kaliningrad, Russia, 2019, pp. 91–98.

9. Sorokin, D.A., Dordopulo, A.I., Levin, I.I.: Implementation of docking for molecular modeling on reconfigurable computing systems. Izvestiya SFedU. Engineering Sciences 7, 217–224. (2011).

10. APEgate project. https://apegate.roma1.infn.it/?page_id=656, accessed: 2026-04-08

11. Kalyaev, A.V., Levin, I.I.: Modular scalable multiprocessor systems with structural-procedural organization of computations. Janus-K Publ., Moscow (2003).

12. Kalyaev, A.V.: Training procedure for a multilayer neuroprocessor network using feedback. Information Technologies 6 (2002).

13. AMD. UltraScale Architecture GTH Transceivers User Guide (UG576). Version 1.7.1. San Jose, 2021. https://www.amd.com/content/dam/xilinx/support/documents/user_guides/ug576-ultrascale-gth-transceivers.pdf, accessed: 2026-04-08

14. Lee, J.-C., Kim, J., Kim, K.W., *et al.*: High bandwidth memory (HBM) with TSV technique. 2016 International SoC Design Conference, ISOCC, pp. 181–182 (2016). https://doi.org/10.1109/ISOCC.2016.7799847

15. Levin, I.I., Doronchenko, Yu.I.: Advanced reconfigurable supercomputer with immersion cooling. Proceedings of the XIV All-Russian Conference on Control Problems, VSPU-2024, pp. 2256–2260 (2024). https://doi.org/10.25699/vspu2024.2256

16. NVIDIA DGX A100 user guide. https://docs.nvidia.com/dgx/dgxa100-user-guide/introduction-to-dgxa100.html, accessed: 2026-04-08

17. Lohrmann, B., Warneke, D., Kao, O.: Nephele streaming: stream processing under QoS constraints at scale. https://doi.org/10.1007/s10586-013-0281-8 (2013), accessed: 2026-04-08

18. Intel Corporation. FPGA AI Suite – AI inference development tools for FPGAs. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/fpga-ai-suite.html, accessed: 2026-04-08

19. AMD. Vitis AI software for adaptive SoCs and FPGAs. https://www.amd.com/en/products/software/vitis-ai.html, accessed: 2026-04-08

20. AMD Research I& Advanced Development. FINN: dataflow compiler for QNN inference on FPGAs. https://xilinx.github.io/finn/, accessed: 2026-04-09

21. Zhao, W., Fu, H., Luk, W., *et al.*: F-CNN: an FPGA-based framework for training convolutional neural networks. Proc. 27th IEEE Int. Conf. Application-Specific Systems, Architectures and Processors, ASAP 2016, London, UK, 2016. pp. 107–114. IEEE (2016). https://doi.org/10.1109/ASAP.2016.7760779

22. Levin, I.I., Dordopulo, A.I.: Reconfigurable computing systems: resource-independent programming. Southern Federal University Publ., Rostov-on-Don (2025).

23. Chu, P.P.: RTL hardware design using VHDL: coding for efficiency, portability, and scalability. Wiley, Hoboken, NJ (2006).

24. NVIDIA DeepLearningExamples: ResNet-50 v1.5. https://github.com/NVIDIA/DeepLearningExamples/blob/master/PyTorch/Classification/ConvNets/resnet50v1.5/README.md, accessed: 2026-04-09

25. NVIDIA A100 datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf, accessed: 2026-04-09

26. NVIDIA AI Enterprise sizing guide (archived). https://web.archive.org/web/20251110193817/https://docs.nvidia.com/ai-enterprise/sizing-guide/latest/appendix.html, accessed: 2026-04-09

27. MLCommons training benchmarks. https://mlcommons.org/benchmarks/training/, accessed: 2026-04-09

DOI: 10.14529/jsfi260205

"Theseus" Parallel Compiler for Multichip Reconfigurable Computer Systems

Vyacheslav A. Gudkov[1]***, Ilya I. Levin***[1]

Modern system software for field-programmable gate arrays (FPGA) and FPGA-based computer systems implements individual task fragments as IP cores. It requires their further integration into a unified computing structure, as well as ensuring correct information of data flows. The "Theseus" parallelizing compiler was developed and characterized by a comprehensive implementation of the entire problem for a variety of FPGA chips. The created automatic parallelizing compiler significantly reduces the requirements for developer qualifications in FPGA programming and does not require marking up the source code of a sequential program with service directives. The parallelizing compiler implements a technology for converting sequential calculations into the most parallel form an information graph of an application task algorithm that is automatically mapped to the target configuration of a multichip reconfigurable computer system using formal methods for reducing the performance of the computing structure. The parallelizing compiler makes it possible to significantly reduce the conversion time of sequential programs to parallel-pipelined solutions for reconfigurable computer systems containing multiple FPGA chips connected by a spatial communication system, while ensuring a guaranteed level of real performance. Due to the efficient organization of calculations and rational use of the available FPGA hardware resources, the "Theseus" compiler will provide significantly higher real performance of a reconfigurable computer system compared to the similar HLS compilers (High-Level Synthesis Compiler). The results of creating application problems from various subject areas for the "Arktur" reconfigurable computer systems using the proposed methods for performance increasing are presented.

Keywords: high-level synthesis, HLS, program translation, C language, performance reduction, reconfigurable computer system, programming of multiprocessor computer systems, increase performance.

Introduction

The high complexity of developing hardware configurations for field-programmable gate arrays (FPGAs) and computer systems based on them require the creation of new development tools with significant reduction in the implementation time for FPGA solutions and the vendor qualification requirements.

Currently, HLS compilers, allowing the FPGA programming in high-level languages, are replacing automated circuit design tools based on HDL (Hardware description language). This reduces the requirements for developers of computer systems and accelerates the programming process by quickly creating individual IP cores, which are digital machines or specialized processors, for computationally laborious program fragments. Automating of IP core generation by an HLS compiler makes it possible to quickly port segments of sequential programs, for example, written in C language, to the FPGA architectures. However, this does not guarantee the efficient implementation, since dataflow organization remains the vendor responsibility, and tools for scaling and synchronization (even within a single chip) are lacking. For reconfigurable computer systems (RCS) [1, 2] with multiple FPGAs, connected via a spatial interconnection network, the configuring of the synchronization and coordination of simultaneous operation of

[1] "Scientific Research Center of Supercomputers and Neurocomputers" Co Ltd ("SRC SC & NC" Co Ltd), Taganrog, Russian Federation

multiple IP cores is a complex and labor-intensive process, comparable in complexity to the development of a new circuit design.

The "Theseus" parallelizing compiler described in this paper is designed to convert the sequential C programs in the ISO/IEC 9899:1999 standard into multichip RCS solutions with automatic synchronization of data and control signals for all used FPGAs.

The paper presents the results of the development of the "Theseus" parallelizing compiler with automatic conversion of sequential C programs to an available hardware resource of multichip RCS. The paper consists of an Introduction, five sections and a Conclusion.

The first section provides a brief overview of known HLS tools. In the second section, the synthesis of the cadr structure of the parallel-pipeline program is considered. The third section discusses methods for increasing the productivity of cadr structures. The fourth section presents the structure of the modern "Theseus" complex, as well as the main operation stages of the new Ariadne component. The fifth section presents the results of operation of the modern parallelizing "Theseus" compiler on the modern high-performance computer system Arcturus. In Conclusion, the obtained results are summarized, and directions are formulated for further research.

1. Overview of Known High-level Synthesis Tools

For FPGA programming, along with traditional hardware design systems such as Xilinx Vivado and Intel Quartus Prime, high-level synthesis (HLS) tools or HLS compilers [3, 4] are increasingly being used. These tools convert a program in one high-level languages into configuration files of specialized hardware in Hardware Description Languages (HDL). Depending on the input language, HLS compilers are either problem-oriented language translators, i.e., versions of programming languages adapted to a specific problem domain, or general-purpose language translators, i.e., high-level language dialects for traditional microprocessors with some features and limitations. The classification of high-level synthesis tools according to these criteria is given in Fig. 1.

It can be noted that despite the differences in code conversion methods and data formats, HLS compilers have common limitations [3, 5, 6]:

- synthesis of the project is performed for a single FPGA chip located on a development board or a single-chip reconfigurable computer system (RCS);
- to create an effective project, it requires pre-marking the code with directives (e.g., #pragma in Vivado), which requires deep and sometimes expert knowledge about the synthesized project;
- synchronization and scaling of IP cores within a solution are performed by the vendor independently.

The disadvantage of traditional HLS compilers is the focus on programming a single single-chip FPGA, which simplifies synthesis but limits scalability. Problems requiring simultaneous operation of several IP cores (for example, big data processing or neural network models) force the development of complex synchronization systems manually. This increases development time and increases the likelihood of errors, especially when working with multiple FPGAs.

The synthesis of multichip solutions is significantly more complicated, since it requires the support and synchronization of a large number of computing devices for various forms of computing organization, as well as the analysis and selection of rational options from a variety of possible ones. Therefore, it is comparable in complexity to automatic parallelization.

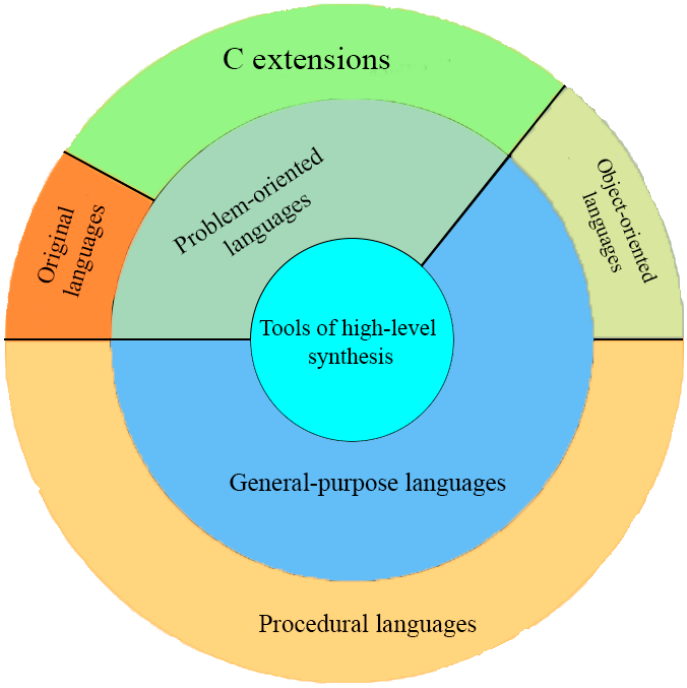


Figure 1. Classification of high-level synthesis tools

The "Theseus" parallelizing compiler developed at SRC SC and NC [7] converts a sequential C program to an available hardware resource of multichip RCS. The differences between the "Theseus" complex and the most well-known Xilinx Vivado HLS and Vitis HLS compilers are as follows:

- there are no requirements for code annotation in sequential programs, which reduces the requirements for application developers, and sequential programs become universal;
- the conversion is based on the analysis of the data dependencies in the input program, rather than manual code annotation directives, similar to #pragma in Vivado HLS;
- for high-level synthesis and scaling of its results, the parameters of parallelism in the cadr structure are reduced rather than parallelized, as is typical for a sequential program;
- arbitrary memory access in a sequential program is converted to regular read/write operations for data streams;
- automatic adaptation of the application to the architecture and configuration of a user-defined RCS;
- automatic synchronization of data and control signals across all utilized FPGA chips;
- support for problems from various application domains.

However, the parallelizing compiler [7] has significant disadvantages:

1. Creation of only one type of computing structure – information graphs with parallelization by layers and iterations.
2. The expediency of creating procedural blocks in which nested conditional transitions and recurrent expressions are effectively implemented is ignored.
3. No means for integrating pipeline and procedural blocks within a unified computing structure, limiting the parallelizing compiler capabilities.
4. A parallelizing compiler does not provide rational generation of program constructs of parallel-pipelined programs. It reduces the efficiency of synthesized applications and requires a large number of iterations in a parallel-pipelined program, which leads to significant

translation time reducing the performance of synthesized applications and requiring extensive exploration of design alternatives, which leads to long compilation times.

In this regard, it is necessary to develop a new version of the compiler, which will implement a fundamentally different approach to converting a parallel program to a parallel-pipeline program, based on the use of performance reduction. Performance reduction is defined as a proportional decrease in performance in all fragments of the information task graph with a possible reduction in hardware costs for implementing the computing structure [8]. Performance reduction can be performed by: memory channels, hardware resources, data width, and frequency.

The use of performance reduction makes it possible to adapt the cadr structure to the available hardware resources of the target RCS. However, it can lead to inefficiency in the data flows of the cadr structure and significant decrease in the performance, which is unacceptable.

2. Synthesis of the Cadr Structure of a Parallel-pipeline Program

The operation of the parallelizing compiler consists of two stages: converting a sequential program to a parallel-pipeline program and adapting the parallel-pipeline program to the configuration of the target reconfigurable computer system (RCS).

At the first stage, the algorithmic scheme of a sequential program is transformed into an information graph, the vertices (subgraphs) of which correspond to data operations, and the edges represent data dependencies between them. Information-independent vertices are located in layers corresponding to the layers of the algorithm graph [9], while the information dependencies between subgraphs from different layers are reflected by iterations. The distribution of subgraphs by layers and iterations makes it possible to visualize the information dependencies between task fragments at different levels of the hierarchy.

In addition, the information graph is invariant to various forms of description of calculations in a sequential program. Therefore, sequential programs of the same task that differ in syntax and form are represented by the same information graph. By replacing the operational vertices with computing devices, and the arcs with the connections of the switching system, the information graph is transformed into a maximally parallel cadr structure [10] that combines the computing structure of the task and the rules for organizing data flows. The cadr structure is characterized by the number of layers and iterations, data width and number of devices in the base subgraph, data processing interval, latency and operation frequency. Together these parameters determine the degree of parallelism, execution time, and hardware resource utilization.

The cadr structures of sequential programs contain recursive expressions and nested conditional transitions, specific to individual problems of digital signal processing, symbolic processing, optimization and graph theory. For their formation, it is necessary to develop new methods for converting sequential programs into an information graph and methods for displaying it in the syntax of the COLAMO high-level language, taking into account the irregularity of information exchanges and the specifics of the problem being solved.

At the second stage, the methodology [7] for converting the cadr structures is used to adapt the cadr structure to the available resources of various RCS configurations.

At selecting transformations, the dynamics of the nonlinear variation of hardware costs $A_{CS} = (a_1^{CS}, a_2^{CS}, ..., a_n^{CS})$ of the cadr structure is analyzed as its main performance parameters

$P_{CS} = (p_1, p_2, ..., p_n)$ change, taking into account data dependencies both within and between subgraphs (task fragments).

The performance parameters of the cadr structure P_{CS} include: L_i^{it} is the number of data-independent subgraphs in the layer F_{ij}; It_i^{it} is the number of iterations in the function f_i; Ch_i^{it} is the number of external channels of the cadr structure; Op_i^{it} is the number of devices in the cadr structure FG^{it}; ρ_i^{it} is the data bit width, the method of hardware implementation of the base subgraph g_i and the data processing interval I.

Each of these parameters, depending on the subject area, structure, and problem dimension, can be crucial for the critical hardware resource of the transformed program [11].

For RCS based on FPGAs, the total number of analyzed variants of rational cadr structures with different parameters is relatively small; in the limiting case, for each base subgraph it does not exceed the number of permutations of performance parameters, which in the considered case equals 5!=120. As a rule, depending on the first variable cadr parameter, the remaining transformations of the cadr structure are determined.

The influence of performance characteristics on the hardware resources occupied by the cadr structure makes it possible to unify them in one view and present the transformation of the cadr structure to the available RCS resources (the local goal of transformations) as a search for rational values of performance parameters (the global goal of transformations).

The hardware cost A_{CS} of the cadr structure depends nonlinearly on the performance parameters P_{CS} : $A_{CS} = \Phi(P_{CS})$, where a single performance parameter p_i may simultaneously affect several hardware resource characteristics.

The main problem of cadr structure transformations is to search the performance parameters P'_{CS} that will be the solution of the system:

$$\begin{cases} A'_{CS} = \Psi(P'_{CS}) \le A_{HCS} \\ Perf(P'_{CS}) \ge Perf_{req} \end{cases},$$

where A_{HCS} is the available hardware resource of the target RCS; $Perf(P'_{CS})$ is the performance of the cadr structure; $Perf_3$ is the specified level of real performance.

However, the performance of the cadr structure depends not only on the rational use of available hardware resources, but also on the organization of data flows.

If the computational resources are not sufficient to fully implement iteratively interconnected subgraphs in the cadr structure, then feedback will be generated at performing the performance reduction by subgraphs in the cadr structure. In this case, the processing of a single input data requires $M = \frac{N}{r}$ passes, where N is the number of subgraphs and r is the degree of performance reduction across subgraphs. This leads to a necessity to provide the input data with data initiation interval $S = N * Lat$, where Lat is the latency of the reduced subgraph.

Note that regardless of the degree of performance reduction r, the processing time of one data remains will not change. Therefore, according to [12], if there exists an information graph G, consisting of a set of interconnected isomorphic subgraphs $p_1, p_2, \ldots, p_N$, and the system resources are insufficient for hardware implementation of the entire graph G, then the maximum efficiency (the specific performance) is achieved through the hardware implementation of a single subgraph p_i.

The optimal structure of graph implementation if the available hardware resources are insufficient to implement all subgraphs F_2, is presented in *Fig.* 2.

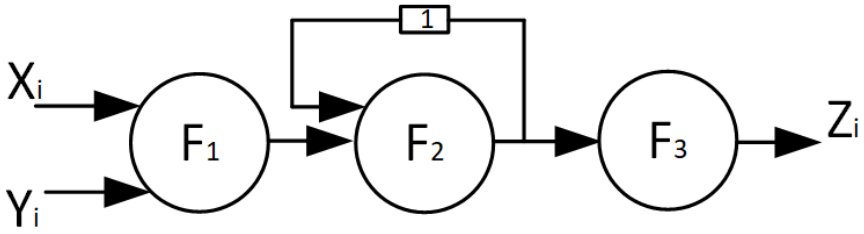


Figure 2. Optimal graph structure

To ensure a uniform data flow processing rate, the reduction transformations over the chain of F_2 subgraphs require to use the performance reductions by the value r and to subgraphs F_1 and F_3.

The use of performance reduction for iterative and procedural subgraphs leads to a significant decrease in performance, which is unacceptable at synthesizing cadr structures by a parallelizing compiler.

To improve the performance of cadr structures, a modified methodology for cadr structure synthesis has been proposed, consisting of the following steps:

1. Calculation of cadr structure parameters

1.1. Determine the critical resource and the reduction coefficient of hardware costs $K_{cr} = max\left(\frac{a_i^{CS}}{a_i^{HCS}}\right) > 1$.

1.2. Determine the reduction coefficient $R = \lceil K_{cr} \rceil$.

1.3. Arrange the performance parameters p_i in the tuple $< P_{CS} >$ according to the influence degree on the critical resource.

1.4. If an unreduced parameter exists in $< P_{CS} >$, then select the performance parameter with the maximum impact on the critical resource: $p_i : \psi(p_i) = max\ K_{cr}$. else: go to item 7 to select a card structure with the maximum performance.

2. Determination of the effective reduction step R^*.

2.1. For the selected performance parameter p_i, determine one of the possible options for the effective reduction step R^*:

1) $R^* = R$;

2) $R^* = f(R,\ p_i),\ R^* < R$;

3) $R^* = \|pi\|, R* > R$.

3. Reduction (decreasing of the performance parameters) of the cadr structure

3.1. Reduce the parameter p_i with the effective step R^*: $p'_i = \Theta(p_i, R^*)$ and save it to a tuple: $< P'_{CS} >= < P_{CS} > \leftarrow p'_i$

4. Evaluation of the current cadr structure and selection of alternatives

4.1. Calculate hardware costs and performance of the cadr structure $A\prime_{CS} = \Psi(P\prime_{CS})$, $Perf(P'_{CS})$.

4.2. Evaluate the achievement of porting purpose from the conditions $A\prime_{CS} = \Psi(P\prime_{CS}) \leq A_{HCS}$ and $Perf(P'_{CS}) \geq Perf_3$.

4.3. If both conditions are performed, then go to item 7 to save the current parameters of the cadr structure

else: Analysis of alternatives for further reduction:

If $A\prime_{CS} > A_{HCS}$ the hardware costs exceed the available RCS resource, then:

If $R^* = R$, then

– increase the reduction coefficient $R :- R + \triangle$, where $\Delta = \left\lceil \frac{A'_{CS}}{A_{HCS}} - 1 \right\rceil$;

– restore the original value of the performance parameter $< P\prime_{CS} >= < P_{CS} > \leftarrow p_i$;

– go to item 1.3 for reduction with the increased coefficient R.

Else

– go to item 1 to decrease the critical resource using the next performance parameter p_{i+1}.

If $A\prime_{CS} \leq A_{HCS} and Perf(P'_{CS}) < Perf_3$, then we go naturally to induction.

5. Induction (increasing of the previously reduced performance parameters) of the cadr structure

5.1. If the tuple of parameters $|S| = 0$, go to item 6.

5.1. Form the tuple of the previously reduced parameters $< p_{i-1}, \ \ldots p_1 >$.

5.2. Select the next item of the tuple p_k, determine its scaling coefficient according to the hardware resource $K_M = min\left(\frac{A'_{CS}}{A_{HCS}}\right)$ and the induction coefficient $Ind = \lfloor K_M \rfloor$.

5.3. For the selected performance parameter p_k, determine one of the possible effective induction steps:

1) $Ind^* = Ind$;

2) $Ind^* = \varphi\,(Ind, p_k) < Ind$;

3) $Ind^* = pk$.

5.4. If increasing the parameter p_k leads to the increased data initiation interval of the cadr structure, then go to item 5.6.

5.5. Increase p_k by the selected step Ind^* : $p\prime_k = \Theta^{-1}(p_k, Ind^*)$ and save it into the tuple: $< P\prime_{CS} >= < P\prime_{CS} > \leftarrow p\prime_k$.

5.6. Go to item 4.

6. Performance improvement

6.1. Form a tuple of parameters S associated with changes in data initiation interval within the cadr structure $\langle p_{i-1}, \ldots, p_1 \rangle$. If $\mid S \mid = 0$, go to item 7.

6.2. Select the next element of the tuple p_k and determine the corresponding interval value s_k.

6.3. Determine the degree of influence of the element p_k on the data initiation interval s_k.

6.4. If the data initiation interval value $s_k > 1$, then increase the scaling factor by s_k for all parameters $p_t \notin S$.

6.5. Go to item 4.

7. Save the cadr structure

7.1. Add the current parameters of the cadr structure to the list, ordered by the minimum performance.

7.2. If $Perf(P'_{CS}) < Perf_3$ and there were other critical resources then select the next critical resource and go to item 1.1.

8. Select a reasonable variant for reduction of the cadr structure

8.1. Issue the first item of the list of cadr structures with the highest performance.

The considered methodology changes the parameters of the cadr structure in such a way as to ensure the rational use of available hardware resources and achieve a given level of real performance. If the target performance level $Perf_{req}$ is unattainable, the result will be a cadr structure with the highest performance level among those analyzed.

Unlike the methods of structural and procedural organization of calculations, which find a rational cadr structure for a predetermined and fixed resource, the transformation is a function that depends on the architecture and configuration of the computing system in this methodology. This ensures exploration of the cadr structure area not only “downward” toward reduced

parallelism (reduction), but also "upward" toward increased parallelism (induction) if hardware resources permit. Combining the reduction and induction makes it possible to define the rational performance parameters for cadr structures even when there is a shortage of hardware resources for the structural implementation of the basic subgraph.

The modified methodology for the synthesis of personnel structures allows for a finite number of permutations to find a solution that ensures the rational use of hardware resources. It also provides performance improvement of the cadr structure by reducing the complexity of data flow organization by converting the cadr structure to efficient calculation forms.

3. Proposed Performance Increasing of the Cadr Structure

The modification of the cadr structure synthesis methodology is caused by the fact that the cadr structure performance depends not only on the rational use of the available FPGA hardware resources in a reconfigurable computer system, but also on the data initiation interval. Data initiation interval depends on the processing intensity in different subgraphs of the cadr structure.

Let the computing structure consist of a set of subgraphs $P = \{p_1, p_2, \ldots, p_n\}$, where the intensity of input and output data flows is defined as $p_i = \langle s_{in}, s_{out} \rangle$ for each subgraph p_i. The data initiation interval of a subgraph p_i is calculated by the formula $S = \frac{1}{v}$, where $v = \frac{S_{out}}{S_{in}}$. The total data initiation interval of the cadr structure is determined by the maximum interval among all its subgraphs. Data initiation interval leads to the equipment downtime and, as a result, reduced performance. Methods for converting computing structures can be used to reduce the data initiation interval: nested pipelining (pipeline-to-pipeline) [13], macro-pipelining [14], and auto-substitution [15].

If the cadr structure contains several data-independent subgraphs $G = \{g_1, g_2, \ldots, g_n\}$ with feedback, the most efficient "pipeline-to-pipeline" transformation can be performed to increase specific performance. The "pipeline-to-pipeline" transformation reduces the subgraphs G by a factor r, replacing the reduced subgraphs with registers in the feedback loop. Maximum efficiency is achieved at $r = Lat$, where Lat is the latency of subgraph g_i, provided that $\mid G \mid \geq r$. In this case, instead of feeding a single datum into subgraph g_i, a data packet can be supplied, representing a slice of operands entering the reduced subgraphs G. The length of the operand packet depends on the number of registers in the feedback loop.

Let the input data flows to the subgraphs G have the form:

$$\langle x_0^n, \ldots, x_0^2, x_0^1, x_0^0 \rangle$$

$$\langle x_1^n, \ldots, x_1^2, x_1^1, x_1^0 \rangle$$

$$\langle x_2^n, \ldots, x_2^2, x_2^1, x_2^0 \rangle$$

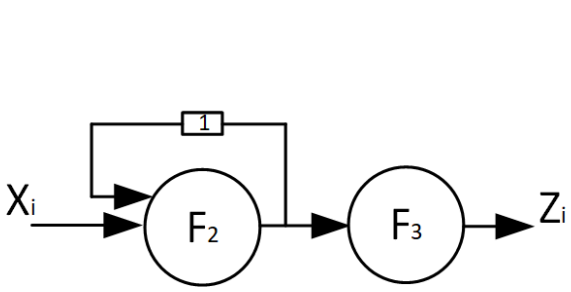
$$\vdots$$

$$\langle x_{r-1}^n, \ldots, x_{r-1}^2, x_{r-1}^1, x_{r-1}^0 \rangle$$

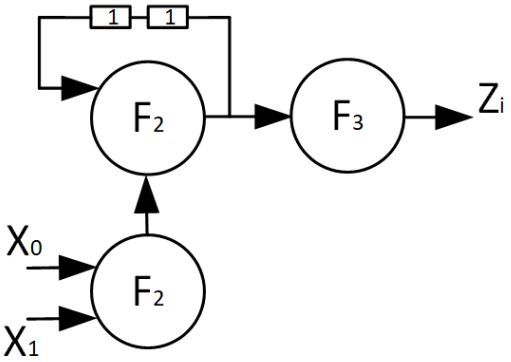
Then, the data packets for subgraph g_i will consist of r elements and have the form:

$$\langle x_{r-1}^0, \ldots, x_2^0, x_1^0, x_0^0 \rangle, \langle x_{r-1}^1, \ldots, x_2^1, x_1^1, x_0^1 \rangle, \ldots, \langle x_{r-1}^n, \ldots, x_2^n, x_1^n, x_0^n \rangle$$

The data initiation interval of the reduced cadr structure will decrease proportionally, becoming $S = \frac{Lat}{r}$. The "pipeline-to-pipeline" transformation does not reduce the task execution time, but it significantly reduces hardware costs by a factor of r and increases specific performance. The released hardware resources can be used to implement the other subgraphs of the cadr structure. If the cadr structure lacks multiple data-independent subgraphs G, i.e., $\mid G \mid = 1$, the "auto-substitution" method is appropriate for recursive procedures and recurrent expressions. In general, an expression of the form $x_i = f(x_{i-1})$ can be rewritten as $x_i = f(f(x_{i-2}))$. By expanding the expression and performing this transformation with the depth k, the feedback length and consequently the data initiation interval of linearized recurrent functions can be reduced by k times. For example, we consider an iterative process, described by the formula $x_{i+1} = ax_i + b$. Expressing x_i through x_{i-1} and substituting it into the original expression, we obtain $x_{i+1} = a(ax_{i-1} + b) + b$. Continuing substitutions for x_{i-2}, x_{i-3} and g.e., a formula expressing x_{i+1} in terms of any $x_{i-r} : x_{i+1} = a^{r-1}x_{i-r} + b + c$. As shown in [16], in the case of a linear function describing the iterative process, the coefficients can be simplified and calculated for any auto-substitution depth without intermediate steps. Thus, the latency of the iterative procedure can be reduced depending on the depth of the performed auto-substitution, which leads to a significant reduction in interval for linearized recurrent functions (Fig. 3).



(a) Computing structure before transformation



(b) Computing structure after transformation

Figure 3. Application of the "auto-substitution" method

The "auto-substitution" method leads to a certain increase in hardware costs for implementation of the computing structure.

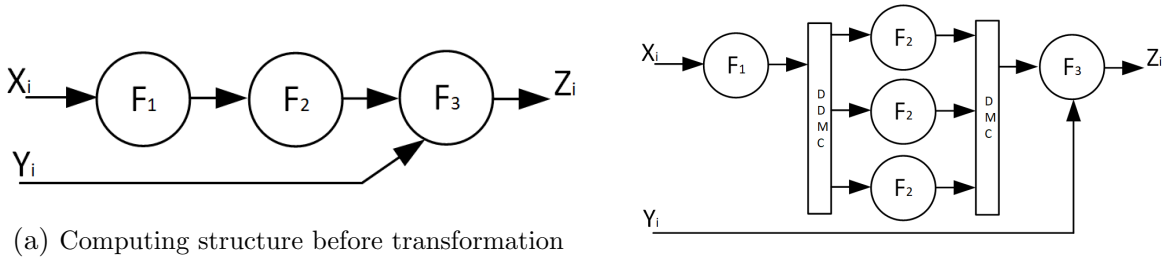
If a subgraph describes a non-linearized recurrent computational function, or if the structural implementation of the subgraph is inefficient (e.g., deep nesting of conditional branches or indirect addressing), then, according to [4], it is advisable to replace the computing structure of the subgraph with a procedural block. Procedural blocks are characterized by the number of clock cycles required to process a single operand, rather than the latency typical for pipeline blocks. In this case, the cadr structure can consist of a combination of structural and procedural blocks, which can result in varying data processing intensities between blocks and increased data initiation interval in the cadr structure.

To reduce the execution time of the procedural block, it is advisable to develop a macro-pipeline as proposed by V.M. Glushkov [14]. The essence of macro-pipeline development is scaling the procedural block by a factor L, where each procedural block sequentially receives data at intervals of $\frac{T_p}{L}$, with T_p being the execution time of the procedural block. Maximum efficiency

of the macro-pipeline is achieved when L equals the procedural block execution time T_p. In this case, the next data element from the stream is supplied to a free procedural block on every clock cycle, ensuring a dense data flow.

