Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.4'2 + 004.51
https://doi.org/10.32362/2500-316X-2022-10-6-7-19

RESEARCH ARTICLE

Computational complexity when constructing rational plans for program execution in a given field of parallel computers

Valery M. Bakanov @

MIREA – Russian Technological University, Moscow, 119454 Russia
@ *Corresponding author, e-mail: e881e@mail.ru*

Abstract

Objectives. The construction of rational plans (schedules) for parallel program execution (PPE) represents a challenging problem due to its ambiguity. The aim of this work is to create methods for developing such plans and specialized software for implementing these methods, which are based on the internal properties of algorithms, primarily on the property of internal (hidden) parallelism.

Methods. The main method for developing PPE plans was the construction, analysis, and purposeful transformation of the stacked-parallel form (SPF) of information graphs of algorithms (IGA). The SPF was transformed by transferring operators from tier to tier of the SPF (this event was taken as an elementary step in determining the computational complexity of scenario execution). As a transformation tool, a method for developing transformation scenarios in the scripting programming language Lua was used. Scenarios were created by a heuristic approach using a set of Application Programming Interface (API) functions of the developed software system. These functions formed the basis for a comprehensive study of the parameters of the IGA and its SPF representation for the subsequent construction of a PPE plan applying to a given field of parallel computers.

Results. Features of the internal properties of the algorithms that affect the efficiency of SPF transformations were identified during the course of computational experiments. Comparative indices of the computational complexity of obtaining PPE plans and other parameters (including code density, etc.) were obtained for various SPF transformation scenarios. An iterative approach to improving heuristic methods favors developing optimal schemes for solving the objective problem.

Conclusions. The developed software system confirmed its efficiency for studying the parameters of hidden parallelism in arbitrary algorithms and rational use in data processing. The approach of using a scripting language to develop heuristic methods (scenarios) for the purposeful transformation of IGA forms showed great flexibility and transparency for the researcher. The target consumers of the developed methods for generating schedules for parallel execution of programs are, first of all, developers of translators and virtual machines, and researchers of the properties of algorithms (for identifying and exploiting the potential of their hidden parallelism). The developed software and methods have been successfully used for a number of years for increasing student competence in data processing parallelization at Russian universities.

Keywords: algorithm graph, fine information structure of program, stacked-parallel form of graph, rational execution parameters of parallel program, execution plan of parallel program

• **Submitted:** 30.01.2022 • **Revised:** 29.03.2022 • **Accepted:** 05.09.2022

For citation: Bakanov V.M. Computational complexity when constructing rational plans for program execution in a given field of parallel computers. *Russ. Technol. J*. 2022;10(6):7–19. https://doi.org/10.32362/2500-316X-2022-10-6-7-19

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

The author declares no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Вычислительная сложность построения рациональных планов выполнения программ на заданном поле параллельных вычислителей

В.М. Баканов [@]

МИРЭА – Российский технологический университет, Москва, 119454 Россия
[@] *Автор для переписки, e-mail: e881e@mail.ru*

Резюме

Цели. Построение рациональных планов (расписаний) выполнения параллельных программ (ВПП), вследствие неоднозначности, является сложной задачей. Цель работы – создание методик разработки таких планов и специализированного программного обеспечения для реализации этих методик, полагающихся на внутренние свойства алгоритмов, в первую очередь на свойство внутреннего (скрытого) параллелизма.

Методы. Основными методами при разработке планов ВПП являются построение, анализ и целенаправленное преобразование ярусно-параллельной формы (ЯПФ) информационных графов алгоритмов (ИГА). Преобразование ЯПФ осуществляется путем переноса операторов с яруса на ярус ЯПФ (именно это событие и принято за элементарный шаг при определении вычислительной сложности выполнения сценария). В качестве инструмента преобразования применен метод разработки сценариев преобразования на скриптовом языке программирования Lua. Сценарии создаются на основе эвристического подхода и используют набор API-функций (API – Application Programming Interface) разработанной программной системы, позволяющих всесторонне изучить параметры ИГА и его ЯПФ-представления для последующего построения плана ВПП на заданном поле параллельных вычислителей.

Результаты. Результаты вычислительных экспериментов выявили особенности внутренних свойств алгоритмов, влияющих на эффективность преобразований ЯПФ. Получены сравнительные показатели вычислительной сложности получения планов ВПП и иных параметров (включая плотность кода и др.) при применении различных сценариев преобразования ЯПФ. Итерационный подход к улучшению эвристических методов позволит приблизиться к оптимальным схемам решения целевой задачи.

Выводы. В целом разработанный программный комплекс подтвердил эффективность в исследовании параметров скрытого параллелизма в произвольных алгоритмах и рационального его использования при обработке данных. Подход применения скриптового языка для разработки эвристических методов (сценариев) целенаправленного преобразования форм ИГА показал большую гибкость и прозрачность для исследователя. Целевыми потребителями разработанных методов генерации расписаний параллельного выполнения программ в первую очередь являются разработчики трансляторов и виртуальных машин, исследователи свойств алгоритмов (в направлении нахождения и использования потенциала скрытого их параллелизма). Разработанное программное обеспечение и методики несколько лет применяются при обучении студентов в университетах России, что позволило повысить компетенции учащихся в области параллелизации обработки данных.

Ключевые слова: граф алгоритма, тонкая информационная структура программы, ярусно-параллельная форма графа, рациональные параметры выполнения параллельной программы, план выполнения параллельной программы

• **Поступила:** 30.01.2022 • **Доработана:** 29.03.2022 • **Принята к опубликованию:** 05.09.2022

Для цитирования: Баканов В.М. Вычислительная сложность построения рациональных планов выполнения программ на заданном поле параллельных вычислителей. *Russ. Technol. J.* 2022;10(6):7–19. https://doi.org/10.32362/2500-316X-2022-10-6-7-19

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

Автор заявляет об отсутствии конфликта интересов.

INTRODUCTION

A currently popular method for reducing computation time is parallelization, involving the simultaneous processing of data in parts on a set of multiple computers with subsequent combination of the obtained results. While the possibility of using extra hardware to overcome existing fundamental and purely technological limitations of increasing performance is intuitively apparent, the possibility of identifying analogous parallelism potentials in algorithms is less so [1].

A separate and nontrivial problem when organizing parallel computing involves the construction of a plan (schedule) for the execution of parallel programs (PPE). Here, almost every sequential program (algorithms forming the basic components of any program conventionally represented in sequential form) can be represented in a parallel form using methods that preserve the set of operations and causal relationships between them to support different execution efficiencies on parallel computers of a given architecture. According to this formulation of the problem, it is the algorithms themselves, comprising the real building blocks of programs, whose internal properties become an increasingly important area of study.[1]

Each of the considered PPE plans is associated with certain program execution quality parameters (time, required computational resources, memory load, etc.). The formulation of the objective function and solution to the multi-parameter optimization problem is dependent on the stated problem.

