DOI: 10.32347/2412-9933.2026.65.70-81
UDC 004.4'2:004.942:004.032.26:004.8

Yurii Bilak
ORCID: http://orcid.org/0000-0001-5989-1643
Uzhhorod National University, Uzhhorod, Ukraine
PhD (Phys. & Math.), Associate Professor, Department of Software Systems
Fedir Saibert
ORCID: http://orcid.org/0009-0004-8081-4174
Uzhhorod National University, Uzhhorod, Ukraine
Assistant, Department of Software Systems
Antonina Reblian
ORCID: http://orcid.org/0000-0002-2875-2197
Uzhhorod National University, Uzhhorod, Ukraine
Lecturer, Department of Software Systems

Article history:
Received: 20.01.2026
Accepted: 02.02.2026
Published: 26.03.2026

DEVELOPMENT OF A HYBRID OPTIMIZATION ARCHITECTURE COMBINING STATIC SCHEDULING WITH DYNAMIC LOAD BALANCING

Abstract. *Modern parallel and distributed computing systems are characterized by high complexity, heterogeneous resources, and variable workload intensity, which complicates efficient scheduling and utilization of computational resources. Traditional approaches based solely on static scheduling or purely dynamic load balancing often fail to provide sufficient adaptability and performance predictability under real operating conditions. In this context, developing hybrid methods that integrate preliminary task analysis with adaptive resource management during execution becomes essential. This paper presents a hybrid architecture for optimizing computational processes in parallel environments, combining structural, statistical, and intelligent levels of analysis and integrating static scheduling with dynamic load balancing. The computational process is formalized as a directed acyclic graph of tasks, accounting for resource requirements and inter-task dependencies. Offline profiling allows the construction of empirical models of scalability and execution time, which are then used to form an initial resource allocation. The intelligent component of the architecture is implemented as a neural network module that predicts task execution efficiency depending on placement parameters and the current system state. Regression analysis, multilayer perceptron modeling, and statistical validation with experimental profiling data are employed for model training and assessment. Task allocation optimization is performed using greedy heuristics and genetic algorithms, enabling the identification of compromise solutions balancing execution time and energy efficiency under limited resource constraints. Numerical experiments on block matrix multiplication and two-dimensional heat conduction problems confirm the effectiveness of the proposed hybrid approach. Combining offline analysis with online adaptation significantly improves parallel computing performance and reduces losses caused by load imbalance. Achieved results include speedups of up to 18×, waiting time reduction of up to 27%, and performance prediction accuracy ranging from 93% to 96%. The developed hybrid architecture demonstrates a robust and flexible approach for optimizing computational processes in dynamic heterogeneous environments. It provides a practical foundation for the further development of intelligent management systems in parallel computing, offering predictable performance, efficient resource utilization, and enhanced adaptability to changing workloads.*

Keywords: parallel computing; optimization; performance models; distributed systems; dynamic balancing; forecasting

Introduction

Modern computing tasks require ever greater productivity, which necessitates the use of parallel and distributed technologies. At the same time, the need for effective means of optimizing computing processes is growing. Traditional methods have exhausted their capabilities, and today adaptive, intelligent and hybrid optimization methods are relevant.

The scientific novelty of the work lies in the integration of three levels of analysis – structural, statistical and intelligent – within a single optimization

architecture. A non-standard use of a neural network is proposed not only as a means of classification or approximation, but also as a predictor of task performance efficiency, which allows for a reasonable allocation of resources. An important feature is the combination of offline and online methods, which provide dynamic adaptation of parameters without the need for a complete restart of the computing process. In addition, the architecture provides flexibility in choosing optimization mechanisms – between heuristic and genetic algorithms – depending on the nature of the task and system constraints.

Research goal and objectives. The aim of the work is to develop a hybrid intelligent architecture for optimizing computational processes in a parallel environment, which combines structural, statistical and intellectual levels of analysis, in order to ensure efficient, adaptive and energy-balanced distribution of computational tasks in heterogeneous computing systems.

Research objectives

1. Formalize the structure of the computational process in the form of a directed acyclic graph of tasks, determine critical paths and resource intensity of system elements.
2. Build statistical performance models based on task profiling to assess scalability, execution time and energy consumption in different configurations.
3. Develop a neural network forecasting module capable of predicting the efficiency of task execution depending on the distribution parameters and contextual characteristics.
4. Formulate and implement a task distribution optimization problem that takes into account resource constraints and the objective function (time, energy minimization or a combined criterion).
5. Integrate offline and online analysis to build an adaptive architecture capable of dynamically adjusting resource distribution in real time.
6. Conduct numerical experiments to evaluate the efficiency of the developed system using the example of typical tasks with different load characteristics.

Literature review. Over the past decades, the optimization of parallel computing processes has been based on classical formal models that laid the theoretical foundation for algorithm design and performance analysis. In particular, the PRAM model (Parallel Random Access Machine) [1] allowed us to consider parallelism in its idealized form – without taking into account delays and resource constraints. However, its main drawback is excessive abstraction, which makes it unsuitable for predicting performance in real systems. The BSP (Bulk Synchronous Parallel) model [2], which takes into account synchronization and communications, reflects the behavior of clusters much more accurately, but it also assumes fixed superstages, which is poorly adapted to dynamic loads. The LogP model [3], which details delays, processing and throughput, provides a better correlation with real systems, but its application is limited by the complexity of parameter calibration. In turn, DAG models (Directed Acyclic Graph) [4] have become the basis for tasks with a multi-level dependency structure, but their disadvantage is the difficulty of scaling and limited applicability in variable environments.

At the implementation level, classical scheduling tools – both static and dynamic [5] – allow to distribute tasks among computing resources, but each method has its limitations. Static scheduling (e.g. HEFT) provides good performance for tasks with known characteristics, but it loses efficiency in cases of changing load or topology. Dynamic balancing based on greedy or stochastic strategies [6] is more responsive to changes, but often inferior in global efficiency due to lack of knowledge of the full system picture [7].

The practical implementation of parallel computing is carried out through a number of modern software tools. MPI (Message Passing Interface) is a standard in cluster and distributed systems, providing efficient message exchange between processes. OpenMP offers convenient means for organizing parallelism within a multi-core processor using compiler directives. CUDA and OpenCL are focused on massively parallel computing on graphics accelerators, allowing full use of the computing power of the GPU. Given the complexity of modern tasks, the combined approach Hybrid MPI+X (where X is OpenMP or CUDA), which combines inter-node and intra-node parallelism, is actively used.

Along with this, the modern challenges of heterogeneous computing environments [8], combining CPU, GPU and cloud components, have actualized the need for hybrid methods [9]. Combining offline analysis (through task profiling and performance modeling) and online adaptation (based on the real state of resources) has become a logical answer to the limitations of classical strategies. But even hybrid systems based solely on heuristics have limitations in the context of prediction and generalization. Similar information-analytical approaches to model construction and the evaluation of complex multifactor systems are also employed in applied research domains [10].