Implementation of a macro-pipeline does not require additional memory channels, but it does require additional hardware proportional to the value of L.

Let the execution time of subgraph F_2 be equal to T; then the data initiation interval in the computational structure (see Fig. 4a) should also be equal to T.



(a) Computing structure before transformation

(b) Computing structure after transformation

Figure 4. Application of the “macro-pipeline” method

If sufficient hardware resources are available to scale subgraph F_2 by a factor T, then the data output from subgraph F_1 can be fed to a free F_2 subgraph on every clock cycle, ensuring a dense data flow. The data flow for device F_3 is generated according to the order in which data is received at the F_2 subgraphs. The application of the discussed methods for converting the computing structure is implemented within a unified cadr structure synthesis algorithm, a generalized diagram of which is shown in Fig. 5.

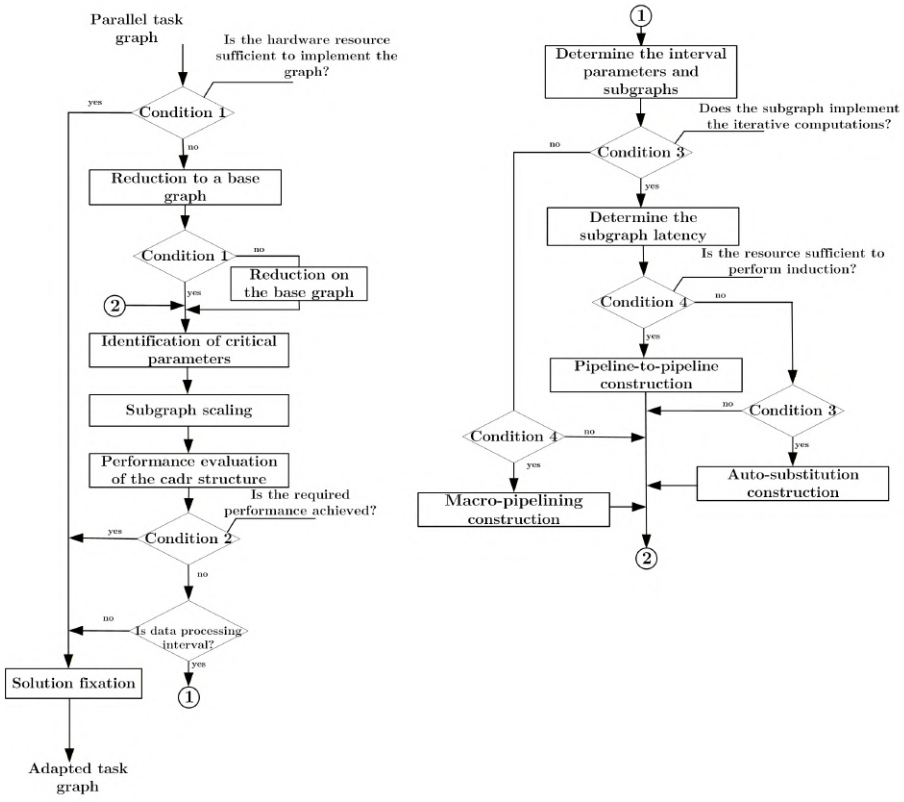


Figure 5. Diagram of the solution search for the cadr structure

The selection of the first transformation of the cadr structure depends on the type of critical resource (external memory channels or hardware resources). The subsequent transformations depend on the current hardware costs and performance parameters at each transformation stage. If the data initiation interval is in the synthesized cadr structure, then transformations must be performed to reduce it.

The combining of the solution search procedures (reduction and induction transformations) with performance improvement procedures into a single synthesis scheme – or using performance improvement procedures as an intermediate step – can lead to repeat the re-adjustment of previously determined scaling parameters of the cadr structure subgraphs and may exceed the limits of the permissible hardware resources of the target RCS. Therefore a two-pass organization of the synthesis is necessary for an efficient cadr structure of the parallel-pipeline program, which cannot be implemented within the current structure of the parallelizing compiler.

4. "Theseus" Parallelizing Compiler

To implement a new trajectory selection scheme at searching effective cadr structures, a new architecture of the "Theseus" parallelizing compiler is proposed, shown in Fig. 6.

The parallelizing compiler was extended with the "Ariadne" cadr structure synthesis program for parallel-pipeline programs and the interaction architecture between programs was modified to repeat invocation of programs for evaluating hardware costs at any stage of cadr structure synthesis and for generating different representations of the parallel-pipeline program depending on cadr structure parameters.

The "Ariadne" cadr structure synthesis program is designed to synthesize efficient implementations of cadr structure transformations for various application domains. In general, the operation of the "Ariadne" component consists of the following stages:

1. Generation of a universal parallel-pipeline program by the "Centaur" component.
2. Evaluation of the impact of scaling coefficients on the hardware costs of the computing structure.
3. Search for the efficient scaling parameters of the basic subgraphs of the task graph, with the calculation of the performance of the resulting solutions.
4. Performance improvement of the obtained solution.
5. Selection of the most efficient solution and generation of a specialized parallel-pipeline program based on it by the "Centaur" component.

The "Ariadne" operation begins with the creation of a universal parallel-pipeline program by the "Centaur" program. The universal parallel-pipeline program contains constants for describing the scaling coefficients of computing structures and memory channels, transformed computing structures into a parallel-pipeline form, as well as mechanisms for matching unbalanced data flows and synchronization elements with each other. As a rule, channel matching is required for adjacent subgraphs, and data synchronization for correct processing of data flows both within the subgraph and between them is caused by data displacement in the channels.

The number of switching structures can be large, and their analysis and processing require significant CPU time during the program translation. However, it is a necessary to assess the impact of scaling coefficients on hardware costs of computing structure, memory channels, and the data initiation interval in the channels. Based on the obtained coefficients, a search is performed for possible solutions (a set of generated values for the program scaling coefficients),

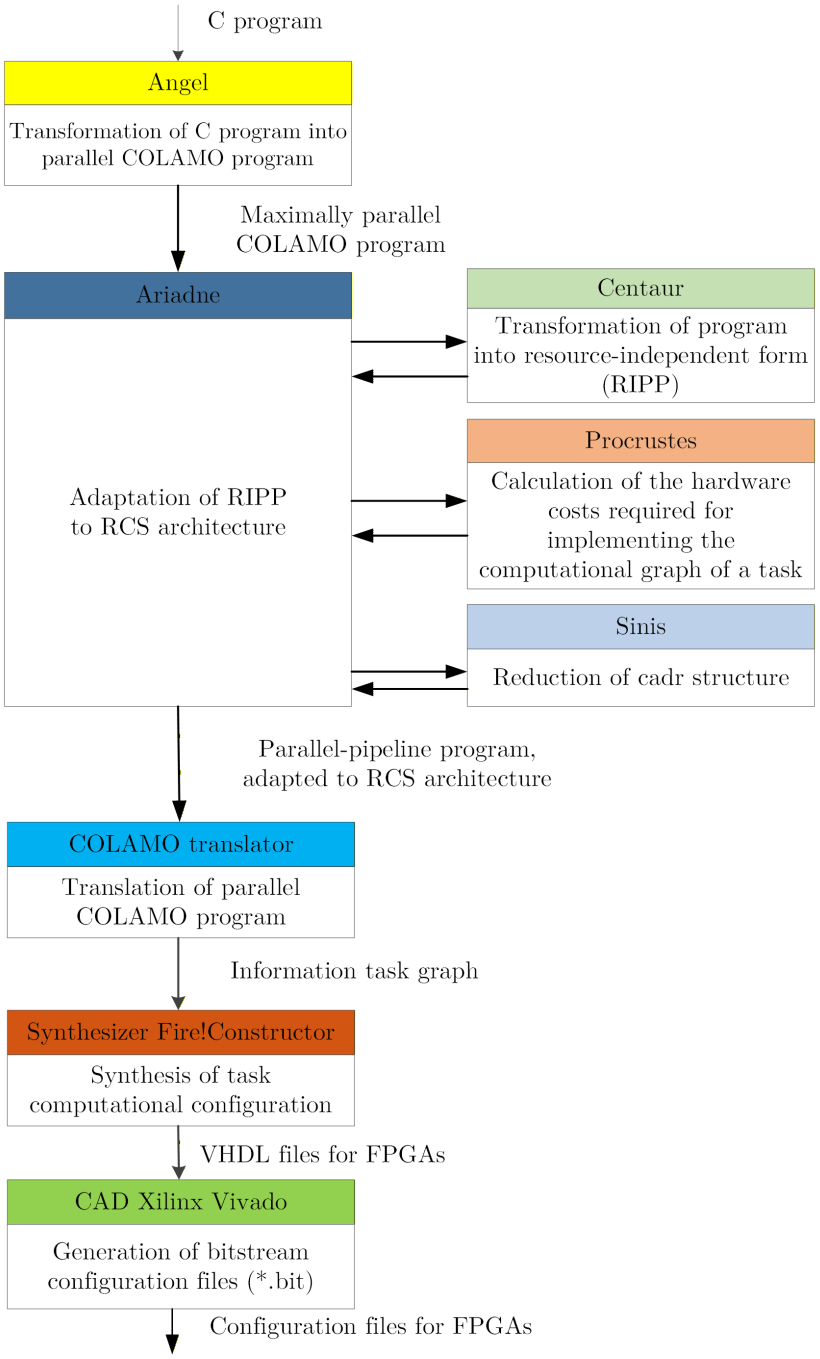


Figure 6. Parallelizing compiler architecture

the cadr structures of which satisfy with the available hardware resources of the target RCS. The resulting combinations represent potential parameters for creating the cadr structure.

Next, a performance evaluation is performed for each result. If the obtained results do not respond the given performance, then transformations are performed to increase the productivity of the cadr structure.

For each obtained solution, a sequential analysis of the scaling (reduction) coefficients is performed, corresponding to subgraphs in which input and output data flow intensity does not correspond to each other. For each subgraph, data dependencies are analyzed both within the subgraph and within adjacent subgraphs, and an efficient transformation method is determined

(pipeline-to-pipeline, auto-substitution, or macro-pipeline). The transformation begins with the subgraph exhibiting the maximum data initiation interval. After completing transformations for the current subgraph, the interval of the cadr structure is analyzed before transfer to the next subgraph.

Let us consider an example to illustrate the synthesis process of a parallel-pipeline program. Suppose that for a parallel program, the information graph of which is given in Fig. 7, the channels of external memory are a critical resource. Let us assume that the total number of all external memory channels should not exceed 10.

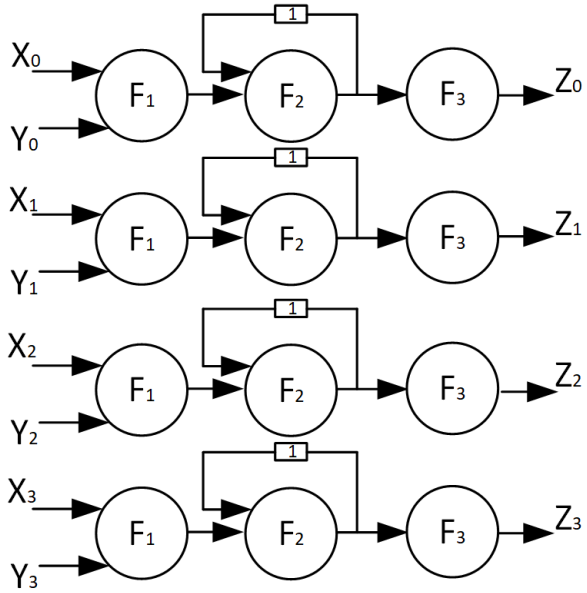
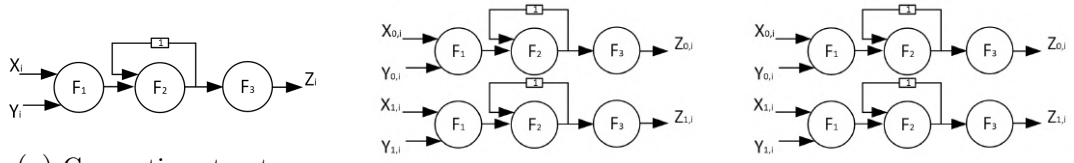


Figure 7. Computing structure of the parallel program

As a result of the synthesis, three variants of cadr structures will be obtained (Fig. 8), ensuring balanced data processing in data channels.



(a) Computing structure before transformation

(b) Computing structure after transformation

(c) Computing structure after transformation

Figure 8. Variants of cadr structures for different scaling factors

If all obtained variants of the cadr structures correspond to the specified performance level, the final solution will be the cadr structure with the maximum performance. In this case, the cadr structure given in Fig. 8c is obtained.

If the obtained cadr structures do not provide the required performance, a search for performance-improving solutions will be launched for each cadr structure. All solutions in Fig. 8 have iterative dependencies at calculating the next data input to the subgraph F_2. This indicates the necessity to organize the data initiation interval for input to subgraph F_2. The pipeline-

to-pipeline cadr transformation method can be used to increase the performance in the cadr structures with iterative dependencies.

Let us consider the pipeline-to-pipeline transformation using the cadr structure given in Fig. 8a as an example. Let the data initiation interval of the functional device correspond to its latency and be equal to 4 cycles. Then all subgraphs and external memory channels should be scaled according to the latency of the subgraph F_2, and markings in the graph is performed defining the application field of the pipeline-to-pipeline method. The cadr structure with a scaling factor of 4 and the logical elements of the method application are given in Fig. 9.

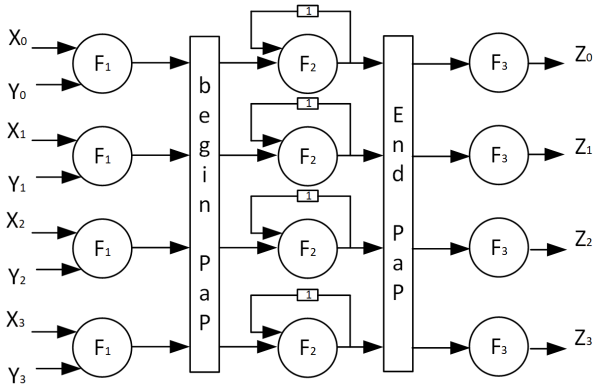


Figure 9. Cadr structure synthesized by the “Ariadna” program

The “Begin PaP” and “End PaP” serve as syntactic brackets for describing subgraphs for which the pipeline-to-pipeline transformations are applied, and they appear as directives in the parallel-pipeline program. Based on the placement of the “Begin PaP” and “End PaP” blocks in the information task graph, the Fire!Constructor synthesizer (Fig. 7) will transform the cadr structure into a single subgraph F_2 with a set of registers in the feedback loop. The modified cadr structure produced by the Fire!Constructor synthesizer is given in Fig. 10.

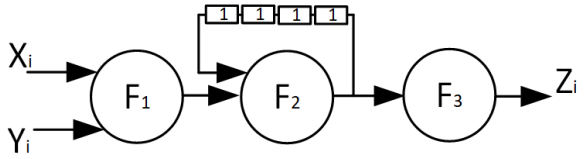


Figure 10. Modified graph using the pipeline-to-pipeline method

The scaling by a factor of 4 did not lead to an excess of elements in the memory channels and ensures correct data delivery to the computing structure. Note that the application of this approach to the cadr structures in Fig. 8b, Fig. 8c will lead to an excess of the number of elements in channels X, Y, and Z, and incorrect effect at data processing in the computing structure. Such transformation variants of the cadr structure will be discarded and not be considered as potential cadr structure options.

The “auto-substitution” method can be applied for subgraph F_2. In this case, additional solution variants for computing structure in Fig. 8 will be obtained, among which the solution with the maximum performance will be selected.

The stages of transforming the cadr structure of the input parallel program are not fixed and are largely determined by the information task structure and the type of information

dependencies. Thus, a number of symbolic processing problems are characterized by the absence of iterations, but the number of layers is extremely large. Therefore, transformations of the iterative structure for such problems are meaningless.

At the same time, in mathematical physics problems, the rational implementation of the iterative component of the cadr structure is crucial for the efficiency of the overall solution. In addition, the RCS hardware resource may not be sufficient to implement even the basic subgraph in some cases. This requires the application of the reduction transformations described in [8] to the basic subgraph, but it allows to subsequently use scaling methods and algorithms to the reduced frame structure to reduce the time for problem solution.

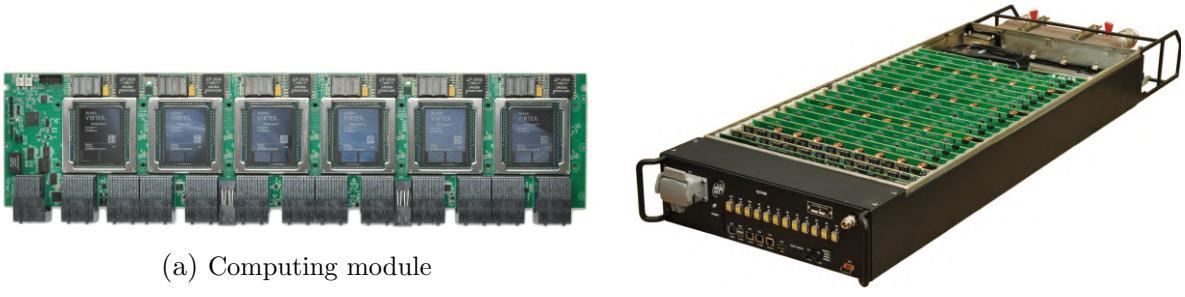
Next, the "Centaur" component generates a specialized pipeline-parallel program, based on the selected solution. The specialized application contains data synchronization and alignment fragments that are strictly necessary, based on the obtained scaling coefficient values. The output of the "Ariadna" program is a pipeline-parallel program adapted to the target architecture and configuration of the reconfigurable computer system, significantly simplified compared to a similar program produced by the previous version of the "Theseus" compiler [7]. This leads to a significant reduction in the total synthesis time of FPGA configuration files of multichip RCS.

In general, the efficiency of the generated pipeline-parallel program depends on the specifics of the information task structure and data flows, which may differ significantly in different task classes, as well as on the available hardware resources of the target reconfigurable computer system.

5. Results of Parallel-pipeline Program Synthesis

The analysis of the effectiveness of the proposed methods for performance increasing of parallel-pipelined programs, created for reconfigurable computer systems, was performed for a number of model problems. They were tested for operability by launching on the high-performance reconfigurable computer system "Arktur" (Fig. 11).

The "Arktur" computing unit, with a performance of 230 TFLOPS, contains 16 computational modules, 96 Virtex UltraScale+ XCVU37P FPGAs, 4GB of RAM, and 768GB of HBM memory.



(a) Computing module

(b) Computing unit

Figure 11. The RCS hardware for testing "Arktur" applications

The "Arktur" RCS is a universal reconfigurable computer system developed by the "Scientific Research Center of Supercomputers and Neurocomputers" (Taganrog, Russia). It is designed to solve problems in various domains: mathematical physics, digital signal processing, symbolic processing, and artificial intelligence.

Two classes of problems have been selected as model problems: linear algebra and symbolic processing. The implementation of problems from different domains makes it possible to verify the effectiveness and correctness of the modified methodology for synthesis the cadr structures and performance-enhancing methods, implemented in the "Ariadna" application. The effectiveness of the synthesis of cadr structures and the method of increasing productivity "auto-substitution" have been tested on linear algebra problems: the problem of solving systems of linear algebraic equations (SLA) by the Gauss method and the Gauss-Seidel method.

The effectiveness of cadr structure synthesis and the performance-enhancement method "autosubstitution" were tested for linear algebra problems: the specifically solving systems of linear algebraic equations (SLAEs) using the Gauss method and the Gauss-Seidel method.

Note that the Vivado HLS-synthesized solution for the Gauss model problem contains a single iterative step, compared to 3512 steps at the problem translation using the "Theseus" parallelizing system. Using manual code annotation with #pragma directives in Vivado HLS allowed obtaining a solution for two iterative steps of the forward elimination algorithm. It is still significantly lower than by the parallelizing compiler. It can be noted that there is a slight decrease in the number of pipeline steps at synthesizing a solution using the performance enhancement methods and the "Theseus" parallelizing compiler, compared to the creation of a specialized solution by application programmers – 4210.

The efficiency of SLAE solution by the Gauss-Seidel method depends on the available hardware resources; insufficient resources lead to feedback loops in the computing structure. At the same time, calculations in feedback have associative properties, which allows using the "auto-substitution" performance-enhancing method. The solution, obtained as a result of using the "auto-substitution" method, contains 5376 data processing steps and the data initiation interval of 21. This is inferior to the specialized solution, obtained by application programmers – 6563, but significantly exceeds the data initiation interval of the existing version of the compiler by 1.7 times.

The performance results for the linear algebra model problems using the parallelizing compiler, application vendors, and the HLS compiler (Vivado HLS) is presented in the Tab. 1. Performance was defined as the number of basic graphs in synthesized solutions and the data initiation interval value.

Table 1. Performance results for model problems in mathematical physics

Researches	Gauss	Gauss–Seidel
Porting time, sec	5073	7752
	Number of graphs / interval	
The existing version of the "Theseus" compiler	3512/1	5380/36
Application vendors (application of performance increasing methods)	3512/1	5376/21
Application vendors (specialized solution)	4210/1	6563/17
HLS compilers	2	-
Level of real performance	0.68	0.81

The effectiveness of using methods for converting cadr structures has been evaluated on symbolic processing problems: the encryption problem, the SSL authentication problem, and

the password hashing problem in the FreeBase database a macro conveyor and a pipeline-to-pipeline.

The DES encryption problem showed that the number of implemented subgraphs by the parallelizing compilers coincided, as the cadr structure did not contain procedural blocks or feedback loops. For the synthesis of the SSL authentication cadr structure, based on the MD5 and RC4 algorithms, the "macro-pipeline" performance-enhancing method was used. The scaling degree of the procedural subgraph implementing RC4 by the "Theseus" parallelizing compiler was 1253 subgraphs in a single pipeline, compared to 970 subgraphs implemented by application vendors. This indicates to a more efficient implementation of procedural calculations in the RC4 block. In contrast, Vivado HLS implemented only a single subgraph. The data initiation interval in the first two cases was 1, the Vivado HLS solution has 2100 cycles.

Due to the insufficient available hardware resources, the feedback loop with the latency of 12340 cycles has been implemented in the cadr structure of the password hashing problem in the FreeBase (MD5-Crypt) database. As a result of the problem implementation by application vendors using the "pipeline-to-pipeline" performance increasing method, the latency of the feedback subgraph implemented by the parallelizing compiler was reduced to 8870 cycles, which is 1.6 times higher than that achieved by application vendors (5544 cycles). This leads to the decrease of specific performance.

The results obtained for symbolic processing problems are presented in the Tab. 2.

Table 2. Performance results for model symbolic processing problems

Researches	DES	MD5-RC4	MD5-Crypt
Porting time, sec	28952	53779	36095
		Number of graphs / interval	
The existing version of the "Theseus" compiler	1335	106/1253	112/132056
Application vendors (application of performance increasing methods)	1335	7518 (6 pipe)/1	26210 (3 pipe)/24
Application vendors (specialized solution)	1740	6790 (7 pipe)/1	22176 (4 pipe)/24
HLS compilers	1	1/2100	-
Level of real performance	0.76	0.9	0.84

Solving MD5-RC4 and MD5-Crypt problems by application vendors and using pipeline-in-pipeline and macroconveyor methods and specialized solutions shows that the application vendors perform more efficient implementation of subgraphs in feedback, despite achieving the same data flow rate in the cadr structure. This results in lower latency of all feedback and reduced hardware costs. The released resource was directed by application vendors to implement an additional pipeline in the computing structure, which increased specific performance by 1.16 and 1.33 times, respectively.

Researches performed on model performance problems of synthesized solutions, presented in Tab. 1 and Tab. 2, have shown that the application of the presented methods to increase the productivity of cadr structures the pipeline-to-pipeline, auto-substitution and macro-pipeline demonstrates the productivity increasing up to 234 times on model problems by various classes, depending on the type and computing structure of the problem.

Comparing the results obtained for model problems using the previous version of the parallelizing compiler, the new compiler version, and application vendors demonstrates the significant advantage of the new version of the "Theseus" parallelizing compiler.

Conclusion

The differences between the proposed version of the "Theseus" automatic parallelizing compiler and the most well-known Xilinx Vivado HLS and Vitis HLS compilers are fully automatic transformation of the input program without requiring manual code annotations (e.g., #pragma directives), support for complex parallelism forms (macro-pipeline, pipeline-to-pipeline) unlike the previous version of the "Theseus" compiler, automatic scaling of solution to a given RCS resources and synchronization of data and control signals.

The "Theseus" compiler will significantly reduce the conversion time of sequential programs to parallel-pipeline solutions for reconfigurable computer systems with multiple FPGA modules connected via a spatial communication network, while ensuring a guaranteed level of real performance. Due to the efficient organization of calculations and rational use of the available FPGA hardware resource, the "Theseus" compiler will provide significantly higher real performance of the reconfigurable computer system compared to the similar compilers.

The new version of the "Theseus" parallelizing compiler will enable to significantly expand the class of problems to be solved. Thanks to the efficient use of the target reconfigurable computer systems hardware resources and the transformation of the CADR structure into effective computational forms (pipelinetopipeline, autosubstitution, and macropipeline), the systems actual performance is guaranteed to reach at least 60% of its peak capacity.

Future research directions for the "Theseus" compiler include expanding the classes and ranges of solving problems and reducing requirements on C-code, in particular, support for dynamic memory allocation for arrays, data loading/unloading, and support of structures and unions.

Acknowledgements

The study was supported by the Russian Science Foundation grant No. 26-11-00209, `https://rscf.ru/project/26-11-00209/`.

References

1. Guzik, V.F., Kalyaev, I.A., Levin, I.I.: Reconfigurable computer systems. SfedU Publishing, Taganrog, 2016. 472 p.

2. Hauck, S., DeHon, A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann, Burlington, MA (2008).

3. Nane, R., *et al.*: A Survey and Evaluation of FPGA High-Level Synthesis Tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(10), 1591–1604 (2016). `https://doi.org/10.1109/TCAD.2015.2513673`

4. Numan, M.W., Phillips, B.J., Puddy, G.S., Falkner, K.: Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and

Toolchains. IEEE Access 8, 174692–174722 (2020). https://doi.org/10.1109/ACCESS.2020.3024098

5. Kamkin, A.S., Chupilko, M.M., Lebedev, M.S., *et al.*: Comparison of High-Level Synthesis and Hardware Construction Tools. Trudy ISP RAN/Proc. ISP RAS 34(5), 7–22 (2022). https://doi.org/10.15514/ISPRAS-2022-34(5)-1

6. Liang, Y., Rupnow K., Li Y., *et al.*: High-Level Synthesis: Productivity, Performance, and Software Constraints. Journal of Electrical and Computer Engineering 2012, 649057, (2012). https://doi.org/10.1155/2012/649057.

7. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: High-Level Synthesis Toolchain "Theseus" for Multichip Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations 10(2), 18–31 (2023). https://doi.org/10.14529/jsfi230202

8. Dordopulo, A.I., Levin, I.I.: Performance Reduction For Automatic Development of Parallel Applications For Reconfigurable Computer Systems. Supercomputing Frontiers and Innovations 7(2), 4–23 (2020). https://doi.org/10.14529/jsfi200201

9. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A. High-Level Synthesis Software for Multicrystal Reconfigurable Computing Systems. Bulletin of the South Ural State University. Series: Computational Mathematics and Computer Science 11(3), 5–21 (2022). https://doi.org/10.14529/cmse220301

10. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: High-Level Synthesis Strategies for Multicore Reconfigurable Computing Systems. In: Voevodin Vl.V. (eds.) Supercomputer Days in Russia, Proceedings of the International Conference, Moscow, September 29-30, 2025. pp. 195–206. MAKS Press, Moscow (2025). https://doi.org/10.29003/m4750.978-5-317-07451-7

11. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A.: Software Complex for High-Level Synthesis of Configuration Files for Multicrystal Reconfigurable Computing Systems. In: Parallel Computing Technologies (PaCT'2023): Short Papers and Posters. Proc. of the 17th All-Russian Scientific Conference with Int. Participation, Saint Petersburg, March 28-30, 2023. pp. 133–142. South Ural State University Publishing Center, Chelyabinsk (2023).

12. Ivanov, A.I., Konovalchik, P.M.: Methods of organizing parallel-pipeline calculations for solving computationally intensive problems. Information Technologies 12, 38–43 (2004).

13. Kotlyarov, A.S., Levin, I.I.: Processing of Network Data Streams in Reconfigurable Computing Systems, Izvestiya SFedU. Engineering Sciences 2(204), 48–56 (2019).

14. Glushkov, V.M.: On Macro-Pipeline Computations. Computational Processes and Systems. Issue 1. pp. 61–70. Nauka, Moscow (1983).

15. Koughi, P.M. Architecture of Pipelined Computers. Radio and Communications, Moscow (1985).

16. Mikhailov, D.V., Levin, I.I.: A method for determining the stability of a recursive adaptive pipelined filter. Radio Engineering 3(6), 5–11 (2020). https://doi.org/10.18127/j00338486-202003(06)-01

DOI: 10.14529/jsfi260206

Optimizing Synthesizer of Parallel-pipelined Programs for Reconfigurable Computer Systems

Andrey A. Gulenok[1]***, Evgeniy A. Semernikov***[1]

Existing synthesizers (software tools that convert models implemented in hardware description languages into a list of logic gate connections) for field-programmable gate arrays (FPGAs), such as Vivado (Xilinx), Quartus II (Intel), and Libero (Microsemi), are designed to implement applications within a single chip. These tools perform optimization of the computational components of algorithms only at the level of logical primitives. Such optimization is inherently local and, therefore, cannot substantially reduce the hardware resources utilized on an FPGA. Moreover, conventional FPGA synthesizers do not analyze the implemented algorithms with respect to potential computational redundancies or inefficient design realizations. To date, the only synthesizer for multichip reconfigurable computer systems, developed at the Scientific Research Center of Supercomputers and Neurocomputers, provides automated mapping of the algorithms information graph onto multiple FPGA devices, as well as inter-chip routing and synchronization of control and data flows within the reconfigurable computing system. The novel version of the optimizing multichip synthesizer for parallel-pipelined programs for reconfigurable computer systems, presented in this paper, incorporates a library and a general algorithm for information-equivalent transformations, along with an original method of nested auto-substitutions for recursive expressions. These features enable a substantial and automated improvement in the performance of generally specified algorithms when implemented on reconfigurable computing systems.

Keywords: synthesis of parallel-pipelined programs, information graph, optimizing synthesizer, reconfigurable computing system.

Introduction

Reconfigurable computer systems (RCS), whose primary hardware resources are field-programmable gate arrays (FPGAs) integrated into a unified multichip computational device, are currently being effectively applied to solve problems across various fields of science and engineering [1, 2]. The high achieved performance and energy efficiency of RCS, in comparison with traditional cluster systems, demonstrate their competitiveness in the domain of high-performance computing [3].