Despite the differences in the architecture and systems of machine instructions of various parallel computing systems and parallel programming technologies, there are scientifically substantiated general approaches to the construction of enlarged plans (schedules) for program execution. Abstracting from the specifics of parallel programming technologies, it is logical to refer to such plans in terms of a *framework* for the execution of a parallel program.

[1] Voevodin V., Dongarra J. (Eds.). *AlgoWiki: Open Encyclopedia of Parallel Algorithmic Features.* http://algowiki-project.org. Accessed October 20, 2022 (in Russ.).

This paper proposed methods and their implementation (in the form of problem-solving program scenarios) for developing rational PPE plans on a given (possibly heterogeneous) field of parallel computers. The developed scenarios are intended for embedding as parallelizing blocks in newly developed systems for creating executable program code. In this case, the emphasis is on achieving maximum computational speed, since program debugging requires multiple translations with complex optimization while ensuring required quality.

METHODS

The problem of finding methods for constructing rational PPE plans was solved by creating a specialized software system of the instrumental level, whose generalized flow diagram and information flow diagram are shown in Fig. 1.

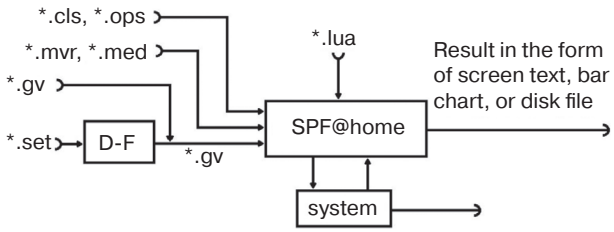

Fig. 1. Instrumental software complex for constructing PPE plans [2]

The input of the software system (Fig. 1) receives a description of an algorithm to be analyzed in the form of a conventional sequential program in an imperative style or its formal description in the form of a directed acyclic information graph of the algorithm (IGA), i.e., an ‘operators → operands’ dependence. In this case, the graph vertices are associated with operators (groups of operators) of the program; and the graph edges, with data transmission lines. In Fig. 1, the files of types *.set and *.gv are the program and information graph files of the analyzed program, respectively; *.mvr and *.med, the files of the metrics of vertices and edges of the graph of the algorithm, respectively; *.cls and *.ops, the files of parameters of computers and program operators, respectively; and *.lua, the Lua text file containing scripts

for PPE plans. When the results of the development are used as an algorithmic basis for the functioning of the components of parallelizing translators, the IGA is built on the basis of the results of parsing the source code by a translator (and the library functions are processed separately). When the application is implemented as a separate *Code Morphing Software* component, the IGA is built on the basis of the generated sequential executable code. A useful feature of using the representation of programs as IGA is that the graph can be processed in separate blocks (on the level of subroutines and files) with subsequent assembly of the graph of the complete application.

The internal logical parallelism in algorithms is identified and analyzed by simulating an actor model (D-F (Data Flow) module) and constructing special sections of the IGA in the stacked-parallel form (SPF) [3] (SPF@home module) [4, 5]. Both modules, which are developed in the C/C++ language in a GUI style for Win'32 model (a command line mode was additionally implemented for massive computations), are completely open-source and can be downloaded for free use (installation file format).[2] The D-F module builds an PPE plan for the operator execution asynchronism model; schedules in the SPF@home module style are designed for synchronous calling of groups of operators.

The D-F module is a universal computer of the *Symmetric MultiProcessing* (SMP) architecture (shared-memory systems) [6], whose input receives a sequential program in an imperative assembler-like language (3-character command mnemonics and a 3-address system with the AT & T operand order). The program is executed in a simulator of a computer of the Data-Flow static architecture; here, the order of execution of processor instructions is determined not by the order in which they appear in the machine code, but by the readiness of the operands [7, 8].

The inverse problem of optimizing the parameters of a computer from the characteristics of the computation process can be solved by varying the number of parallel computers and the rules for fetching instructions from the buffer [4]. Export of IGA to third-party programs in DOT format is provided along with a detailed simulation protocol.

Conditional execution is implemented by the predicate method [9, 10], with cycles being implemented using a system of macros, which unfold cyclic structures. The convenience of visualizing the solution is contributed by the output of the program execution data as a function of the computation intensity (the number of simultaneously executed operators as a function of time) and Gantt charts.

The SPF@home module [5] is designed to model and select the best (in a given sense) scenarios for transforming the SPF as a plan for the parallel execution of operators on a computational system of a given architecture. A significant advantage of using the SPF is the satisfactory temporal computational complexity of its derivation, which is quadratic polynomial with respect to the number of graph vertices. The SPF can initially be constructed in "upper" or "lower" form (where all the operators are located on the tiers of the SPF as close as possible to the beginning or end of the program execution, respectively). An important user advantage of the SPF is its improved visualization of the representation of the identified parallelism in the algorithm: even the initially obtained SPF of the algorithm is already a certain initial (usually far from optimal) PPE plan.

Groups (bundles) of operators located on the tiers of the SPF are executed in parallel within each tier, while bundles of operators on each tier are executed sequentially, beginning with the initial tier. The minimum possible height of the SPF (the number of tiers), which is determined by the length of the critical path in the IGA [11, 12], determines the shortest execution time of the algorithm. In the SPF@home module, the obtained SPF is visualized in text and graphical forms (the bar diagram of the SPF widths is an easy representation of the height distribution function of the SPF widths). The mechanism for setting operator execution metrics is used to set parameters for operators (for example, their execution time or required resources for a heterogeneous field of computers), computers (types of operators that can be executed on a given computer), as well as data transmission lines (transmission time, data sizes). The practice of using the metrics mechanism is described in detail in the subsection that presents PPE scheduling on a given heterogeneous field of computers.

Additionally, the SPF@home module provides the ability to obtain information on the data lifetime required to execute a given algorithm. These data, which exist as a consequence of the execution of individual operators, serve as input operands for other operators of the algorithm. This information (in fact, an estimate of the local *capacitive complexity* of the program execution) is important for determining the required parameters of the internal registers of the processor and/or solving questions about the optimal placement of data between the processor registers and RAM.

The main method of transforming the SPF is the purposeful movement of operators between the tiers of the SPF while maintaining information links in the IGA. In general, this is an *NP*-complete problem, which belongs to the class of constrained scheduling

[2] http://vbakanov.ru/dataflow/content/install_df.exe. Accessed October 24, 2022 (in Russ.); http://vbakanov.ru/spf@home/content/install_spf.exe. Accessed October 24, 2022 (in Russ.).

problems [13] and can be used in the construction of rational (iteratively tending to optimal) PPE plans. In the present work, a heuristic approach was used to obtain a solution whose scenarios (in this case, SPF transformations in the required direction) are implemented using the Lua scripting language [14]. Lua was chosen because it is completely open-source, close in syntax to common programming languages (C/C++ style), and compact when embedded in a parent application.

