Research Article **Open Access**

Valery Bakanov*

Software complex for modeling and optimization of program implementation on parallel calculation systems

https://doi.org/10.1515/comp-2018-0019
Received June 29, 2018; accepted December 5, 2018

Abstract: The paper considers the problem of developing rational methods for the creation of a framework (a plan, execution timetable) of parallel programs for real parallel computing systems. To solve this problem, a software environment (software stand) has been developed that allows implementing different strategies for building a framework for parallel programs and assessing the quality of these strategies. The built-in script Lua programming language is used to increase the flexibility of modeling and optimization capabilities. Results of applying some of the proposed strategies for constructing rational plans for parallel programming are outlined.

Keywords: graph algorithm presentations, analysis of information program structure, line-parallel form (LPF) of information graph, rational parameters for execution of parallel programs, strategy for constructing a rational plan for the parallel program execution

1 Introduction

The task of developing effective parallel programs (i.e. the fastest ones and those which make the most use of the given computing system)remains to be solved completely solved yet [1]. The problems are associated with a huge variety of both computational algorithms and architecture of parallel computing systems. In this regard, research in the field of modeling and optimizing the execution of programs on parallel computing systems are relevant today.

Almost every application program contains several standard algorithms (linear algebra, statistics, signal analysis, etc) united by "glue", which defines the overall structure of the program in question. In this case, the "glue" described often represents a sequential algorithm which cannot be parallelized efficiently. The main execution time of the program falls on separate blocks, representing (in case of well-designed application architecture) well-parallelized algorithms. It is these parts of the overall program that must be subjected to the efficient parallelization.

In programming the parallel parts of a common program, one of the known parallel programming technologies is usually applied. However before this it is necessary to define the rational plan ("framework") for performing the parallel parts of the program. To determine such plans, it is advisable to apply the computer simulation method for executing the given algorithms on a certain field of parallel calculators. The research requires flexibility in the description and purposeful modification of plans for parallel programs. One of the ways to achieve this is to use the script programming language to implement the tasks of modeling and optimization.

2 Methods of research

To solve these problems, the specialized software toolkit (actually a software platform) was created using the C++ programming language; the client part represents a 32-bit GUI-application for Windows and includes a text editor and an interpreter of Lua script language as components [2] (Figure 1). The developed program system in the form of portable executable files is available for free use (from the article author's website at: http://vbakanov.ru/spf@home/content/spf@home.rar).

The personal contribution of the author consists in setting the task, designing, algorithmizing, developing the software interface and the user interface of the system, programming, debugging and supporting the specified software system as well as, developing strategies for building rational (aspiring to optimal) execution schedules.

***Corresponding Author: Valery Bakanov:** National Research University "Higher School of Economics" (HSE), 101000, Myasnitskaya Str. 20, Moscow, Russia; E-mail: vbakanov@hse.ru

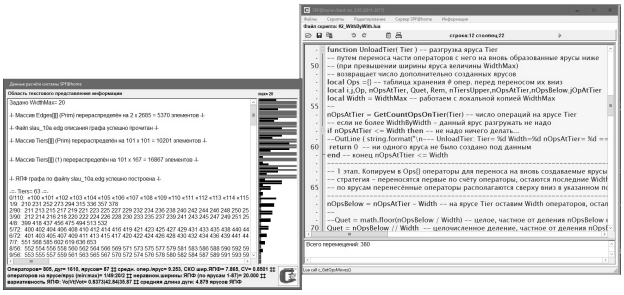

Figure 1: User interface of the developed software system.

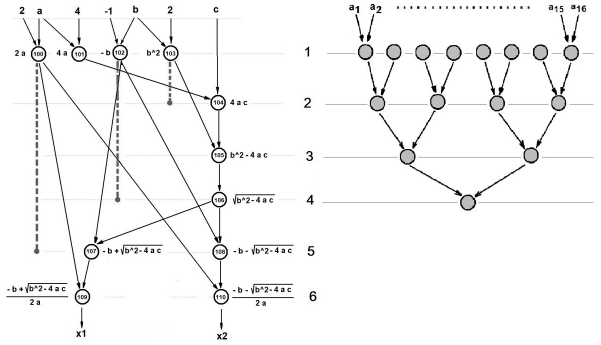

Figure 2: Examples of a line-parallel form of algorithms: on the left – with an opportunity of moving operators between tiers, on the right – in the absence of such an opportunity (doubling procedure for 16 numbers), whereas the figures in the center represent LPF tier numbers.