Existing FPGA synthesis tools, such as Vivado (Xilinx), Quartus II (Intel), and Libero (Microsemi), are designed to implement applications within a single-chip or across a set of independent FPGAs. If efficient problem implementation requires a large number of interconnected FPGAs, the use of single-chip synthesis tools becomes labor-intensive, as developers must manually partition program fragments across individual devices and ensure proper alignment of data flows both within and between FPGAs. It should be noted that automatic synthesis tools exhibit low efficiency with a high density of chip filling, as exemplified by the Vivado synthesis tool [4]. It leads to reducing the effective performance of RCS or increasing the development time for FPGA-based solutions.

The only synthesis tool for multichip RCS, developed at the Scientific Research Center of Supercomputers and Neurocomputers, provides automatic mapping of the algorithms information graph (IG) across multiple FPGA devices, inter-chip routing, and synchronization of control and data flows in RCS [5]. The information graph of an algorithm for solving the problem is

[1] "Scientific Research Center of Supercomputers and Neurocomputers" Co Ltd ("SRC SC & NC" Co Ltd), Taganrog, Russian Federation

defined as a directed graph in which vertices correspond to operations, data sources, and data sinks, while edges represent data dependencies between them [6].

This paper presents a new version of the multichip synthesis tool and describes the implemented methods of information-equivalent transformations of the algorithms IG. These methods take into account the properties of associative and distributive operations, as well as the regularity of graph structures. Such transformations make it possible to reduce the number of operations in many times for some problems and automatically obtain efficient parallel-pipeline solutions that are often not obvious or even known to the vendor at application development.

Section 1 provides a brief overview of existing synthesis tools, high-level synthesis (HLS) compilers, and the multichip synthesis tool, as well as their application in the development of programs for multichip RCS. Section 2 presents a general algorithm for information-equivalent graph transformations developed for the new version of the optimizing synthesizer, which has been verified for problems from three distinct application domains. Section 3 introduces a method of nested auto-substitution for pipelining calculations of recursive expressions. Section 4 presents the results obtained using the optimizing synthesizer for a number of applied problems. Finally, the obtained results are analyzed in the conclusion.

1. Development of Efficient Programs for Reconfigurable Computer Systems

Currently, standard synthesis tools, such as Xilinx Vivado or Intel Quartus Prime, are used to obtain highly efficient implementations of RCS solutions. Due to the use of low-level digital circuit description languages such as VHDL or Verilog, the vendor can detail the design of the computing structure of a parallel-pipeline program and achieve a high device utilization taking into account the FPGA architectural features.

However, the development time for an efficient solution on a multichip RCS using this approach typically ranges from several months to a year, even for highly qualified hardware engineers. This is due both to the use of low-level design tools and to the necessity of developing multiple projects simultaneously, each of which implements a fragment of the overall program mapped onto a separate FPGA device.

The use of high-level synthesis (HLS) compilers [7], such as Vivado HLS or Vitis, reduces development time by automatically generating complex functional units (IP-cores) from C-language program fragments. However, this approach does not guarantee efficient implementation on RCS for arbitrary C programs. Moreover, issues related to scaling and integration of IP-cores across a multichip RCS remain within the vendors responsibility.

At the Scientific Research Center of Supercomputers and Neurocomputers (Taganrog), a Fire!Constructor multichip synthesis tool is being developed and improved to relieve vendors of a number of routine problems, associated with designing solutions across multiple FPGAs. The inputs to this synthesizer are the IG of a parallel-pipelined program and a description of the RCS architecture. The synthesizer automatically performs automatic layout of the IG into non-overlapping subgraphs, each of which is subsequently located on a separate FPGA device.

The objective of the partition and placement stages is to create favorable conditions for subsequent inter-chip routing. However, even “optimal” solutions of partition and placement problems according to selected criteria cannot guarantee successful routing. Therefore, these

tasks are solved repeatedly using different algorithms, which increaces the probability of successful routing in Fire!Constructor.

As a rule, the operation time of graph partition algorithms has a polynomial or even exponential dependence on the number of vertices in the graph. Therefore, a multilevel partition scheme is employed for large-scale graphs (from several thousand and more) in the synthesizer Fire!Constructor [8], which iteratively contracts pairs of adjacent vertices (Coarsening Phase). Afterward, partition and placement tasks are performed by a graph refinement stage in which the contraction process is iteratively reversed (Uncoarsening Phase). During inter-chip routing, the synthesizer assigns external connections of subgraphs to the available physical communication links between FPGAs in the RCS.

Additionally, Fire!Constructor performs synchronization of control and data flows by inserting delays to align input streams at each pipeline stage of the computational structure. The synchronization problem is solved on the IG by assigning each vertex a latency value corresponding to its pipeline stage. Then, for each incoming edge, the cumulative signal latency from data sources is calculated. Based on these values, flow alignment is achieved by inserting delays into leading signals.

The resulting synchronization subsystem can consume FPGA resources comparable to those required for the computational logic itself. Three types of FPGA resources are used to implement delay elements: flip-flops, programmable delay elements (MLUTs), and block memory (BRAM). Therefore, Fire!Constructor incorporates a set of procedures aimed at reducing both the total number of delays and the utilization of critical hardware resources allocated for synchronization [9].

Using single-chip synthesis tools, all of the above tasks must be handled manually by the vendor. Thus, the use of a multichip synthesizer such as Fire!Constructor significantly reduces development time for RCS-based solutions.

At the final stage of Fire!Constructor synthesis, a single-chip project for the Xilinx Vivado environment is generated for each FPGA involved in the multichip RCS. Each single-chip project contains the corresponding fragment of the circuit-level implementation of the target algorithm, which is subsequently compiled into FPGA configuration files.

Existing FPGA synthesis tools perform optimizations primarily at the level of logical primitives. However, such transformations are local in nature, including the removal of unused elements or simplification of constant logic. The computing structure of the implemented algorithm largely remains as specified by the vendor and may exhibit significant redundancy or inefficiency. Therefore, the efficiency of the final implementation depends heavily on the vendors expertise in parallel-pipelined computing.

A new version of the multichip synthesizer is being developed. In addition to the previously implemented functions for displaying a parallel-pipelined program on the RCS, it automatically identifies fragments of the IG program that are ineffective from the point of view of their parallel-pipelined implementation on the RCS. Analyzing and upgrading the identified fragments through information-equivalent transformations, the synthesizer can significantly increase the real or specific performance of the RCS compared to the execution of the original program.

2. General Algorithm of Information-Equivalent Transformations on a Graph

The structure of a parallel-pipelined program is transmitted to a multichip synthesizer in the form of IG. It is independent of the source programming language and, consequently, is not overloaded by unique syntactic constructions (such as switch-case, union, etc.). Note that the same IG can correspond to multiple program texts (including written in different languages), differing both in the order of independent operators and syntactic constructions, which determine the sequence of operators. This contrasts with program text, where related calculations can be placed in different lines of code, procedures, or even program modules. These IG features described above enable algorithmic simplification of structural analysis procedures, identifying redundant and inefficient structures and procedures for information-equivalent transformations, unlike the program text.

The term "optimizing compiler" refers to the implementation of various algorithms to generate a more optimal program code while preserving its functionality. Similarly, the new developed version of the multichip synthesizer is referred to as an "optimizing synthesizer", since it performs transformations analogous to those of optimizing compilers, but applied to configuration files for multiple FPGAs.

A library of information-equivalent transformations is realized in the optimizing synthesizer. It includes such transformations on individual vertices as calculating expressions for which all operators are constants, exclusion of multiplications and logical "AND" operations with the constant "1" and additions and logical "OR" with a constant "0", and replacing the multiplication or division of an integer by a constant equal to 2, to an integer extent, by shifts (since due to the FPGA architectural features, the shift is implemented in it by bit recomputation, which does not take up hardware resources), etc.

The library also includes transformations for groups of operations which take into account the associativity and distributivity properties. Based on the transformation method of homogeneous IG from sequential to parallel-pipelined form [10], the optimizing synthesizer implements a procedure for graph fragment transformation consisting of sequential associative operations of the same type (e.g., addition, multiplication, logical AND) into a pyramidal form (Fig. 1) or a series of intermediate forms.

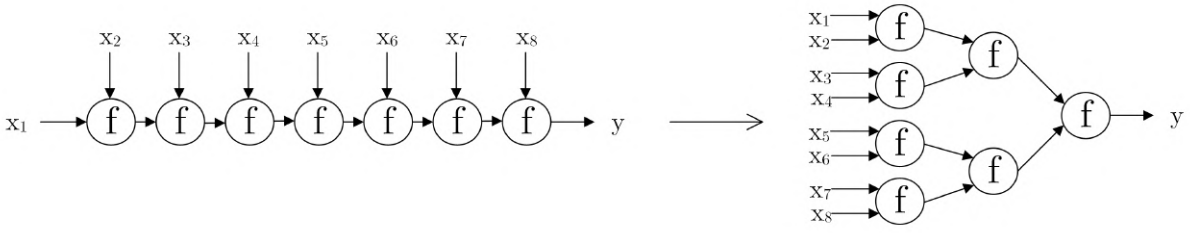


Figure 1. Transformation of a homogeneous IG from sequential to parallel-pipelined form

Note that such transformation does not reduce the number of operations, but can be performed to decrease the latency of the IG fragment. The optimizing synthesizer can more easily analyze input operands in the pyramidal form. If they are constants, then identify functional dependencies between them. Two smaller pyramids can be obtained, which can be further transmitted for analysis, excluding the closing vertex from such a fragment.

Based on the transformation method of heterogeneous IG from the sequential to parallel-pipelined form [10], the optimizing synthesizer also implements transformations of graph fragments the analogues of which are putting the common multiplier in parentheses and opening parentheses in mathematics (Fig. 2).

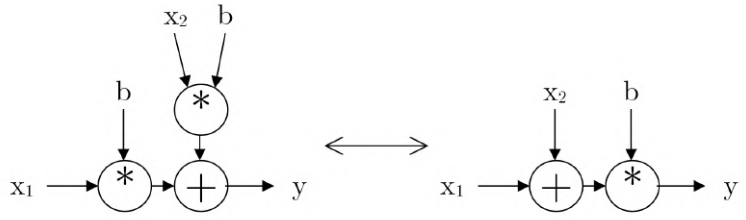


Figure 2. Transformations involving distributive operations

The IG transformation, similar to opening parentheses, is performed to reduce either the latency of the IG fragment or the number of operations. The transformation analogous to expansion may reduce the number of operations when expressions contain identical operands or constants. After opening parentheses, similar terms are combined, and part of the expression with constants can be calculated in advance.

Note that all these transformations, as well as transformations in single-chip synthesizers, are local in nature. Their implementation violates the regularity of the IG structure of algorithm to solve the problem, although it reduces the hardware resource. This complicates its analysis for redundancy and inefficiency. Therefore, in the optimizing synthesizer, the first stage involves the search and reduction of calculation redundancy in initial graph structure and the transformation of inefficient fragments to more efficient ones from the point of view of the procedural pipeline organization of calculations.

A general algorithm has been developed to perform information-equivalent transformations on graphs, representing solutions to various problems, including those from different application domains:

1. The initial graph G is placed into the array Mas of processed graphs.

2. All graphs in Mas are transformed into structures that can be decomposed into pairs of equivalent subgraphs solving identical subtasks of the original problem, along with "merge" subgraphs that combine the results of these pairs.

3. If such decomposition is not possible, proceed to step 9.

4. All processed graphs are partitioned into subgraphs.

5. Isomorphic subgraphs (with respect to operations) are analyzed, and computational redundancy is eliminated.

6. Information-equivalent transformations are performed both in computational subgraphs and in "merge" subgraphs.

7. The array Mas is cleared and filled with the resulting subgraphs.

8. Return to step 2.

9. End.

This algorithm is based on a divide-and-conquer approach, in which the original IG of the problem is divided into two subgraphs, each of which implements the processing of its own half of the original input data. These subgraphs are checked for redundancy and transformed in order to reduce it. Each of resulting subgraphs becomes a new graph for analysis, and so on until not

obtained subgraphs of minimal dimension that cannot be separated. The information-equivalent transformations described in step 6 are specific for each type of initial IG. The transformations described in step 2, as well as the methods for partitioning graphs into subgraphs in step 4, also depend on the original graph type.

Let us consider the application of a general algorithm for functionally equivalent transformations using the example of IG algorithms for problems in various subject areas.

2.1. Transformation of an Information Graph via Vandermonde Matrix Multiplication by a Vector of Unknowns

We demonstrate how an optimizing synthesizer automatically performs an IG transformation of the discrete Fourier transform (DFT) into an IG similar to the fast Fourier transform (FFT) using a general algorithm.

The DFT problem is solved through the multiplication of the complex Vandermonde matrix W of dimension $N \times N$ by a vector X. Figure 3 shows a fragment of the IG of an 8-point DFT algorithm for calculation of the elements of the output vector Y with indices 0 and 4.

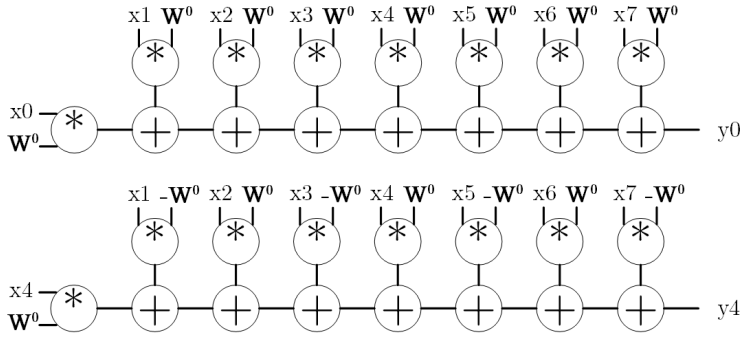


Figure 3. Fragment of the DFT IG

The elements of the complex Vandermonde matrix for the DFT have a functional dependence, using which it is possible to reduce the number of operations using information-equivalent transformations on IG. According to step 1 of the general algorithm, the entire DFT IG is loaded into an array of processed graphs. If the matrix-vector multiplication algorithm uses a sequential for-loop, the adders on the graph are connected sequentially (Fig. 3). Further, in accordance with step 2, the original graph is transformed so that the structure of the adders takes on a pyramidal form (Fig. 4).

According to step 4, in this graph fragment, the two original pyramids of adders are split into four equivalent subgraphs $G1$, $G2$, $G3$ and $G4$, as shown in Fig. 4. Each of them is a pyramid of three adders and four multipliers, and two "merge" subgraphs $G5$ and $G6$ containing one adder each.

Figure 4 shows that the sequence of multipliers differs from the original (Fig. 3). Since the sequence of addition of the multiplication results can be any, due to the associativity of the addition operation, the input arcs of the pyramid of adders can be reversed. It is performed in the implementation of step 5 in the analysis of isomorphic operations of subgraphs.

Subgraph $G1$ is isomorphic to $G3$ by vertices, with identical input data. Subgraph $G2$ is isomorphic to $G4$ by vertices, but the input data (constants) differ. Their difference can be expressed as $W_{G2} = (-1) \cdot W_{G4}$. Since they have a common functional dependency, using the

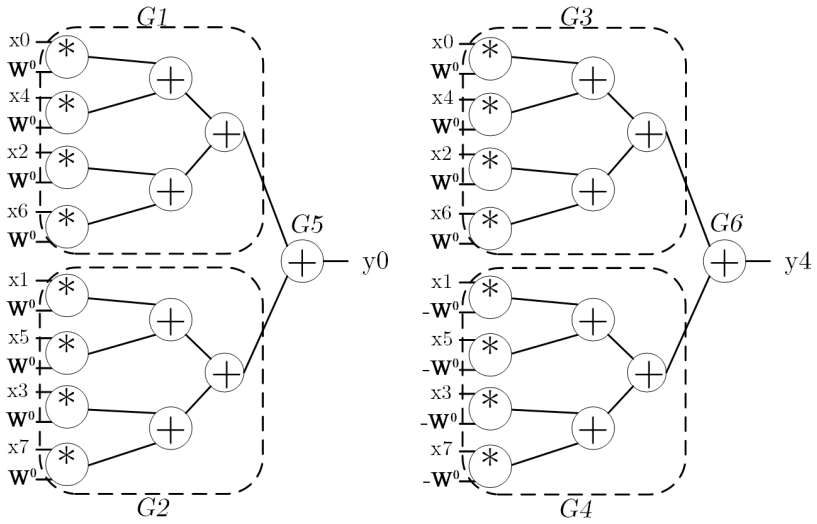


Figure 4. Fragment of the transformed DFT graph

corresponding information-equivalent transformation in the step 6, the graph $G4$ can be made such that the input data will coincide with graph $G2$, namely, by making a transformation similar to taking the common denominator out of brackets. To do this, multiply the coefficients W_{G4} by "-1" and add the multiplication by "-1" after the last adder of the $G4$ subgraph (we will add this multiplication to the $G6$ merge subgraph).

By combining the subgraphs $G1$ with $G3$ and $G2$ with $G4$, we obtain the graph structure shown in Fig. 5.

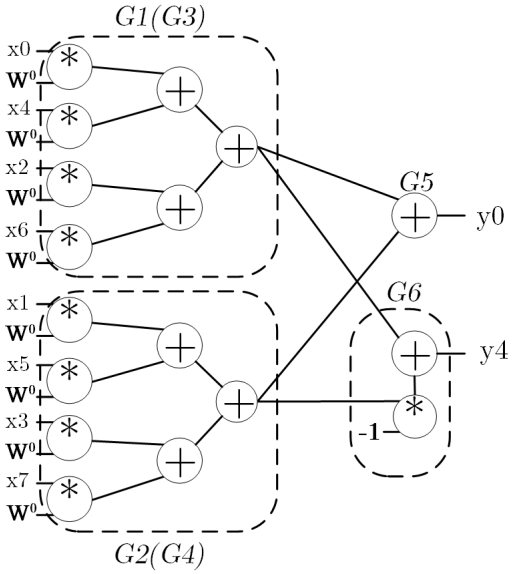


Figure 5. Fragment of the transformed DFT graph

As given in Fig. 5, the addition and multiplication by "-1" can be replaced by subtraction. However, such transformation can generally disrupt the regularity of the IG structure. Therefore, such transformations are performed in the optimizing synthesizer after executing the general information-equivalent transformation algorithm.

For subgraphs that calculate other values of the output vector, similar subgraphs are generated, differing only in the constants of the multipliers, corresponding to their functional dependencies. After step 7, the array of processed graphs includes subgraphs $G1(G3)$, $G2(G4)$, and similar subgraphs for computing other output values.

By iteratively dividing processed graphs into pairs of smaller subgraphs, the algorithm terminates once the minimal subgraphs containing a single addition are processed. Consequently, the original DFT IG is automatically transformed into an IG analogous to the FFT graph with computational complexity $O(N) = \frac{3}{2}N \cdot \log_2 N$. The original DFT IG has the complexity $O(N) = N \cdot (2N - 1)$.

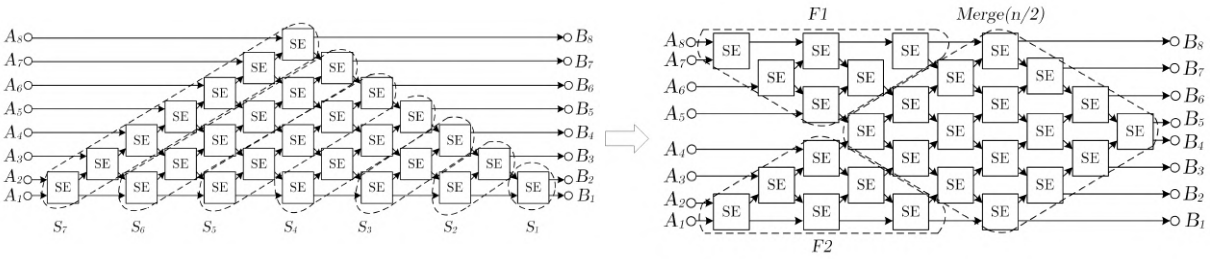
The general algorithm of information-equivalent transformations for matrix-vector multiplication can be supplemented with additional transformations to ensure that the resulting graph represents the FFT algorithm itself, rather than a merely equivalent structure. These transformations rely on the uniqueness and functional dependencies of the complex Vandermonde coefficients W and are not generally applicable to arbitrary matrix-vector multiplication tasks.

The proposed general algorithm was also verified on another problem, the Walsh-Hadamard transform [11], which, like the DFT, is a matrix-vector multiplication problem. Similar computational complexity estimates were obtained: $O(N^2)$ for the original graph and $O(N \cdot \log_2 N)$ for the "fast" version [12].

Due to the developed general algorithm of information-equivalent graph transformations, the optimizing synthesizer automatically produces a solution with higher computational efficiency than the original, general implementation.

2.2. Transformation of an Information Graph for a Sorting Network

The developed general algorithm of information-equivalent transformations was also used for graphs solving problems from a different area, the sorting of an input sequence. Let us consider the operation of this algorithm using the transformation of the bubble sorting algorithm into an initial array A containing eight elements. The original IG (Fig. 6a), developed on the "head-tail" combining steps principle described in [13], is transformed, according to step 2 of the algorithm, into a sorting network, developed using the "divide-in-half" principle (Fig. 6b). According to step 4, pairs of equivalent subgraphs $F1$ and $F2$ are formed, each of which implements subtask of sorting half of the array, along with a merge subgraph $Merge(n/2)$ that combines the results of the subgraph pairs.



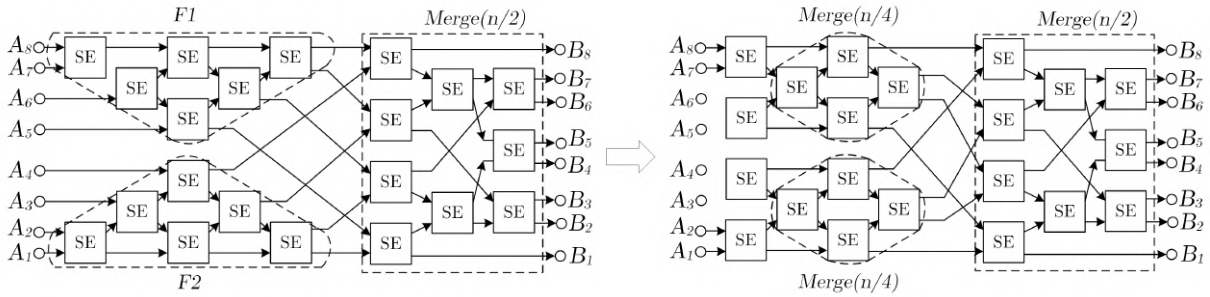
(a) Bubble sorting IG

(b) IG after combinig steps

Figure 6. Transformations of sorting networks

Since the graph structure of the sorting algorithm does not contain isomorphic subgraphs with identical input data, step 5 is skipped. Further, due to the functional redundancy shown

in [13], the described transformation methods and changing the order of operations in accordance with step 6, information-equivalent transformations of the $Merge(n/2)$ subgraph are performed (Fig. 7a).



(a) Transformations of the $Merge(n/2)$ (b) Subgraphs $F1$ and $F2$ transformation

Figure 7. Transformations of sorting networks

According to step 7, subgraphs $F1$ and $F2$ are placed into the array of processed graphs. At the next iteration, in accordance with step 2, transformations are performed to subgraphs $F1$ and $F2$. Each of subgraphs is further divided into two subgraphs that sort one-quarter of the original array, along with two merge networks $Merge(n/4)$ (Fig. 7b). After performing step 6 transformations on the $Merge(n/4)$ networks, the resulting sorting algorithm structure corresponds to the Batchers odd-even sorting network (Fig. 8).

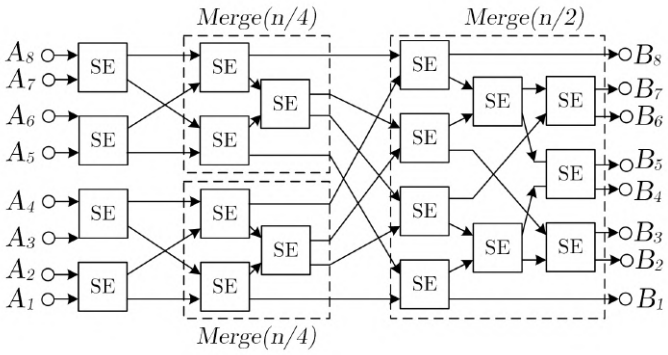


Figure 8. Batchers odd-even sorting network

The resulting four sorting subgraphs contain a single sorting element and cannot be further transformed, so the algorithm completes its work. Therefore, the original IG of the bubble sorting algorithm with computational complexity $O(N) = \frac{1}{2}N^2$, is automatically transformed into the IG of Batchers odd-even sorting network with computational complexity $O(N) = \frac{1}{4}(N \cdot \log_2^2 N + 2 \cdot \log_2 N)$. The sorting networks automatically obtained through information-equivalent transformations have been simulated and analyzed in [13, 14], which also verify the correctness of such transformations.

2.3. Transformation of an Information Graph for Solving Tridiagonal SLAEs Using the Jacobi Method

The general algorithm of information-equivalent transformations was verified on problems of solving tridiagonal systems of linear algebraic equations (SLAEs) by the Jacobi method. In

this method, the vector of unknowns is calculated as:

$$Y_i^v = \frac{1}{2}(Y_{i+1}^{v-1} + Y_{i-1}^{v-1}) + \frac{\Delta^2 C_i}{2}, \tag{1}$$

where v is the current iteration number; i is the iteration layer from which the data is taken; Δ is the grid step; C_i are the free terms. The computational complexity of the classical Jacobi method for tridiagonal SLAEs is $O(N) = N \cdot K$ in a sequential implementation, where N is the matrix dimension and K is the number of iterations required to reach a specified accuracy. Figure 9 shows the IG for SLAE of dimension $N = 9$ and the number of iterations K=4, where the vertex f implements the calculation of the function $f = \frac{1}{2}(Y_{i+1}^{v-1} + Y_{i-1}^{v-1})$. Calculations related to the free terms C_i vector (1) have been omitted so as not to overload the illustrative material, but the transformations described below take into account all operations of the original formula for calculating the vector of unknowns.

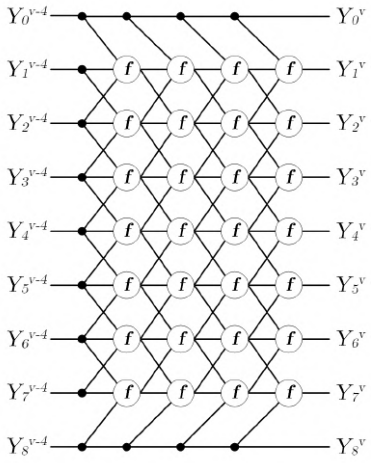
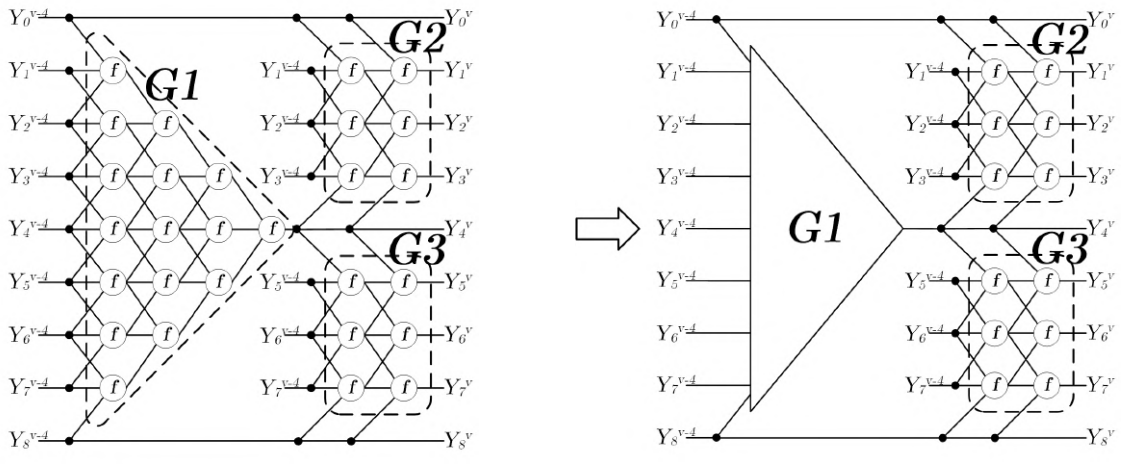


Figure 9. IG of the Jacobi method for tridiagonal SLAEs

According to step 1 of the general algorithm, the initial IG of the algorithm for solving tridiagonal SLAEs by the Jacobi method is placed in an array of processed graphs. According to paragraphs 2-4 of the general algorithm, the IG is transformed to the following form (Fig. 10a), where a subgraph $G1$ is formed to calculate the Y_4 central element of the vector of unknowns with a given depth of iterations and two subgraphs $G2$ and $G3$, each of which implements a half-size SLAE calculation.

Since the resulting graph structure (Fig. 10a) does not contain isomorphic subgraphs with identical input data, step 5 is skipped. According to step 6, the subgraph $G1$ is processed: the vertices f are expanded into arithmetic addition and multiplication operations, and information-equivalent transformations corresponding to the expansion of brackets and combination of like terms are applied. As a result, the subgraph $G1$ acquires a pyramid structure of adders with multiplications at the base, where the total number of computational operations scales logarithmically with the input array size (Fig. 10b).

According to step 7, the array of processed subgraphs is cleared, and subgraphs $G2$ and $G3$ are added with the general algorithm continuing from step 2. Each of $G2$ and $G3$, is further subdivided, according to steps 2-4, into subgraphs $G4$, $G5$, $G6$ and $G7$, $G8$, $G9$ respectively (Fig. 11a). Subgraphs $G4$ and $G7$ are transformed into pyramid structures as in $G1$ after applying

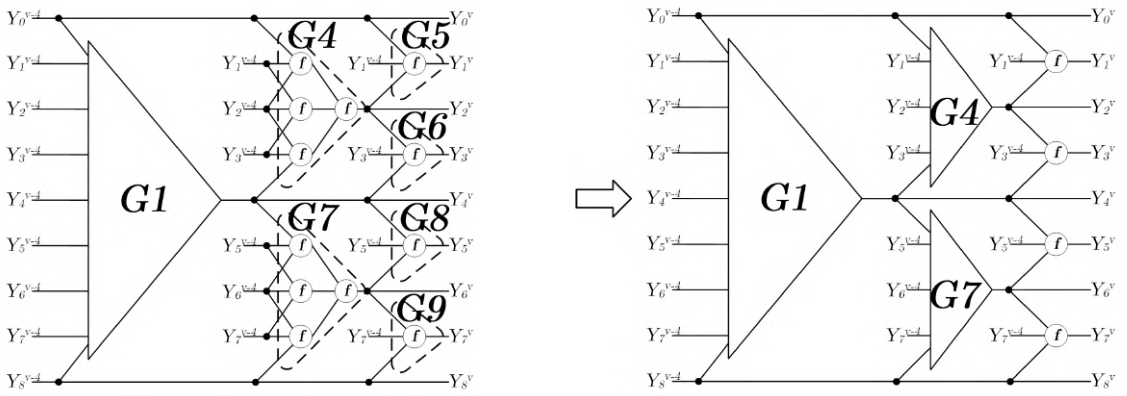


(a) Divide Jacobi IG by two

(b) Transformation subgraph $G1$

Figure 10. Transformations of the Jacobi IG

step 6 (Fig. 11b). The remaining subgraphs $G5$, $G6$, $G8$ and $G9$ contain only a single function f and cannot be further transformed, marking the completion of the algorithm.