Each Lua function call is actually a wrapper over the corresponding Application Programming Interface (API) call of the parent program. The set of APIs of the SPF@home system, which covers almost all foreseeable actions on the IGA, can be used to analyze a graph of any complexity (limited by the resources of the computing device). In this sense, the use of the SPF graph is only one of the possible solution methods. A total of three types of calls can be distinguished:

- Information calls—serve to obtain information on the IGA and its SPF, on the basis of which data the specific IGA processing method is selected for solving the problem. Examples include determining the total number of tiers of the SPF, the number of operators in a given tier, the range of possible locations of a given operator in the tiers of the SPF, etc.
- Action calls—serve to implement specific methods for solving the problem of constructing an PPE schedule. Examples are to build the upper or lower form of the SPF, add an empty tier under the data, move an operator from tier to tier, etc.
- Auxiliary calls—serve to output the computed data in text and graphical forms for data exchange with other applications, work with the file system, etc.

The information graph of the algorithm in the SPF form can formally be represented by a 2D list of identifier elements (e.g., unique numbers) of operators $[a_{ij}]$, where $i = 1 \ldots W$ is the number of a row; the quantity W or SPF height is determined by the length of the critical path in the IGA; and $j = 1 \ldots j_i^{\max}$ is the number of operators in row i. The quantity $H = \max_i (1 \ldots j_i^{\max})$ is referred to as the SPF width. In a real IGA, the position of each operator on the tiers is limited by the presence of information links in the algorithm and by the range $i^{\min} \le i \le i^{\max}$, where $i^{\min}$ and $i^{\max}$ are the admissible numbers of tiers of placement of a given operator in the SPF; the range $i^{\min} \le i \le i^{\max}$ can be logically described in terms of the *variability* of positions on the tiers of the ith operator. The SPF is actually the (initial, naive) PPE plan (schedule). For a given description, the direction of the unit vector of time coincides with the increase in the number of a tier of the SPF.

In general, the proposed approach is fully consistent with the Explicitly Parallel Instruction Computing (EPIC) style [15] intended for software implementation of parallelizing translator blocks. At the same time, for the Very Long Instruction Word (VLIW) computing architecture [10], the term *operator* should be understood as a machine instruction in order to fully adhere to the Instruction-Level Parallelism (ILP) concept [4]. For multithreaded systems on multicore processors, it is logical to correlate the "operator" with a *parallelism granule* of a much larger size, e.g., on the level of an operator/operators or procedures of a high-level programming language [16]. The latter fits well with the concept of interpreters. In both cases, the presented general methodology for constructing a rational PPE plan remains unchanged.

The internal implementation of the data, which does not have to be provided for the explicit construction of the SPF in the form of a 2D array, can be in any format convenient for computer implementation. For example, in the naive case, it can be that establishing a one-to-one correspondence between the IGA in the form of a set of directed edges $\{k, l\}$ (adjacency matrix) identified by pairs of vertex numbers i_k, j_k and i_l, j_l, where i and j are the numbers of a row and a column in the SPF.

The examples for the study comprised popular data processing algorithms (linear algebra, statistics, array operations, etc.). Additional, artificial (not corresponding to any of the applied algorithms, but generated in accordance with the specified parameters) IGAs were prepared. A disadvantage of the experimental material was the relatively small dimensions of the processed data due to the considerable difficulty of manual programming. However, the performed experiments developed an increase in the identified trends including an increase in the dimension of the processed data throughout the studied dimension range.

The computational complexity of executing scenarios of SPF transformations was determined using an analog (applied in array sorting operations) of the classical method of estimating the target parameter, namely, determination of the number of *elementary steps* (permutations of two elements of an array being sorted) that is required to complete the operation. In our case, it is logical to define the elementary step as a permutation of an operator from tier to tier of the SPF. This approach has all the advantages and disadvantages of the classical method, including the failure to take into account the complexity of analyzing the situation and making decisions about taking a specific elementary step.

In this study, the estimated *code density* characterizes the resource use of a parallel computing system (number of computers) when executing this algorithm (formally,

the deviation of the widths of the tiers of the transformed JPF from a given value). When computing resources are not fully used, the translator has to insert NOP instructions into empty places in bundles of parallel-executed instructions, leading to decreased code efficiency.

The following series of experiments was carried out using the SPF@home module as offering the greatest flexibility in transforming the SPF of algorithm graphs; IGAs were generated by the D-F module based on the program code. During the computational experiments, the SPF@home module saves the most detailed simulation protocol for subsequent analysis. For this work, the following parameters obtained in the course of the target transformation of the SPF are of particular interest:

- The height H and width W of the obtained SPF (width constraints were set as a parameter that ensures the execution of the algorithm on a given number of parallel computers).
- The uniformity of the tier width distribution (*code density*) in this SPF was estimated by the coefficient of variation $CV = \frac{1}{\bar{W}}\sqrt{\frac{\sum(W-\bar{W})^2}{H-1}}$, where $\bar{W}$ is the arithmetic mean of the tier widths over the SPF.
- The computational complexity of the performed SPF transformation (in units of the number of permutations of operators from tier to tier of the SPF).

The efficiencies of SPF transformations were compared by using two heuristic methods (scenarios), which were represented in the form of Lua scripts and based on different general approaches to SPF transformations. The first (*Strategy_01*) used a dichotomy-based approach of massive transfer of operators from tier to tier of the SPF, whereas the second (*Strategy_02*) uses a gradual transfer of operators from "more loaded" tiers to "less loaded" ones. In both cases, if necessary, additional (initially empty) tiers are created in necessary places to ensure the execution of the algorithm on a given number of parallel computers.

RESULTS

Scheduling of program execution on a fixed number of parallel computers with the possibility of increasing the program execution time

This subsection considers the most general case corresponding to the condition $\bar{W} >> P$, where P is the number of parallel computers. It is in this case that the SPF height (program execution time) has to be increased.

The effectiveness of the above heuristic methods was tested by successively applying them to the SPF of the studied algorithms in the range of the numbers P of parallel computers from W_0 (the width of the initial SPF) to 1 (fully sequential execution of the algorithm).

Figures 2–5 show the change in the target quantities, specifically, (a) computational complexity, (b) SPF height, and (c) coefficient of variation of the widths of the SPF tiers (ordinate axis) as functions of a given number P of parallel computers (abscissa axis). The SPF transformations of the corresponding algorithms were carried out according to the scenarios *Strategy_01* and *Strategy_02* (curves 1 and 2, respectively).

As can be seen from the data in Figs. 2–5, the application of both methods to the studied algorithms gives similar results: the *Strategy_02* scenario is faster than its rival (panels (a)), whereas the algorithm execution times differ little (panels (b)). In code density (panels (c)), both scenarios show similar tendencies toward minimizing the objective function using a small (much smaller than the SPF width) number of parallel computers. The bizarre shapes of the curves are a consequence of the complexity of the scripts used and the processing of integer values, while the curves using the *Strategy_02* script are visually more monotonic.