The initial data for the system under discussion include description of the program in the form of an Information Graph Algorithm (IGA); meanwhile the decomposition stage of the given algorithm is deemed completed. Hereby the graph nodes (operators) can have any size (depending upon the chosen level of parallelization). Output information is given in text and (in part) graphical form and the data of each experiment is carefully logged in a file with a unique name.

The first step of analysis is to reveal the hidden parallelization of the given algorithm and as a result of its execution we obtain the so-called Line-Parallel Form (LPF) of the information graph of the algorithm [3]. Actually, the LPF is obtained by a fast algorithm of computing complexity $O(N^2)$, where N is the number of IGA vertices. This method is included in a set of functional capabilities of this software system. In fact, LPF already determines a certain (but in most cases far from the optimal ones albeit in most cases a suboptimal) plan for parallel execution of the given algorithm. The operators on the same LPF tier are information-independent and can be executed simultaneously – that is, in parallel.

Let's consider a simple example of processing one of the simplest algorithms – solving the complete quadratic equation in real numbers, where the level of parallelization is the level of machine instructions and for this task, the solution coincides with the order of arithmetic operations.

Figure 2 (on the left) shows the canonical LPF form of this algorithm, which is characterized by 11 arithmetic operations (operators), 6 data inputs (coefficients of equations a, b, c and constants 2.0, 4.0 and −1.0) and 2 data outputs (two real roots). The height of this LPF is equal to 6 (the minimum path value in the IGA), whereas the width is 4.

Analysis of the LPF algorithm for solving the complete quadratic equation shows significant unevenness of the distribution of operators over the tiers (which is indicative of very inefficient use of resources of the parallel computing system). Nevertheless, it is possible to move operators between tiers while maintaining the LPF height (which determines the total parallel algorithm execution time), without violating algorithm information dependencies to move operators from tier to tier while maintaining the LPF height (namely, it determines the total time of parallel algorithm execution). The potential displacements range is shown in Figure 2 (on the left by a dotted line). Additionally, Figure 2 (on the right) shows the LPF algorithm, which basically does not allow the redistribution of operators over LPF tiers – in case of maintaining the height thereof.

By moving two of the operators on tier 1 to the lower tiers, the LPF width is reduced to 2. At the same time, for execution of the given algorithm (during the same time period) only 2 parallel calculators are sufficient instead of 4. These transformations illustrate the first variant of optimization, where the target function is to balance the LPF width. Balancing entails minimizing the LPF width while maintaining its height, which is equivalent to minimizing the computational resources (provided the execution time for single-tasking the parallel calculator does not increase), this action can be called "balancing" the LPF widths. The second optimization option involves transforming the LPF form to execute the algorithm on the given field (number, first of all) of parallel calculators (which are computationally heterogeneous) while allowing an increase in execution time, i.e. increasing the LPF height.

This problem is closely related to the problem of partitioning the graphs and according to the work outlined in [4], in the general case belongs to the class of NP-complete (for a large-sized IGA, the computational complexity of obtaining an exact solution by way of full exhaustive search is prohibitively large). Approximate solutions are achieved by the well-known methods (genetic

algorithms, branch and border method, etc.) that can be developed, debugged and tested by means of the built-in script language in the developed software system.

However, this approach is suitable mainly at the stage of researching execution of algorithms and is unlikely to be applied to Run-Time compilers, where compilation speed is especially important. This is true for the conceptual architecture of processors with a very long machine word VLIW – *Very Large Instruction Word* in the EPIC – *Explicitly Parallel Instruction Computing* model, known since 1983 [5] and still active to this day [6]. Proceeding from this, one of the applications of this system is development of fast heuristic methods (scenarios) for determining the rational transformations of the LPF form for the given formulation of the optimization problem. By consistent improvement (based on certain quality indicators) of the developed scenarios, their aspiration to optimal scenarios is realized.