(a) Divide subgraphs $G2$ and $G3$ by two

(b) Transformation subgraphs $G4$ and $G7$

Figure 11. Transformations of the Jacobi IG

The resulting "fast" Jacobi method represents an iterative calculation of the midpoint of the computational grid, where the next step processes two halves of the original range. The number of operations for the classical Jacobi method is given by $S_{classic} = (4 \cdot N - 2) \cdot K \approx 4 \cdot N \cdot K$, while for the fast Jacobi method it is $S_{fast} = 67 \cdot K \cdot (\log_2(N-1) - 1) \approx 67 \cdot K \cdot \log_2 N$, where N is the row length of the SLAE, and K is the number of iterations required to achieve the required accuracy. Due to information-equivalent transformations, the optimizing synthesizer automatically generates a new computational method that can be structurally implemented on a RCS, achieving higher specific performance than the original method. Such transformations for tridiagonal SLAEs using the Jacobi method have been simulated and analyzed in [15], which also verify their correctness.

3. Nested Auto-Substitution Method

In addition to the general algorithm and the library of information-equivalent transformations, the optimizing synthesizer implements an original nested auto-substitution method

described in [16], which increases the efficiency of parallel-pipelined implementation of recursive expressions.

The input data rate to the structure implementing a recursive expression depends on the execution time of all operations along the critical path of the feedback loop at pipelined computing. In other words, you can submit the next input count only after processing the previous count is completed. It is known that the velocity of such calculations can be increased using the well-known method of recursive expression pipelining-sequential auto-substitution.

The principle of this method is that any recursive expression of the form:

$$y_i = f(y_{i-1}, a_i), \tag{2}$$

where y_i is the current calculated value; y_{i-1} is the previous value; a_i are certain coefficients of the function f, can be expanded recursively. Expressing y_{i-1} as

$$y_{i-1} = f(y_{i-2}, a_{i-1}), \tag{3}$$

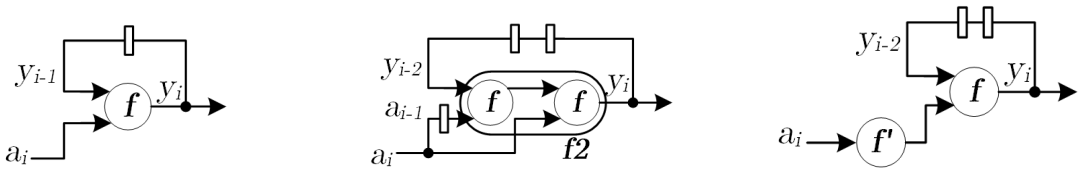
and substituting (3) into (2), we obtain

$$y_i = f(f(y_{i-2}, a_{i-1}), a_i) = f2(y_{i-2}, a_i, a_{i-1}) \tag{4}$$

Continuing this procedure, y_i can be expressed recursively:

$$y_i = fn(y_{i-n}, a_i, a_{i-1}, ..., a_{i-n+1}) \tag{5}$$

Sequential auto-substitution can be implemented on a graph using information-equivalent transformations. The initial graph corresponding to (2) is given in Fig. 12a, where the vertex corresponding to the delay of the operand stream by one sample is indicated as empty rectangle in the feedback.



(a) Recursive expression IG

(b) Auto-Substitution IG

(c) Transformed auto-Substitution IG

Figure 12. Sequential auto-substitution on a graph

Graph transformation corresponding to (2) is performed by cascading the node representing f and adding delays to its input arcs (Fig. 12b). If f is a linear function, further transformations, corresponding to bracket expansion and combining like terms, can be performed resulting in the graph shown in Fig. 12c. In this resulting graph, the critical path length remains the same as in the original graph, while the number of registers in the feedback loop increases, allowing a higher input data rate for pipeline processing [16]. Repeating such transformations allows continuous data flow; however, computing fn requires more operations than the original expression, with the number of additional operations scaling linearly with the recursion depth.

The new nested auto-substitution method improves upon sequential auto-substitution. After obtaining formula (4), instead of expressing y_{i-2} from the original formula (2), it is expressed

from formula (4) itself:

$$y_{i-2} = f2(y_{i-4}, a_{i-2}, a_{i-3}) \tag{6}$$

Substituting (6) into (4) we obtain:

$$y_i = f2(f2(y_{i-4}, a_{i-2}, a_{i-3}), a_i, a_{i-1}) = f4'(y_{i-4}, a_i, a_{i-1}, a_{i-2}, a_{i-3}) \tag{7}$$

Note that $f4$ from sequential auto-substitution differs from $f4$ in the nested method. Subsequent steps follow the same principle, applying the previously obtained formula at each iteration:

$$y_i = f4'(f4'(y_{i-8}, a_{i-6}, ...), a_i, ...) = f8'(y_{i-8}, a_i, ...),$$

$$y_i = f8'(f8'(y_{i-16}, a_{i-14}, ...), a_i, ...) = f16'(y_{i-16}, a_i, ...),$$

$$y_i = fN'(y_{i-N}, a_i, ...),$$

where N is a power of two corresponding to the number of recursion-unfolding steps.

On a graph, transformations in accordance with the nested auto-substitution method, unlike sequential, consist in the fact that for cascading, not the subgraph describing the original function f is selected, but the subgraph, obtained at the previous step. Therefore, for linear function f (Fig. 12c) the second iteration of graph transformations is given in Fig. 13.

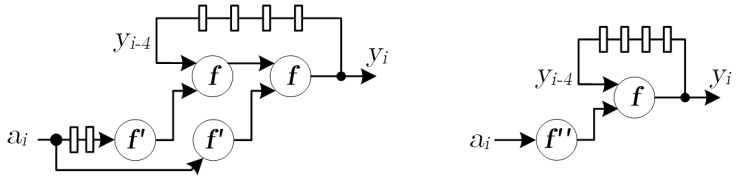


Figure 13. Nested auto-substitution on a graph

In sequential auto-substitution, the growth of additional hardware from pipelining is estimated as $O_{seq}(N) = K_1 \cdot N$, where N is the degree of pipelining and K_1 depends on function f. In the new nested method, the additional hardware growth is reduced to $O_{nested}(N) = K_2 \cdot \log_2 N$, where N is always a power of two corresponding to the recursion-unfolding steps. Synthesized solutions were simulated on RCS of various configurations, confirming the correctness of the information-equivalent transformations.

The nested auto-substitution method can also be used for the simple iteration method for solving SLAEs [17], described by $\overrightarrow{x}_i = T \times \overrightarrow{x}_{i-1} + \overrightarrow{\phi}$. In this case, the function on the right-hand side involves matrix and vector operations. The computational complexity of the original algorithm is $O(n) \approx (2 \cdot M^2 + M) \cdot n$, where n is the number of iterations, M is the SLAE dimension. Using nested auto-substitution, the "fast" version reduces the complexity to $O(n) \approx (2 \cdot M^2 + M) \cdot \log_2 n$.

4. Results of Application of the Optimizing Synthesizer

The effectiveness of the new version of the multichip optimizing synthesizer for parallel-pipelined programs for RCS was investigated on a number of benchmark tasks on the RCS Tertius architecture [18] with a peak performance of 2.5 TFLOPS. The system includes four Xilinx Kintex UltraScale XCKU095 FPGAs, each with 100 million equivalent logic gates, connected in a ring using LVDS and GTY/GTH channels.

Note that the solutions generated by the developed optimizing synthesizer can also be directly implemented by a vendor in the program code of a parallel-pipelined program for RCS. However, in this case, the program size, debugging time and the vendors skill requirements increase, often reducing the program readability. In addition, the source C programs almost never contain an effective structural and procedural organization of calculations for parallel-pipelined RBC programs obtained during the automatic conversion of an input sequential program in C.

The performance of the optimizing synthesizer was tested on graphs for solving discrete Fourier transform problems, sorting networks, solving tridiagonal SLAE using the Jacobi method, and for a second-order recursive filter with constant coefficients. The results of the optimizing synthesizer implementation were the structures of "fast" algorithms that were automatically transformed in accordance with the general algorithm and nested auto-substitution, where the effectiveness of the solutions obtained was evaluated.

The result of the optimizing synthesizer for DFT problem with dimension 8 is given in Fig. 14. The resulting transformed graph is not isomorphic to the graph of the well-known FFT method, but contains the same number of computational elements. The original DFT IG includes 56 adders and 64 multipliers for complex numbers. The resulting IG contains 24 adders and subtractors and 6 multipliers (25% from the original). The synthesizer was also verified on larger DFT problems, producing graphs with computational complexity equivalent to FFT graphs.

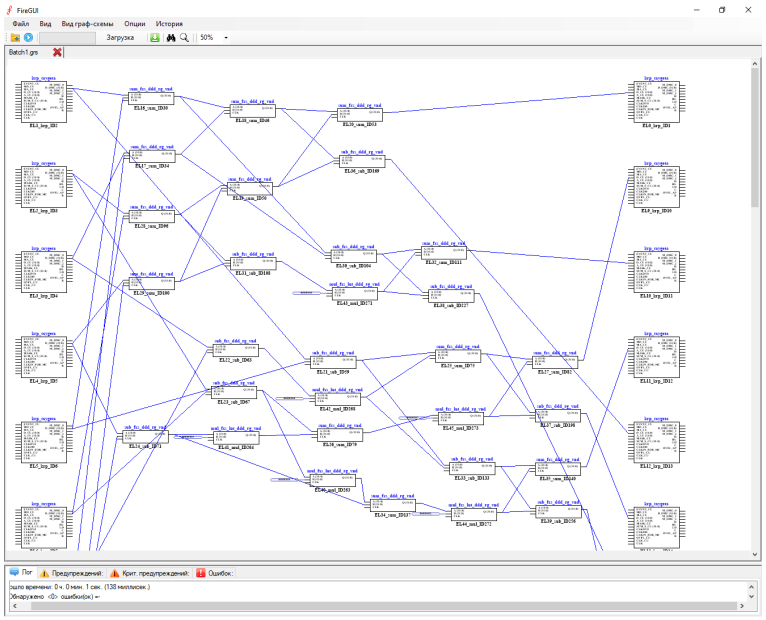


Figure 14. Result of the optimizing synthesizer for DFT problem

An odd-even Batcher network for sorting 16 elements, automatically generated by the synthesizer from a bubble-sort network, is given in Fig. 15. The synthesizer was tested on networks with dimensions 16, 32, 128 and 256.

For solving a tridiagonal SLAE using the Jacobi method with dimension 128, the synthesizer automatically produced a "fast" Jacobi IG, containing approximately five times fewer operations than the classical Jacobi method. As the matrix dimension increases, this indicator increases, since for the classical Jacobi method the number of operations depends linearly on the size of the matrix, and in the "fast" Jacobi method this dependence is logarithmic.

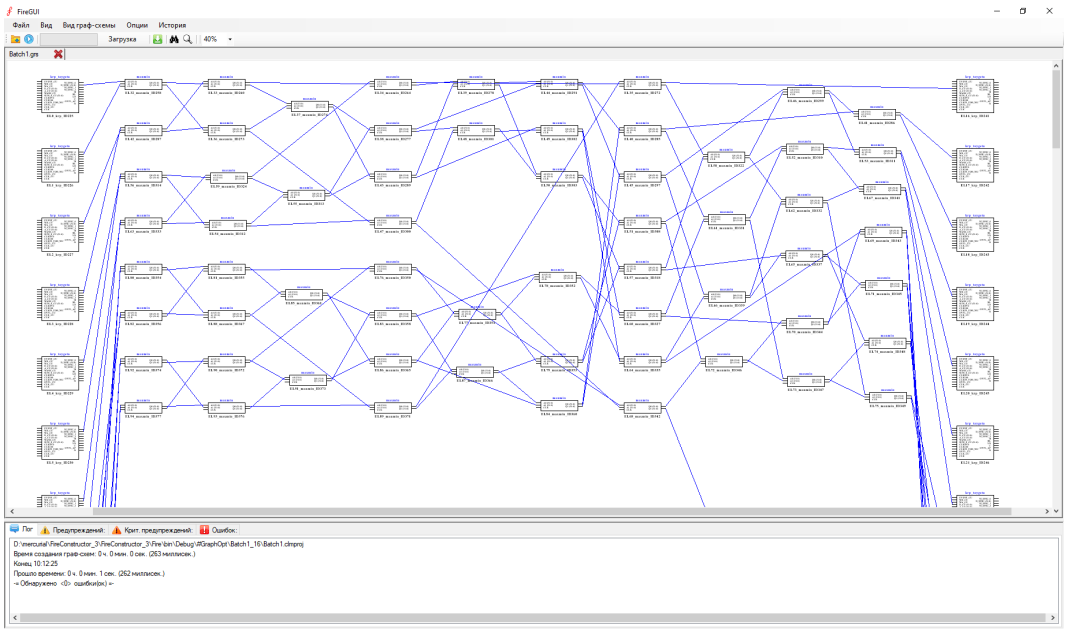


Figure 15. Result of the optimizing synthesizer by sorting problem

Figure 16 presents the result of the synthesizer on a second-order recursive filter problem with constant coefficients.

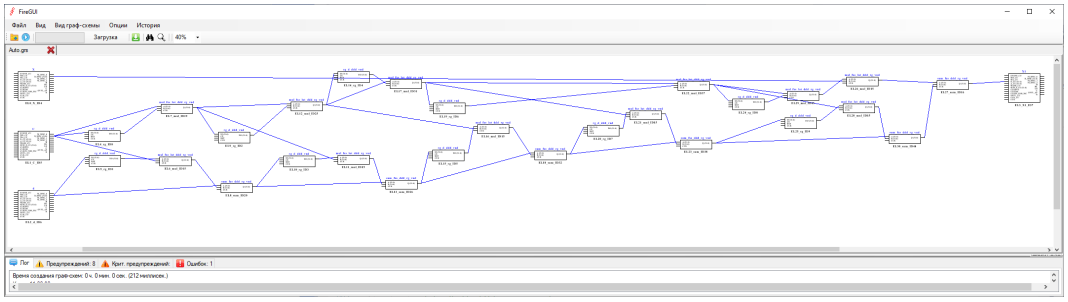


Figure 16. Result of the optimizing synthesizer by recursive filter problem

The additional hardware required for pipelining the second-order recursive filter using sequential auto-substitution is equal to $O_{seq}(N) = 2 \cdot N$, where N is the pipeline depth, while for nested auto-substitution it is equal to $O_{nested}(N) = 6 \cdot \log_2 N$. Based on these formulas, it is obvious that for this filter, using the nested auto-substitution becomes more effective than sequential auto-substitution at $N > 8$.

To ensure that adders and multipliers operate at the maximum RCS Tertius frequency of 500 MHz, they were implemented in a pipelined configuration. The adder and multiplier contains 12 and 8 pipeline registers, respectively. In the case of development a recursive part of the filter on this element base, the feedback circuits contain two adders connected in series and one multiplier. Therefore, the result is formed after 32 cycles in feedback circuits. It determines the choice of the pipelining degree. The efficiency of the second-order recursive filter solution automatically created in the synthesizer with constant coefficients on the Tertius RCS implementation was approximately twice that of the previously known sequential auto-substitution method. Similar results were obtained for adaptive filters with variable coefficients.

Conclusion

The new version of the optimizing synthesizer for parallel-pipelined programs described in this paper not only maps the IG algorithm of the solved problem on multichip RCS but also significantly reduces the number of operations in programs created by an unqualified user. The library implemented in the optimizing synthesizer and the general algorithm of information-equivalent transformations, as well as the original method of nested auto-substitutions, allow to automatically obtain the "fast" algorithms, the effectiveness of which is not inferior to previously known methods or new ones based on original methods developed by the research team at Scientific Research Center of Supercomputers and Neurocomputers (Taganrog).

The number of computational operations in traditional algorithms for solving the considered model problems is determined by the formula $O(N) = C_1 \cdot N^m$, where C_1 and m depend on the specific problem. In all resulting "fast" algorithms, the number of operations is determined by the formula $O(N) = C_2 \cdot N^{m-1} \cdot \log_2 N$. Therefore, the efficiency of the optimizing synthesizer only increases with increasing problem dimension. So, for the problem dimension $N = 1024$, the number of operations in the algorithm can be reduced by two decimal orders of magnitude, and the specific productivity will increase a hundredfold, which is unattainable for similar synthesis tools.

Future research directions include the development of optimization methods for computing structures in problems with variable operand stream intensity and IGs containing structural analogs of unconditional jumps.

Acknowledgements

The study was supported by the Russian Science Foundation grant No. 26-11-00209, https://rscf.ru/project/26-11-00209/.

References

1. Levin, I.I., Dordopulo, A.I., Fedorov, A.M., Kalyaev, I.A.: Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems. Supercomputing Frontiers and Innovations 3(1), 22–40 (2016). https://doi.org/10.14529/jsfi160102

2. Kalyaev, I.A., Levin, I.I.: Reconfigurable Computing Systems Based on FPGAs. Publishing House of SSC RAN, Rostov-on-Don (2022). 506 p.

3. Antonov, A.S., Afanasyev, I.V., Voevodin, Vl.V.: High-performance computing platforms: current status and development trends. Computational Methods and Programming 22(2), 135–177 (2021). https://doi.org/10.26089/NumMet.v22r210

4. Dichenko, A.A., Levin, I.I., Sorokin, D.A.: Method for generating topological constraints of computational structures for reconfigurable computing system. Izvestiya SFedU. Engineering Sciences, 33–46 (2022). https://doi.org/10.18522/2311-3103-2025-6-33-46

5. Gulenok, A.A.: Synthesizer of Structural Parallel Application Programs for Multichip Reconfigurable Computing Systems: Candidate of Technical Sciences Dissertation. Taganrog, 2011.

6. Kotlyarov, D.V., Kutepov, V.P., Osipov, M.A.: Graph-based stream parallel programming and its implementation on cluster systems. RAS Publishing, Theory and Control Systems 1, Moscow (2005).

7. Nane, R., *et al.*: A survey and evaluation of FPGA high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35(10), 1591–1604 (2016). https://doi.org/10.1109/TCAD.2015.2513673

8. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. Technical Report TR 98-019, Department of Computer Science, University of Minnesota (1998).

9. Gulenok, A.A., Dudko, S.A.: Hardware resource management during automatic placement of synchronization triggers. 12th Multiconference on Control Problems, MKPU-2019, Divnomorskoye, Russia, vol. 3, pp. 86–88 (2019).

10. Mikhaylov, D.V.: Methods for creating pipeline-parallel programs for tasks with sequential information graphs: Candidate of Technical Sciences dissertation, specialty 05.13.11 "Mathematical and software support for computing machines, complexes, and networks", advisor: Prof. I.I. Levin. Taganrog (2022). 213 p.

11. Golubov, B.I., Efimov, A.V., Skvortsov, V.A.: Walsh Series and Transforms: Theory and Applications. Nauka, Moscow, (1987). 360 p.

12. Semernikov, E.A., Gulenok, A.A.: Structural optimization of the information graph of matrix-vector multiplication. Proceedings of the XIV All-Russian Meeting on Control Problems VSPU-2024, pp. 3024–3028. IPU RAS, Moscow (2024).

13. Levin, I.I., Alekseev, K.N.: Transformation of sorting networks for different degrees of parallelism. Izvestiya SFedU. Technical Sciences 5(235), 104–118 (2023). https://doi.org/10.18522/2311-3103-2023-5-104-118

14. Alekseev, K.N., Levin, I.I., Gulenok, A.A.: Transformation of simple sorting networks into Batchers odd-even networks. Izvestiya SFedU. Technical Sciences 4(240), 50–63 (2024). https://doi.org/10.18522/2311-3103-2024-4-50-63

15. Levin, I.I., Chekina, M.D.: Fast Jacobi method for solving SLAEs on reconfigurable computing systems. Science in the South Russia, vol. 2, pp. 145–165. Nauka Publisher, Rostov-on-Don (2026).

16. Levin, I.I., Gulenok, A.A., Semernikov, E.A.: Nested auto-substitution method for pipelining computations in recursive filters. PAO Radiofizika. Radar and Communication 2(40), 17–28 (2025). https://doi.org/10.18127/j00338486-202510-02

17. Samokhin, A.B.: Integral Equations and Iterative Methods in Electromagnetic Scattering. Moscow: Radio i Svyaz (1998).

18. Computing units of the NIT supercomputers and neurocomputers. http://superevm.ru/index.php?page=modern-developments, accessed: 2026-04-01

DOI: 10.14529/jsfi260207

Improving Performance of HPC Systems Based on Robust Survival Machine Learning Methods

Lev V. Utkin[1], ***Andrei V. Konstantinov***[1], ***Vladimir S. Zaborovsky***[1], ***Vladimir A. Muliukha***[1]

High-performance computing (HPC) systems face growing complexity and uncertainty that limit the accuracy of classical methods for predicting computational processes and task execution. To address the problem of tasks failing to complete within allotted time windows, termed "censored" events, survival machine learning methods are proposed for predicting execution time and minimizing such losses. Survival analysis plays an important role for solving the task of time-to-event prediction. While traditional methods handle censored data effectively, they often rely on strong parametric assumptions that may limit their flexibility. This paper introduces a novel survival analysis framework, called CiSurv (Contaminated imprecise Survival model), that integrates imprecise probability theory with attention-based multi-label classification. By reformulating survival prediction as an imprecise classification problem, CiSurv provides an approach to modeling uncertainty in censored observations while improving predictive accuracy. The proposed framework incorporates the imprecise contaminated model to refine interval-valued probabilities associated with censored data. Two models are developed: CiSurvN, which uses the neural network-based attention mechanism for solving the classification task, and CiSurvG, which employs Gaussian kernel-based attention with a single kernel parameter. Experiments on real and synthetic data demonstrate that CiSurvN generally outperforms CiSurvG due to its ability to learn complex feature dependencies. Key findings reveal that intermediate contamination parameter values yield optimal performance, outperforming a special case when the parameter is 1 in most cases. Codes implementing the proposed models are publicly available.

Keywords: high-performance computing, survival analysis, censoring, attention mechanism, imprecise contaminated model.

Introduction

High-performance computing (HPC) is a powerful tool used for a wide range of scientific research requiring large volumes of data and high levels of algorithmic complexity. Although the peak performance of HPC systems continues to grow steadily, reflecting the potential of hybrid computing electronic component, the complexity of using such systems is simultaneously increasing. Furthermore, the high uncertainty in describing the characteristics of computational environment and multifunction application workflows limits the accuracy of classical methods for planning and predicting both stochastic and deterministic characteristics of computational processes. One of the important problems of a functioning supercomputer system is to successfully complete the execution of a specific applied task executed on behalf of a specific user in the time interval allotted for this by the dispatcher and to minimize losses arising when the task does not complete the execution due to various reasons. The actual execution time of tasks depends on many factors and always differs from the estimated time. When the task is not completed, then the corresponding event can be regarded as the censored event which means that the task cannot be executed in a predefined period of time. To predict the execution time of the task and to minimize the number of "censored events, the survival machine learning methods can be applied.

[1]Higher School of Artificial Intelligence Technologies, Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russian Federation

Survival analysis, commonly known as time-to-event analysis, is a core statistical methodology used to model the time until a specific event occurs [10]. Its broad applicability has made it indispensable in such fields as medicine, engineering, finance, and industrial reliability [1]. A major strength of this methodology is its capacity to incorporate censored data, where event times are incompletely observed, enabling more accurate and interpretable results for evidence-driven decisions. Survival analysis supports engineering tasks such as predicting system failures, optimizing maintenance schedules, and evaluating reliability [18].

Survival analysis has spurred the development of numerous machine learning models to address time-to-event prediction challenges [23, 27]. These models include adaptations of traditional machine learning methods, for example, random forests, neural networks, transformer-based approaches, to survival data [4, 12, 28]. Methodological perspectives of survival analysis are offered in [8].

Attention-based approaches [20] have enhanced survival analysis. Transformers capture long-range dependencies [11, 24], with applications spanning medical survival analysis [13] and competing risks scenarios [25]. These methods demonstrate improved performance in high-dimensional settings while providing feature importance insights.

Alternatively, survival analysis can be reframed as classification by discretizing time intervals [7, 19] with likelihood-based loss functions [15]. The main idea of the models is to consider the event probabilities over intervals. Another approach [14] for survival analysis with discretizing time intervals is to consider non-parametric probability distributions over time intervals for each instance in the training set. For uncensored observations, the event time distribution is deterministic, assigning probability 1 to the true interval and 0 elsewhere. For censored observations, uncertainty arises: the event could occur in any interval after censoring, yielding a set of possible distributions and reflecting partial knowledge. It is assumed that the probability of the event occurring in any interval after the censored time can range from zero to one, reflecting prior ignorance. To aggregate the probability distributions generated for each instance from the training set, it is proposed in [14] to employ kernel Nadaraya-Watson regression [17, 26] with trainable dot-product attention weights [16, 20]. A model implementing the approach is called iSurvJ.

We propose to extend the approach introduced in [14] by applying the imprecise ϵ-contaminated model [21] to reduce the set of probability distributions over the time to event intervals. A goal of incorporating the imprecise ϵ-contaminated model into the model [14] is to enhance the prediction accuracy of the model by searching for the optimal hyperparameter ϵ which controls imprecision of probabilities over the time intervals. The proposed model, called CiSurv (**C**ontaminated **i**mprecise **Surv**ival model), aims to open a door for incorporating various imprecise models into survival analysis. It illustrates that there are optimal parameters of the imprecise models which can improve a survival model analyzed. Two modifications of CiSurv are studied: CiSurvN and CiSurvG. The first model, CiSurvN, uses the neural network to compute the matrix of attention weights. The second model, CiSurvG, uses the Gaussian kernels with a single parameter for computing the weights. On the one hand, the model CiSurvN is more powerful due to using the neural network. On the other hand, when a dataset is rather small, the model CiSurvG may provide better results due to a small number of training parameters.

It should be noted that imprecise models have been proposed in survival analysis offering robust alternatives when distributional assumptions are violated. Key approaches include the Hill model framework for right-censored data [6] and the imprecise Dirichlet model [5, 22].

However, the models provide lower and upper bounds for the survival concepts, for example, for survival functions. Moreover, in contrast to the proposed approach, they do not take into account relationships between the feature vectors.

Various numerical experiments with real and synthetic datasets are provided to compare the proposed survival models with each other for different values of the hyperparameter ϵ. It is shown by numerical examples that there is an "optimal value" of ϵ which provides the best accuracy measure for a certain dataset.

The corresponding codes implementing the proposed models are publicly available at: `https://github.com/NTAILab/isurv_reducing_imprecision`.

The paper is organized as follows. Section 1 provides basic definitions of survival analysis. The model iSurvJ proposed in [14] is considered in Section 2. The approach to reduce imprecision of probabilities in iSurvJ and the model CiSurv are described in Section 3. Numerical experiments with well-known public real and synthetic data illustrating CiSurv are provided in Section 4. Concluding remarks are provided in Section 5.

1. Preliminaries: Survival Analysis

Datasets in survival analysis are represented by a set of triplets $\mathcal{A} = \{(\mathbf{x}_1, \delta_1, T_1), ..., (\mathbf{x}_N, \delta_N, T_N)\}$ [10]. Here the vector $\mathbf{x}_i \in \mathbb{R}^d$ consisting of d features characterizes the i-th object. The time T_i corresponds to one of two types of the event observations. The first type is when the event is observed. In this case, the observation is called uncensored and the censoring indicator δ_i is equal to 1. The second type is when the event is not observed, it is greater than T_i, but its 'true' value is unknown. In this case, the observation is called censored and the censoring indicator δ_i is equal to 0. A survival model is trained on the set $\mathcal{A}$ to predict probabilistic measures of an event time T for a new object represented by the vector $\mathbf{x}$.

An important concept in survival analysis is the survival function (SF) $S(t \mid \mathbf{x})$, which is defined as the probability of surviving up to time t, i.e., $S(t \mid \mathbf{x}) = \Pr\{T > t \mid \mathbf{x}\}$. Most survival measures can be derived through the SF. Comparison of survival models is often carried out by means of the Brier Score (BS) [3, 9] which is defined as

$$BS(t, \hat{S}) = \mathbb{E}\left[\left(\Delta_{new}(t) - \hat{S}(t \mid \mathbf{x}_{new})\right)^2\right], \tag{1}$$

where $\Delta_{new}(t) = \mathbf{1}\left(T_{new} > t\right)$ is the true status of a new test subject; $\hat{S}(t \mid \mathbf{x}_{new})$ is the predicted survival probability.

The Integrated Brier Score (IBS) extends the BS by averaging it over a range of time points, typically from $t = 0$ to a maximum time $t_{\max}$. It is defined as:

$$IBS = \frac{1}{t_{\max}} \int_0^{t_{\max}} BS(t, \hat{S})\, \mathrm{d}t. \tag{2}$$

2. Survival Analysis as an Imprecise Multi-Label Classification Problem

First, we introduce the model iSurvJ proposed in [14] which considers the survival analysis problem as an imprecise classification task under condition of total ignorance.

Consider a partition of the time axis into discrete intervals:

$$[0,\infty) = [0,t_1] \cup [t_1,t_2] \cup ... \cup [t_{T-2},t_{T-1}] \cup [t_{T-1},\infty), \tag{3}$$

such that

$$0 = t_0 < t_1 < t_2 < ... < t_{T-1} < \infty. \tag{4}$$

The partition divides the time axis into T intervals denoted as $\tau_i = [t_{i-1}, t_i]$ for $i = 1, \ldots, T$. Let us define the probability $\pi_k^{(i)}$ that an event corresponding to the i-th object is observed in the interval τ_k. This generates a probability distribution for each subject $\pi^{(i)} = (\pi_1^{(i)}, ..., \pi_T^{(i)})$, $i = 1, ..., N$. If the observation is uncensored, then there holds

$$\pi_j^{(i)} = 0, \text{ if } j \neq k, \quad \pi_j^{(i)} = 1, \text{ if } j = k. \tag{5}$$

For censored observations, the true event time must occur in some interval after t_{k-1}, though the specific interval remains unknown. Consequently, the probability that the event is at each interval after the time t_{k-1} can be from 0 to 1, i.e., there holds

$$\pi_j^{(i)} = 0, \text{ if } j < k, \quad \pi_j^{(i)} \in [0,1], \text{ if } j \geq k. \tag{6}$$

The interval $[0, 1]$ is a form of prior ignorance. We have the constraint $\sum_{j=1}^{T} \pi_j^{(i)} = 1$ for precise probabilities. However, this condition is not fulfilled when the upper or lower probabilities are summed. Due to imprecision of probabilities $\pi_j^{(i)}$, we suppose that the i-th instance produces a set of probability distributions $\pi^{(i)}$ denoted as $\mathcal{R}^{(i)}$.