Quantitatively, the *Strategy_02* scenario has a lower (by a factor of approximately 2–4 in the studied range of processed data sizes) computational complexity than *Strategy_01*, although the converse was expected at first glance. However, the *Strategy_02* script has more complex internal logic compared to *Strategy_01* (in the latter case, it is primitive), which cannot be taken into account by the accepted computational complexity estimation system. In view of the above, it may be more logical to use the *Strategy_01* script in the components of parallelizing systems for fast, but rather rough construction of PPE plans, while the *Strategy_02* method should be used for constructing these plans in the optimization mode.

Scheduling of program execution on the minimum number of parallel computers with the possibility of increasing the program execution time

In the case $\bar{W} \approx P$ (the arithmetic mean of the widths of the initial SPF is comparable to the number of parallel computers), the problem arises of PPE scheduling with the maximum code density without increasing the SPF height ("balancing" of operators over the tiers of the SPF). The developed empirical methods for balancing the SPF gave contradictory results: in some cases, it was possible to achieve almost 100% code density, whereas some algorithms were virtually unmodifiable (due to restrictions on moving operators between the tiers because of the need to preserve the information dependences in the algorithm).

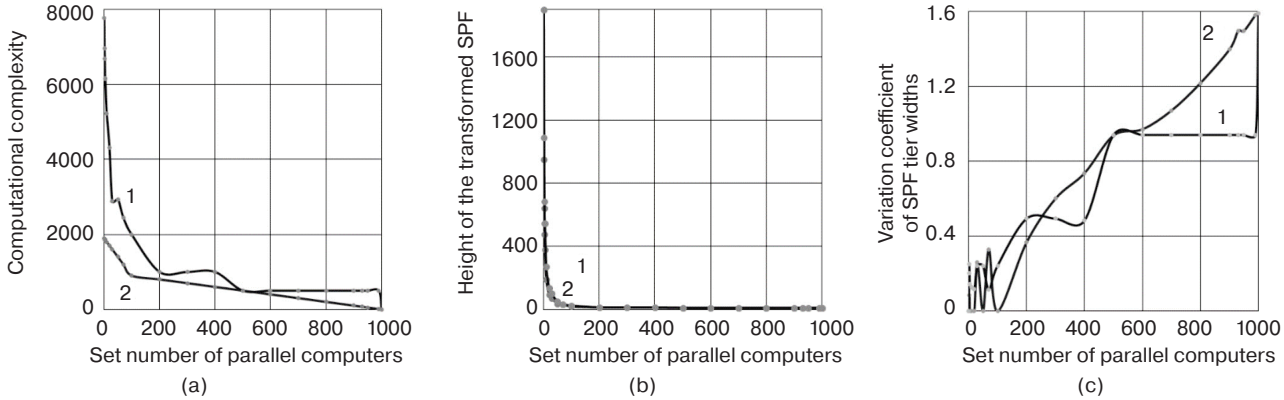

Fig. 2. Algorithm for multiplying 10th-order square matrices by the classical method

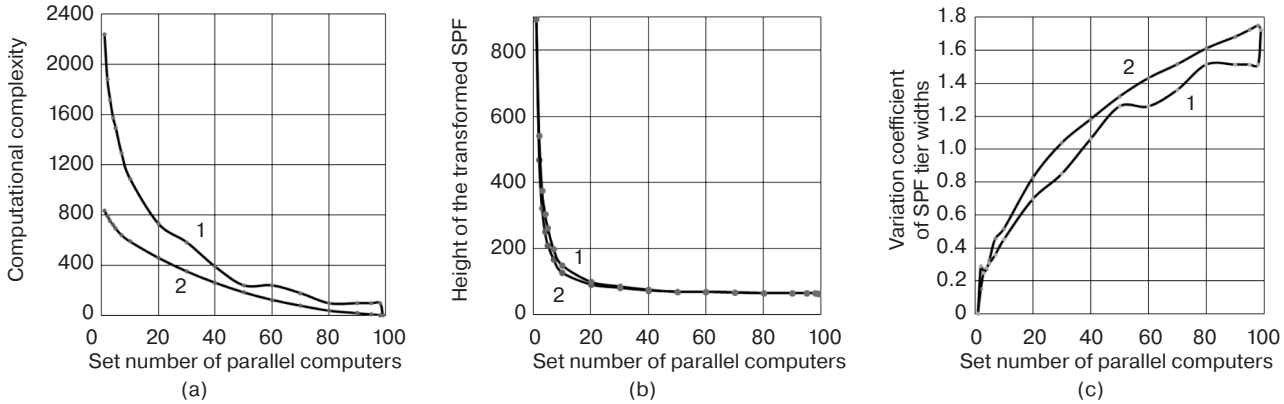

Fig. 3. Algorithm for solving systems of linear algebraic equations (SLAE)
of the 10th order by the direct (non-iterative) Gauss method