To implement the above transformations of IGA forms, the program interface of the system under consideration includes three types of API calls (total up to 70, each of them is the Lua "wrapper" of the corresponding C function), as follows:

- Information calls (serve to obtain information about IGA and its LPF; based on these data, one of the IGA processing strategies is subsequently selected to solve the set task, implemented subsequently in Lua). Examples: get the total number of LPF tiers, the number of operators on the given tier, the range of possible location of this operator by LPF tiers.
- Promotional calls (serve to implement the specific strategies for solving the problem of constructing a parallel schedule of algorithm execution). Examples: to build the canonical form of LPF, add an empty tier under the data, and transfer the operator from tier to tier.
- Auxiliary calls (output of calculated data in text and graphical format for data exchange with the other applications, operations with file system, etc.).

The degree of uneven distribution of operators over the LPF tiers (one of the optimization parameters) will be determined as:

- k is the coefficient of uneven distribution of operators over the tiers of the LPF graph $k = \frac{max(W_i)}{min(W_i)}$, should be W_i is width of the i-th LPF tier;
- σ is the root-mean square deviation of the widths of tiers, $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{i=N}\left(W_i - \bar{W}\right)^2}$, where $\bar{W}$ is arithmetic mean width of LPF tiers, N is the number of tiers;
- CV is the coefficient of variation of the LPF tier widths, $CV = \frac{\sigma}{\bar{W}}$.

The complexity of performing each scenario of transforming the LPF form is proposed to be evaluated in the number of transpositions of operators from LPF tier to tier (similarly to determining the complexity of array sorting operations).

Parallel computation fields can be heterogeneous allowing computations between fields with different memory volume parameters, data types, or different data structures, e.g. vector computations. A requirement to achieve this heterogeneity along with the choice of selecting the execution operator (from the set of appropriate parameter calculating machines) would necessitate the use of tags within the model (extra parameters) for operators and calculators. The principle of determining the opportunity to execute the particular operator on the given calculator is based on attributing the numeric parameter of the operator tag to the range of the same type of parameters of the corresponding calculator tag (unlimited number of tag names is allowed, all comparisons are related by conjunction). The opportunity is presumed for the group (in "from" and "up to" range) setting the tags for numbers of operators and calculators and setting the relevant default values.

The tagging system is also implemented to describe the metrics of IGA vertices and arcs (i.e. the operators and their message exchange lines) and allowing their values to be set. Operator metrics characterize the algorithm's runtime computational complexity, message exchange line metrics (message length or transmission time) or, for multi-core processors, the parameters for temporary data storage (e.g. in-process registers). The option to group (in "from" and "up to" range) assignment metrics for numbers of operators and data exchange lines and to set the default values is also supported.

Accounting for these characteristics makes it possible to obtain maximum adequacy of the software model, i.e. successfully executing the data processing on parallel computer systems of various types.

Similar solutions to the system under discussion can be METIS and ParMETIS (University of Minnesota, 1998-2003), but these packages are mainly aimed at operations with graphs and do not "and do not enable the features of the optimization problem of implementing specific algorithms on preconfigured parallel calculators. V-Ray and PARUS (both developed by Moscow State University named after M. V. Lomonosov, Russia), the purpose of which, however, differs significantly from those presented in the work and can be attributed to related topics with this project.

For the case of heterogeneity of the field of parallel calculating machines, the strategy is proposed (and imple-

mented on Lua) that allows the logical division of (split up) LPF tiers into the individual sub-tiers. If there are no more operators within the given tier than the number of calculators capable of executing operators of this exact type, then they are executed in parallel. Otherwise these operators are forced to be executed sequentially on sub-tiers within the given tier (Figure 3).

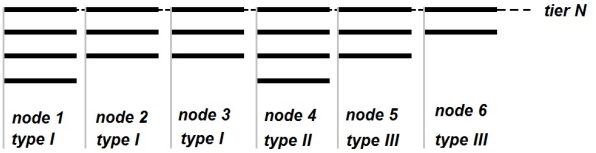

Figure 3: Execution of parallel program on a heterogeneous field of calculators according to splitting scheme applicable to tiers for six parallel computers (nodes) of type I, type II and type III, by 3, 1 and 2 pieces respectively for the *N*-th LPF tier.