Let us consider the following *classification problem*. There is a set of feature vectors $\mathbf{x}_1, ..., \mathbf{x}_N$. The k-th interval $[t_{k-1}, t_k]$ can be viewed as the k-th class represented by the vector $\pi_k = (\pi_k^{(1)}, ..., \pi_k^{(N)})^{\mathrm{T}}$. Then the Nadaraya-Watson kernel regression [17, 26] can be applied to find the class probability distribution $\mathbf{p}(\mathbf{x}) = (p_1(\mathbf{x}), ..., p_T(\mathbf{x}))$ of a new instance $\mathbf{x}$. Since a part of probabilities $\pi_k^{(i)}$ are interval values, then it is obvious that the predicted class probabilities $p_1, ..., p_T$ are also interval-valued. They are determined as follows:

$$p_k(\mathbf{x}) = \sum_{i=1}^{N} w(\mathbf{x}, \mathbf{x}_i)\pi_k^{(i)}, \ k = 1, ..., T, \tag{7}$$

where $w(\mathbf{x}, \mathbf{x}_i)$ is an attention weight expressed through kernels $K(\mathbf{x}, \mathbf{x}_i)$, $i = 1, ..., N$.

The attention weights satisfy the condition:

$$\sum_{i=1}^{N} w(\mathbf{x}, \mathbf{x}_i) = 1. \tag{8}$$

We extend the above weights $w(\mathbf{x}, \mathbf{x}_i)$ to a more general form accepted in transformers [16, 20]. Introduce a matrix of the input keys $\mathbf{K} \in \mathbb{R}^{N \times d}$ (the matrix of N vectors $\mathbf{x}_i$), and a matrix of the input queries $\mathbf{Q} \in \mathbb{R}^{N \times d}$ (the matrix of the same feature vectors). We also introduce the trainable matrices $\mathbf{W}_K \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_Q \in \mathbb{R}^{d \times d}$. Then the matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ of attention weights $w(\mathbf{x}_j, \mathbf{x}_i)$, $i = 1, ..., N$, $j = 1, ..., N$, can be computed as [16, 20]:

$$\mathbf{W} = \mathrm{softmax}\left(\frac{(\mathbf{Q}\mathbf{W}_Q)\,(\mathbf{K}\mathbf{W}_K)}{\sqrt{d}}\right). \tag{9}$$

Hence, (7) is rewritten as:

$$p_k(\mathbf{x}) = \mathbf{W}_k \pi_k, \ k = 1, ..., T, \tag{10}$$

where $\mathbf{W}_k$ is the k-th row of the matrix $\mathbf{W}$.

By having the probability distribution $\mathbf{p}(\mathbf{x})$, the predicted SF at time $t \in [t_{j-1}, t_j]$, $j = 1, ..., T$, can be computed as follows:

$$\hat{S}(t \mid \mathbf{x}) = \sum_{k=j+1}^{T} p_k(\mathbf{x}). \tag{11}$$

The matrix $\mathbf{W}$ can be computed in a more simple way by using the Gaussian kernel

$$w(\mathbf{x}_j, \mathbf{x}_i) = \text{softmax}\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\psi}\right), \tag{12}$$

where ψ is the kernel parameter.

On the one hand, this way simplifies the model because it learns only the parameter ψ for computing the matrix $\mathbf{W}$ and can be applied to cases of small datasets. On the other hand, this simplified model may provide worse accuracy results.

2.1. Model iSurvJ

To solve the above classification problem under condition of the interval-valued class probabilities, we have to define how to train the classifier when the class probabilities are interval-valued.

In the model, *iSurvJ*, the class probabilities and the attention weights are jointly learned. To implement the learning, the probability logits are initially initialized as zeros. At each epoch, the softmax(logits) operation is applied to the logits to obtain probabilities of intervals $\widetilde{\pi}_k^{(i)}$ and ensuring the condition $\sum_{k=1}^{T} \widetilde{\pi}_k^{(i)} = 1$. Here $\widetilde{\pi}_k^{(i)}$ are precise analogues of interval-valued probabilities $\pi_k^{(i)}$ obtained by means of learning.

Let $c(i)$ be an index of a time interval in which the event, corresponding to the i-th instance, occurs, and $p_{c(i)}(\mathbf{x}_i)$ be the probability of the interval with the index $c(i)$. The whole loss function for learning the model is defined as

$$\mathcal{L}(\mathbf{p}) = \sum_{i=1}^{N} l(\mathbf{p}(\mathbf{x}_i)), \tag{13}$$

where $l(\mathbf{p}(\mathbf{x}_i))$ is determined as

$$l(\mathbf{p}(\mathbf{x}_i)) = -\mathbf{1}[\delta_i = 1] \cdot \log\left(p_{c(i)}(\mathbf{x}_i)\right) - \mathbf{1}[\delta_i = 0] \cdot \log\left(\sum_{j=c(i)+1}^{T} p_j(\mathbf{x}_i)\right)$$
$$- \gamma \sum_{k=1}^{T} \widetilde{\pi}_k^{(i)} \cdot \log\left(\widetilde{\pi}_k^{(i)}\right). \tag{14}$$

For the i-th instance, $\mathbf{p}(\mathbf{x}_i) = (p_1(\mathbf{x}_i), ..., p_T(\mathbf{x}_i))$ represents a precise probability distribution over time intervals, computed at each epoch using $\mathbf{W}$ and $\widetilde{\pi}_k^{(i)}$ via equation (10). The hyper-parameter γ controls the influence of the regularization term in the loss function (14), which is

defined as the entropy of the probability distribution $\widetilde{\pi}^{(i)}$. This regularization prevents iSurvJ from overconfident decision-making during the joint optimization of weights and probabilities.

An implementation of training the model iSurvJ is presented as Algorithm 1.

Algorithm 1 The training algorithm for iSurvJ [14]

Require: Training set $\mathcal{A}$; hyperparameters p_{mask}, T, d; the number of epochs E
Ensure: Weight matrix $\mathbf{W}$; probability distribution $\mathbf{p}(\mathbf{x}_i)$
1: Initialize randomly $\mathbf{W}_Q$, $\mathbf{W}_K$ and compute $\mathbf{W}$
2: Initialize logits of $\widetilde{\pi}_k$, $k = 1, ..., T$, as zeros
3: **while** Number of epochs $\leq E$ **do**
4: Compute $\widetilde{\pi}_k$, $k = 1, ..., T$, using the softmax applied to logits y
5: Compute the distribution $\mathbf{p}(\mathbf{x}_i)$ using $\mathbf{W}$ and $\widetilde{\pi}_k$, $k = 1, ..., T$
6: Learn $\mathbf{W}$ and $\widetilde{\pi}_k$, $k = 1, ..., T$ using (14) and (13)
7: Compute logits of $\widetilde{\pi}_k$, $k = 1, ..., T$
8: **end while**

It is important to point out that the constraints for some probabilities in the form of intervals $[0, 1]$ are automatically fulfilled due to the use of the softmax operation. It always produces a probability distribution.

3. Reducing Imprecision of Probabilities

3.1. Precise Estimates of Interval Probabilities

One of the problems of using the above approach is that the probability distributions $\pi^{(i)}$ are too imprecise if the i-th observation is censored. In order to reduce the imprecision, it is proposed to apply the imprecise ϵ-contaminated model. However, if we do not exploit any additional information about the i-th object behavior, it is impossible to reduce the probability intervals $[0, 1]$ for $\pi_j^{(i)}$. One of the ways to overcome this difficulty is to use a statistical model constructed from other observations available in the dataset $\mathcal{A}$. It should be noted that the probability distributions $\pi^{(i)}$ corresponds to the i-th object and does not take into account other objects because they have different feature vectors. There are several approaches to take into account the difference of the feature vectors. All the approaches are based on a common idea.

Let us consider the probabilities $\pi_j^{(i)}$ for $j \geq k$ when the corresponding event will occur after the interval τ_k (an uncensored observation). Since the observation is censored at t_k, we know that the random time to event X is satisfied to the condition $X > t_k$. Thus, the conditional probability that the event will occur in the interval $\tau_j = [t_{j-1}, t_j]$ for $j > k$ is of the form:

$$\pi_j^{(i)} = P(t_{j-1} \leq X_i \leq t_j \mid X_i > t_k) = \frac{\widetilde{S}(t_{j-1} \mid \mathbf{x}_i) - \widetilde{S}(t_j \mid \mathbf{x}_i)}{\widetilde{S}(t_k \mid \mathbf{x}_i)}, \tag{15}$$

where $\widetilde{S}(t \mid \mathbf{x}_i)$ is an estimate of the conditional SF for the i-th object.

It is important to note that $\pi_j^{(i)} = 0$ for all $j < k$ and there holds:

$$\sum_{j=1}^{T} \pi_j^{(i)} = 1. \tag{16}$$

SFs $\widetilde{S}(t_j \mid \mathbf{x}_s)$ can be obtained in several ways. One of the ways taking into account relationships between feature vectors is the Beran estimator [2] which estimates the SF as follows:

$$\widetilde{S}(t \mid \mathbf{x}) = \prod_{t_i \le t} \left\{ 1 - \frac{\alpha(\mathbf{x}, \mathbf{x}_i)}{1 - \sum_{j=1}^{i-1} \alpha(\mathbf{x}, \mathbf{x}_j)} \right\}^{\delta_i}, \tag{17}$$

the weight $\alpha(\mathbf{x}, \mathbf{x}_i)$ corresponds to location of $\mathbf{x}_i$ relative to $\mathbf{x}$ under condition that the closer $\mathbf{x}_i$ to $\mathbf{x}$, the greater the weight $\alpha(\mathbf{x}, \mathbf{x}_i)$.

Weights are defined through kernels as

$$\alpha(\mathbf{x}, \mathbf{x}_i) = \frac{K(\mathbf{x}, \mathbf{x}_i)}{\sum_{j=1}^{n} K(\mathbf{x}, \mathbf{x}_j)}. \tag{18}$$

If $K(\mathbf{x}, \mathbf{x}_i)$ is the Gaussian kernel with a parameter ω, then weights are of the form:

$$\alpha(\mathbf{x}, \mathbf{x}_i) = \text{softmax}\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{\omega} \right). \tag{19}$$

It should be noted that the set $\mathcal{A}$ can be restricted to the M nearest neighbors of $\mathbf{x}$ to simplify computations. In this case, SFs $\widetilde{S}(t_j \mid \mathbf{x}_i)$ can be efficiently computed by fitting a Kaplan-Meier model to these M neighbors. This implementation corresponds precisely to the Beran estimator with uniform weighting.

3.2. Imprecise Statistical Models

3.2.1. Imprecise contaminated model

The linear-vacuous mixture, or imprecise ϵ-contaminated model [21], produces a set $\mathcal{P}(\epsilon, \pi)$ of the probability distributions $\pi^* = (\pi_1^*, ..., \pi_T^*)$ such that $\pi_j^* = (1 - \epsilon)\pi_j + \varepsilon \cdot h_i$, where $\pi = (\pi_1, ..., \pi_T)$ is an elicited probability distribution, $\mathbf{h} = (h_1, ..., h_T)$ is an arbitrary probability distribution, and $\epsilon \in [0, 1]$ is a model contamination parameter which reflects imprecision of the model. The set $\mathcal{P}(\epsilon, \pi)$ is a subset of the unit simplex with the dimension d. Moreover, it coincides with the unit simplex when $\epsilon = 1$. The ϵ-contaminated model for the i-th object can be represented as

$$\pi^{*(i)} = (1 - \epsilon) \cdot \pi^{(i)} + \epsilon \cdot \mathbf{h}. \tag{20}$$

Probabilities $\pi_j^{(i)}$, $j = k, ..., T$, are defined through (15). Using the imprecise ϵ-contaminated model and the definition of $\pi_j^{(i)}$ for uncensored observations given in (15), we can reduce the interval $[0, 1]$ in (6) and control the obtained interval by means of the contamination parameter ϵ. The lower probability of $\pi_j^{(i)}$ is determined under condition $h_j = 0$ as follows:

$$\underline{\pi}_j^{*(i)} = (1 - \epsilon) \cdot \pi_j^{(i)}. \tag{21}$$

The upper probability of $\pi_j^{(i)}$ is determined under condition $h_j = 1$ as follows:

$$\overline{\pi}_j^{*(i)} = (1 - \epsilon) \cdot \pi_j^{(i)} + \epsilon. \tag{22}$$

The lower and upper predicted class probabilities $p_1, ..., p_T$ from (7) are determined as

$$\underline{p}_k(\mathbf{x}_0) = \sum_{i=1}^{N} w_{0,i} \underline{\pi}_k^{(i)}, \ \overline{p}_k(\mathbf{x}_0) = \sum_{i=1}^{N} w_{0,i} \overline{\pi}_k^{(i)} \ k = 1, ..., T. \tag{23}$$

However, according to Algorithm 1, we do not need to compute the lower and upper class probabilities. Algorithm 1 could be applied to compute the SF for a new subject without these bounds. However, it assumes that elements of the probability distributions are restricted by the interval $[0, 1]$ to simply use of the softmax operation. An advantage of the imprecise ϵ-contaminated model is that the probability distribution $\mathbf{h}$ in 20 belongs to the unit simplex. Hence, the i-th elements $\widetilde{\pi}_k^{(i)}$ of learned vectors $\widetilde{\pi}_k$, $k = 1, ..., T$, in Step 4 of Algorithm 1 are replaced with

$$\pi_k^{*(i)} = (1 - \epsilon) \cdot \pi_k^{(i)} + \epsilon \cdot \widetilde{\pi}_k^{(i)}. \tag{24}$$

Therefore, Algorithm 1 is extended by additional calculations of probabilities $\pi_k^{*(i)}$ through learned probabilities $\widetilde{\pi}_k^{(i)}$ and probabilities $\pi_k^{(i)}$ obtained by using (15) and the Beran estimator (17).

4. Numerical Experiments

The main goal of numerical experiments is to show that the proposed model can provide better results for certain values of the contamination parameter ϵ in comparison with the model iSurvJ which can be regarded as a special case of CiSurv under condition $\epsilon = 1$. The model iSurvJ has been compared with different survival models and demonstrated outperforming results [14]. Therefore, we show that the contamination parameter ϵ impacts on the model CiSurv performance.

4.1. Conditions of Experiments

For research purposes, we consider two modifications of the proposed model: CiSurvN and CiSurvG. The first model, CiSurvN, uses the neural network to compute the matrix $\mathbf{W}$. The model CiSurvG uses the Gaussian kernels with a single parameter for computing $\mathbf{W}$. On the one hand, the model CiSurvN is more powerful due to using the neural network. On the other hand, when a dataset is rather small, the model CiSurvG may provide better results due to a small number of training parameters.

20 independent experiments are conducted for models CiSurvN and CiSurvG. In each experiment, the dataset is diveded into training and test sets in a 60% to 40% ratio. The well-established metric IBS is used for comparison. The resulting metric values are averaged over 20 runs.

In all experiments, the following values of hyperparameters are used for both models: number of training iterations: 300; learning rate: 10^{-2}; regularization: $\alpha = 2 \cdot 10^{-3}$; entropy regularization coefficient: $\gamma = 2.0$; kernel parameter (temperature) in the Beran estimator: 0.4; batch size: 100% of the training set.

Prior to model training, all numerical features are standardized using z-score normalization, while categorical features are encoded using one-hot encoding to ensure compatibility with the machine learning algorithms.

4.2. Results of Experiments: Real Datasets

To evaluate the performance of the proposed model, we study several publicly available survival datasets: Veterans, Breast cancer, LND, GCD. These datasets from medical observations are used in experiments instead of real data with times to the successful execution of tasks in

Table 1. IBS obtained for real datasets by using CiSurvN for different ϵ

ϵ	Veterans	Breast cancer	LND	GCD
0.0	0.0993	0.2414	0.2574	0.1984
0.1	0.0990	0.2268	0.2491	0.1936
0.2	0.0982	0.2101	0.2290	0.1907
0.3	**0.0975**	0.2010	0.2283	0.1864
0.4	0.0980	**0.1924**	0.2273	0.1855
0.5	0.0993	0.2015	0.2175	0.1844
0.6	0.0993	0.2022	**0.2120**	0.1843
0.7	0.0993	0.2053	0.2234	0.1820
0.8	0.0993	0.2153	0.2288	**0.1785**
0.9	0.0994	0.2160	0.2300	0.1845
1.0	0.0995	0.2166	0.2303	0.1867

HPC systems due to two reasons. First, data on the success or failure of tasks on HPC systems is generally unique and typically inaccessible to many users and readers. This prevents the requirement for reproducibility of results, as the model parameters are tuned to specific data. Second, real datasets from observations of task solving on HPC systems are quite large and have complex structures, which significantly complicate or make impossible the study of features of the proposed models. Therefore, we test and investigate CiSurvN and CiSurvG on the open datasets having restricted sizes.

The *Veterans' Administration Lung Cancer Study (Veteran) Dataset* includes data on 137 instances characterized by 6 features. It is available through the R package "survival".

The *Breast Cancer Dataset* consists of 198 samples described by 80 features. It can be obtained from the Python library "scikit-survival".

The *Lupus Nephritis Dataset (LND)* consists of 87 observations. The dataset is available at `http://www.stat.rice.edu/~sneeley/STAT553/Datasets`.

The *Gastric Cancer Dataset (GCD)* includes 90 observations with 4 features. It is available through the R package "coxphw".

Table 1 shows IBS obtained for real datasets by using the CiSurvN modification for different values of ϵ from 0 to 1. The best metric values are highlighted in bold. It can be seen from Tab. 1 that there exist "optimal" values of ϵ different from 1, which correspond to the smallest values of IBS. This implies that incorporation of the imprecise ϵ-contaminated model into CiSurvN may lead to outperforming results.

Figure 1 illustrates how IBS depends on the parameter ϵ for some datasets when the CiSurvN modification of CiSurv is used.

Table 2 shows the IBS metrics obtained for real datasets by using the CiSurvG modification of CiSurv for different values of ϵ from 0 to 1. It is interesting to point out that the results differ from the results obtained by using CiSurvG and presented in Tab. 1. First of all, one can see from Tab. 2 that CiSurvG provides worse results in comparison with CiSurvN because only one training parameter of the attention weight is used (the parameter of the Gaussian kernel). Second, we can see that some datasets show the best IBS in "extreme" cases when the contamination parameter ϵ is 1 or close to 1. Figure 2, where the dependence of IBS on the parameter ϵ is shown for real datasets using CiSurvG, illustrates this peculiarity of results.

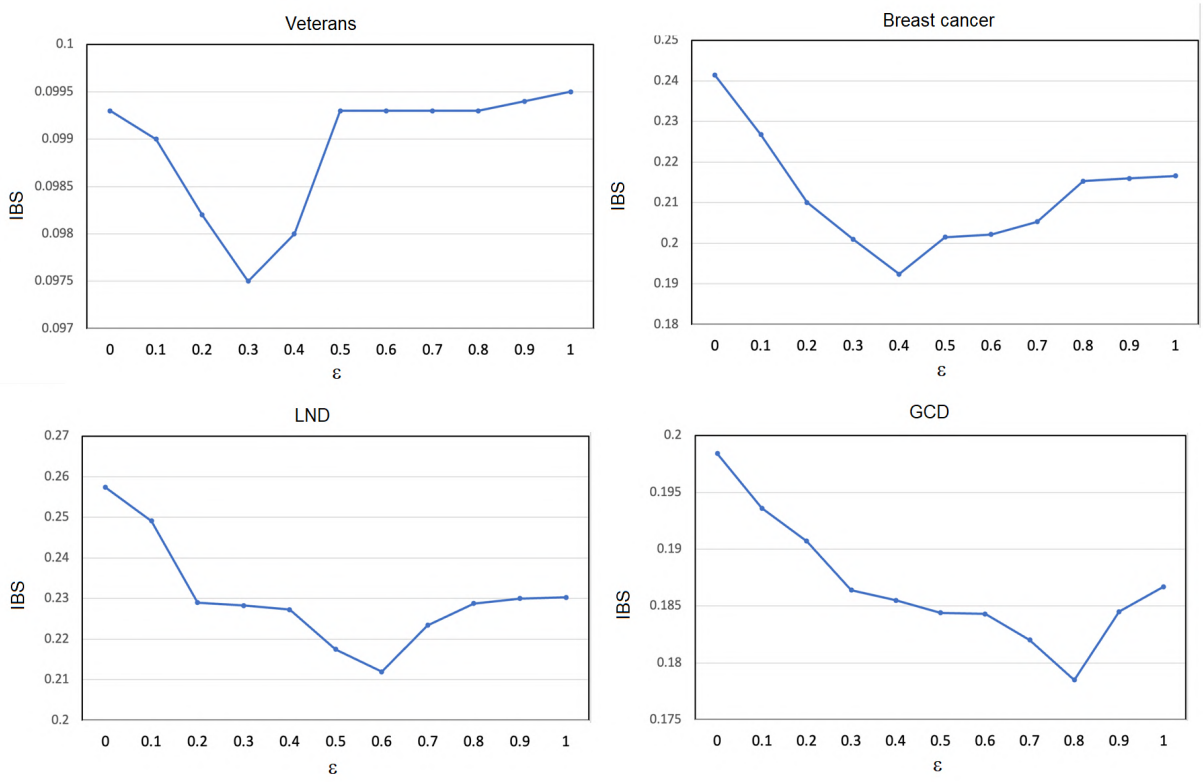


Figure 1. IBS metrics as functions of the contamination parameter ϵ for some real datasets obtained by using CiSurvN

Table 2. IBS obtained for real datasets by using CiSurvG for different ϵ

ϵ	Veterans	Breast cancer	LND	GCD
0.0	0.1585	0.3378	0.2867	0.2089
0.1	0.1571	0.3285	0.2802	0.2037
0.2	0.1560	0.3216	0.2750	0.2004
0.3	0.1550	0.3167	**0.2706**	0.1980
0.4	0.1542	0.3136	**0.2690**	0.1964
0.5	0.1536	**0.3124**	0.2698	0.1952
0.6	0.1531	0.3128	0.2706	0.1944
0.7	0.1527	0.3152	0.2714	0.1932
0.8	0.1524	0.3193	0.2723	0.1926
0.9	0.1522	0.3245	0.2730	**0.1925**
1.0	**0.1521**	0.3297	0.2734	0.1928

4.3. Synthetic Datasets

To rigorously benchmark the capabilities of the proposed survival analysis models under diverse and adversarially designed conditions, we construct synthetic datasets Linear, Quadratic, Friedman 1, Noisy. These datasets incorporate structured complexity, including non-linear de-

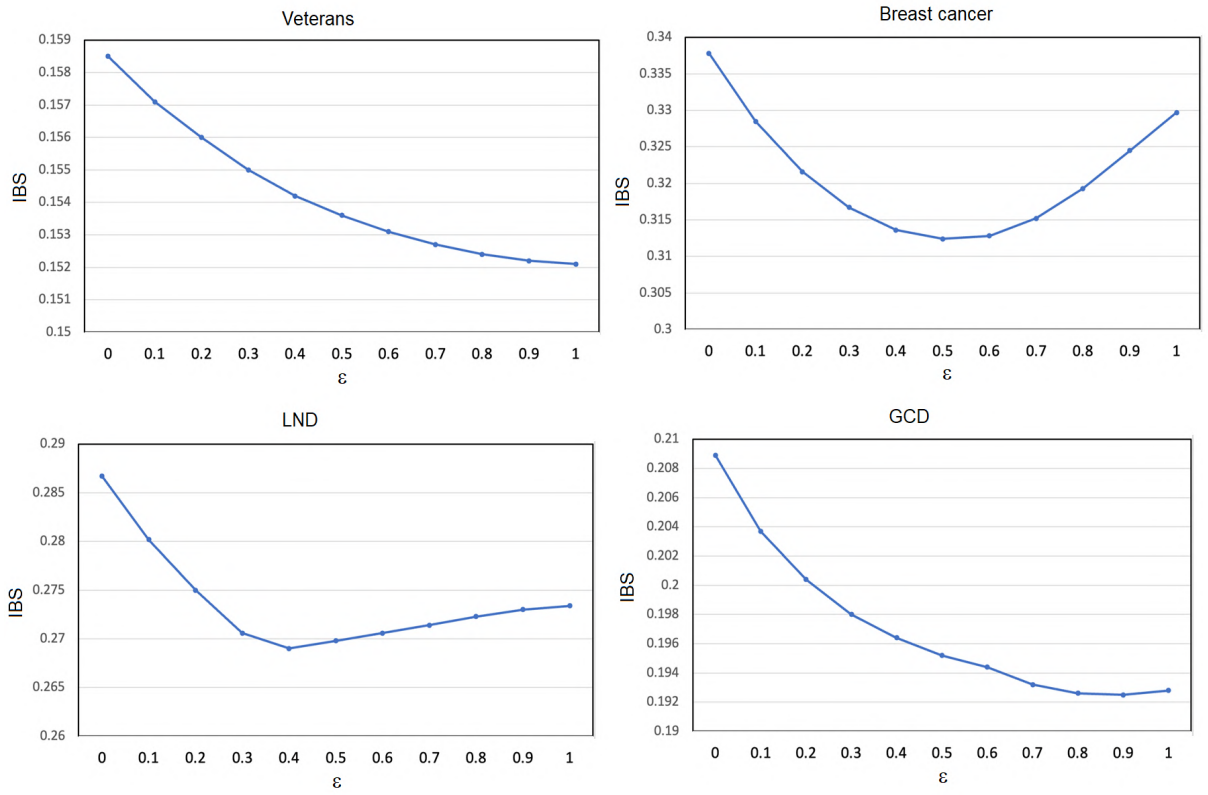


Figure 2. IBS metrics as functions of the contamination parameter ϵ for some real datasets obtained by using CiSurvG

pendencies, feature interactions, and noise regimes. Let $\mathbf{x} = (x_1, x_2, \ldots, x_d)^\top \in \mathbb{R}^d$ be a feature vector.

For all synthetic datasets, censoring indicators $\delta_i \in \{0, 1\}$ are generated as independent Bernoulli random variables with fixed probabilities $\Pr\{\delta_i = 0\}$ (censored) and $\Pr\{\delta_i = 1\}$ (observed). Event times T are sampled from a Weibull distribution with shape parameter $k > 0$ and are of the form:

$$T = \frac{y}{\Gamma\left(1 + \frac{1}{k}\right)} \cdot (-\log(u))^{\frac{1}{k}}, \tag{25}$$

where $u \in \mathcal{U}(0, 1)$ is a random number uniformly distributed in the interval $[0, 1]$; y is a function of features, which is defined for each dataset and shown below; $\Gamma(\cdot)$ is the gamma function.

The *Friedman1* dataset defines y as:

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \tag{26}$$

where $x_i \sim \mathcal{U}(0, 1)$ for $i \leq 5$, the remaining $d - 5$ features (if $d > 5$) are considered as noise variables and do not influence y.

The *Noisy* dataset generates T by applying the function $y = X\mathbf{w}$ and $k \leq 1$. The parameter k leads to high-variance noise. Here $X \in \mathbb{R}^{n \times d}$ is a feature matrix with elements $x_{ij} \sim \mathcal{U}(0, 1)$; $\mathbf{w} \in \mathbb{R}^d$ is a coefficient vector whose elements are generated as: $w_j \sim \mathcal{U}(0, 1)$.

Table 3. IBS obtained for synthetic datasets by using CiSurvN for different ϵ

ϵ	Linear	Quadratic	Friedman1	Noisy
0.0	0.2021	0.2892	0.1336	0.1208
0.1	0.1924	0.2783	0.1310	0.1175
0.2	0.1870	0.2741	0.1240	0.1161
0.3	0.1892	0.2601	0.1202	0.1162
0.4	0.1848	0.2622	0.1160	0.1152
0.5	0.1824	**0.2538**	0.1142	0.1120
0.6	0.1839	0.2561	**0.1139**	**0.1107**
0.7	0.1826	0.2541	0.1144	0.1103
0.8	0.1811	0.2580	0.1158	0.1136
0.9	**0.1780**	0.2569	0.1164	0.1177
1.0	0.1825	0.2574	0.1175	0.1190

The *Linear* dataset uses the linear function:

$$y = \sum_{i=1}^{d} w_i x_i = \mathbf{x}^\top \mathbf{w}, \tag{27}$$

where $x_i \sim \mathcal{U}(0,1)$; $\mathbf{w} = (w_1, w_2, \ldots, w_d)^\top \in \mathbb{R}^d$ is a vector of weights, $w_i \sim \mathcal{U}(0,1)$.

The *Quadratic* dataset implements the quadratic dependency between features such that $y = \mathbf{x}^\top Q \mathbf{x}$, where $x_i \sim \mathcal{U}(0,1)$, $Q \in \mathbb{R}^{d\times d}$ is a symmetric positive semi-definite matrix defined as $Q = A^\top A$, $A \in \mathbb{R}^{d\times d}$ is a matrix with normally distributed elements $a_{ij} \sim \mathcal{N}(0,1)$.

In each experiment, a new training set of 500 samples with 5 features and the censoring rate 0.2 is generated. The test set is the same across all experiments and consisted of 300 samples with the same censoring rate. In all other aspects, the experimental procedure is identical to that used for the real datasets.

Numerical results provided by CiSurvN with synthetic data are given in Tab. 3. The results are depicted in Fig. 3. One can see that from the results that there exist "optimal" values of ϵ between 0 and 1, corresponding to the smallest values of IBS. Numerical results provided by the model CiSurvG with synthetic data are given in Tab. 4. The same results are depicted in Fig. 4. It can be seen from Tab. 4 and Fig. 4 that, for half of the datasets, experimental results obtained using the CiSurvG model show that the optimal value of ϵ is 1. This indicates that with a small number of trainable parameters, the model attempts to relax constraints on the probability distribution set in the ϵ-contaminated model to allow greater freedom for the parameter of the Gaussian kernel.

Experiments conducted on real and synthetic datasets demonstrate the effectiveness of the proposed CiSurv model and its two modifications, CiSurvN and CiSurvG. For the CiSurvN model, which utilizes a neural network to compute the weight matrix $\mathbf{W}$, experiments showed that values of the parameter ϵ (typically between 0.3 and 0.7) yielded the best performance across most datasets. This confirms a hypothesis that the ϵ-contaminated model can indeed improve upon the special case of iSurvJ for which $\epsilon = 1$.