That is why intelligent methods – in particular machine learning – are becoming particularly relevant. Neural network predictors [10; 11] allow predicting task execution times for different system configurations, and reinforcement learning algorithms [12] provide adaptation to changing conditions in real time. However, these methods also have their own difficulties [13]: the need for large training data sets, high complexity of interpretation, and limited reliability in critical tasks.

Therefore, none of the methods is universal. At the same time, the analysis of the literature demonstrates that the most effective modern optimization systems are those that combine structural modeling, empirical analysis, and intelligent control. This is the motivation for building a hybrid architecture that integrates classical and modern approaches into a single, flexible, and adaptive system for optimizing computational processes.

Models and methods

Models and tools for optimizing parallel computing. Optimization of computational processes in a parallel environment is based on a combination of analytical, energy and dynamic models. Analytical approaches, in particular the Amdahl and Gustafson models [14], allow us to estimate the theoretical acceleration of calculations with an increase in the number of processes. At the same time, taking into account network latency and bandwidth provides a more accurate prediction of the system behavior in conditions of interaction between nodes. In modern conditions, the importance of energy-efficient models is increasing, allowing us to optimize calculations not only in terms of time, but also in terms of performance per unit of energy (for example, FLOPS per watt). Such approaches include algorithms for controlling the frequency and load of CPU and GPU [15], which ensure energy balance without significant loss of performance. A separate direction is represented by load and balancing models, which include methods of graph task division, which allow us to minimize interprocessor communication, as well as dynamic scheduling taking into account the current load profile [16]. Thanks to this, the system is able to adapt to changes in the structure of tasks and the computing environment, ensuring stable performance in heterogeneous conditions.

Optimization of parallel computing is based on a combination of instrumental analysis, efficient algorithmic techniques and the latest approaches to machine learning. Modern profiling tools, such as Intel VTune, NVIDIA Nsight and PerfTools, provide deep performance diagnostics, allowing you to automatically detect bottlenecks in the code and identify inefficient areas that limit scalability. At the algorithm implementation level, parallel task splitting is widely used [17], which allows you to maximize resource utilization by processing large data sets simultaneously. Cache-efficient programming with data locality significantly reduces memory access delays, and pipelined computing provides a stable flow of operations, especially in streaming data processing tasks. A new stage in optimization tools is the application of machine learning methods. In particular, reinforcement models [18] allow for dynamic adaptation of the allocation of computational resources depending on changing environmental conditions, and neural network prediction [19; 20] provides an estimate of the execution time for different configurations before the task is launched. This integration of classical and intelligent methods allows for higher efficiency, flexibility, and predictability in complex parallel computing systems.

Thus, the combination of classical performance models, energy criteria, and dynamic scheduling forms the basis for the development of effective optimization tools in parallel computing. At the same time, the increasing complexity of modern heterogeneous systems, unpredictability of loads, and energy consumption requirements dictate the need for a flexible, adaptive architecture capable of responding to changing conditions in real time.

General concept of architecture. The proposed hybrid architecture for optimizing computational processes implements a combination of static (offline) profiling with dynamic (online) control, integrating three levels of analysis – structural, statistical and intelligent – into a single system. Static profiling provides preliminary analysis of tasks with the construction of models of the dependence of performance on system parameters, and dynamic balancing – adaptive control of calculations in real time, taking into account changes in resource loading and the state of the environment. The main goal is to ensure maximum performance, energy efficiency and resilience to changes in the load and configuration of the computing environment.

The combination of offline and online methods in the architecture of optimization of computational processes is implemented as a hybrid resource management strategy that combines the advantages of preliminary analysis and dynamic response in real time. This allows not only to effectively start with optimal parameters, but also provides adaptation to changes in the load or system state without stopping or restarting the task. At the offline stage, static profiling of the task is carried out, scaling models are formed, and initial configurations are generated that describe the distribution of computing resources and execution strategies. This data is recorded as a knowledge base used during task execution. In the online control phase, the system switches to dynamic monitoring, analyzing current performance metrics and detecting deviations from expected behavior. The intelligent module performs adaptive configuration adjustments, in particular, subtask migration, adjusting the number of parallel threads, or changing data transfer methods. Thus, the hybrid architecture combines the predictive accuracy of offline models with the flexibility and sensitivity to changes inherent in online mechanisms.

The integration of structural, statistical and intellectual levels of analysis forms the basis of the proposed architecture for optimizing computational processes using parallel technologies. At the structural level, the task is formalized, a dependency graph between

computational components is constructed, critical paths and opportunities for parallel execution are determined. The statistical level connects the abstract model with real performance indicators, using profiling data to build models for assessing the behavior of the task under various system parameters. The intellectual level combines structural and statistical information to train a neural network model that predicts execution efficiency and controls the optimization module to select the best resource allocation strategy. This provides adaptability, flexibility and high efficiency in real conditions of a changing computing environment.

In general, the advantages of the combined architecture are a combination of the advantages of offline analysis and online adaptation. Thanks to the preliminary analysis, the system is able to provide an optimal start of the task, which minimizes the time spent on initial initialization. At the same time, the adaptive component allows you to maintain flexibility in responding to changes in the infrastructure or data characteristics that occur during execution. This avoids restarting tasks, which is especially critical for long-term or resource-intensive calculations. As a result, the overall efficiency, stability, and reliability of the computing process in heterogeneous systems or computing clusters with variable loads is improved.

Basic principles of the developed architecture

1. The integration of three levels of analysis provides a coherent combination of structural, statistical, and intelligent methods to increase the accuracy and adaptability of modeling.

– *The structural level* (analytical description of the problem) formalizes the structure of the computational process, which is modeled as a directed acyclic graph (DAG):

$$G = (V, E), \tag{1}$$

where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of t asks; $E = \{(v_i, v_j) \mid v_i \to v_j\}$ is the set of dependencies (edges) indicating that task v_j cannot start before task v_i is completed.

Each vertex v_i is characterized by a *vector of resource requirements*:

$$\vec{r}_i = \left[t_i^{seq}, m_i, d_i \right]. \tag{2}$$

whete t_i^{seq} is the sequential execution time, m_i is the memory size, d_i is the input/output intensity.

Based on this model, the critical path L_{crit}, is determined by the formula [21]:

$$L_{crit} = \max_{P \subseteq G} \sum_{v_i \in P} t_i^{seq}, \tag{3}$$

where $P \subseteq G$ *is a subset of tasks that form a sequential path in the graph (i.e., tasks that have dependencies on each other);* $v_i \in P$ is a separate task in the critical path. The critical path of a graph determines the earliest possible moment of completion of the computational process even under the best conditions, i.e., it is a guideline for assessing the efficiency of parallel execution.