In this case, the total time T for solving problems is determined by the sum over all tiers, i.e. by the maximum time values related to execution of operators on sub-tiers of the given tier, as follows:

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

where j-th is the number of tiers, i is the number of sub-tiers within the given tier, k_j are the types of calculators in j-th tier, whereas t_{ik} is the execution time of type i operator installed on type k calculator.

The task of minimizing a total solution time becomes more complicated in case of possibility when it is possible to perform each operator on several calculators due to the ambiguity of t_{ik} in the above expression; the additional balancing by sub-tiers is required here. This allows the inverse problem to be solved as well, i.e. optimizing the parameters of heterogeneous computational field for specific algorithm.

Therefore, the overall processing diagram for processing representations of information graphs of algorithms (IGA) in the program system under consideration can be presented in Figure 4.

When applying this system as a component of a parallelizing compiler, the algorithm graph already exists, as long as an operating compiler performs the program analysis for information dependencies and actually builds the graph. To obtain the correct IGAs, the operator has been using the DATA-FLOW calculator [7] program simulator, which allows the program to be debugged and is exported to the file format of the list of adjacent vertices (i.e. input to

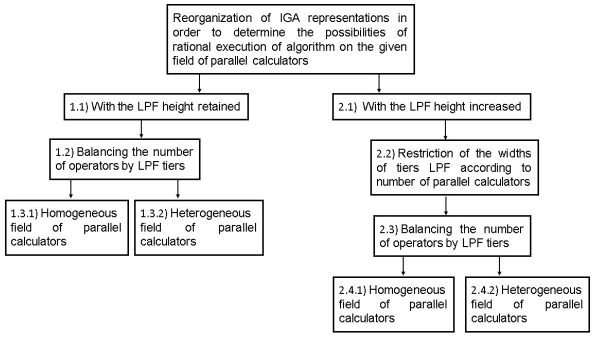

Figure 4: Enlarged diagram of IGA processing stages aimed at definition of parameters of their rational performance on the given field of parallel calculators.

the software system). A specific feature of this simulator is the ability to model the asynchronous mode of operators execution. For each of them the time of execution is set (in conventional units).

3 Results

The efficiency of data processing is illustrated by data in Table 1: strategy with the metaphorical name "Bulldozer" by analogy with the function of a bulldozer, i.e. moving soil from elevations to fill surface depressions.

Considered information graphs (homogeneous parallel calculators field):

- slau_5a and slau_10 – represent the procedures for solving SLAE of orders 5,10 respectively by the direct Gauss method,
- doubling_32 – the method of doubling for 32 numbers,
- mnk_10 and mnk_20 – the procedures for linear least-squares fit for 10 and 20 points,
- korr_10 and korr_20 – the procedures for determining coefficient of pair correlation for 10 and 20 points,
- m_matr_5 and m_matr_10 – the procedures for multiplying square matrices of orders 5 and 10 by the classical method.

The uneven distribution of the number of operators over LPF tiers was estimated by unevenness of distribution of operators over the tiers and the variation coefficient of widths of *CV* tiers .

It can be seen from the data in Table 1 that the LPF's "balancing" potential at a constant height is limited, so strategies with increasing LPF height are of more practical interest (see Table 2 which compares the two strategies with metaphorical names "Dichotomy" and "Width-

Table 1: Effectiveness of applying the "Bulldozer" strategy of balancing the LPF width under a constant height thereof (numerator – for the original LPF, denominator – for the LPF converted).