The improvement in the IBS scores across multiple datasets when using optimal values of ϵ provides strong evidence for the value of the contamination model framework in survival analysis. The numerical results demonstrate that the proposed CiSurv framework, particularly the

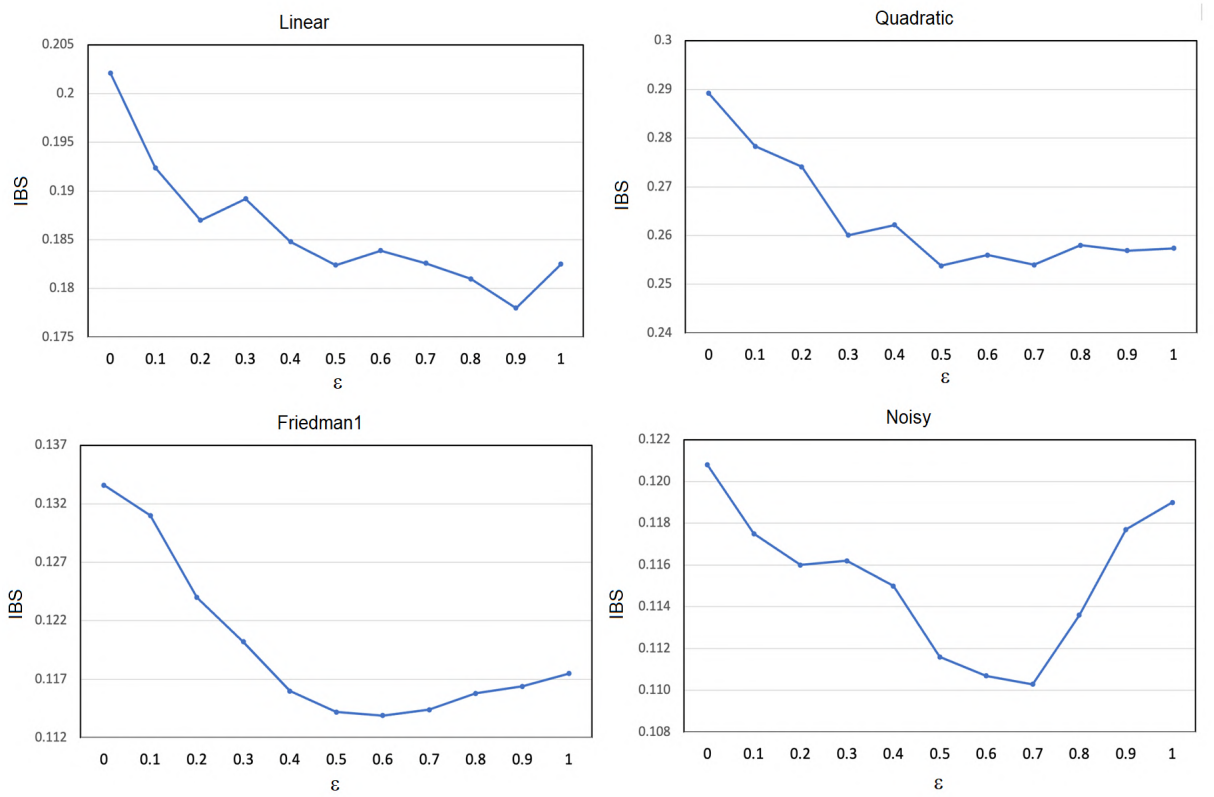


Figure 3. IBS metrics as functions of the contamination parameter ϵ for some synthetic datasets obtained by using CiSurvN

Table 4. IBS obtained for synthetic datasets by using CiSurvG for different ϵ

ϵ	Linear	Quadratic	Friedman1	Noisy
0.0	0.2032	0.3264	0.1222	0.1322
0.1	0.2018	0.2961	0.1210	0.1263
0.2	0.2008	0.2741	0.1194	0.1231
0.3	0.1998	0.2670	0.1175	0.1213
0.4	0.1989	0.2650	0.1166	0.1209
0.5	0.1982	0.2632	0.1157	0.1207
0.6	0.1976	0.2601	**0.1151**	0.1205
0.7	0.1971	0.2584	0.1154	0.1203
0.8	0.1968	0.2578	0.1157	0.1202
0.9	0.1967	**0.2569**	0.1161	0.1202
1.0	**0.1965**	0.2574	0.1170	**0.1201**

CiSurvN variant, offers an approach to survival analysis that can adapt to different data characteristics through appropriate selection of the contamination parameter ϵ. The findings suggest that relaxing the strict assumptions of traditional survival models through the ϵ-contaminated approach can lead to improved predictive performance.

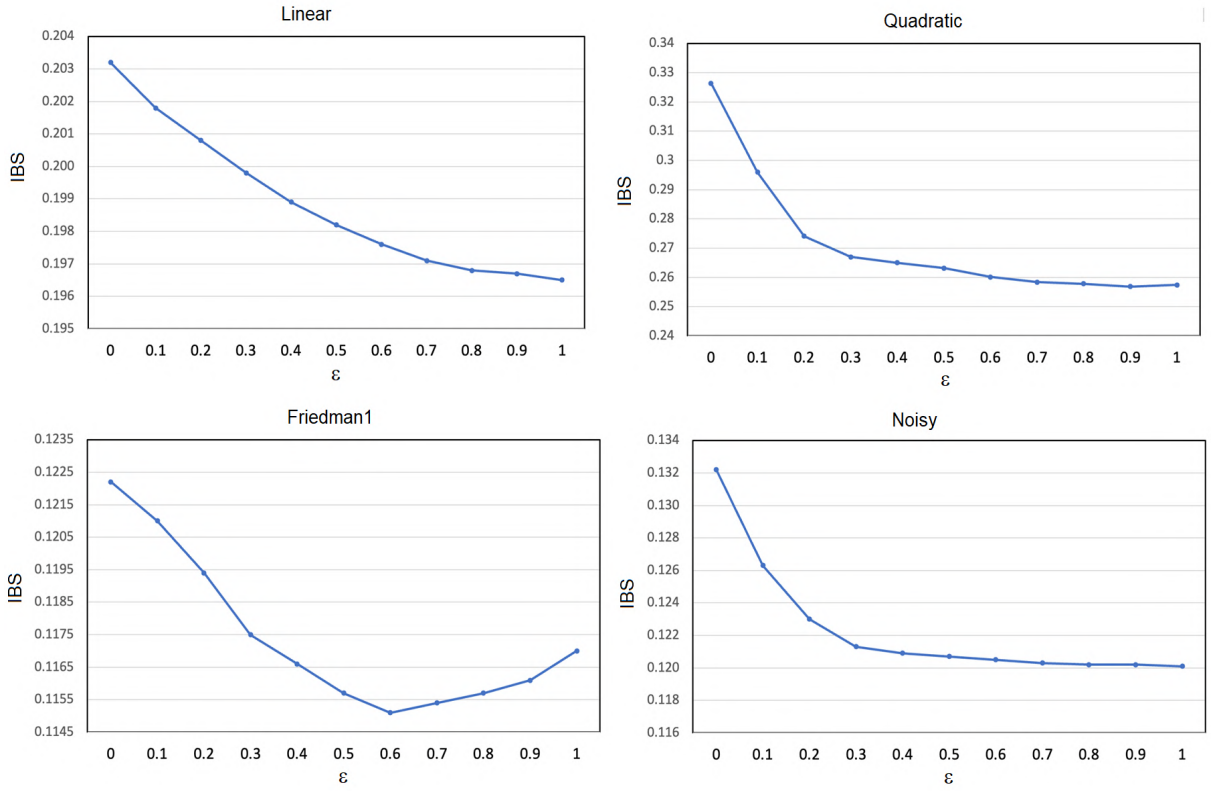


Figure 4. IBS metrics as functions of the contamination parameter ϵ for some synthetic datasets obtained by using CiSurvG

Conclusion

The proposed CiSurv framework introduces a novel approach to survival analysis by integrating the imprecise ϵ-contaminated model into an attention-based multi-label classification method. This reformulation of survival analysis as an imprecise probability problem provides a flexible and robust solution for time-to-event prediction, particularly in scenarios involving censored data.

The main contributions of this work include the successful integration of imprecise probability theory through the ϵ-contaminated model, which reduces uncertainty in censored observations by refining interval-valued probabilities. This refinement leads to more accurate survival predictions compared to the existing iSurvJ approach, where $\epsilon = 1$.

Numerical experiments confirmed that intermediate values of ϵ often yield the best predictive performance. The results have illustrated that modeling uncertainty explicitly via the ϵ-contaminated framework offers a promising pathway for developing robust, flexible survival analysis tools capable of managing real-world data heterogeneity and uncertainty. It should be noted that the neural network-based CiSurvN tends to outperform simpler kernel-based variants, underscoring the benefit of increased model capacity in complex data regimes.

The model capacity to handle imprecision in probabilistic estimates makes it well-suited for real-world applications characterized by data uncertainty, limited sample sizes, or complex censoring patterns. Extending CiSurv to incorporate other imprecise probability models, for example, the pari-mutuel model, the constant odds-ratio model [21] could enhance its applicability.

This is a direction for further research. It is interesting to extend the approach to competing risks and time-varying covariates. These are also directions for further research.

The proposed survival analysis model CiSurv and its variants CiSurvN and CiSurvG can be employed to predict system reliability and proactively schedule maintenance in HPC environments minimizing downtime. By modeling task execution times and failure probabilities, CiSurv enhances resource allocation and workload management in high-performance computing systems. The imprecise probabilistic framework enables more accurate failure prediction under uncertain and incomplete HPC system monitoring data. Finally, applying CiSurv can improve fault-tolerance strategies in HPC, ensuring more resilient operation and optimal utilization of computational resources.

Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation (topic FSEG-2024-0027).

References

1. Abbasi, A., Asim, M., Ahmed, S., *et al.*: Survival prediction landscape: an in-depth systematic literature review on activities, methods, tools, diseases, and databases. Frontiers in Artificial Intelligence 7, 1428501 (2024). https://doi.org/10.3389/frai.2024.1428501

2. Beran, R.: Nonparametric regression with randomly censored survival data. Tech. rep., University of California, Berkeley (1981).

3. Brier, G.: Verification of forecasts expressed in terms of probability. Monthly Weather Review 78(1), 1–3 (1950). https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2

4. Chen, G.H.: An introduction to deep survival analysis models for predicting time-to-event outcomes. Foundations and Trends® in Machine Learning 17(6), 921–1100 (2024). https://doi.org/10.1561/2200000114

5. Coolen, F.: An imprecise Dirichlet model for Bayesian analysis of failure data including right-censored observations. Reliability Engineering and System Safety 56, 61–68 (1997). https://doi.org/10.1016/S0951-8320(96)00131-7

6. Coolen, F., Yan, K.: Nonparametric predictive inference withright-censored data. Journal of Statistical Planning and Inference 126, 25–54 (2004). https://doi.org/10.1016/j.jspi.2003.07.004

7. Craig, E., Zhong, C., Tibshirani, R.: A review of survival stacking: a method to cast survival regression analysis as a classification problem. The International Journal of Biostatistics 21(1), 37–51 (2025). https://doi.org/10.1515/ijb-2022-0055

8. Emmert-Streib, F., Dehmer, M.: Introduction to survival analysis in practice. Machine Learning & Knowledge Extraction 1, 1013–1038 (2019). https://doi.org/10.3390/make1030058

9. Graf, E., Schmoor, C., Sauerbrei, W., Schumacher, M.: Assessment and comparison of prognostic classification schemes for survival data. Statistics in Medicine 18(17-18), 2529–2545 (1999). https://doi.org/10.1002/(SICI)1097-0258(19990915/30)18:17/18<2529::AID-SIM274>3.0.CO;2-5

10. Hosmer, D., Lemeshow, S., May, S.: Applied Survival Analysis: Regression Modeling of Time to Event Data. John Wiley & Sons, New Jersey (2008).

11. Hu, S., Fridgeirsson, E., van Wingen, G., Welling, M.: Transformer-based deep survival analysis. In: Survival Prediction-Algorithms, Challenges and Applications. pp. 132–148. PMLR (2021).

12. Ishwaran, H., Kogalur, U.: Random survival forests for r. R News 7(2), 25–31 (2007).

13. Jiang, S., Suriawinata, A.A., Hassanpour, S.: MHAttnSurv: Multi-head attention for survival prediction using whole-slide pathology images. Computers in Biology and Medicine 158, 106883 (2023). https://doi.org/10.1016/j.compbiomed.2023.106883

14. Konstantinov, A., Utkin, L., Efremenko, V., *et al.*: Survival analysis as imprecise classification with trainable kernels. Mathematics 13(18), 3040 (2025). https://doi.org/10.3390/math13183040

15. Kvamme, H., Borgan, Ø.: Continuous and discrete-time survival prediction with neural networks. Lifetime Data Analysis 27(4), 710–736 (2021). https://doi.org/10.1007/s10985-021-09532-6

16. Luong, T., Pham, H., Manning, C.: Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1412–1421. Association for Computational Linguistics, Lisbon, Portugal (2015). https://doi.org/10.18653/v1/d15-1166

17. Nadaraya, E.: On estimating regression. Theory of Probability & Its Applications 9(1), 141–142 (1964). https://doi.org/10.1137/1109020

18. Softic, S., Hrnjica, B.: Case studies of survival analysis for predictive maintenance in manufacturing. International Journal of Industrial Engineering and Management 15(4), 320–337 (2024). https://doi.org/10.24867/IJIEM-2024-4-366

19. Suresh, K., Severn, C., Ghosh, D.: Survival prediction models: an introduction to discrete-time modeling. BMC Medical Research Methodology 22(1), 207 (2022). https://doi.org/10.1186/s12874-022-01679-6

20. Vaswani, A., Shazeer, N., Parmar, N., *et al.*: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017).

21. Walley, P.: Statistical Reasoning with Imprecise Probabilities. Chapman and Hall, London (1991).

22. Walley, P.: Inferences from multinomial data: Learning about a bag of marbles. Journal of the Royal Statistical Society, Series B 58, 3–34 (1996). https://doi.org/10.1111/j.2517-6161.1996.tb02065.x

23. Wang, P., Li, Y., Reddy, C.: Machine learning for survival analysis: A survey. ACM Computing Surveys (CSUR) 51(6), 1–36 (2019). https://doi.org/10.1145/3214306

24. Wang, Y., Kong, X., Bi, X., *et al.*: ResDeepSurv: A survival model for deep neural networks based on residual blocks and self-attention mechanism. Interdisciplinary Sciences: Computational Life Sciences 16(2), 405–417 (2024). https://doi.org/10.1007/s12539-024-00617-y

25. Wang, Z., Sun, J.: SurvTRACE: Transformers for survival analysis with competing events. In: Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. pp. 1–9 (2022). https://doi.org/10.1145/3535508.3545521

26. Watson, G.: Smooth regression analysis. Sankhya: The Indian Journal of Statistics, Series A, 359–372 (1964).

27. Wiegrebe, S., Kopper, P., Sonabend, R., *et al.*: Deep learning for survival analysis: a review. Artificial Intelligence Review 57(65), 1–34 (2024). https://doi.org/10.1007/s10462-023-10681-3

28. Zhang, X., Mehta, D., Hu, Y., *et al.*.: Adaptive transformer modelling of density function for nonparametric survival analysis. Machine Learning 114(2), 31 (2025). https://doi.org/10.1007/s10994-024-06686-w

DOI: 10.14529/jsfi260208

An Analog Photonic Computing Device Based on a Diffractive Neural Network for Video Stream Processing

Roman V. Skidanov[1], ***Leonid L. Doskolovich***[1], ***Nikolay L. Kazanskiy***[1], ***Alexandr E. Morozov***[1], ***Alexey S. Pronin***[1], ***Daniil M. Sorokin***[1], ***Yuriy V. Khanenko***[1], ***Victor A. Soifer***[1]

This paper investigates the design principles of analog photonic computing devices for pattern recognition tasks. As a result of the conducted research, an analog photonic computing device (APCD) based on a diffractive neural network (DNN) is developed and implemented in two configurations. The DNN is realized using a phase spatial light modulator (SLM) placed in the Fourier plane of a two-lens optical system (4F system). Benefits of combining an optical DNN with a compact computer neural network for post-processing the optical recognition results are demonstrated. The presented results show the effectiveness of using the APCD for processing and recognition of high-dimensional imagery in real-time video streams. A comparative performance analysis of the APCD and a family of GPU-based YOLO neural networks is conducted for object image recognition in video streams. The results demonstrate that when processing high-dimensional frames, the power consumption of the APCD is an order of magnitude lower than that of modern GPU-class graphics platforms such as the RTX 4090.

Keywords: optical computing, diffractive neural network, image processing and classification, spatial light modulator, YOLO neural networks, video-streams.

Introduction

Artificial neural networks have achieved significant success in a wide range of machine learning tasks, including image and video classification and analysis, medical diagnostics, speech recognition, solving various inverse problems, and so on. At the same time, the training and inference processes of neural networks used, for example, in modern models of video imagery recognition and analysis impose enormous demands on digital processors in terms of computational volume and speed, which cannot always be met. These constraints have driven the rapid advancement of a research field focused on the optical implementation of neural networks [1–3]. It should be noted that this line of research lies within the general trend of augmenting or substituting digital computing devices with their optical analogs, which are expected to overcome the limitations of digital computing systems related to insufficient computational speed and high energy consumption [4]. As shoun in Fig. 1, interest in this research area has grown exponentially since 2017 The data presented in demonstrate an exponential growth in interest in this research direction since 2017.

Among optical neural networks, particular interest is attracted by so-called diffractive neural networks (DNNs), which are realized as a cascade of phase diffractive optical elements (DOEs) [5–16], (see Fig. 2). It is important to note that the research field concerned with the design of DOEs – including cascaded DOEs – for solving various problems of laser light transformation and focusing has a long history and has been actively evolving for over 50 years [17–24]. At the same time, the use of cascaded DOEs to optically solve machine learning problems was first demonstrated only in 2018 in the seminal work [5]. In that work, several analogies were drawn between a cascade of DOEs and an artificial neural network. Specifically, it was noted that, by the Huygens-Fresnel principle, each pixel of a DOE emits a spherical wave that illumi-

[1]S.P. Korolyov Samara National Research University, Samara, Russian Federation

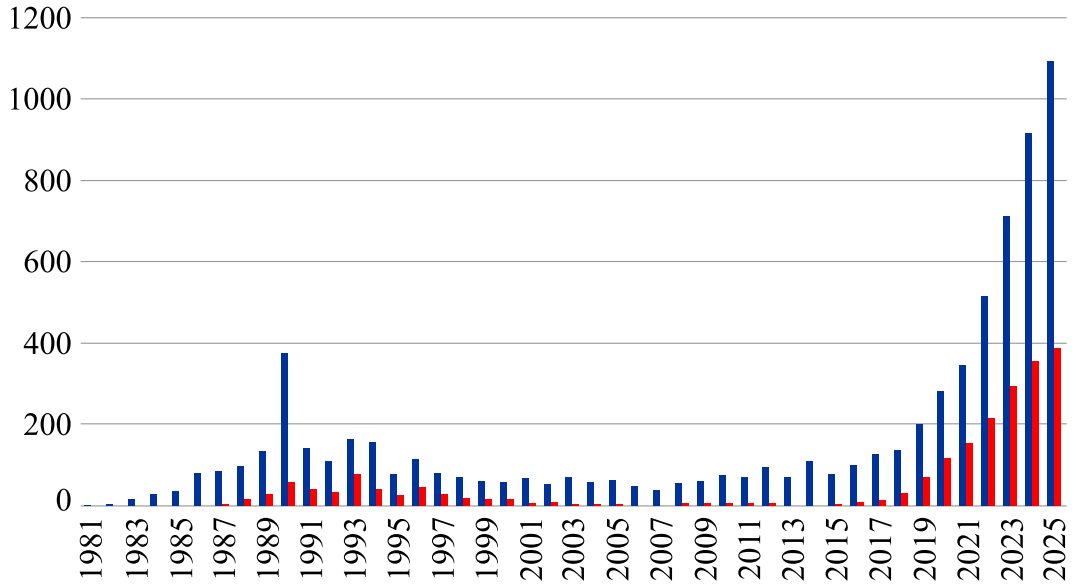


Figure 1. Annual number of publications in the field of optical computing. Blue: total publications in the Scopus database; Red: including publications on the subject of optical neural networks

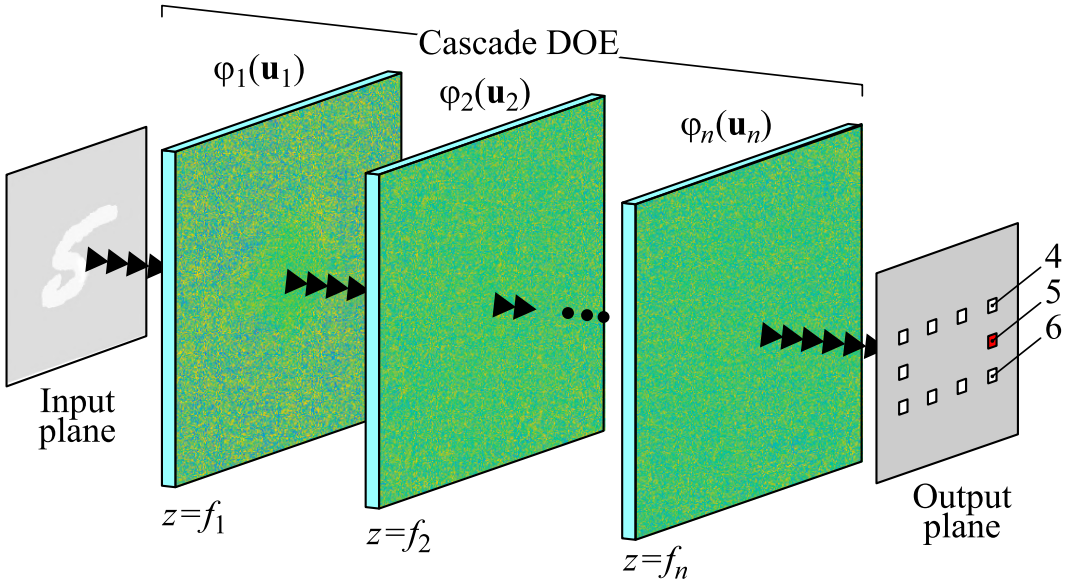


Figure 2. Geometry of a DNN implemented as a cascade of DOEs for solving a classification problem

nates the pixels of the subsequent DOE in the cascade, thereby mimicking the interconnections between neurons in a multilayer perceptron. On this basis, the term "diffractive neural network" was introduced, and the possibility of optically solving classification problems using cascaded DOEs was demonstrated both theoretically and experimentally.

The optical solution to the classification problem is schematically shown in Fig. 2. We assume that optical images belonging to N distinct object classes – for instance, images of handwritten digits – are generated in an input plane by means of an SLM. These images propagate through the optical system (a DOE cascade) towards the output plane. In the output plane, a set of energy detection regions are specified, with the number of regions corresponding to the number

of input image classes. In a particular case, the registration regions can be implemented as an array of photodetectors that measure the total energy of the incident optical signal. The number of the region (photodetector) with the highest energy then determines the class of the input image. By way of example, Fig. 2 schematically illustrates that for an input image of the digit '5', the maximum energy is attained in the region associated with that digit. The above-described operation of the DNN presupposes its prior 'training', which involves solving the inverse problem of determining the phase function of the constituent DOEs such that the desired operational condition is satisfied (i.e., forming the maximum energy in the registration region that corresponds to the class of the input object). Typically, the afore-mentioned inverse problem is solved in the framework of scalar diffraction theory under the thin-element approximation. The phase functions of the DOEs are calculated using a gradient-based method within the artificial neural network methodology, utilizing a training dataset (a set of characteristic input images) [5–16].

It is important to note that in the DNN shown in Fig. 2, the relationship between the fields in the input and output planes is described by a linear operator. Considering that a multilayer neural network without nonlinearities is equivalent to a single-layer linear neural network, the use of a cascade of DOEs instead of a single DOE is not always justified. In particular, the authors of [25] showed that with properly chosen parameters, a single DOE can solve classification tasks with high accuracy, not only matching but even exceeding the classification accuracy achieved with complex cascaded DOEs. Moreover, it is well known that in the physical realization of a DNN designed to operate with visible light, a major challenge is the stringent requirement for precise positioning of the constituent DOEs [10, 12, 15, 16]. Moreover, cascaded DOEs prove to be substantially more sensitive to misalignment than single DOEs, so that with positioning errors of only 10–15 wavelengths, they become inferior to single DOEs in terms of operational performance [15, 16]. In this regard, the present authors consider DNNs implemented as a single DOE as the most suitable option for practical applications. Furthermore, studies conducted by the present authors (not reported in this paper) have shown that the variant most tolerant to positioning errors is the DNN realization in a 4F-system configuration (see Fig. 3), in which a single DOE is placed between two lenses that perform Fourier transforms. This particular DNN configuration is discussed further in this paper.

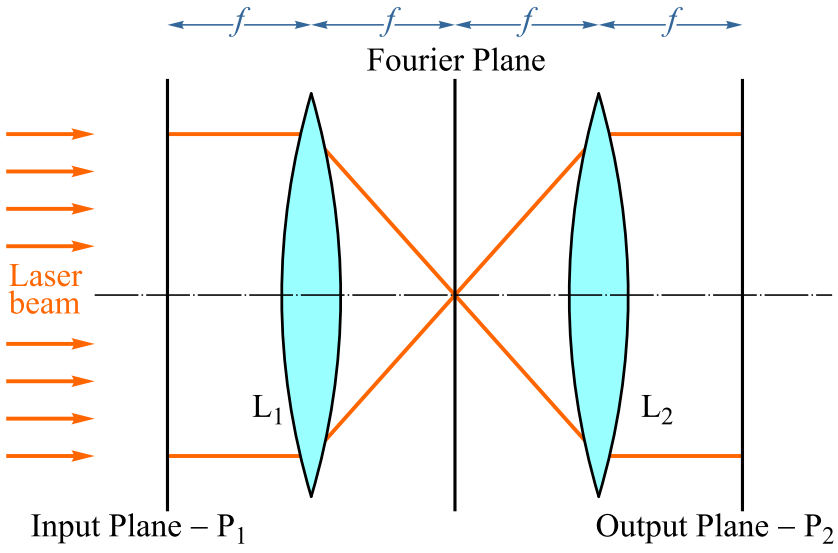


Figure 3. A 4F-system [26] (f are focal lengths of lenses L_1 and L_2)

In this work, we discuss two versions of an analog photonic computing device (APCD) employing a DNN in a 4F-system configuration. These devices were developed at S. P. Korolyov Samara University under a contract with RFNC-VNIIEF in 2024 and 2025. The APCD uses two UPOLabs HDSLM36R liquid-crystal SLMs with a resolution of 4096×2140 and a maximum frame rate of 60 Hz. One modulator serves for image input, whereas the other realizes a diffractive neural network as a single phase-only DOE positioned in the frequency plane. The results of the processing are captured using two DAHENG ME2P-1231-32U3M cameras with a resolution of 4096×3000 and a maximum frame rate of 60 Hz. A more detailed description of the APCDs is provided in Section 1. The methods for designing DOEs employed in the APCD are set forth in [15, 16].

1. APCD-2024 and APCD-2025

In 2024 and 2025, two experimental APCD prototypes based on a 4F-system were designed and studied.

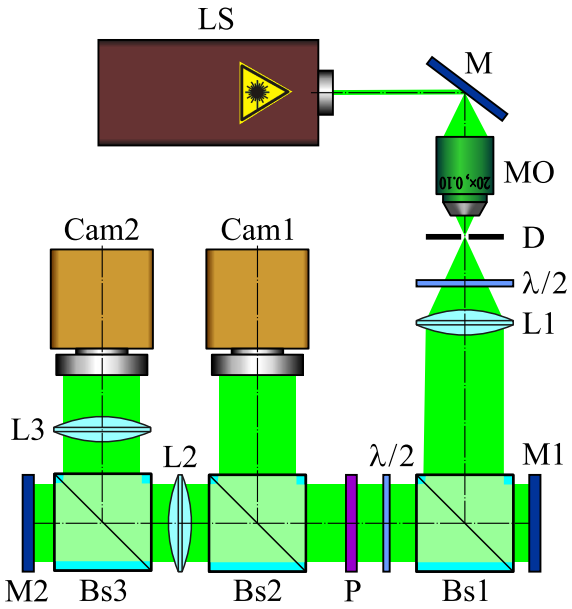
1.1. APCD-2024

Figure 4a depicts a schematic optical arrangement of the developed APCD. The linearly polarized output beam from a solid-state laser LS (LSR532HX, $\lambda = 532$ nm, maximum output power 2.5 W) was converted into a near-plane-wave distribution by using a collimation system comprising a microscope objective MO (20×) and a pinhole D with a 10-μm diameter. The plane wave passed through a half-wave plate (denoted $\lambda/2$ in Fig. 4a), which served to rotate the linear polarization plane of the laser beam. This linearly polarized beam was then directed via a beam splitting cube Bs1 onto a reflective SLM M1 (SPSLM36R, 4096×2240 pixels, pixel size 3.6 μm). To perform amplitude modulation, a polarizer P was placed behind the SLM. A combination of lens L2 (focal length 150 mm) and reflective SLM M2 (SPSLM36R) was utilized to implement a 4F optical correlator system, wherein the phase mask of the DNN, displayed on modulator M2, served as the spatial filter. We note that the beam splitting cube Bs2 served as an extra correlator component to observe the reflected modulated light field with video camera Cam1 (Daheng ME2P-1231-32U3M, 4096×3000 pixel array, pixel size 3.45 μm). Light splitting cube BS3 combined with lens L3 (focal length 75 mm) enabled observation of the intensity distribution of the optical field generated in the display plane of the reflective modulator M2. Here, the lens distance to both the modulator display plane and the video-camera Cam 2 sensor (Daheng ME2P-1231-32U3M, sensor size 4096×3000, pixel size 3.45 μm) was equal to double the focal length of the lens. In this case, the actually recorded intensity corresponded to the optical field spectrum generated in the display plane of the first light modulator M1. The experimentally realized device (Fig. 4b and 4c) basically corresponds to the optical setup in Fig. 4a; however, to minimize the system's footprint, a large number of rotating mirrors were utilized, and the beam splitter cubes Bs2 and Bs3 were replaced with semi-transparent mirrors. To further reduce the device footprint, a two-tier optical setup was realized, with the second tier containing both cameras (Fig. 4b and 4c). A 3D model of the APCD assembly is shown in Fig. 4b and a APCD photograph is shown in Fig. 4c.

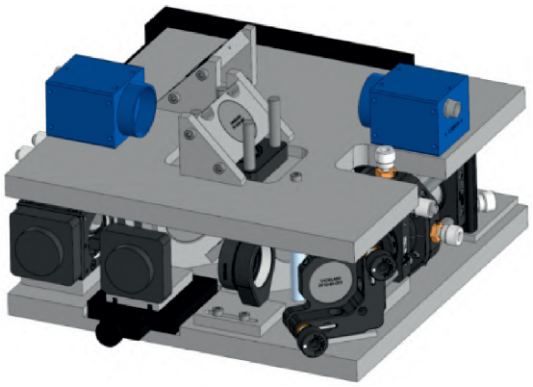
Camera Cam 2 is intended for capturing the intensity distribution in the Fourier plane. Data from this distribution allows implementation of an electronically controlled nonlinear operational mode of the APCD-2024, with the transmittance of each pixel varying nonlinearly as a function

of the detected intensity level. Nonlinearity is introduced into the modulator M2 via correcting the DNN phase function.