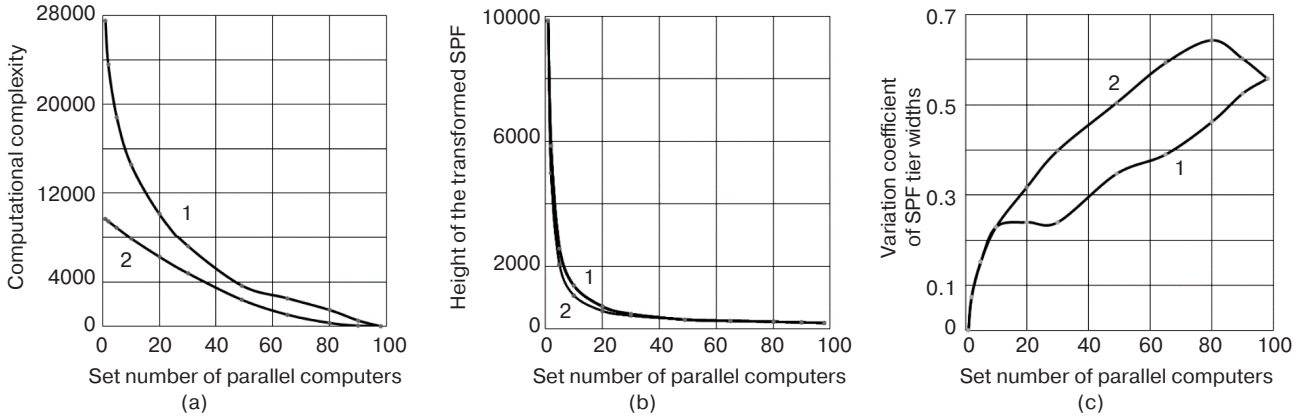

Fig. 4. Artificially generated algorithm e19039_o9853_t199

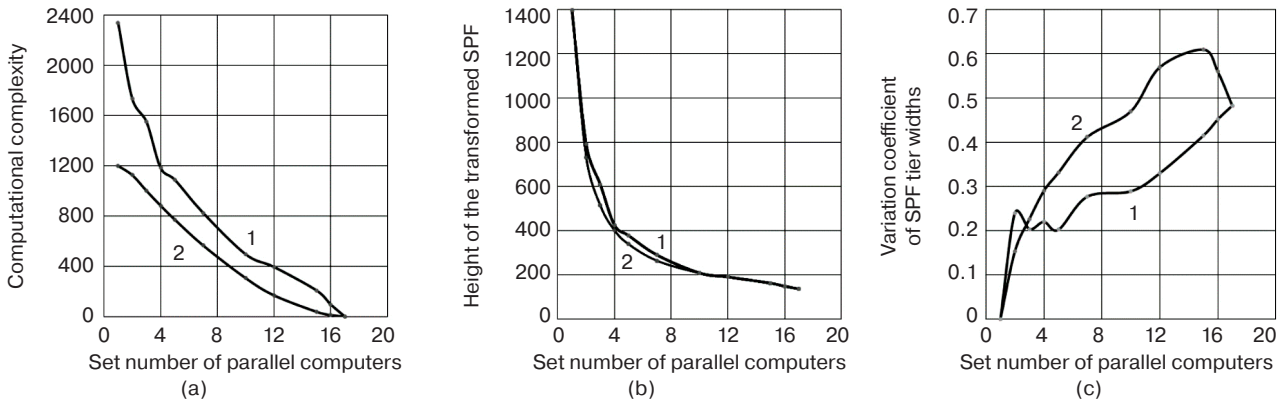

Fig. 5. Artificially generated algorithm e2367_o1397_t137

The SPF@home software module can also be used for solving an inverse problem, e.g., the determination of the parameters of the parallel computing system from the performance–cost balance of the system itself.

Promising PPE scheduling methods

In all the above experiments, the SPF was initially computed in the upper form (all the operators were located as high as possible in the tiers of the SPF). In this case, the operators are moved between the tiers mainly from top to bottom: from the initial tiers to the final ones (this does not, of course, prohibit multiple movements with repeated changes in the direction of movement of operators between the tiers).

The described sequence of actions is logically justified by the characteristic shape of the computation intensity curves under the conditions of unlimited parallelism, in which a sharp increase in the initial part is followed by a peak and a smooth decrease by the end of the program execution. Such a curve shape, which occurs at significant amounts of input data, once again confirms that the execution of operators on a field of parallel computers belongs to one of the varieties of multichannel queuing systems. In this case, since the variability of the operators located in the region of the peak of the computation intensity function is high, the efficiency of obtaining a satisfactory PPE plan by moving down the SPF operators is also significant. We will refer to the algorithms whose initial (unmodified) form of width distribution in the upper SPF corresponds to the one described above as belonging to the Π class.[3]

As a matter of discussion, it was of interest to consider the lower form of the initial SPF (in this case, all the operators are moved as far as possible toward the end of the program execution). Such an SPF can be obtained from the upper form by moving the operators through the tiers as low as possible or, more easily, by constructing the SPF in the direction from the end of the program to its beginning (in the latter case, the computational complexity of obtaining the SPF remains the same as when obtaining the upper SPF). The distributions of the widths of the SPF in the upper and lower forms are compared below.

Figure 6 shows the bar diagram of the distribution of the SPF widths at given sizes of processed data (indicated in the captions) that was obtained by the SPF@home module. Each of the four columns of a row presents two diagrams: the one on the left is for the upper form of the SPF of this algorithm and the one on the right is for its lower SPF. Here, H, W, and $\bar{W}$ are the height, width, and arithmetic mean width of the SPF (the last is shown in Fig. 6 by the dotted line; the forward slash symbol separates the parameters for the upper and lower SPF).

The data in Fig. 6 are interesting because of the possibility of significant balancing of the SPF without using complex heuristic algorithms for its reorganization. In fact, all the possible solutions for the reorganization of the SPF are within the range between the upper and lower forms; however, when choosing the lower form as the initial form, the priority movement of operators is upward.

The last statement raises the question of whether or not there are states that are balanced still better than the two boundary (upper and lower) forms. To answer this question, an experiment was carried out to study the stepwise transformation of the SPF from the upper to the lower form (in Fig. 7, the numbers on the abscissa axis are the numbers of movements of operators from tier to tier), in which along with the CV index (solid lines, left-hand ordinate axis), the irregularity of the widths of the SPF tiers was estimated by the ratio of the width of the widest tier to that of the narrowest one (dashed line, right-hand ordinate axis).

Despite the entire set of SPF states not being completely covered in these experiments (possible movements of operators along the SPF tiers were downward to the maximum variability), it can be argued with a high degree of probability that the target values do not increase in the range of existence and that it is in the region of the lower SPF that they are minimal.

When determining rational PPE plans in the case of obtaining easily determined conditions where the algorithm belongs to the Π class, significant savings in computations are possible: instead of implementing the above rather complex scenarios, it is enough to construct the lower SPF. The previously used quantitative characteristics of the irregularity of the tier widths do not provide information on the shape of the curve with this irregularity. As an additional estimate of the irregularity of the distribution of operators over the SPF tiers, the well-known graphic-analytical method can be used to determine the income inequality, which consists in calculating the numerical parameters of the stratification (Lorenz curve and Gini coefficient)[4], despite the mirror-opposite shape of the analyzed curves.

The example shows the importance of studying the properties (including classification) of algorithms from the side of their essential facet that consists in their internal parallelism in order to make the best practical use of these properties.

[3] We will refer to the Π class such algorithms that are characterized by the distribution of the tier widths in the SPF of the information graph in the upper form (when all operators are as close as possible to the beginning of execution) with the presence of a pronounced maximum at the beginning and of a gentle decrease at the end.

[4] Gini coefficient: Are everyone equal? *Open Journal*, an investment and finance medium. https://journal.open-broker.ru/economy/koefficient-dzhini/. Accessed March 31, 2022 (in Russ.).