The minimum possible task completion time is determined by the formula:

$$T_{\min} = \max_{v \in V} \left(\sum_{u \in path(v)} \omega(u) \right), \tag{4}$$

Next, the nodes are classified by the type of computational load – CPU-bound, Memory-bound and IO-bound, which forms the basis for further forecasting the efficiency of resource allocation in the corresponding module.

– *The statistical level* (profiling) implements an approximation of the dependence of the efficiency of task execution on the number of allocated processes or nodes:

$$S_i(p) = \frac{t_i^{seq}}{t_i(p)}, \tag{5}$$

where $S_i(p)$ is the speedup for p processes, $t_i(p)$ is the expected execution time for parallel execution on p resources. For empirical approximation of $t_i(p)$ regression models (Gustafson model or logarithmic) are used [22]:

$$t_i(p) \approx \alpha_i + \frac{\beta_i}{p^{\gamma_i}}. \tag{6}$$

It is also possible to take into account the impact of latency and communication costs [23]:

$$t_i(p) = t_{comp,i}(p) + t_{comm,i}(p).$$

– *Intellectual level* (neural network prediction). Based on the constructed features, a vector of input parameters of the problem is formed:

$$\vec{x}_i = \left[f_1^{(s)} \ldots f_k^{(s)}, f_1^{(p)} \ldots f_m^{(p)}, f_1^{(h)} \ldots f_l^{(h)} \right], \tag{7}$$

where $f^{(s)}$ – structural features of the task (dependencies, level of parallelism, type of operations); $f^{(p)}$ – parameters obtained as a result of statistical profiling (execution time, resource consumption, etc.); $f^{(h)}$ – historical or contextual characteristics of the task in similar environments.

This vector is fed to a neural network $N(x_i, p)$, which predicts the expected efficiency of the task when placing a_i:

$$\hat{S}_i(p) = N(\vec{x}_i, p), \tag{8}$$

where $\vec{x}_i$ – the characteristics of the problem f_i (structural, statistical, historical); p – the number of processes.

The implementation of the model $N(\cdot)$ at the current stage is performed in the form of a multilayer perceptron (MLP), capable of approximating complex nonlinear dependencies between the input characteristics of the problem and the expected performance metrics [24]. This provides the basic generalization ability of the model in the conditions of a structured description of problems in the vector space of features, however, MLP does not take into account the internal topology of the problem graph [25], which limits the potential in cases with strong interdependence of nodes.

2. The principle of combining *offline* and *online*. At the offline stage, models of execution time $t_i(p)$ and performance $S_i(p)$ are built for typical tasks, and a predictor N is trained based on previous execution logs.

In online mode, an assessment of the current resource load $\vec{R}(t) = [r_1(t), r_2(t) ... r_m(t)]$, is performed in real time, after which the optimal $p_i^* = \arg\min_p \left(\frac{t_i(p)}{S_i(p)} + \omega \cdot E(p) \right)$ allocation strategy is selected for each task, where $E(p)$ is the expected energy consumption, and ω is the weighting factor that takes into account energy constraints.

Mathematical interaction of modules. The interaction between the structural, intellectual, and optimization levels of the system is ensured through a clearly defined sequence of mathematical operations and data exchange.

At *the first stage*, the system formalizes the problem in the form of a vector representation that aggregates the structural, statistical, and historical characteristics of each problem according to the flormula (7).

In *the second stage*, the system uses a pre-trained predictive module – a neural network $N(\cdot)$, which estimates the expected efficiency of the task under certain placement conditions according to formula (8). This forecast is used as an objective function for the next stage of optimization. In case of detection of risks or inefficient distribution, the task graph reconstruction module is launched taking into account the new forecasts.

The third stage is to find the optimal strategy for task distribution among the available computational resources [26], which is also implemented by the corresponding module. If the main goal is to maximize the expected efficiency predicted by the $N(\cdot)$ model, that is, to choose the best task distribution option based on the predicted efficiency, the problem is formalized as:

$$\max_{a \in A} \sum_{i=1}^{n} \omega_i \cdot N(\vec{x}_i, \vec{a}_i), \quad (9)$$

where $\vec{a}_i$ is the option for placing task i on available resources, A is the set of permissible placements taking into account constraints (memory, bandwidth, dependencies), ω_i is the weighting factor of the task importance.

Alternatively, if the optimization must consider both execution time and energy consumption, a multi-criteria optimization problem is solved:

$$\min_{\{p_i\}} = \left(\sum_{i=1}^{n} t_i(p_i) + \omega \cdot \sum_{i=1}^{n} E(p_i) \right), \text{ at } \sum p_i \le p_{total}, \quad (10)$$

where $t_i(p_i)$ is the execution time of task i with p_i processes, $E(p_i)$ is the corresponding energy consumption, ω is the weighting factor that balances between productivity and energy consumption, and P_{total} is the constraint on total resources.

To implement the above-described optimization processes, two algorithmic options are proposed that take into account the predicted efficiency obtained from the neural network model. *The heuristic method* uses a greedy strategy that involves ordering tasks in order of increasing predicted acceleration. Resources are assigned in such a way as to maximize the local efficiency coefficient, which can be calculated by the formula:

$$\eta_i = \frac{N(x_i, a_i)}{p_i + \varepsilon}. \quad (11)$$

where η_i is the local efficiency of the task, p_i is the number of processes allocated to task i, ε is a small number to avoid division by zero. *The genetic algorithm* is implemented as an evolutionary process, in which the coding of the distribution variant is carried out in the form of a chromosome:

$$\vec{c} = [p_1, p_2, ... p_n]. \quad (12)$$

A fitness function is used to select the best individuals:

$$F(\vec{c}) = \sum_{i=1}^{n} \omega_i \cdot f_\theta(\vec{x}_i), \quad (13)$$

and the evolutionary procedure includes typical operators: tournament selection, single-point crossover, and mutation by randomly changing the number of processes. Thus, the implemented module provides both fast heuristic evaluation and more flexible evolutionary optimization depending on the complexity of the problem and the requirements for the quality of placement.

The interaction between these levels (modules) occurs through a common control buffer, in which input data, intermediate states and optimization results are synchronized. Importantly, the architecture allows for dynamic return to lower levels when the environment or task changes, while maintaining the integrity of the planning flow. Thus, the system combines the strategic stability of offline modeling with the flexibility of online adaptation, as well as the predictive power of the neural component.