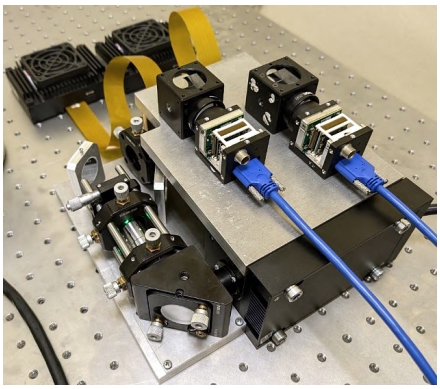
The APCD-2024 setup mounted in a compact housing demonstrated a fairly good performance: in an experiment on recognition of handwritten digits from the MNIST dataset [27], which comprises 60,000 training images and 10,000 test images, a recognition error rate of 3.14% was achieved (corresponding to an accuracy of 100% – 3.14% = 96.86%). According to the original website of MNIST creators [28], error rates of 12% are observed when using simple linear classifiers without preprocessing [29].



(a) Schematic optical layout: LS – a 532-nm laser, M – a rotating mirror, MO – a 20× microlens, D – a pinhole, $\lambda/2$ – half-wave plates, L1 – a collimating lens, Bs1, Bs2, Bs3 – beam splitters, M1 – an input SLM, P – a polarizer, L2 – a lens performing a Fourier transform, L3 – an imaging lens generating an intensity pattern in the Fourier plane, Cam1 – an output camera, and Cam2 – a camera capturing the intensity pattern in the Fourier plane



(b) A 3D model of the assembled APCD



(c) An APCD photograph

Figure 4. A 4F-system-enabled APCD

The two-tier arrangement suffers from several drawbacks: access to the components on the first tier for alignment is only possible after removal of the second tier. The high density of

heat-dissipating elements within a confined volume induces distortions in the optical layout. Furthermore, the forced-air cooling system calls for frequent cleaning of the optical surface.

In addition, the optical scheme depicted in Fig. 4a proved to be suboptimal in terms of the position of beam splitter Bs3, thus leading to extra light losses.

In 2025, a new version of the APCD was developed, in which the optical components and electronic modules were integrated into a standard form-factor housing designed for installation in a server rake.

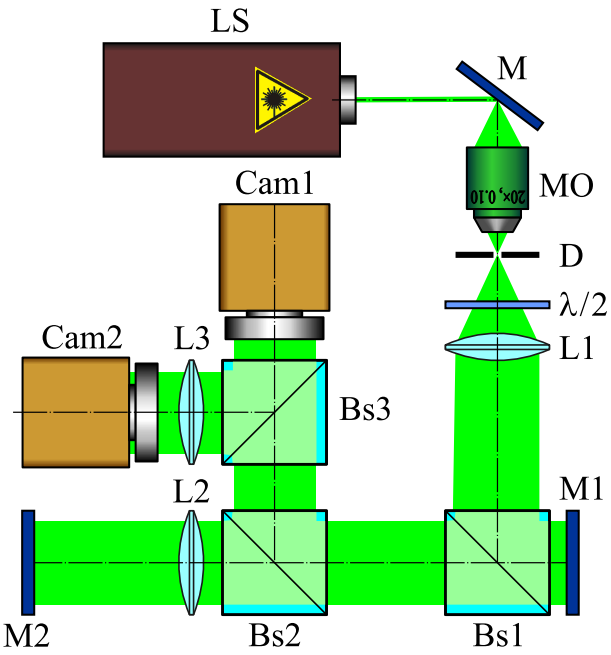
1.2. APCD-2025

Two main modifications were introduced into the APCD-2025 optical scheme. The first modification consists in image feeding via phase – rather than amplitude – modulation of the electromagnetic wave. All known approaches – starting from the classic work [26] to recent publications [30–32] – have relied on amplitude-based input of the image under recognition. This stems from the inertia of human visual perception, as the eye perceives the intensity distribution, which is determined by the light amplitude. The use of phase-modulated input nade it possible to exclude the half-wave plate located behind the beam splitter and the polarizer from the optical scheme. The second modification affected the beam splitter Bs3 deflecting light to the camera Cam2. With the light passing through the beam splitter twice, it was moved to the arm of back reflection from beam splitter Bs2. Figure 5a shows the new optical layout of the APCD-2025, while Fig. 5b presents a photograph of the APCD-2025 in a housing intended for mounting in a server rack.

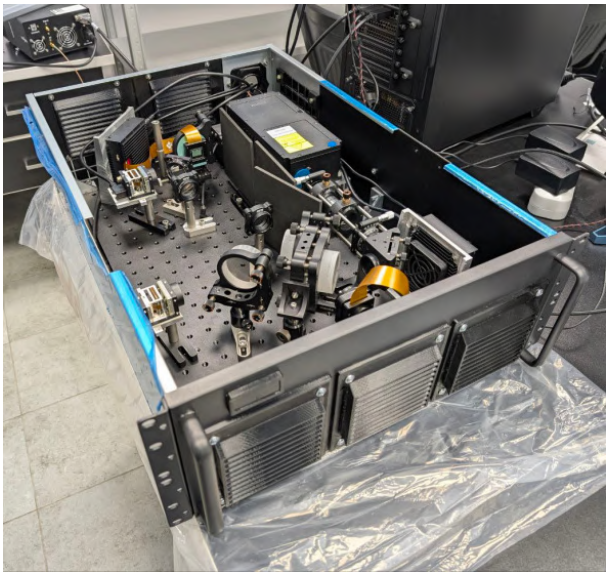
As seen from Fig. 5, the beam splitter Bs3 is relocated from modulator M2 closer to cameras Cam1 and Cam2, enabling a single passage of light through the beam splitter, so reducing energy losses in the element by more than twice (minus one beam splitting and two Fresnel reflections). With the half-wave plate and polarizer not found behind the beam splitter Bs1, the modulator M1 can be used in a regular mode of phase modulation. Another benefit is an essential reduction in light energy losses. A polarizer typically transmits only half of the incident power and about 4% is additionally reflected at each surface of the two elements. Thus, their combined transmittance was $0.5*0.96^4=0.42$, so that their removal from the optical scheme reduced energy losses by nearly a factor of 2.5. Overall, by eliminating both the half-wave plate and the polarizer, reduced the energy losses by approximately a factor of 5, allowing a five-fold reduction in the laser power in the APCD. The total energy consumption decreased from 72 W to 56 W.

2. Experiments on Hand-Wwritten Digit Recognition Using the APCD-2024

In image recognition (classification) tasks, a sequence of images from several distinct classes is fed to the input of the APCD-2024 (with image dimensions up to 2048×2048 pixels). Typically, handwritten digit images from the MNIST database [29] are used as a standard test [5–16]. In accordance with the description of the classification task in the Introduction, several energy registration zones are defined at the output plane of the optical system, with the zones used in the experiment shown in Fig. 6a. The diffractive neural network realized as a phase-only DOE (the DOE design methods employed are described in detail in [15, 16]) on phase modulator M2 is required to generate the maximum energy in the detection zone whose index matches the class of



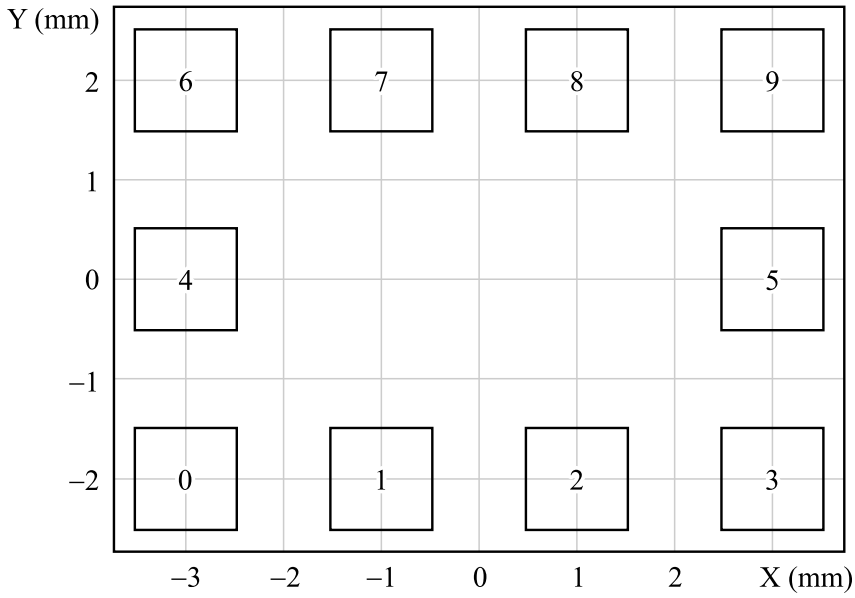
(a) Optical layout of the APCD-2025: LS – a 532-nm laser, M – a rotating mirror, MO – a 20× microlens, D – a pinhole, $\lambda/2$ – a half-wave plate, L1 – a collimating lens, Bs1, Bs2, Bs3 – beam splitters, M1 – an input modulator, L2 – a Fourier transforming lens, M2 – a modulator in the Fourier plane, L3 – a lens for imaging the Fourier-plane intensity distribution, Cam1 – an output camera, Cam2 – a camera for capturing the Fourier-plane intensity distribution



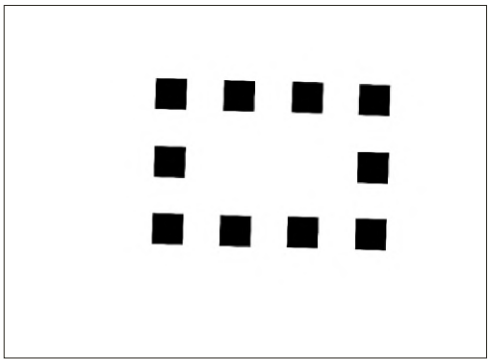
(b) A photograph of the APCD in the housing

Figure 5. APCD-2025

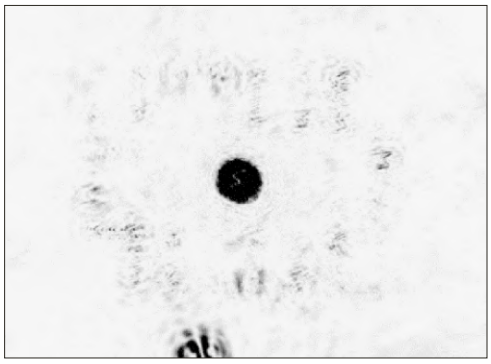
the input image. In the experimental setup, it is hardly possible to ensure the exact alignment of the modulator in the Fourier plane as well as precise camera positioning, so the positions of the registration zones are determined experimentally by means of a special phase mask uniformly illuminating each zone. Figure 6b shows actual positions of the detection zones within the full field of view of camera Cam1, while Fig. 6c represents the actual intensity distribution when an image of the digit 5 is fed to the input.



(a) Layout of the detection zones of the diffractive neural network (diagram)



(b) Actual positions of the zones in the field of view of camera Cam1



(c) Output intensity pattern when an image of 5 is fed to the input

Figure 6. Layout of the detection zones of the diffractive neural network

A series of experiments on handwritten digit recognition and classification using the MNIST database was conducted with the APCD-2024. In the experiments, a sequence of 5,000 digit images was loaded into the input SLM. The resulting intensity pattern across 10 detection zones at the optical system output plane was recorded and classification outcomes were subsequently determined. The classification accuracy achieved did not exceed 85%.

The relatively low classification accuracy can be attributed to several shortcomings of the optical system:

1. The wavefront from the collimator deviates from an ideal plane wave;
2. There are spurious reflections from numerous optical surfaces;
3. The effective active area of the modulators is only 80% of their total active surface area;
4. A significant proportion of the incident light is directed to the zero diffraction order (at the center of Fig. 6c);
5. Geometric inaccuracies result in errors when locating the detection zones.

A technique for reducing the recognition error that eliminates all systematic inaccuracies of the optical system was proposed. To achieve this, every other image in the sequence should be

formed as a background image. Then, the intensity pattern from the object under recognition is analyzed and the intensity distribution corresponding to the background input image is subtracted from it. Using this technique, the classification accuracy reached 96% in the very first experiments. Unfortunately, this approach reduced the APCD's performance by half. Therefore, to compensate for the influence of systematic errors on the classification results, a tiny digital neural network – a linear perceptron digitally implemented on a computer – was introduced to post-process the recognition results. A similar approach was discussed in [33, 34].

The input layer of the digital neural network was formed by the integrated intensities measured in the registration zones. The hidden layer comprised 10 neurons and the output layer also comprised 10 neurons, associated with the classes of the input objects. Training of this neural network was performed using the output images generated by the APCD. From each image, between 10 and 90 numerical features were extracted – namely, the total intensities within each registration zone, or within subdivisions of the zones into 4 and 9 subregions. These features were computed for 50,000 handwritten digit images taken from the MNIST database. The neural network employed a linear neuron activation function. At the output layer, the computer neural network returned the object class, enhancing recognition accuracy. Table 1 presents the experimental recognition results obtained using the APCD-2024 augmented with the above-described neural network, evaluated on 5,000 characters from the MNIST test set.

The first column of Tab. 1 specifies the character presented at the APCD-2024 input, and the rows indicate the counts of correct and incorrect recognition outcomes. In the confusion matrix shown in Tab. 1, the rows represent the true classes, while the columns give the predicted classes. Recognition accuracy is computed as the sum of all values on the diagonal from the top-left to the bottom-right corner (highlighted in green in Tab. 1) divided by the total number of digits under recognition (5,000): $(476+552+475+477+528+418+490+535+449+443)/5000 = 0.9686$. In the case of perfect system performance, the diagonal of the table should contain the largest values, while all off-diagonal entries should be zero.

Table 1. Recognition results for 5,000 handwritten characters from the MNIST database obtained with the APCD-2024 augmented with a supplementary neural network with 90 input neurons

	0	1	2	3	4	5	6	7	8	9
0	476	0	0	0	0	1	1	1	0	2
1	0	552	4	2	1	0	0	1	3	0
2	0	1	475	3	1	2	0	1	4	1
3	1	1	2	477	0	3	0	3	4	2
4	0	0	0	0	528	1	1	1	0	4
5	0	1	1	5	0	418	3	1	4	1
6	0	0	1	0	6	4	490	0	1	0
7	0	1	3	3	2	0	0	535	1	5
8	0	1	1	3	2	3	0	0	449	3
9	4	2	1	4	15	2	1	17	6	443

The confusion matrix presented in Tab. 1 enables computation of the primary classification quality metrics for the digits. For instance, the precision (which indicates the proportion of digits predicted as class "X" that truly belong to class "X") for the digit "5"

is 418/(0+1+1+5+0+418+3+1+4+1)=0.9631. The recall – representing the proportion of true class "X" digits that the classifier successfully identifies – for the digit "5" equals 418/(1+0+2+3+1+418+4+0+3+2)=0.9631. The F1-score, defined as the harmonic mean of precision and recall and considered the most informative metric for imbalanced data for the digit "5", equals

$$\mathrm{F1} = 2 \times (\mathrm{Precision} \times \mathrm{Recall})/(\mathrm{Precision} + \mathrm{Recall}) = 0.9631. \quad (1)$$

A compact auxiliary neural network enables the APCD to achieve substantially improved recognition performance within a time frame of approximately several tens of microns. A recognition rate error of 3.14% was achieved. The processing time required by the auxiliary neural network to analyze the output data is on the order of several tens of microseconds, which is substantially shorter than the 16.7 ms operational cycle of the APCD-2024. Thus, the proposed hybrid approach practically does not slow down the operation of APCD-2024.

3. Experiment with the APCD-2025

In the APCD-2025, images are input via phase modulation of the optical field. This results in a high level of background illumination around the images under recognition and hence a pronounced intensity peak at the center of the spatial spectrum in the Fourier plane. To suppress this on-axis intensity peak, the phase function of a binary phase grating with a high spatial frequency (Nyquist frequency) is superimposed on the DNN phase function on modulator M2 in the Fourier plane. This grating redirects the light originally concentrated in the central region away from the APCD-2025 detection region. Table 2 presents recognition results from experiments conducted with the APCD-2025 in combination with the aforementioned DNN when recognizing 10,000 characters from the MNIST test set.

Table 2. Recognition results for 10,000 handwritten digits from the MNIST database obtained with the APCD-2025 augmented by a neural network featuring 90 input neurons

	0	1	2	3	4	5	6	7	8	9
0	998	0	0	0	0	0	0	0	0	2
1	0	997	0	0	1	0	0	2	0	0
2	0	17	981	1	0	0	0	1	0	0
3	0	0	1	989	0	9	0	0	0	1
4	0	1	0	0	986	0	0	2	0	11
5	0	1	0	0	9	983	6	0	0	1
6	9	0	0	0	7	0	982	0	1	1
7	0	7	0	0	0	1	0	984	0	8
8	0	0	0	2	1	0	7	0	982	8
9	0	0	0	0	6	0	0	1	0	993

Employing phase-based image input together with post-processing of the intensity distribution in the target regions using a linear perceptron resulted in a 1.3% classification error rate for handwritten digits from the MNIST database. The fairly high classification accuracy achieved on tests images provided a sufficient basis for proceeding to the recognition of real-world objects. Meanwhile, it is worth noting that developers of computer neural networks for the MNIST

database reported reaching an error rate of 0.21% as early as 2013 using DropConnect regularization [35], while researchers employing Random Multimodel Deep Learning (RMDL) achieved a 0.18% error rate in 2018 [36].

4. Assessment of the APCD Performance

The authors deem it incorrect to compare the performance of the APCD and a conventional computer by evaluating the time required for a computer to solve the mathematical relationships describing the APCD's operation (computing two Fourier integrals via the fast Fourier transform and so on). We propose comparing the image recognition speed of the APCD with that of a modern graphics processor employing a convolutional neural network, such as YOLO family [37].

Currently, the YOLO (You Only Look Once) family [37, 38] serves as a kind of a benchmark among computer networks for object recognition. This neural network was specifically developed to achieve maximum processing speed while ensuring guaranteed recognition accuracy. Studies have demonstrated that YOLO performs efficiently in real time and can reliably detect small objects in video streams [37]. The authors of [37, 38] conducted a comparative analysis of the inference times required for object recognition using different neural networks from the YOLO family. However, those results were achieved on outdated graphics cards, namely, NVIDIA RTX 2080 (≈4 TFLOPS) and NVidia Tesla T4 16 Gb (8.1 TFLOPS in FP32). Because of this, in this paper, the comparison is made with a contemporary GPU, the RTX 4090 (52.2 TFLOPS).

The APCD can process images with resolutions ranging from 512×512 up to 4096×2140 (the dimension of the UPOLabs HDSLM36R modulator used) at a 60Hz frame rate, i.e. in 16.7 ms. Importantly, the APCD-aided image processing time is independent of the input image dimensionality and determined by the slowest input/output component in the system.

In this paper, we examined the YOLO neural network in a regime where it does not downsample the input image but instead processes every pixel of the frame. This is motivated by the fact that digit images from the MNIST dataset are only 28×28 pixels in size [28]. Downsampling the entire frame would inevitably downsample the digit images as well, leading to a deteriorating recognition quality. When no downsampling is applied, the inference speed of the YOLO neural network depends on the input frame dimensionality. Additionally, the operating speed of the YOLO neural network also depends on the size (type) of the neural network itself. Table 3 gives the image recognition times for images of varying resolution on a GPU RTX 4090 using the YOLO v.8, which was selected as the fastest modification of this family. Three network variants were evaluated (medium, large and extra-large), each providing recognition accuracy comparable to that of the APCD-2025.

Table 3 shows that, considering the APCD's constant processing time of 16.7 ms per frame, YOLO exhibits a superior performance for image resolutions up to 1024×1024 pixels. However, for larger-dimension images, the APCD-2025 outperforms YOLO with a significant margin. This advantage of the APCD-2025 could be further extended by operating the SLM at a higher clock frequency.

A further metric for comparing the performance of the graphic card and the APCD is energy efficiency – specifically, the energy consumed per processed frame. The RTX 4090 GPU consumes about 420 W (WGPU) [39], whereas the APCD-2025 consumes about 56 W (WAPCD). The energy per frame for the GPU is given by EGPU=WGPU×YOLO. For the APCD-2025, the energy required per frame is $E_{APCD} = W_{APCD} \times 16.7 \times 10^{-3}\text{s} = 0.94J$. Table 3 presents the per-

frame energy consumption of the RTX 4090 GPU. Here, the energy advantage of the APCD-2025 already starts to be seen at 512×512-pixel resolution, and beginning at 2000×2000 pixels, the APCD's energy consumption is an order of magnitude lower than that of the RTX 4090 GPU, even for the lightest YOLO modification, YOLO v.8 medium. Thus, the energy efficiency of optical computing with the APCD-2025 is an order of magnitude higher than that of graphics accelerators for frame resolutions of 2000×2000 pixels and above.

Table 3. Recognition results for 10,000 handwritten digits from the MNIST database obtained with the APCD-2025 augmented by a neural network featuring 90 input neurons

Frame size	YOLO v.8 medium (ms/J)	YOLO v.8 large (ms/J)	YOLO v.8 xlarge (ms/J)
512×512	5.5 / 2.3	6.6 / 2.8	7.8 / 3.3
1024×1024	13.8 / 5.8	13.1 / 5.5	17 / 7.1
2048×2048	30.3 / 12.7	47.6 / 20	58.8 / 24.7
4000×3000	100 /42	154 / 64.7	232 / 97.4

Conclusion

Summing up, we have analyzed two versions of an analog photonic computing device (APCD) and have demonstrated the advantages of the APCD-2025 over the APCD-2024, with the former implementing phase modulation for image input and featuring several design simplifications. The effectiveness of combining the APCD with a compact digital neural network for post-processing the optical recognition results has also been demonstrated. The APCD-2025 has shown clear advantages over GPU-based neural networks in both processing speed and energy efficiency for frame resolutions exceeding 1024×1024 pixels. Specifically, at resolutions of 2000×2000 and higher, the APCD-2025 outperforms a state-of-the-art RTX 4090 GPU by an order of magnitude in energy efficiency. Hence, the use of the APCD enables real-time object recognition in video streams.

Acknowledgements

This study was conducted within the framework of the scientific program of the National Center for Physics and Mathematics, section 1 "National Center for Research on Supercomputer Architectures". Stage 2026-2027.

References

1. Shen, Y., Harris, N., Skirlo, S., *et al.*: Deep learning with coherent nanophotonic circuits. Nature Photon. 11, 441–446 (2017). https://doi.org/10.1038/nphoton.2017.93

2. Harris, N.C., Carolan, J., Bunandar, D., *et al.*: Linear programmable nanophotonic processors. Optica 5(12), 1623–1631 (2018). https://doi.org/10.1364/OPTICA.5.001623

3. Zhu, H.H., Zou, J., Zhang, H., *et al.*: Space-efficient optical computing with an integrated chip diffractive neural network. Nat. Commun. 13(1), 1044 (2022). https://doi.org/10.1038/s41467-022-28702-0

4. Kazanskiy, N.L., Khorin, P.A., Golovastikov, N.V., Khonina, S.N.: Analog optical computing: principles, progress, and prospects. Opt. Laser Technol. 193(A), 114220 (2026). https://doi.org/10.1016/j.optlastec.2025.114220

5. Lin, X., Rivenson, Y., Yardimci, N.T., *et al.*: All-optical machine learning using diffractive deep neural networks. Science 361(6406), 1004–1008 (2018). https://doi.org/10.1126/science.aat8084

6. Chang, J., Sitzmann, V., Dun, X., *et al.*: Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification. Sci. Rep. 8, 12324 (2018). https://doi.org/10.1038/s41598-018-30619-y

7. Yan, T., Wu, J., Zhou, T., *et al.*: Fourier-space diffractive deep neural network. Phys. Rev. Lett. 123(2), 023901 (2019). https://doi.org/10.1103/physrevlett.123.023901

8. Luo, Y., Mengu, D., Yardimci, N.T., *et al.*: Design of task-specific optical systems using broadband diffractive neural networks. Light Sci. Appl. 8, 112 (2019). https://doi.org/10.1038/s41377-019-0223-1

9. Zhou, T., Fang, L., Yan, T., *et al.*: *In situ* optical backpropagation training of diffractive optical neural networks. Photon. Res. 8(6), 940–953 (2020). https://doi.org/10.1364/PRJ.389553

10. Mengu, D., Zhao, Y., Yardimci, N.T., *et al.*: Misalignment resilient diffractive optical networks. Nanophotonics 9(13), 4207–4219 (2020). https://doi.org/10.1515/nanoph-2020-0291

11. Zhou, T., Lin, X., Wu, J., *et al.*: Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit. Nat. Photonics 15(5), 367–373 (2021). https://doi.org/10.1038/s41566-021-00796-w

12. Chen, H., Feng, J., Jiang, M., *et al.*: Diffractive deep neural networks at visible wavelengths. Engineering 7(10), 1483–1491 (2021). https://doi.org/10.1016/j.eng.2020.07.032

13. Zheng, S., Xu, S., Fan, D.: Orthogonality of diffractive deep neural network. Opt. Lett. 47(7), 1798–1801 (2022). https://doi.org/10.1364/OL.449899

14. Motz, G.A., Doskolovich, L.L., Soshnikov, D.V., *et al.*: Design of diffractive neural networks for solving different classification problems at different wavelengths. Photonics 11(8), 780 (2024). https://doi.org/10.3390/photonics11080780

15. Soshnikov, D.V., Doskolovich, L.L., Motz, G.A., *et al.*: Designing robust diffractive neural networks with improved transverse shift tolerance. J. Opt. Soc. Am. A 42(5), 699–709 (2025). https://doi.org/10.1364/JOSAA.561906

16. Soshnikov, D.V., Doskolovich, L.L., Motz, G.A., *et al.*: Design of DOE robust to positioning errors for optical classification purposes. Computer Optics 50(1), 1693 (2026). https://doi.org/10.18287/COJ1693

17. Lohmann, A.W., Paris, D.P.: Binary Fraunhofer holograms, generated by computer. Appl. Opt. 6(10), 1739–1748 (1967). https://doi.org/10.1364/ao.6.001739

18. Fienup, J.R.: Phase retrieval algorithms: a comparison. Appl. Opt. 21(15), 2758–2769 (1982). https://doi.org/10.1364/ao.21.002758

19. Soifer, V.A., Kotlyar, V.V., Doskolovich L.L.: Iterative methods for diffractive optical elements computation. Taylor & Francis (1997). https://doi.org/10.1201/9781482272918

20. Ripoll, O., Kettunen V., Herzig, H.P.: Review of iterative Fourier transform algorithms for beam shaping applications. Opt. Eng. 43(11), 2549–2556 (2004). https://doi.org/10.1117/1.1804543

21. Glses A.A., Jenkins, B.K.: Cascaded diffractive optical elements for improved multiplane image reconstruction. Appl. Opt. 52(15), 3608–3616 (2013). https://doi.org/10.1364/AO.52.003608

22. Wang, H., Piestun, R.: Dynamic 2D implementation of 3D diffractive optics. Optica 5(10), 1220–1228 (2018). https://doi.org/10.1364/OPTICA.5.001220

23. Doskolovich, L.L., Mingazov, A.A., Byzov, E.V., *et al.*: Hybrid design of diffractive optical elements for optical beam shaping. Opt. Express 29(20), 31875–31890 (2021). https://doi.org/10.1364/oe.439641

24. Kazanskiy, N.L., Khonina, S.N., Butt, M.A.: Exploring the functional characteristics of diffractive optical element: A comprehensive review. Opt. Laser Technol. 183, 112383 (2025). https://doi.org/10.1016/j.optlastec.2024.112383

25. Zheng, M., Shi, L., Zi, J.: Optimize performance of a diffractive neural network by controlling the Fresnel number. Photonics Res. 10(11), 2667–2676 (2022). https://doi.org/10.1364/PRJ.474535

26. Lugt, A.V.: Signal detection by complex spatial filtering. IEEE Trans. Inf. Theory 10(2), 139–145 (1964). https://doi.org/10.1109/TIT.1964.1053650

27. Kussul, E., Baidyk, T.: Improved method of handwritten digit recognition tested on MNIST database. Image Vis. Comput. 22(12), 971–981 (2004). https://doi.org/10.1016/j.imavis.2004.03.008

28. LeCun, Y., Cortes, C., Burges, C.J.C.: The MNIST database of handwritten digits. https://web.archive.org/web/20210407152035/http://yann.lecun.com/exdb/mnist/, accessed: 2026-05-14

29. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE 86(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791

30. Zhang, Y., *et al.*: Memory-less scattering imaging with ultrafast convolutional optical neural networks. Sci. Adv. 10(24), eadn2205 (2024). https://doi.org/10.1126/sciadv.adn2205

31. Zhang, Z., Feng, F., Gan, J., *et al.*: Space-time projection enabled ultrafast all-optical diffractive neural network. Laser Photonics Rev. 18(8), 2301367 (2024). https://doi.org/10.1002/lpor.202301367

32. Li, B., Zhu, Y., Fei, J., *et al.*: Multi-functional broadband diffractive neural network with a single spatial light modulator. APL Photonics 10(1), 016115 (2025). https://doi.org/10.1063/5.0245832

33. Goncharov, D.S., Starikov, R.S., Zlokazov, E.Y.: Pattern recognition system based on a coherent diffractive correlator with deep learned processing of downsampled correlation responses. Appl. Opt. 63(36), 9196–9204 (2024). https://doi.org/10.1364/AO.541305

34. Volkov, A.A., *et al.*: Optoelectronic spatial filtering system for neural network applications. Vestnik natsional'nogo issledovatel'skogo yadernogo universiteta 'MIFI' 15(2), 134–142 (2026). https://doi.org/10.26583/vestnik.2026.2.4

35. Wan, L., Zeiler, M., Zhang, S., *et al.*: Regularization of neural network using DropConnect. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, Georgia, USA, June 17-19, 2013. vol. 28, pp. 1058–1066 (2013).

36. Kowsari, K., Heidarysafa, M., Brown, D.E., *et al.*: RMDL: Random multimodel deep learning for classification. In: ACM International Conference Proceeding Series, ICISDM18, Lakeland, FL, USA, April 9-11, 2018. pp. 19–28 (2018). https://doi.org/10.1145/3206098.3206111

37. Wang, G., *et al.*: TridentYOLO: Improving the precision and speed of mobile device object detection. IET Image Process. 16(1), 145–157 (2022). https://doi.org/10.1049/ipr2.12340

38. Vychegzhanin, S.V., Tatarinova, A.G., Myshkin, R.E.: Comparative analysis of neural network models for detecting unmanned aerial vehicles. Computer Optics 50(2), 1728 (2026). https://doi.org/10.18287/COJ1728

39. https://technicalcity.b-cdn.net/ru/video/Tesla-V100-PCIe-32-GB-protiv-H100--PCIe-80-GB , accessed: 2025-03-04

DOI: 10.14529/jsfi260209

Supercomputer-Aided Space-Time Processing Methods for MIMO Radars in Advanced Airborne Earth Remote Sensing Systems

Vladimir S. Verba[1], *Vyacheslav A. Miheev*[1], *Viktor A. Plushchev*[1], *Andrey A. Chernienko*[1,2]

This paper addresses the problem of escalating computational complexity arising from the transition from classical Active Electronically Scanned Arrays to fully-fledged Multiple-Input Multiple-Output (MIMO) radars for airborne Earth remote sensing (ERS) systems. It is shown that the formation of a virtual aperture increases the number of virtual channels to 10^5–10^6, making classical adaptive processing methods based on direct access to covariance matrices of dimension NN impossible for existing onboard computers. Modifications of the MUSIC and ESBM (Estimation of Signal Parameters via Beamspace Mapping) algorithms have been developed. These algorithms are adapted to operate with sparse covariance matrices, which makes it possible to radically reduce the amount of stored and processed data. Based on the proposed methods, a pipelined processing architecture has been developed and implemented on hybrid CPU+FPGA computers, providing latency compatible with real-time requirements for detection tasks in air-ground and air-sea circuits. The results obtained pave the way for the creation of next-generation airborne radar systems with unprecedented spatial resolution and high noise immunity.