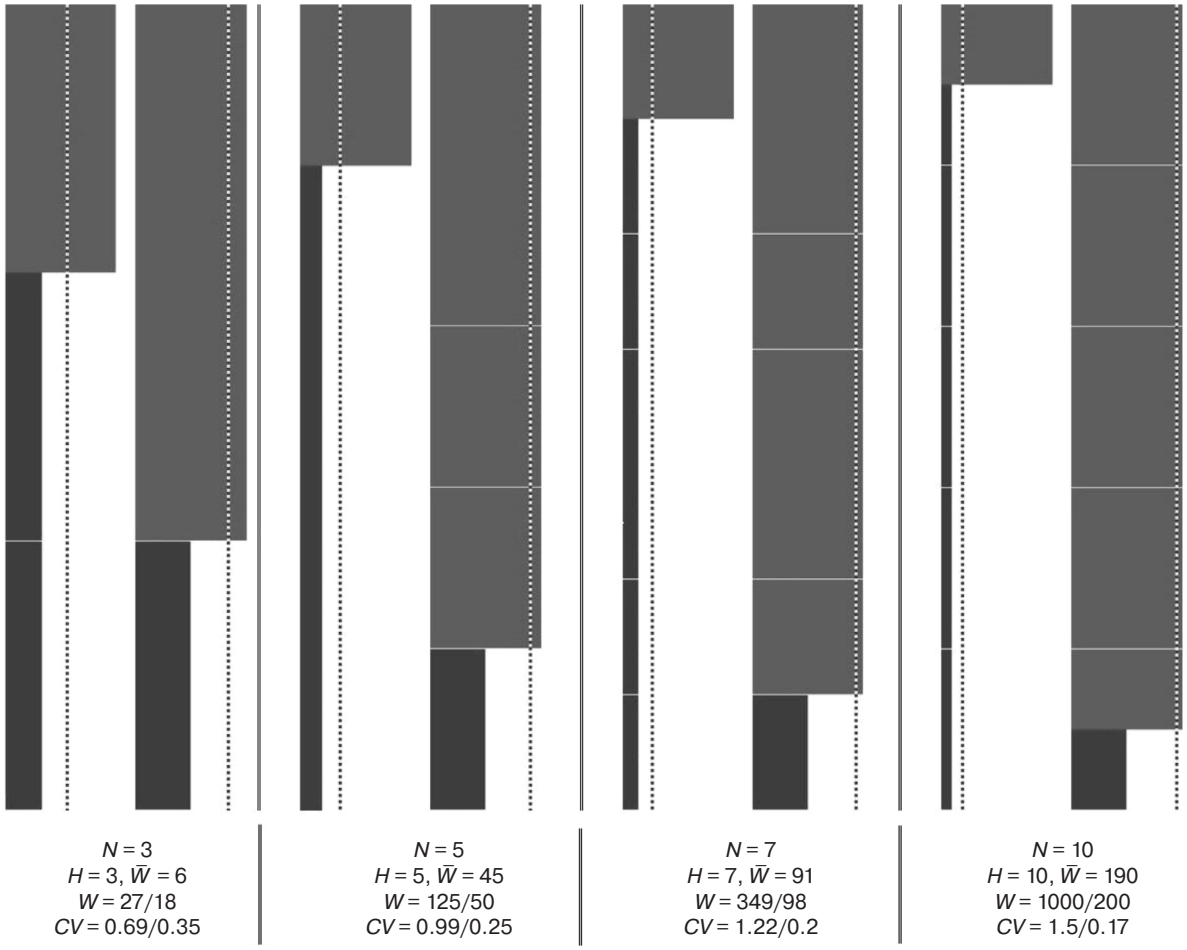

Fig. 6. Algorithm for multiplying square matrices by the classical method (Abscissa axis is the tier width for $N = 3, 5, 7, 10$ orders of the matrices). The light- and dark-gray areas contain tiers of maximum and minimum widths, respectively

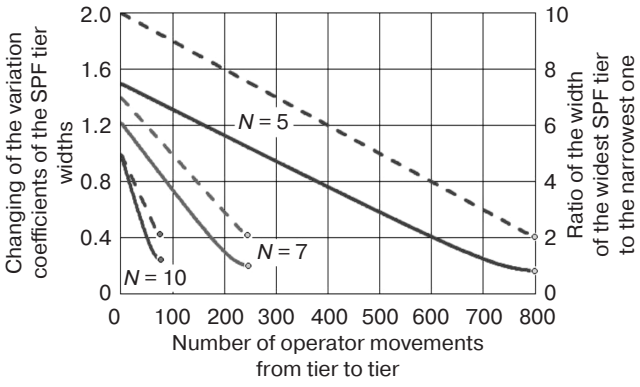

Fig. 7. Irregularity of the distribution of the widths of the SPF tiers for the algorithm of matrix multiplication by the classical method (*N* is the order of the multiplied matrices)

PPE scheduling on a given number of heterogeneous computers

Modern multicore processors are more and more often developed with computing cores of various capabilities. Therefore, it is practically useful to be able to schedule the PPE for such systems (with a heterogeneous field of parallel computers).

The SPF@home module supports this feature by comparing information from two metrics files for operators and computers (*.ops and *.cls, respectively, Fig. 1). It is possible to set a match on a set of freely assignable attributes for any range of operators/computers. The condition for the feasibility of a given operator on a given computer is the relation $minVal\_i \le Val\_i \le maxVal\_i$ at the same i, where Val_i, $minVal\_i$, and $maxVal\_i$ are the numerical values of this parameter for the operator and the computer, respectively.

Since PPE scheduling on a heterogeneous field of parallel computers is a more complex procedure than those described above, the emphasis here is on Lua programming. Since one SFP tier may contain operators the execution of which require different computers, it can be useful to apply a metaphor for splitting the SFP tiers into families of subtiers, each of which corresponds to a block of computers with certain capabilities. All the operators on a given tier have the same execution capabilities, the sequence of processing them within a tier/subtier is, to a first approximation, arbitrary. Figure 8a illustrates the splitting of operators on one of the tiers of the SPF in the case of 11 parallel computers of