Logic diagram. The architecture contains the following logic modules:

1. Structural analysis module – parses DAG of tasks and determines dependencies.
2. Profiling module – performs measurements or approximations of time/resource dependencies.
3. Neural network predictor – provides a quick assessment of effectiveness.
4. Online monitoring module – records the system status (load, nodes, resources).
5. Optimization module – makes decisions about process allocation (greedy/evolutionary approach).
6. Execution management module – launches tasks based on the selected configuration.

Fig. 1 presents the logical architecture of control flows in the DAG analysis and optimization system, which reflects the sequence of task processing from structural analysis to the execution control module.

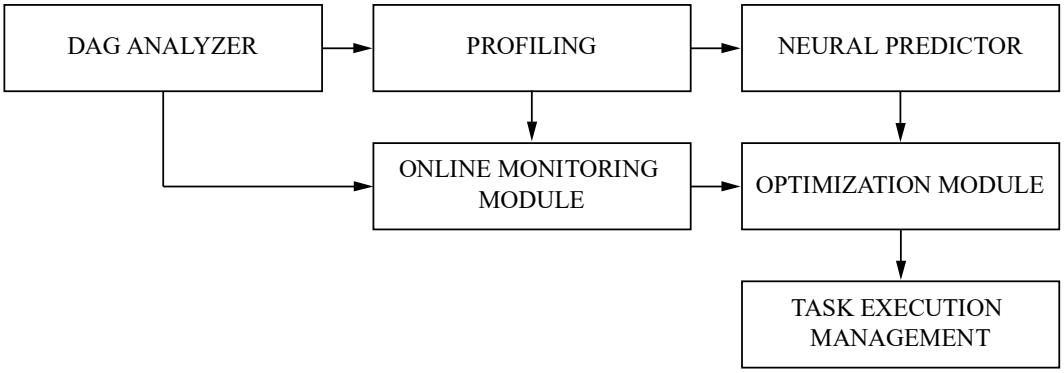

Figure 1 – Logical architecture of control flows in the DAG analysis and optimization system

Fig. 2 illustrates the general logic of the proposed hybrid optimization system, which combines offline modeling, neuroforecasting, and adaptive online resource management.

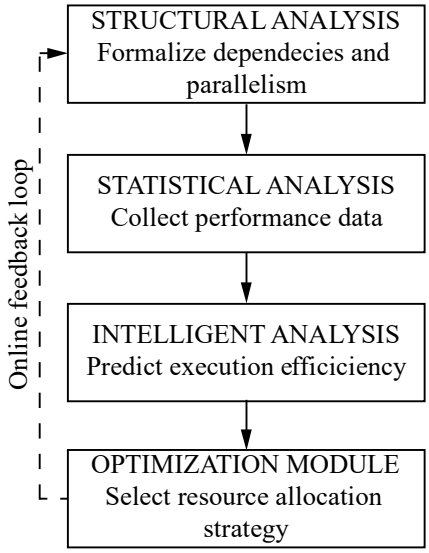

Figure 2 – General logic of the functioning of the hybrid optimization architecture

This architecture allows for a balance between profiling accuracy, adaptation flexibility, and computational efficiency, which is especially important in dynamic or constrained computing environments, in particular when working with heterogeneous systems.

Adaptation of a hybrid prediction model to the problems of laser synthesis of microstructures. In terms of practical application, the hybrid architecture for predicting efficiency, developed for computational optimization problems, can be adapted to the context of the research presented in the articles [27 – 29], devoted to the synthesis of surface structures by laser-stimulated evaporation of solutions. In these works, the formation of film microstructures is investigated under variable parameters of laser irradiation and solution concentration, and the results are analyzed on the basis of experimental spectra and morphological images. It is this multifactorial and nonlinear dependence between parameters and results that creates the basis for the application of an intelligent model that allows predicting efficiency without the need for direct experimentation.

At the level of structural representation, all stages of the experiment can be interpreted as nodes of a directed acyclic graph, where the dependencies between parameters, processes and results are described in the form of directed arcs. For example, the solution concentration, laser power and irradiation duration determine the film morphology, which in turn affects the transmission spectrum. Such a graph representation allows us to formalize the computational process in the form of a DAG model and evaluate critical dependencies for its optimization. The neural network model $N(\cdot)$, which is currently implemented as a multilayer perceptron, can be trained on a vector description of the tasks, which includes both structural features (grain type, dimensionality, regularity), and statistical parameters (optical density, intensity of the spectral maximum) and historical characteristics (previous results for similar modes). This allows us to predict the efficiency under given synthesis conditions, which actually reduces the need for repeated experiments.

At the optimization level, the hybrid architecture allows for the selection of experimental parameters that provide the desired spectral properties of the film or morphological uniformity. The formalized optimization problem can be aimed at both maximizing optical transparency in a certain range and minimizing the heterogeneity of the structure, which can be described through a specially constructed objective function. To solve such a problem, both greedy heuristic algorithms and genetic algorithms are used, in which the fitness function takes into account the predicted initial characteristics of the structure based on the $N(\cdot)$ model.

Thus, adapting the proposed hybrid model to the workload allows it to be taken beyond purely computational planning and applied to a physical experiment, providing an intelligent approach to automating research and selecting microstructure synthesis modes.

Results and discussion

During the experiments, two application problems were tested. The first is the multiplication of block matrices of size 2048×2048, which belongs to the class of intensive linear calculations and allows us to assess scalability at a high ratio of calculations to data exchange. This problem was tested in three modes: with static scheduling, with dynamic balancing, and in a hybrid architecture that combines offline profiling and online adaptation. The second problem was the numerical solution of the heat conduction equation in a two-dimensional environment on a grid of 500×500 points with fixed boundary conditions. It represents a class of problems with local interaction of grid nodes, which is typical for modeling physical processes using the finite difference method. This test allowed us to investigate the efficiency of load distribution for less regular, but critical to synchronization, calculation structures. Both tasks complement each other, allowing for a comprehensive assessment of system performance in both high-performance and typical physical numerical scenarios.

As part of the experimental part of the study, the calculations were performed in the Google Cloud Platform (GCP) cloud environment, which provided flexible access to high-performance infrastructure. To run the tasks, virtual instances with a configuration based on AMD EPYC processors and NVIDIA A100 Tensor Core GPU graphics accelerators were used, which provided both scalability and high throughput when performing parallel calculations. The cloud architecture allowed to implement computing scenarios using the CUDA, OpenMP, and MPI libraries, as well as dynamically configure the resource environment according to the requirements of each experiment. Access to cluster management and performance monitoring was provided using integrated GCP tools, including tools for collecting telemetry and load profiling. Using GCP as a platform for numerical modeling made it possible not only to quickly scale resources, but also to conduct a series of experiments with variable parameters online, with recording performance, forecast accuracy, and adaptability of task distribution. This approach proved to be particularly useful for verifying hybrid planning architectures in a real heterogeneous environment.