Keywords: MIMO radar, supercomputer processing methods, reduced-rank algorithms.

Introduction

The evolution of airborne Earth remote sensing (ERS) systems over the past decades has been characterized by a consistent transition from mechanically scanned antenna systems to Active Electronically Scanned Arrays (AESAs) and, more recently, to multiple-input multiple-output (MIMO) radars. While classic AESAs have provided significant advances in scanning speed and beamforming flexibility, they have nevertheless reached the physical limit of their resolution with a fixed aperture.

In contrast, MIMO radars open up fundamentally new possibilities for airborne ERS systems. Creating a virtual aperture equivalent to the product of the number of transmitting and receiving elements, they enable a substantial increase in angular resolution without increasing the physical size of the antenna system [3, 4]. In MIMO radars, each channel emits orthogonal (independent) signals. This method implies a virtual receive antenna array whose size is equal to the product of the number of transmitting and receiving channels. This approach significantly improves angular resolution and the accuracy of reflected signal with the same physical antenna size, suppressing fluctuating signals and increasing the probability of detecting small and maneuvering objects [7, 8].

As in many areas of radar, a significant increase in the tactical and technical characteristics of a MIMO radar requires a significant increase in computing resources. As in many areas of radar, a significant increase in the tactical and technical characteristics of a MIMO radar requires a significant increase in computing resources. The transition to full-fledged MIMO radars raises a fundamental problem related to the "curse of dimensionality." Indeed, the formation of a virtual aperture (Virtual Array) leads to the fact that the number of virtual receiving channels reaches

[1]JSC "Vega Radio Engineering Corporation", Russian Federation
[2]JSC "New Start", Russian Federation

the value of $N_{rx} \times N_{tx}$, where N_{rx}, N_{tx} are the number of receiving and transmitting elements, respectively. In promising remote sensing complexes, the number of virtual channels reaches 10^5.

Classical Space-Time Adaptive Processing (STAP) algorithms, which form a space-time filter to protect against interference, the parameters of which adapt depending on the motion parameters of the radar carrier and the interference environment, require computational costs of the order of $O(N^3)$ for an $N \times N$ correlation matrix, which at $N \approx 10^5$, it makes them physically unrealizable on existing onboard computers.

At the same time, the tasks of detecting small objects: unmanned aerial vehicles of a small class, submarine periscopes or hypersonic missiles, impose strict requirements on processing latency [1, 2]. The acceptable processing window for detecting small objects is units-tens of milliseconds. Thus, there is a "computational gap" between the potential capabilities of MIMO radars and the actual performance of existing onboard computing systems.

A number of approaches to reducing the computational complexity of STAP algorithms have been proposed in the literature. Rank reduction methods such as multichannel Wiener filtering (MWF) and auxiliary channel filter (ACS) can reduce the effective dimension of the processing space by projecting onto a subspace of a smaller dimension. Knowledge-Aided (KA) methods use a priori information about terrain and interference conditions to reduce the size of the training sample.

However, as the number of virtual channels grows to 10^5, the performance of these methods decreases significantly: the required subspace size is still too large, and the gain in computational complexity is insufficient to achieve real-time mode. In addition, classical spectral estimation algorithms (MUSIC, ESPRIT) in their traditional formulation also face insurmountable computational difficulties when processing such a high dimension [5, 6].

This article aims to overcome the described computational gap by synthesizing new algorithmic and architectural solutions. The main hypothesis is the transition to a heterogeneous computing architecture of the onboard supercomputer core, combining general-purpose central processors (CPUs) and field-programmable gate array (FPGAs).

This work is divided as follows: Section 2 discusses the basics of the method used to reduce computational complexity. Section 3 describes the detection algorithm in detail. The architecture of the onboard computer suitable for the proposed algorithm is presented in Section 4. Possible applications and practical value of the algorithm and the computer architecture presented is discussed in Conclusion.

1. Matematical Signal Model and Computational Complexity

1.1. Geometry of Shooting in MIMO Radars

Let us consider an aviation complex for remote sensing of the Earth (remote sensing), equipped with a MIMO radar. In general, the system configuration can be monostatic, when the transmitting and receiving antenna arrays are placed on the same aircraft, or bistatic, in which the transmitter and receiver are separated into different carrier. For the sake of certainty, the monostatic configuration, characteristic of most promising aviation complexes, is further considered, however, the results obtained can be generalized to the bistatic case.

Let the transmitting antenna array contain N_{tx} radiating elements, and the receiving antenna contain Nrx receiving channels. With time or code separation of signals, each of the N_{tx} transmitters emits an orthogonal (or quasi-orthogonal) signal. Figure 1 shows the geometry of

the MIMO radar survey. After consistent filtering, a virtual aperture is formed at the receiving point, the number of channels of which is equal to the product $N_v = N_{tx} \cdot N_{rx}$. The vector $v \in C^{N_v}$ is the result of convolution of the transmitting and receiving antenna patterns and contains all information about the angular position of objects.

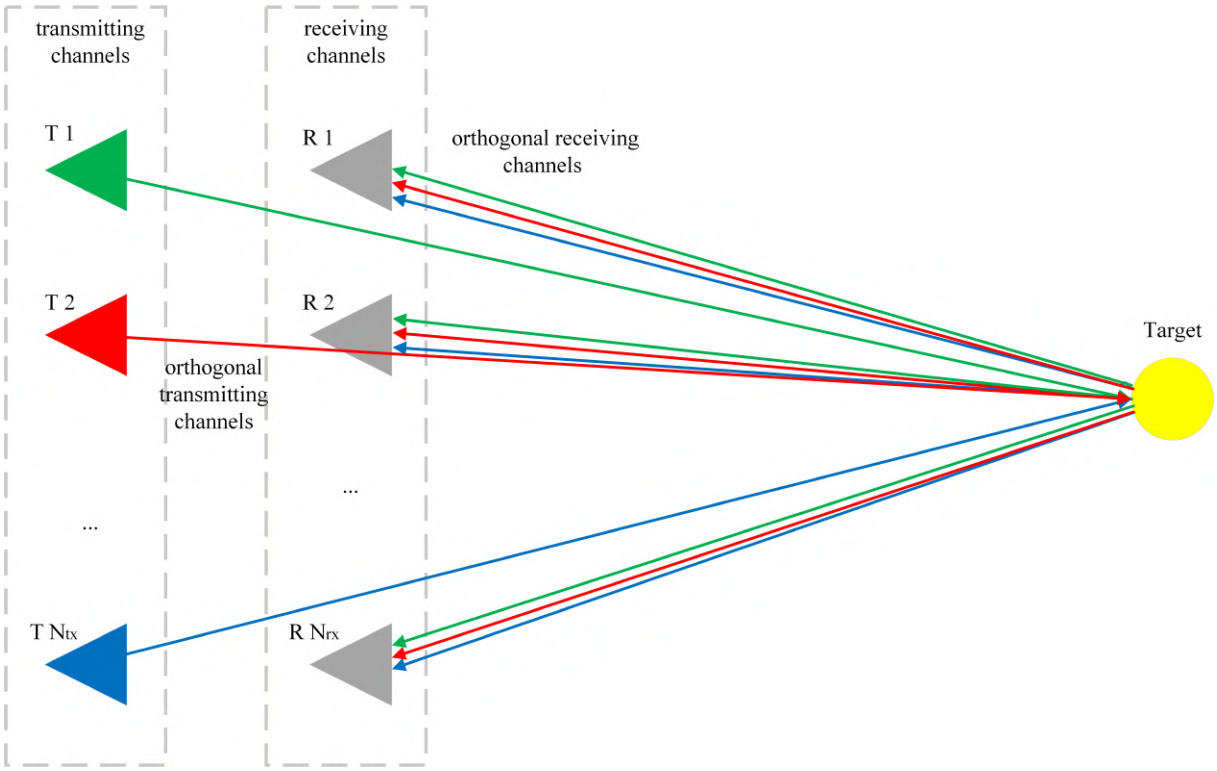


Figure 1. Principle of operation of the MIMO radar

The moving carrier of ERS is characterized by the presence of a Doppler frequency shift due to both the movement of the carrier itself by the possible movement of targets. Let M transmitting pulses with a repetition period of T be processed in a coherent packet. The space-time domain vector of the signal is formed as a result of the Kronecker product of the time and space vectors. The synthesized MIMO covariance matrix (signals from all virtual channels for the entire observation time) makes it possible to consider joint space-time processing as a single task of estimating signal parameters in the dimensional space $N_v \times M$.

1.2. Model of the Signal and Interference Environment for MIMO Radars

In MIMO radars, the received signal is represented as a superposition of reflections from target objects, passive interference from the underlying surface, and active noise interference. Passive interference (reflections from the ground, water surface, urban buildings) is characterized by high space and time correlation and dominates in terms of energy. Active noise interference is created by special electronic warfare equipment and can have both broadband and narrowband structures.

Let $x \in C^{N_v M}$ denote the space-time vector of the received signal samples, where N_v is the number of virtual channels, M is the number of transmitted pulses in a coherent packet. The vector x can be represented as:

$$x = x_s + x_i + x_n, \tag{1}$$

where x_s is the component of the signal reflected from the object, x_i is the component of passive and active interference, and x_n is the component of the receiving channels noise.

The covariance matrix of the received signal is defined as $R = E[xx^H]$, where $E[\cdot]$ is the expectation operator, $(\cdot)^H$ is the Hermitian conjugation. In practice, R is unknown and must be estimated from a training sample:

$$\hat{R} = \frac{1}{L}\sum_{l=1}^{L} x_l x_l^H, \tag{2}$$

where x_l are independent implementations.

For R to be non-degenerate and there to be an inverse matrix, it is necessary that $L \geq 2 \cdot NM$.

1.3. Computational Complexity of the Classical Space-Time Processing Approach

The classical approach to STAP involves solving the optimal filtering problem using a weight vector w that minimizes the output power of interference with a fixed response to the target signal. The optimal weight vector is determined by the expression:

$$w_{opt} = \mu R^{-1} s, \tag{3}$$

where μ is a normalization multiplier that provides $w_{opt}^H s = 1$, s is the "true" reflected signal that we expect to receive from the target in the direction we are interested in. The key computational problem is the need to calculate the inverse covariance matrix R^{-1} with dimension $NM \times NM$.

The computational complexity of solving the inverse of an R^{-1} matrix with dimension $NM \times NM$ by the Gaussian or LU decomposition method is $O((NM)^3)$ arithmetic operations. For promising MIMO radars with N_v on the order of 10^4 virtual channels and M on the order of 100–500 pulses per packet, the NM product can reach 5×10^6.

When using classical space-time processing algorithms to find the inverse R^{-1} covariance matrix with a dimension of $5 \times 10^6 \times 5 \times 10^6$, the computational complexity will be about $O(10^{20})$ floating-point operations, which is physically impossible even on modern supercomputers with a capacity of about 10^{18} operations per second.

As for the required amount of RAM, a preliminary estimate shows that to store only the R matrix, the required minimum amount of memory should be $(NM)^2 \times 16\ bytes = 5 \times 10^{14}\ bytes$ (approximately 500 TB). It is impossible to store a matrix of this size in RAM on board existing and promising ERS.

The classical approach to space-time processing becomes inapplicable when switching to full-fledged MIMO radars with a high-dimensional virtual aperture. This fact forms a fundamental limitation, the overcoming of which requires the development of fundamentally new algorithmic and architectural solutions proposed in the following sections of this work.

1.4. Problem Statement and Initial Data

Suppose there is a MIMO radar with N_{rx} receiving and N_{tx} transmitting channels forming a virtual aperture of dimension $N_v = N_{tx} \cdot N_{rx}$. A coherent packet contains M pulses, so that the space-time vector of the signal has dimension $NM = N_v M$. The received signal is a superposition

of reflections from K targets, passive interference, and noise:

$$x(t) = \sum_{k=1}^{K} \alpha_k a(\theta_k, \varphi_k) \otimes b(f_{d,k}) + x_i(t) + x_n(t), \tag{4}$$

where α_k is complex amplitude reflections from the k-th target, $a(\theta_k, \varphi_k) \in C^{N_v}$ is a vector of the reflected signal in the spatial basis (depends on the azimuth φ_k and elevation θ_k), $b(f_{d,k})$ is a vector of the reflected signal in a temporal basis (depends on the Doppler frequency $f_{d,k}$), $\otimes$ is a work of Kronecker, $x_i(t)$ is the interference component of the reflected signal and $x_n(t)$ is the noise component of the reflected signal.

It is necessary to develop an algorithm for estimating the angular coordinates (θ_k, φ_k) and the Doppler frequency $f_{d,k}$ of all objects K with minimal computational effort and high resolution. To achieve this goal, we will decompose the task. Since calculations based on the covariance matrix are necessary in the framework of space-time adaptive signal processing, it is impossible to solve the problem in this form. At the first stage, it is necessary to reduce the dimension of the covariance matrix. At the second stage, the azimuth, elevation angle, and Doppler frequency are estimated.

2. The Dimensionality Reduction Method

The central problem formulated in the previous sections is that the dimension of the space-time vector of the NM signal for promising MIMO radars reaches values of the order of 10^6, which makes classical processing methods computationally unrealizable. This section proposes several complementary approaches to reducing the effective dimension of the, based on the use of sparse representations, adaptive rank reduction methods, and coherent restoration of the covariance structure.

The first stage is the projection of the initial space-time vector x into the space of rays of a smaller dimension $Q \ll NM$. At the second stage, the projection matrix is optimized. At the third stage, the signal is projected into a smaller dimensional space.

2.1. Projection Matrix Formation

The projection matrix $B \in C^{NM \times Q}$ is formed based on a priori information about the expected directions of arrival of the signal. In general, the pillars B are a set of signal vectors corresponding to the Q reference directions:

$$B = [\tilde{a}(\theta_1, \varphi_1) \otimes \tilde{b}(f_{d,1}), \tilde{a}(\theta_2, \varphi_2) \otimes \tilde{b}(f_{d,2}), ..., \tilde{a}(\theta_Q, \varphi_Q) \otimes \tilde{b}(f_{d,Q})], \tag{5}$$

where $\tilde{a}(\theta_q, \varphi_q)$ is the approximation of the reflected signal in the space basis, $\tilde{b}(f_{d,q})$ is the approximation of the reflected signal in the time (frequence) basis.

2.2. Projection Matrix Optimization

The KAC-JIOAF (Knowledge-Aided Constrained Joint Iterative Optimization Framework for Adaptive Filters) optimization procedure [6] is used to minimize information loss. The problem is formulated as an optimization with confines:

$$\min_{B,W} \|B^H R B - W W^H\|_F^2 + \lambda \|B\|_* + \mu \|W - W_0\|_F^2, \tag{6}$$

where R is the covariance matrix of the initial reflected signal, W is the matrix of weights of the adaptive filter, W_0 is the initial approximation obtained from a priori data, $\|\cdot\|_F$ is the Frobenius norm, $\|\cdot\|_*$ is the nuclear norm providing a low-rank solution and λ, μ are the regularization parameters.

Since optimization is performed using two parameters, the search for a solution is carried out iteratively:

Step 1 (fixing B, optimizing W):

$$W_{t+1} = \frac{(B_t^H R B_t W_t + \mu W_0)}{(B_t^H R B_t + \mu I)}, \tag{7}$$

where I is the identity matrix.

Step 2 (fixing W, optimizing B):

$$B_{t+1} = \arg\min_B \|B^H R B - W_{t+1} W_{t+1}^H\|_F^2 + \lambda \|B\|_*. \tag{8}$$

The solution of the second stage is performed by the algorithm of proximal gradient descent with soft threshold processing of singular values.

2.3. Projecting a Signal into a Smaller Space

After the optimal matrix B_{opt} is found, the projection of the signal is performed:

$$y = B_{opt}^H x \tag{9}$$

and the projection of the reduction in the dimension of the covariance matrix:

$$R_y = B_{opt}^H R B_{opt}. \tag{10}$$

At the same time, the dimension of the processing space is reduced from NM to Q, where Q is selected in the range from 100 to 1000, depending on the complexity of the scene.

3. Estimation of Target Parameters in Space-Time Signal Processing

At the next stage of processing, an adapted version of the MUltiple SIgnal Classification algorithm (MUSIC) is applied to the projected covariance matrix R_y [7, 9, 10], which is based on the decomposition of the covariance matrix $R \in C^{N_v M}$ into signal and noise subspaces.

3.1. Decomposition into Signal and Noise Subspaces

Decomposing the matrix into eigenvectors:

$$R_y = U_y \Lambda_y U_y^H, \tag{11}$$

where $U_y = [u_1, u_2, ..., u_Q]$ – the matrix of eigenvectors, $\Lambda_y = diag(\lambda_1, \lambda_2, ..., \lambda_Q)$ is a diagonal matrix of eigenvalues, ordered in descending order $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_Q$.

The number of objects $\hat{K}$ is estimated using the minimum description length criterion:

$$U_s = [u_1, u_2, ..., u_{\hat{K}}]. \tag{12}$$

The noise subspace is formed from the remaining vectors:

$$U_n = [u_{\hat{K}+1}, u_{\hat{K}+2}, ..., u_Q]. \tag{13}$$

3.2. Projection of Reflected Signals into the Ray Space

For each potential direction (θ, φ) and the Doppler frequency f_d, the original reflected signal is projected into the ray space:

$$c(\theta, \phi, f_d) = B_{opt}^H (a(\theta, \varphi) \otimes b(f_d)). \tag{14}$$

3.3. Calculation of Parameter Estimates

To calculate the estimates parameters using the MUSIC algorithm, a pseudospectrum in the ray space is calculated:

$$P_{MUSIC}(\theta, \varphi, f_d) = \frac{1}{\|U_n^H c(\theta, \varphi, f_d)\|^2} = \frac{1}{c(\theta, \varphi, f_d)^H U_n U_n^H c(\theta, \varphi, f_d)}. \tag{15}$$

The peaks of the pseudospectrum correspond to estimates of the parameters of the objects:

$$\hat{\theta}_k, \hat{\varphi}_k, \hat{f}_{d,k} = \arg \max_{\theta, \varphi, f_d} \hat{P}_{MUSIC}(\theta, \varphi, f_d), \ k = 1, 1, ..., \hat{K}. \tag{16}$$

The proposed hybrid algorithm of MUSIC with a reduction in the dimension of the covariance matrix is an effective solution to the problem of space-time processing of MIMO radar signals for remote sensing aircraft systems, providing an optimal compromise between computational complexity and accuracy of estimates of the parameters of detected objects.

3.4. Analysis of the Computational Complexity of the Proposed Hybrid Algorithm and the Capabilities of onboard Computing Systems

Table 1 presents a comparative analysis of the computational complexity of the classical MUSIC method and the proposed hybrid method.

Table 1. Comparison of computational complexity of classical MUSIC and hybrid method

Stage / Method	Classical MUSIC	Proposed hybrid method
Formation of the covariance matrix	$O((NM)^2 L) \approx 10^{12}$	$O(Q^2 L) \approx 10^8$
Calculation of eigenvectors of the covariance matrix	$O((NM)^3) \approx 10^{18}$	$O(Q^3) \approx 10^9$
Calculation of the pseudospectrum (per object)	$O(NM) \approx 10^6$	$O(Q) \approx 10^3$
Iterative optimization of KAC JIOAF	-	$O(T \cdot Q^3) \approx 10^{10}$
Overall difficulty	10^{18}	10^9

Table 1 shows that the computational complexity of the proposed algorithm is 8 orders of magnitude less than the classical MUSIC algorithm, which allows it to be implemented in modern and promising remote sensing systems.

Considering that the accumulation time of 1 coherent packet is 10 microseconds, the onboard computing system must be able to process the received signals before the next packet arrives (pipelining). Thus, the minimum performance of the onboard computing system should be at least 10^{10} operations/ 0.01s = 10^{12} operations/s. Taking into account the fact that for accurate

calculations it is necessary to use the double precision data type (double 64), the number of operations should be an order of magnitude greater. Thus, in order to perform pipelining using the proposed hybrid method, the onboard computing system performance should be at least 10^{13}–10^{14} operations per second (10–100 TFLOPs).

onboard computing systems are characterized by severe performance, power consumption, and size limitations. The high computational complexity of classical MUSIC makes it inapplicable in such conditions. The hybrid method, on the contrary, has a number of properties that make it promising for implementation in onboard computing systems:

- reduced load on central processing unit (CPU);
- the possibility of parallelism;
- adaptability to resources;
- compatible with hardware accelerators.

Thus, the hybrid method is not only significantly superior to classical MUSIC in computational efficiency, but also optimally meets the requirements for signal processing algorithms in onboard computing systems.

4. Architecture of the Onboard Supercomputer

Overcoming the "computational gap" formulated in Section 2 requires not only the development of new dimensionality reduction algorithms but also the creation of a specialized computing architecture capable of implementing these algorithms in real time under severe restrictions on weight, energy consumption, and vibration loads typical of remote sensing aircraft systems. This section proposes the architecture of an onboard supercomputer core based on the principles of heterogeneity, pipelining, and specialized computing coprocessors.

4.1. The Principle of Heterogeneity: Separation of Computing Load

The classical approach to implementing radar algorithms on a universal central processing unit (CPUs) does not provide the required performance when processing MIMO signals. An alternative is to use a heterogeneous computing architecture that combines different types of computing devices, each of which is optimal for its class of tasks. In the proposed architecture, the computing load is distributed as follows. FPGAs are assigned tasks that require high bandwidth and deterministic low latency: pre-filtering, compressed range sampling, fast Fourier transform (FFT) in range and Doppler frequency, as well as primary signal detection. FPGAs provide hardware parallel processing of the data stream coming from analog-to-digital converters (ADCs), which allows for pipelining with minimal latency.

Multicore processors (CPUs) and graphics processing units (GPUs) are tasked with adaptive processing and spectral analysis, requiring high computational complexity and flexibility of algorithms. Such tasks include: estimation and access to low-dimensional covariance matrices, implementation of MUSIC and KAC-JIOAF algorithms, space-time adaptive processing, as well as post-processing (trajectory filtering, targets classification). For performing sequential and control operations CPUs are efficient, while GPUs provide massively parallel processing of matrix operations typical for spectral analysis.

4.2. Pipelining Calculations: Cascading Processing

A key requirement for an onboard remote sensing system is to ensure continuous processing of radar data in real time. The traditional processing model "assemble frame – process – output result" leads to unacceptable latency due to waiting for the completion of the full processing cycle of the previous frame. To eliminate this disadvantage, a pipeline (cascade) data processing architecture is proposed. Figure 2 shows the structural and functional diagram of an onboard supercomputer.

The structural and functional scheme of the mentioned supercomputer consists of 5 main modules. In module 1, the received reflected signals are digitized, transferred to a zero intermediate frequency, and accumulated to increase the signal-to-noise ratio and reduce data flow. The main computing unit in module 1 are FPGAs with buffer memory and ADC boards. In module 2, a space-time vector of reflected signals is formed, the initial covariance matrix R is estimated, and a fast Fourier transform is performed along the time coordinate. The main computing unit of module 2 are FPGAs.

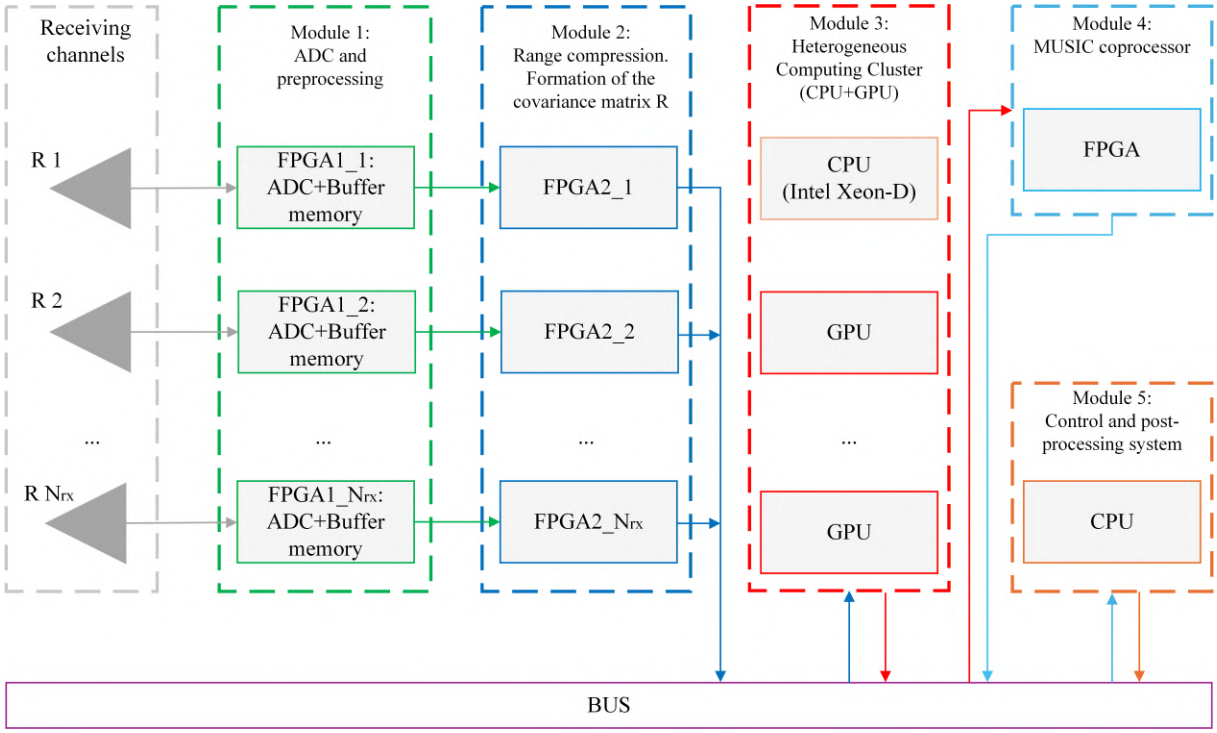


Figure 2. Structural and functional diagram of an onboard supercomputer

Module 3 (heterogeneous computing cluster) implements iterative optimization of KAC-JIOAF for calculating the $\mathbf{B_{opt}}$ projection matrix and the subsequent eigenvalue decomposition of the projected covariance matrix $\mathbf{R_y}$ of dimension $Q \times Q$. The heterogeneous computing cluster includes an Intel Xeon-D multi-core processor and graphics processors based on video cards. The central processor performs the KAC-JIOAF iterative optimization control, the formation $\mathbf{B_{opt}}$ of the projection matrix, and the process stop. Graphics processors perform parallel matrix multiplication.

Module 4 consists of a specialized FPGA computer that performs calculations to find the MUSIC pseudospectrum for the entire grid of directions.

The final stages of processing are performed: filtering of trajectories, iterative refinement of estimates. A central processor is used for these purposes. In addition, modules 3, 4 and 5 are connected by a high-speed bus (bandwidth up to 10 Gbit/s per line), which ensures end-to-end pipelining of processing and a delay of no more than 5 ms per coherent packet. The proposed architecture fully covers the computational needs of the proposed hybrid algorithm and ensures its operation in real time under the conditions of severe weight and size limitations of the airborne remote sensing system.

Conclusions

As a result of the conducted research, it was possible to successfully solve the problem of computational gap in signal processing in MIMO radars for remote sensing aircraft systems. The developed hybrid algorithm demonstrates an impressive reduction in computational complexity by more than 10^8 times compared to the classical MUSIC method.

The practical significance of the work is confirmed by the possibility of implementing real-time signal processing with limited computing resources. The developed approach ensures robustness to small data samples and a high level of compatibility with onboard computing systems.

The prospects of the proposed solution are determined by the possibility of computing parallelism, adaptability to computing system resources, and compatibility with hardware accelerators. It is especially important to maintain high-quality signal processing while significantly reducing the computing load. The practical implementation of the developed algorithms opens up wide possibilities for use in modern aviation complexes. This includes real-time signal processing, improved direction finding accuracy, operation in high-noise environments, reduced power consumption of onboard equipment, and scalability of the solution for various classes of onboard systems.

Thus, the conducted research has made it possible to create an innovative method for processing space-time signals, which can be successfully applied in promising aviation complexes for remote sensing of the Earth. The developed algorithms and architecture of the computing core ensure an optimal balance between computational efficiency and signal processing quality, which makes them suitable for practical use in conditions of severe limitations of onboard systems.

References

1. Bashly, P.N., Denisenko, V.G., Mishchenko, S.E., Shatsky, V.V.: Angular super-resolution MUSIC algorithms based on the NIPALS and Jacobi methods. Antennas (6), 6–14 (2025). https://doi.org/10.18127/j03209601-202506-01

2. Fa, R., Rodrigo, C.: Knowledge-based, low-rank STAP for MIMO radar, based on collaborative iterative optimization of adaptive filters with multiple constraints. In: ICASSP. pp. 2762–2765 (2010)

3. Gupta, P., Kar, S.P.: MUSIC and an improved musical algorithm for estimating the direction of arrival. In: IEEE. pp. 0757–0761 (2015). https://doi.org/10.1109/ICCSP.2015.7322593

4. Lee, J., Stoica, P.: MIMO radar signal processing. Wiley, Hoboken, New Jersey (2008)

5. Lee, J., Strutoy, P.: Transmission diversity in MIMO radar: a look at the radiation pattern. IEEE Signal Processing Letters 13(6), 369–372 (2006)

6. Mohanna, M., Rabeh, M.L., Zier, E.M., Hekala, S.: Optimization of a musical algorithm for estimating the angle of arrival of a signal in wireless communication. NRIAG Journal of Astronomy and Geophysics 2(1), 116–124 (2013). https://doi.org/10.1016/j.nrjag.2013.06.014

7. Strutka, P., Lee, J.: On the identifiability of parameters in MIMO radar. IEEE Transactions on Signal Processing 55(1), 123–134 (2007)

8. Verba, V.S.: Multipurpose tracking in multi-position radar systems. Radio Engineering and Electronics 64(9), 902–909 (2019)

9. Verba, V.S.: Problems of intercepting priority air targets. Information and measurement control systems 17(1), 55–64 (2019)

10. Wang, H., Lee, J.: Space-time coding for MIMO radar. IEEE Transactions on Aerospace and Electronic Systems 45(2), 621–635 (2009)