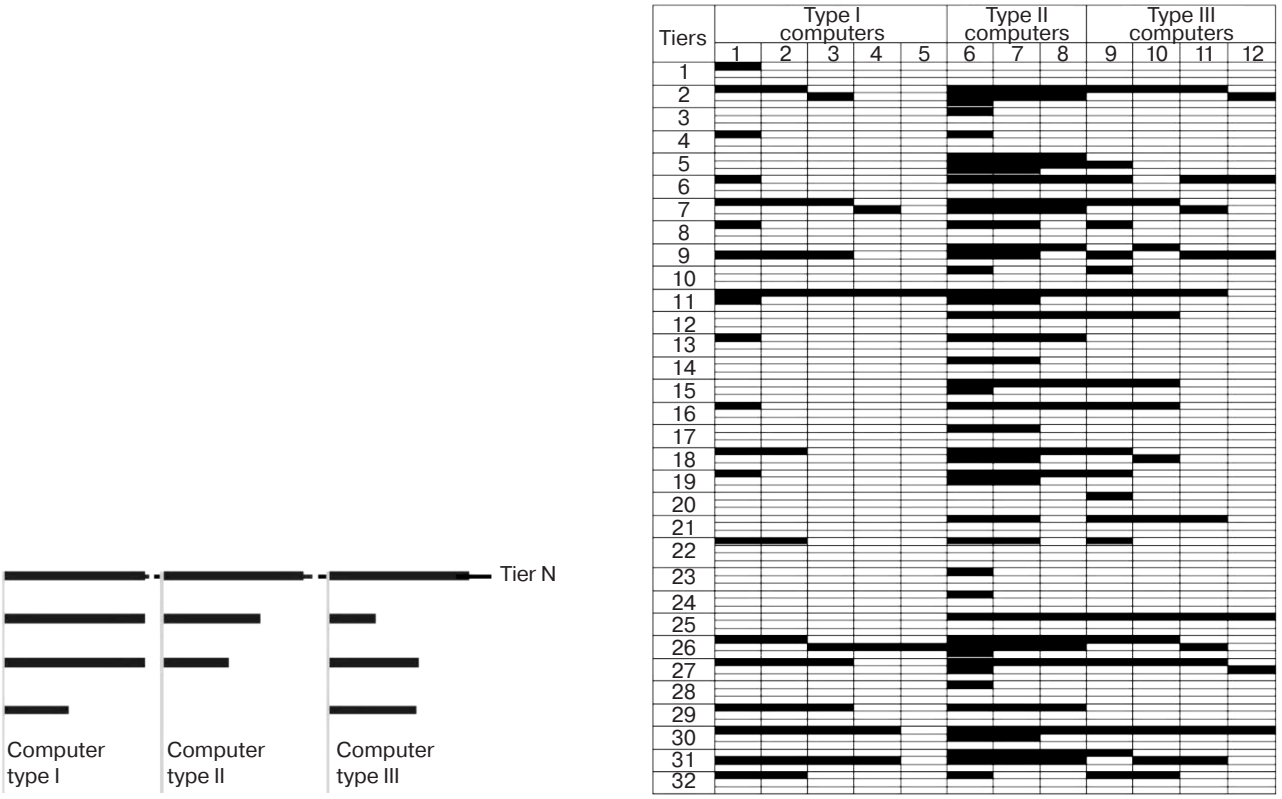

Fig. 8. (a) Splitting of the SPF tiers into families of subtiers in solving the problem of scheduling for a heterogeneous field of parallel computers and (b) the result of computing the schedule for the execution of a real parallel program

three types, and Fig. 8b presents the result of computing the real PPE plan on a heterogeneous field of parallel computers (three types of computers, 5, 3 and 4 units; the numbers of executable operators are hidden).

In this case, the total time T of solving the problem is determined by the sum over all the tiers of the maximum values of the execution times of operators on the subtiers of this tier:

$$T = \sum_j \left(\max_{k_j} \sum_i t_{ik} \right),$$

where j is the number of tiers, i is the number of subtiers on a given tier, k_j are the types of computers on the jth tier, and t_{ik} is the execution time of an operator of type i on a computer of type k.

If the goal is to achieve maximum performance, the optima number of computers of a particular type can be determined by minimizing the parameter T (e.g., by solving the inverse optimization problem to determine the ratio between the numbers of computing g devices of different types). The problem of minimizing the total solution time T becomes more complicated if each operator can be executed on several computers because of the ambiguity of the parameter t_{ik} in the above expression; here, additional balancing over substages is needed.

DISCUSSION

This study confirmed the possibility of gradual iterative improvement (in a given direction) of heuristic scenarios for transforming the initial SPF of various algorithms. By and large, it is possible to develop faster scenarios with a slightly inferior quality for an intended purpose and relatively slower scenarios with a higher quality (in fact, on the level of optimization).

Although this work is focused on the computational complexity of scenarios for obtaining PPE plans (schedules) for the PPE, the presented software system should also show its efficiency in solving multidimensional optimization problems using the known algorithms (this software system in this case is a *mathematical model of the subject of optimization*). To implement this case of the use, the discussed computer system permits working in command line mode.

Despite the low computational complexity of obtaining the SPF from the IGA, the method of using the SPF as the basis for constructing PPE plans has the disadvantage that it is impossible to easily take into account the execution time of operations, as a result of which the execution time of a bundle of operators on one tier of the SPF has to be considered equal to the execution time of the slowest of them. By taking an approach consisting in the purposeful movement of operators through the tiers of the SPF, operators can be

sorted on each tier according to the closest possible time for their execution.

The experiments detected a significant dispersion of the properties of the algorithms represented by information graphs in the possibility of forming PPE plans with the maximum code density. Different algorithms require different methods for their efficient transformation. It seems important to a priori (even before the reorganization of the SPF, at the time of its receipt) define scenarios for its effective purposeful modification with a given goal. The tool here should be the creation of a system for classifying algorithms according to some parameters that determine effective methods for their transformation. A promising approach to solving this problem involves formal methods of artificial intelligence.

All the performed experiments showed that the identified trends increase with increasing dimension of the processed data. This provides confidence that the trends determined by modeling are preserved when scaling by the volume of data being processed.

In accordance with the iterative principle (inherent in the heuristic approach) of gradually approaching the best solution of the problem under consideration, we may have confidence in the possibility of quantitative improvement (with respect to the above parameters) in methods for scheduling of program execution on a field of parallel computers, which is given or determined by the solution of the optimization problem.

An important productive property of the developed software system is the possibility of solving inverse problems of determining the parameters of a computing system in accordance with the specified requirements for the very process of program execution, e.g., for execution time.

CONCLUSIONS

In general, the developed software system confirmed its efficiency in studying the parameters of hidden parallelism in arbitrary algorithms and its rational use in data processing. The approach using a scripting language for the development of heuristic methods (scenarios) for purposeful transformation of the forms of the information graph of algorithms showed greater flexibility and transparency for the researcher. The necessary flexibility is achieved by using an interpreted scripting language, while the processing speed is achieved according to the capabilities of the executable code of the compiled language of the parent application.

The target consumers of the developed methods for generating PPE schedules are, in the first place, developers of translators and virtual machines, as well as researchers into the properties of algorithms in order to identify and exploit the potential of their hidden parallelism. A practical application of the proposed methodology has to take into account a number of applied implementation issues, including known problems of using pointers, conflicts of memory operations in Load/Store instruction bundles, etc., which do not fundamentally change the proposed methodology.

The developed software, comprising methods for detecting hidden parallelism and its parameters in arbitrary algorithms, as well as constructing rational plans (schedules) for the PPE on a given field of computers, has been used for several years teaching students at Russian universities in order to improve their competences in data processing parallelization processes.