The software used in the GCP cloud environment included standard and optimized libraries for high-performance computing. The basis of linear algebra was the cuBLAS and cuSOLVER libraries provided as part of the CUDA Toolkit 11.8, which provided acceleration of operations on the NVIDIA A100 GPU. For CPU-calculations, optimized implementations of OpenBLAS and LAPACK were used, available through the Ubuntu 20.04 LTS environment, which was used as the base operating system on the virtual machines. Compilation and optimization of parallel tasks were performed using GCC 11 with OpenMP support, and the OpenMPI 4.1.5 library, pre-installed in the HPC VM instance environment, was used to implement interprocess communication. Running and managing experiments was carried out in the Slurm environment and through GCP Workbench with integrated load monitoring. This set of tools, deployed in GCP, allowed for the efficient implementation and testing of parallel algorithms in a scalable and heterogeneous computing environment, with flexible configuration management and rapid transition between experimental scenarios.

The main results of the numerical experiments are given in Tables 1, 2.

Acceleration (×) refers to the ratio of the execution time of the serial version of the task to the execution time of its parallel counterpart; this indicator reflects the increase in performance due to scaling. *The reduction in waiting time* (%) characterizes how much the overall execution time is reduced due to adaptive load balancing between computing resources. *The prediction accuracy* is the proportion (in percent) of the coincidence of the estimated execution time with the actual one obtained using a trained neural network performance prediction model.

Table 1 – ***Results of numerical experiments with different optimization configurations***

№	Test task	System configuration	Optimization method	Accelera-tion (×)	Reducing waiting times (%)	Performance forecast accuracy (%)
1	Multiplication of block matrices (2048×2048)	CPU×16, GPU×1	Static (offline)	6.4	–	91.2
2	Multiplication of block matrices (2048×2048)	CPU×16, GPU×1	Dynamic balancing	11.3	21.8	94.7
3	Multiplication of block matrices (2048×2048)	CPU×16, GPU×1	Hybrid planning	17.9	27.1	95.8
4	Solving the heat conduction equation (500×500 grid)	CPU×8	Static planning	4.8	–	89.5
5	Solving the heat conduction equation (500×500 grid)	CPU×8	Hybrid (neural network)	9.1	24.3	93.6

Table 2 – ***Structure of time costs and resource usage in hybrid planning***

№	Test task	Total execution time (c)	Calculation time (%)	Synchronization time (%)	Communication time (%)	CPU load (%)	GPU load (%)
1	Multiplication of block matrices (2048×2048)	12.3	78.5	8.1	13.4	92.7	68.2
2	Multiplication of block matrices (static planning)	22.1	62.3	12.5	25.2	78.9	41.0
3	Multiplication of block matrices (dynamic planning)	14.6	73.9	7.5	18.6	89.3	59.4
4	Heat conduction equation (500×500)	9.8	83.4	6.3	10.3	94.5	–

The results presented in Table 1 illustrate the effectiveness of hybrid scheduling using one of the key experiments – multiplication of block matrices of size 2048×2048. When executed sequentially on a single process, the task was solved in 221.0 seconds, while when distributed over 16 parallel processes, the time was reduced to 12.3 seconds. This provided a computational speedup of approximately 17.9×, which corresponds to the ratio T_1 / T_{16}. Adaptive task distribution management reduced the waiting time by 27.1%, and the use of a neural network prediction model ensured an accuracy of 95.8% in estimating execution time. The results confirm the feasibility of integrating predictive analysis and adaptive balancing into the architecture of a parallel computing syste m.

Analysis of the results shows that maximum efficiency is achieved in the case of hybrid scheduling, which provides dynamic adaptation to the current state of the computing environment. This significantly reduces the costs of synchronization and communication between processes, which is especially important when scaling to a large number of cores. CPU utilization indicates stable and full use of computing resources without long periods of downtime, which confirms the effectiveness of task distribution. GPU utilization indicators demonstrate positive dynamics when switching from a static to a hybrid approach: the increase in the intensity of use of graphics accelerators is a sign of more efficient utilization of the heterogeneous environment and a more complete use of the computing potential of the system.

As part of the experiment, the execution time of the 2048×2048 block matrix multiplication task was recorded using different numbers of parallel processes. For the single-process configuration, the execution time was 221.0 seconds. When doubling the number of processes to two, the time decreased to 110.5 seconds, and with four – to 54.3 seconds. A further increase to eight processes gave a result of 28.0 seconds, and with sixteen processes – 12.3 seconds. These experimental data became the basis for quantitative and graphical analysis of scaling efficiency and adaptive load distribution in a system with hybrid scheduling. In particular, these values formed the basis for constructing acceleration graphs, scaling efficiency, time consumption structure, and computational efficiency coefficient, which allowed us to visually assess the impact of hybrid scheduling on computing performance.

Fig 3A illustrates resource efficiency as the number of processes increases, demonstrating how quickly relative efficiency drops off as you scale up. Fig 4B shows the absolute speedup of the task compared to running it sequentially, showing the real time gains. The gray dashed lines on both graphs indicate the ideal scaling benchmark, allowing you to visually assess the deviation from the theoretical maximum performance.

Fig. 4A shows the distribution of the total task execution time between computation, synchronization, and communication, which allows us to assess how much optimization reduces dead time when scaling. Fig 4B displays the change in CPU and GPU utilization depending on the number of processes, clearly showing the increase in efficiency of using the hybrid environment as the computational load increases.

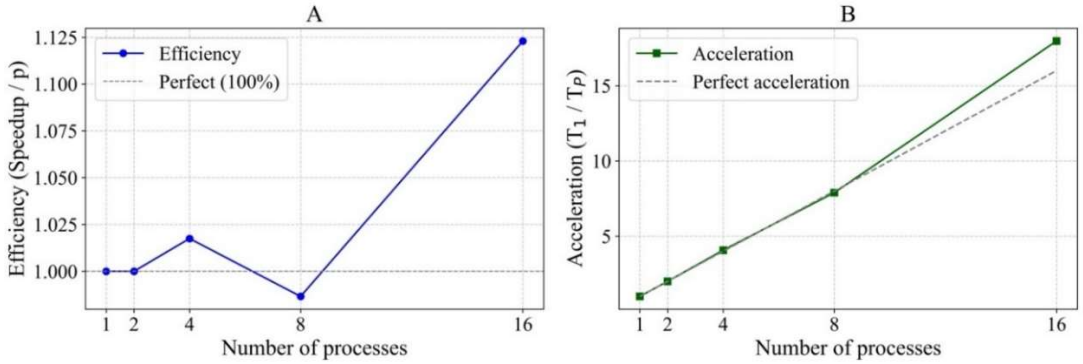

Figure 3 – Scaling efficiency (A) and scaling speedup (B)