IGA name (arcs/vertices)	*Number of LPF tiers*	*Average LPF width*	*Unevenness k (of LPF width)*	*CV*	*Number of vertices (operators) displacements*
slau_5a (230/115)	25	4.60	20/20	1.2/1.1	15
slau_10a (1610/805)	63	12.8	90/90	1.72/1.62	86
doubling_32 (62/31)	5	6.2	16/16	0.984/0.984	0
mnk_10 (132/66)	16	4.13	22/14	1.2/0.766	9
mnk_20 (252/126)	26	4.85	42/24	1.58/0.867	19
korr_10 (152/88)	15	5.87	32/24	1.26/0.915	12
korr_20 (292/168)	25	6.72	62/35	1.73/0.921	44
m_matr_5 (450/225)	5	45.0	5/3,40	0.994/0.514	42
m_matr_10 (3800/1300)	10	190	10/6	1.5/0.763	407

By-Width "). Besides the algorithms described above, the algorithms generated by the IGA graph generating program e313_o206_t32, e451_o271_t30, e2367_o1397_t137) have also been analyzed.

To "unload" the excessively wide tiers, the "Dichotomy" strategy assumes the transfer of half of operators to the newly created tier under the current one; in the meantime, the "Width-By-Width" strategy realizes the gradual transfer of operators to the newly created LPF tiers. It can be seen from Table 2 that in most cases "Width-By-Width" leads to better results than "Dichotomy" (for example, the height of converted LPF is significantly smaller thus corresponding to a reduction in the execution time of parallel program – in addition, for fewer operator displacements).

4 Discussion of research results

As can be seen from the data in the tables above, the previously described strategies (by the way, not excessively sophisticated) reduce the unevenness of LPF widths under the given constraints, but with different efficiency for various IGA's. The data show the polynomial growth of labor intensity of these algorithms (in terms of moving operators from tier to tier) in comparison with the degree of "jamming" the LPF width.

When applying the "Bulldozer" strategy, k and CV have been reduced by 1.5 to 2 times (larger values corresponding to increased data processing), whereas applying the "Dichotomy" and "Width-By-Width" strategies has reduced the same indicators' values by up to 10 times (even further in some cases, depending on the preset condition and specific algorithms). These parameters should be considered the starting points for further improving these strategies. Generally, increased complexity (and accordingly, variability) results in an increased effectiveness of the IGA strategy."

The outlined experimental data allows us to draw conclusions about the quality of functioning of the developed LPF conversion strategies – for example, when applying the "Dichotomy" strategy to the doubling algorithm (doubling_32 in Table 2). Accordingly, this strategy operates more "roughly", where the target parameter experiences notably higher "jumps" at considerably faster speeds. This results in the development of a more effective "Width-By-Width" strategy.

Further improvements may be achieved through the improvement of individual strategies and their reasonable joint application. The task of the priority effectiveness prediction level for these strategies (prior to applying the proper LPF transformations) is particularly interesting and practically important (application of artificial intelligence methods to recognize the situation and choose the most efficient LPF conversion method according to the optimization task).

In general we have succeeded in achieving the preset goal – development of "rapid" rational strategies for drawing up a plan for implementation individual fragments of parallel programs where the use of the described software system proved effective. Quantitative parameters of current approximations are obtained in the course of fine-tuning strategies for optimizing the execution of parallel programs and ways to further develop such strategies have also been revealed.

Analysis of the parallel program execution in multi-task mode requires additional research. Meanwhile in the zero approximation, the optimization (in the aforementioned sense) of each separate program is the basis for op-

Table 2: Comparison of effectiveness of two LPF conversion strategies in compliance with width full stop W_0: (numerator – "Dichotomy" strategy, denominator – "Width-By-Width" strategy).