REFERENCES

1. Voevodin V.V., Voevodin Vl.V. *Parallel'nye vychisleniya* (*Parallel computing*). St. Petersburg: BHV-Petersburg; 2004. 608 p. (in Russ.).
2. Bakanov V. Research and selection of rational methods for obtaining framework of schedules for the parallel programs execution. In: Silhavy R., Silhavy P., Prokopova Z. (Eds.). *Data Science and Intelligent Systems. CoMeSySo 2021. Lecture Notes in Networks and Systems*. V. 231. Springer, Cham. https://doi.org/10.1007/978-3-030-90321-3_22
3. Fedotov I.E. *Parallel'noe programmirovanie. Modeli i priemy* (*Parallel programming. Models and techniques*). Moscow: SOLON-Press; 2018. 390 p. (in Russ.). ISBN 978-5-91359-222-4
4. Bakanov V.M. Dynamics control computing in the processors data flow architecture for different types of algorithms. *Programmnaya inzheneriya = Software Engineering*. 2015;9:20–24 (in Russ.).
5. Bakanov V.M. Software complex for modeling and optimization of program implementation on parallel calculation systems. *Open Comput. Sci*. 2018;8(1): 228–234. https://doi.org/10.1515/comp-2018-0019
6. Padua D. (Ed.). *Encyclopedia of parallel computing*. NY: Springer; 2012. 2195 p. https://doi.org/10.1007/978-0-387-09766-4
7. Dennis J.B., Misunas D.P. A preliminary architecture for a basic data-flow processor. In: *Proc. Second Annual Symp. Computer Architecture* (*ISCA 75*). 1975. P. 126–132. https://doi.org/10.1145/642089.642111
8. Kukunas J. *Power and performance: Software analysis and optimization*. Morgan Kaufman, Elsevier Inc.; 2015. 300 p.
9. McNairy C., Soltis D. Itanium 2 processor microarchitecture. *IEEE Micro*. 2003;23(2):44–55. https://doi.org/10.1109/MM.2003.1196114
10. Tanenbaum E., Ostin T. *Arkhitektura komp'yutera* (*Computer architecture*). Transl. from Eng. St. Petersburg: Piter; 2019. 816 p. (in Russ.). ISBN 978-5-4461-1103-9
11. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. *Introduction to algorithms*. 3rd ed. London: MIT Press; 2009. 1292 p.
12. McConnell J.J. *Analysis of algorithms: An active learning approach*. Jones & Bartlett Publishers; 2008. 451 p.
13. Garey M., Johnson D. *Vychislitel'nye mashiny i trudnoreshaemye zadachi* (*Computing machines and difficult tasks*). Transl. from Eng. Moscow: Kniga po trebovaniyu; 2012. 420 p. (in Russ.). ISBN 978-5-458-26100-5
    [Garey M., Johnson D. *Computers and intractability*. San Francisco; 1979. 338 p.]
14. Ierusalimschy R. *Programming in Lua*. 3rd ed. PUC-Rio, Brasil, Rio de Janeiro; 2013. 348 p.
15. Fisher J.A. Very long instruction word architectures and the ELI-512. In: *Proceedings of the 10th Annual International Symposium on Computer Architecture* (*ISCA '83*). 1983. P. 140–150. https://doi.org/10.1145/800046.801649
16. Babb R.I. Parallel processing with large-grain data flow techniques. *Computer*. 1984;17(7):55–61. https://doi.org/10.1109/MC.1984.1659186

СПИСОК ЛИТЕРАТУРЫ

1. Воеводин В.В., Воеводин Вл.В. *Параллельные вычисления*. СПб.: БХВ-Петербург; 2004. 608 с.
2. Bakanov V. Research and selection of rational methods for obtaining framework of schedules for the parallel programs execution. In: Silhavy R., Silhavy P., Prokopova Z. (Eds.). *Data Science and Intelligent Systems. CoMeSySo 2021. Lecture Notes in Networks and Systems*. V. 231. Springer, Cham. https://doi.org/10.1007/978-3-030-90321-3_22
3. Федотов И.Е. *Параллельное программирование. Модели и приемы*. М.: СОЛОН-Пресс; 2018. 390 с. ISBN 978-5-91359-222-4
4. Баканов В.М. Управление динамикой вычислений в процессорах потоковой архитектуры для различных типов алгоритмов. *Программная инженерия*. 2015;9:20–24.
5. Bakanov V.M. Software complex for modeling and optimization of program implementation on parallel calculation systems. *Open Comput. Sci*. 2018;8(1): 228–234. https://doi.org/10.1515/comp-2018-0019
6. Padua D. (Ed.). *Encyclopedia of parallel computing*. NY: Springer; 2012. 2195 p. https://doi.org/10.1007/978-0-387-09766-4
7. Dennis J.B., Misunas D.P. A preliminary architecture for a basic data-flow processor. In: *Proc. Second Annual Symp. Computer Architecture* (*ISCA 75*). 1975. P. 126–132. https://doi.org/10.1145/642089.642111
8. Kukunas J. *Power and performance: Software analysis and optimization*. Morgan Kaufman, Elsevier Inc.; 2015. 300 p.
9. McNairy C., Soltis D. Itanium 2 processor microarchitecture. *IEEE Micro*. 2003;23(2):44–55. https://doi.org/10.1109/MM.2003.1196114
10. Таненбаум Э., Остин Т. *Архитектура компьютера*: пер с англ. СПб.: Питер; 2019. 816 с. ISBN 978-5-4461-1103-9
11. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. *Introduction to algorithms*. 3rd ed. London: MIT Press; 2009. 1292 p.
12. McConnell J.J. *Analysis of algorithms: An active learning approach*. Jones & Bartlett Publishers; 2008. 451 p.
13. Гэри М., Джонсон Д. *Вычислительные машины и труднорешаемые задачи*: пер. с англ. М.: Книга по требованию; 2012. 420 с. ISBN 978-5-458-26100-5
14. Ierusalimschy R. *Programming in Lua*. 3rd ed. PUC-Rio, Brasil, Rio de Janeiro; 2013. 348 p.
15. Fisher J.A. Very long instruction word architectures and the ELI-512. In: *Proceedings of the 10th Annual International Symposium on Computer Architecture* (*ISCA '83*). 1983. P. 140–150. https://doi.org/10.1145/800046.801649
16. Babb R.I. Parallel processing with large-grain data flow techniques. *Computer*. 1984;17(7):55–61. https://doi.org/10.1109/MC.1984.1659186

About the author

Valery M. Bakanov, Dr. Sci. (Eng.), Professor, Department of Hardware, Software and Mathematical Support of Computing Systems, Institute of Cybersecurity and Digital Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: e881e@mail.ru. Scopus Author ID 6505511355, ResearcherID L-1314-2015, RSCI SPIN-code 8868-4594, https://orcid.org/0000-0002-1650-038X

Об авторе

Баканов Валерий Михайлович, д.т.н., профессор, профессор кафедры КБ-5 «Аппаратное, программное и математическое обеспечение вычислительных систем» Института кибербезопасности и цифровых технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: e881e@mail.ru. Scopus Author ID 6505511355, ResearcherID L-1314-2015, SPIN-код РИНЦ 8868-4594, https://orcid.org/0000-0002-1650-038X

Translated from Russian into English by Vladislav V. Glyanchenko
Edited for English language and spelling by Thomas A. Beavitt