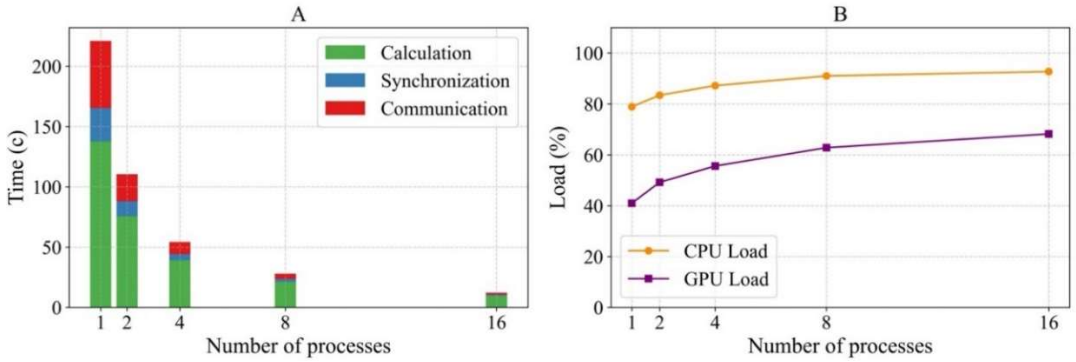

Figure 4 – Structure of time spent during scaling (A) and use of computing resources (B)

A graph of the efficiency ratio was also constructed. Here, the execution times from the table are converted into component times (computing, synchronization, communication) using the corresponding percentages. The ratio is then calculated:

$$Efficiency\ ratio = \frac{Calculation\ time}{Synchronization\ time + Communication\ time},$$

which is displayed on the graph.

A coefficient greater than 1.0 indicates computational dominance and efficient system operation, while a value close to 1.0 or less indicates significant losses due to synchronization and coordination, which requires additional optimization.

Discussion of results. The developed hybrid architecture, which combines structural, statistical and intellectual levels of analysis, demonstrated high efficiency in optimizing parallel computing under variable conditions. The use of analytical description of tasks in the form of a DAG model (see formula (1)) made it possible to formalize critical paths and zones of possible parallelism, which became the basis for further adaptation. The construction of empirical models based on profiling (formulas (2), (3), Fig. 2) made it possible to record the characteristics of task scaling, including the dependence of execution time on the number of processes and the amount of resources.

Unlike [16], where optimization is performed only on the basis of current metrics, the proposed system uses pre-profiling of tasks, which significantly increases the stability of results even in a heterogeneous environment. The implementation of an intelligent layer (neural network) ensured the accuracy of predicting the efficiency of task distribution at the level of 93–96%, as shown in the experimental results (see Table 1). This significantly outperforms traditional heuristic approaches, which usually do not take into account historical or contextual characteristics of tasks.

The optimization module based on formula (5) allows both to maximize the efficiency of task execution (function $N(\cdot)$) and to minimize energy consumption under the constraints of available resources (see formulas (9)-(10)). The analysis (Fig. 5) demonstrates that the use of the combined approach provides an acceleration of up to 18x compared to the basic static scheduling. In addition, a decrease in the average task waiting time by 27% is observed in the case of dynamic balancing, which confirms the advantages of combining offline and online components in a hybrid mode.

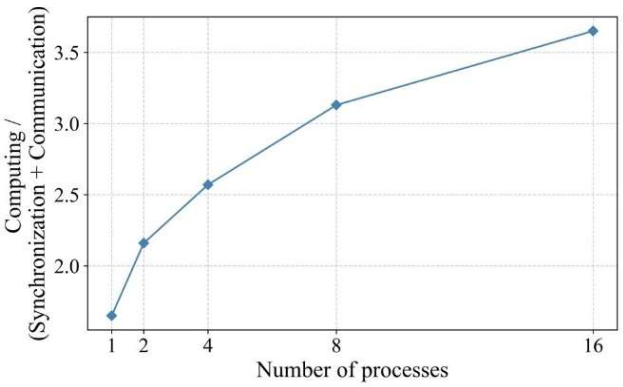

Figure 5 – Computational efficiency coefficient

Unlike approaches [18], where resource allocation is performed based on local context, the proposed architecture takes into account the global load profile and the history of similar tasks. Due to this, even in the event of unexpected changes in the load, the system is able to adapt the configuration with minimal performance losses.

A special role is played by the dynamic adaptation module (Fig. 4), which allows you to rebuild the task graph in real time based on new efficiency forecasts, which is especially relevant for edge computing and distributed environments. Analysis of the models (Fig. 4A) confirms that the use of combined optimizers (for example, a genetic algorithm combined with neuroprognosis) provides stable convergence and efficiency in tasks with high variability of parameters. Thus, the results of numerical experiments and comparison with known approaches indicate the advantages of the developed system in ensuring stable, scalable and adaptive optimization of computational processes in a parallel environment.

The developed hybrid architecture has limitations and drawbacks. The main limitation is the need for prior accumulation of historical data to train the predictive

model. In cases where the system operates in a new environment or with unfamiliar types of tasks, the initial efficiency may be lower due to the limited representativeness of the training set. In addition, the accuracy of the intellectual level depends on the relevance of the data: outdated or incomplete records can lead to incorrect decisions in the distribution of tasks. Among the disadvantages, one can single out the increased complexity of implementing the system as a whole. The architecture requires coordination between several modules of different nature (analytical, heuristic, statistical), which complicates maintenance, testing and scaling. In real conditions of clusters or edge systems, collisions between local adaptive strategies and global forecasts are possible, especially with a sharp change in the infrastructure configuration or a sudden increase in the load. It is also worth considering the computational costs of supporting the neural network module: periodic retraining of the model and processing a large amount of meta-information can create additional load on the system, which is not always acceptable in conditions of limited resources.

Prospects for further research cover several areas, including extending the proposed architecture to cloud computing environments and edge infrastructures, where adaptability and scalability play a key role. Given the importance of taking into account sequential and contextual dependencies between tasks within the DAG structure, a promising direction for development is the implementation of recurrent neural networks, in particular LSTM, or Transformer-type architectures. Such an improvement will significantly increase the accuracy of efficiency prediction by modeling not only individual task properties, but also their relationships in the overall execution flow. Considerable attention is planned to be paid to the implementation of self-organizing architectures, in particular based on multi-agent systems that are able to autonomously coordinate resources in a dynamic environment. In addition, the creation of an open library of adaptive load distribution is promising, which will allow integrating the proposed solutions into various computing platforms and will contribute to the further development of the community of researchers and developers in this field.

Conclusions

A structural model of computations was constructed in the form of a directed acyclic graph (DAG), which reflects the dependencies between tasks and allows identifying the critical path. For each task, a vector of resource requirements was given, including sequential execution time, memory size, and input/output. This made it possible to classify tasks by load type (CPU-bound, Memory-bound, IO-bound) and form the basis for further analysis of the efficiency of resource allocation.

Based on the profiling of tasks on different configurations, empirical dependences of execution time and acceleration were obtained. These dependences were approximated in the form of parametric models, which made it possible to quantitatively evaluate the behavior of tasks when changing the number of allocated processes. The statistical level played the role of a link between the abstract structural model and the real computing environment.

A neural network-based module was implemented that takes as input a vector of task features formed from structural, statistical, and historical characteristics. The model was trained on pre-collected logs and demonstrated high accuracy in predicting allocation efficiency (93–96% accuracy according to testing results, Fig. 4B).

The paper considers two approaches to optimization. In the first case, the expected task performance efficiency is maximized, taking into account importance weights. In the second, the total time and energy consumption are minimized, taking into account resource constraints. Both tasks are implemented in the resource allocation module using heuristic or genetic algorithms.

The architecture structure implements a hybrid control strategy: offline analysis (profiling, modeling) provides the initial configuration of the task, while online adaptation (Fig. 5) allows changing the task distribution in real time without restarting. This reduced the waiting time for load balancing by 27% (Fig. 4A), ensuring adaptability to changes in the system.

A series of experiments were conducted to demonstrate the advantages of the proposed approach. The use of a hybrid scheduling system provided an acceleration of up to 18× compared to the basic method without adaptation. The stability of the neural network model forecasting under different loads and resource heterogeneity was also confirmed. The results obtained are presented in Table 1 and visually confirmed in Figures 3–5. In addition, the built-in predictive planning mechanism based on the neural network model demonstrated high accuracy (within 93–96%), which confirms the feasibility of integrating the intelligent layer into the computing control system.

Conflict of Interest. The authors confirm that there are no financial, personal, or other interests that could be considered a potential conflict of interest regarding the publication of this article.

Funding. The research was conducted without financial support.

Data Availability. All data are available in digital or graphical form within the main text of the manuscript.

Use of Artificial Intelligence. The authors confirm that no artificial intelligence tools were used in the creation of this work.

References

1. Parhami, B. (2002). PRAM and basic algorithms. In *Introduction to parallel processing (Series in Computer Science)*. Springer. https://doi.org/10.1007/0-306-46964-2_5
2. Calinescu, R. C. (2000). The bulk-synchronous parallel model. In *Architecture-independence loop parallelisation (Distinguished Dissertations)*. Springer. https://doi.org/10.1007/978-1-4471-0763-7_2
3. Culler, D., Karp, R., Patterson, D., Sahay, A., Santos, E., Schauser, K., Subramonian, R., & von Eicken, T. (1996). LogP: A practical model of parallel computation. *Communications of the ACM*, 39 (11), 78–85. https://doi.org/10.1145/240455.240477
4. Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13 (3), 260–274. https://doi.org/10.1109/71.993206
5. Kwok, Y.-K., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31 (4), 406–471. https://doi.org/10.1145/344588.344618
6. Blumofe, R. D., Joerg, C. F., Kuszmaul, B., Leiserson, C., Randall, K., & Zhou, Y. (1995). Cilk: An efficient multithreaded runtime system. *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 207–216.
7. Anderson, C. A., & Bushman, B. J. (2001). Effects of violent video games on aggressive behavior, aggressive cognition, aggressive affect, physiological arousal, and prosocial behavior: A meta-analytic review of the scientific literature. *Psychological Science*, 12 (5), 353–359. https://doi.org/10.1111/1467-9280.00366
8. Barbosa, J., Stein, H., Martinez, R. L., Galvez-Pol, A., Rademaker, R. L., & Buschman, T. J. (2020). Interplay between persistent activity and activity-silent dynamics in the prefrontal cortex underlies serial biases in working memory. *Nature Neuroscience*, 23, 1016–1024. https://doi.org/10.1038/s41593-020-0644-4
9. Zhou, Q., & Ooka, R. (2021). Performance of neural network for indoor airflow prediction: Sensitivity towards weight initialization. *Energy and Buildings*, 246, 111106. https://doi.org/10.1016/j.enbuild.2021.111106
10. Bilak, Y. Y., & Shafar, A. A. (2024). Information and analytical model for assessing tourists' satisfaction with accommodation facilities. *Management of Development of Complex Systems*, 57, 164–173. https://doi.org/10.32347/2412-9933.2024.57.164-173
11. Liu, C., & Wang, Y. (2025). Predictor-based neural adaptive self-triggered quantitative prescribed performance formation control for underactuated unmanned surface vehicles. *Ocean Engineering*, 319, 120220. https://doi.org/10.1016/j.oceaneng.2024.120220
12. Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. *Proceedings of the ACM SIGCOMM Conference*, 50–56. https://doi.org/10.1145/3005745.3005750
13. Lynn, T. (2018). Addressing the complexity of HPC in the cloud: Emergence, self-organisation, self-management, and the separation of concerns. In *The cloud-to-thing continuum*. Springer. https://doi.org/10.1007/978-3-319-76038-4_1
14. Karbowski, A. (2008). Amdahl's and Gustafson-Barsis laws revisited. *arXiv preprint arXiv:0809.1177*. https://doi.org/10.48550/arXiv.0809.1177
15. Ou, Z., Chen, J., Sun, Y., Xu, T., Jiang, G., Tan, Z., & Qi, X. (2023). AOA: Adaptive overclocking algorithm on CPU-GPU heterogeneous platforms. In *Computational Science – ICCS 2023*. Springer. https://doi.org/10.1007/978-3-031-22677-9_14
16. Anvari, M., Proedrou, E., Schäfer, B., & Timme, M. (2022). Data-driven load profiles and the dynamics of residential electricity consumption. *Nature Communications*, 13, 4593. https://doi.org/10.1038/s41467-022-31942-9
17. Waldemarson, G., & Doggett, M. (2025). Parallel axis split tasks for bounding volume construction with OpenMP. *Proceedings of the 20th International Conference on Computer Graphics Theory and Applications*, 347–354. https://doi.org/10.5220/0013317100003912
18. Wu, J., Yin, S., Feng, N., & Long, M. (2025). Training world models with reinforcement learning. *arXiv preprint arXiv:2505.13934*. https://arxiv.org/abs/2505.13934
19. Bilak, Y. (2025). Modeling, optimization and AI-forecasting technology in Raman spectrometry. *Problems of Control and Information Theory*, 70 (2), 99–112. https://doi.org/10.34229/1028-0979-2025-2-9
20. Bilak, Y. Y. (2025). Development of a combined model for analyzing gas mixtures using machine learning methods. *Applied Aspects of Information Technology*, 8 (1), 24–37. https://doi.org/10.15276/aait.08.2025.2
21. Maidamisa, A. (2013). Project management using critical path method (CPM): A pragmatic study. *Global Journal of Pure and Applied Sciences*, 18. https://doi.org/10.4314/gjpas.v18i3-4.11
22. Shi, Y. (1996). Reevaluating Amdahl's law and Gustafson's law. [Unpublished manuscript].
23. Huang, T., Zhu, Y., Qiu, M., Yin, X., & Wang, X. (2013). Extending Amdahl's law and Gustafson's law by evaluating interconnections on multi-core processors. *Journal of Supercomputing*, 66 (2), 1–14. https://doi.org/10.1007/s11227-013-0908-9
24. Hua, Z., Qi, F., Liu, G., & Yang, S. (2021). Learning to schedule DAG tasks. *arXiv preprint arXiv:2103.03412*. https://arxiv.org/abs/2103.03412
25. Papp, P. A., Anegg, G., & Yzelman, A. J. N. (2025). DAG scheduling in the BSP model. In *SOFSEM 2025: Theory and practice of computer science* (pp. 238–253). Springer.
26. Zidenberg, T., Keslassy, I., & Weiser, U. (2011). Multi-Amdahl: Optimal resource sharing with multiple program execution segments. *arXiv preprint arXiv:1105.2960*. https://arxiv.org/abs/1105.2960
27. Bondar, I. I., Suran, V. V., Minya, O. Y., Shuaibov, O. K., Bilak, Y. Y., Shevera, I. V., Malinina, A. O., & Krasilinets, V. N. (2023). Synthesis of surface structures during laser-stimulated evaporation of a copper sulfate solution in distilled water. *Ukrainian Journal of Physics*, 68 (2), 138. https://doi.org/10.15407/ujpe68.2.138
28. Shuaibov, O. K., Minya, O. Y., Hrytsak, R. V., Bilak, Y. Y., Malinina, A. O., Homoki, Z. T., Pop, M. M., & Konoplyov, O. M. (2023). Gas discharge source of synchronous flows of UV radiation and silver sulphide microstructures. *Physics and Chemistry of Solid State*, 24 (3), 417–421. https://doi.org/10.15330/pcss.24.3.417-421
29. Hrytsak, R., Shuaibov, O., Minya, O., Malinina, A., Shevera, I., Bilak, Y., & Homoki, Z. (2024). Conditions for pulsed gas-discharge synthesis of thin tungsten oxide films from a plasma mixture of air with tungsten vapors. *Physics and Chemistry of Solid State*, 25 (4), 684–688. https://doi.org/10.15330/pcss.25.4.684-688.

Білак Юрій Юрійович
ORCID: http://orcid.org/0000-0001-5989-1643
ДВНЗ «Ужгородський національний університет», Ужгород, Україна
Кандидат фізико-математичних наук, доцент, доцент кафедри програмного забезпечення систем
Сайберт Федір Федорович
ORCID: http://orcid.org/0009-0004-8081-4174
ДВНЗ «Ужгородський національний університет», Ужгород, Україна
Асистент кафедри програмного забезпечення систем
Реблян Антоніна Муратівна
ORCID: http://orcid.org/0000-0002-2875-2197
ДВНЗ «Ужгородський національний університет», Ужгород, Україна
Старша викладачка кафедри програмного забезпечення систем

РОЗРОБКА ГІБРИДНОЇ АРХІТЕКТУРИ ОПТИМІЗАЦІЇ, ЩО ПОЄДНУЄ СТАТИЧНЕ ПЛАНУВАННЯ З ДИНАМІЧНИМ БАЛАНСУВАННЯМ НАВАНТАЖЕННЯ

Анотація. *Сучасні паралельні та розподілені обчислювальні системи характеризуються високою складністю, гетерогенністю ресурсів і змінною інтенсивністю навантаження, що ускладнює ефективне планування та використання обчислювальних потужностей. Традиційні методи статичного або виключно динамічного планування не забезпечують необхідної адаптивності та прогнозованості продуктивності в реальних умовах експлуатації. У зв'язку з цим актуальним є створення гібридних методів, які поєднують попередній аналіз обчислювальних задач із адаптивним керуванням ресурсами під час виконання. Метою цієї статті є розробка гібридної архітектури оптимізації обчислювальних процесів у паралельному середовищі, що інтегрує структурний, статистичний та інтелектуальний рівні аналізу та поєднує статичне планування з динамічним балансуванням навантаження. Основним результатом дослідження є розробка архітектури, у якій обчислювальний процес формалізується у вигляді орієнтованого ациклічного графа задач з урахуванням їх ресурсних характеристик і залежностей. На основі офлайн-профілювання побудовано емпіричні моделі масштабування та часу виконання, які використовуються для початкової конфігурації розподілу ресурсів. Інтелектуальний рівень реалізовано у вигляді нейромережевого модуля прогнозування ефективності виконання задач залежно від параметрів розміщення та поточного стану обчислювального середовища. Для навчання та оцінювання моделі використано методи регресійного аналізу, багатошарову нейронну мережу типу MLP, а також статистичну валідацію прогнозів за експериментальними даними профілювання. Оптимізація розподілу задач здійснюється з використанням евристичних алгоритмів жадібного типу та генетичних алгоритмів, що дозволяє знаходити компромісні рішення за часовими й енергетичними критеріями в умовах обмежених ресурсів. Чисельні експерименти для задач множення блочних матриць і розв'язання рівняння теплопровідності підтвердили переваги гібридної методики. Результати дослідження показали, що поєднання офлайн-аналізу з онлайн-адаптацією дозволяє підвищити продуктивність паралельних обчислень і зменшити втрати, пов'язані з дисбалансом навантаження. Досягнуто прискорення до 18 разів, зниження часу очікування до 27% та точність прогнозування продуктивності на рівні 93–96%. Розроблена гібридна архітектура є ефективним інструментом оптимізації обчислювальних процесів у динамічних гетерогенних середовищах і може бути використана як основа для подальшого розвитку інтелектуальних систем керування паралельними обчисленнями.*

Ключові слова: паралельні обчислення; оптимізація; моделі продуктивності; розподілені системи; динамічне балансування; прогнозування

Link to publication

APA Bilak, Y., Saibert, F., & Reblian, A. (2026). Development of a hybrid optimization architecture combining static scheduling with dynamic load balancing. *Management of Development of Complex Systems*, 65, 70–81, dx.doi.org\10.32347/2412-9933.2026.65.70-81.

ДСТУ Білак Ю. Ю., Сайберт Ф. Ф., Реблян А. М. Розробка гібридної архітектури оптимізації, що поєднує статичне планування з динамічним балансуванням навантаження. *Управління розвитком складних систем.* Київ, 2026. № 65. С. 70 – 81, dx.doi.org\10.32347/2412-9933.2026.65.70-81.