IGA name (arcs/vertices/tiers)	*Initial LPF*	*LPF converted*					
	DisplacementLPF height CV	W_0=10	W_0=8	W_0=6	W_0=4	W_0=2	W_0=1
slau_10a (1610/805/63)	-	$\frac{926}{504}$	$\frac{1038}{534}$	$\frac{1135}{575}$	$\frac{1340}{632}$	$\frac{1601}{701}$	$\frac{1920}{742}$
	63	$\frac{141}{119}$	$\frac{167}{137}$	$\frac{197}{167}$	$\frac{283}{231}$	$\frac{486}{419}$	$\frac{805}{805}$
	1.72	$\frac{0.472}{0.565}$	$\frac{0.399}{0.49}$	$\frac{0.361}{0.41}$	$\frac{0.294}{0.3}$	$\frac{0.287}{0.14}$	$\frac{0}{0}$
m_matr_10 (3800/1300/10)	-	$\frac{5232}{1800}$	$\frac{5232}{1820}$	$\frac{5828}{1840}$	$\frac{6152}{1869}$	$\frac{6964}{1880}$	$\frac{7776}{1890}$
	10	$\frac{272}{190}$	$\frac{272}{242}$	$\frac{436}{320}$	$\frac{544}{475}$	$\frac{1088}{950}$	$\frac{1900}{1900}$
	1.50	$\frac{0.127}{0}$	$\frac{0.127}{0.0967}$	$\frac{0.229}{0.0587}$	$\frac{0.143}{0}$	$\frac{0.249}{0}$	$\frac{0}{0}$
korr_20 (292/168/25)	-	$\frac{91}{52}$	$\frac{91}{54}$	$\frac{121}{56}$	$\frac{160}{78}$	$\frac{212}{120}$	$\frac{283}{143}$
	25	$\frac{32}{31}$	$\frac{32}{32}$	$\frac{40}{35}$	$\frac{59}{59}$	$\frac{97}{95}$	$\frac{168}{168}$
	1.73	$\frac{0.359}{0.481}$	$\frac{0.359}{0.362}$	$\frac{0.259}{0.294}$	$\frac{0.29}{0.495}$	$\frac{0.257}{0.24}$	$\frac{0}{0}$
mnk_20 (252/126/26)	-	$\frac{51}{32}$	$\frac{61}{34}$	$\frac{61}{36}$	$\frac{79}{38}$	$\frac{127}{78}$	$\frac{183}{100}$
	26	$\frac{31}{30}$	$\frac{33}{31}$	$\frac{33}{32}$	$\frac{44}{36}$	$\frac{70}{65}$	$\frac{126}{126}$
	1.58	$\frac{0.504}{0.611}$	$\frac{0.349}{0.508}$	$\frac{0.349}{0.387}$	$\frac{0.336}{0.302}$	$\frac{0.224}{0.125}$	$\frac{0}{0}$
doubling_32 (62/31/5)	-	$\frac{8}{6}$	$\frac{8}{8}$	$\frac{20}{12}$	$\frac{20}{16}$	$\frac{34}{22}$	$\frac{49}{26}$
	5	$\frac{6}{6}$	$\frac{6}{6}$	$\frac{9}{8}$	$\frac{9}{9}$	$\frac{16}{16}$	$\frac{31}{31}$
	0.984	$\frac{0.629}{0.675}$	$\frac{0.629}{0.629}$	$\frac{0.328}{0.524}$	$\frac{0.328}{0.328}$	$\frac{0.129}{0.129}$	$\frac{0}{0}$
e313_o206_t32 (313/206/32	-	$\frac{52}{28}$	$\frac{60}{46}$	$\frac{103}{68}$	$\frac{138}{104}$	$\frac{210}{148}$	$\frac{293}{174}$
	32	$\frac{40}{40}$	$\frac{42}{42}$	$\frac{56}{51}$	$\frac{70}{68}$	$\frac{123}{112}$	$\frac{206}{206}$
	0.738	$\frac{0.475}{0.626}$	$\frac{0.454}{0.532}$	$\frac{0.388}{0.522}$	$\frac{0.3}{0.42}$	$\frac{0.281}{0.2}$	$\frac{0}{0}$
e451_o271_t30 (451/271/30)	-	$\frac{85}{47}$	$\frac{98}{77}$	$\frac{177}{177}$	$\frac{209}{162}$	$\frac{315}{212}$	$\frac{430}{241}$
	30	$\frac{43}{43}$	$\frac{46}{46}$	$\frac{70}{61}$	$\frac{83}{79}$	$\frac{156}{144}$	$\frac{271}{271}$
	0.522	$\frac{0.32}{0.514}$	$\frac{0.312}{0.412}$	$\frac{0.256}{0.421}$	$\frac{0.214}{0.271}$	$\frac{0.254}{0.172}$	$\frac{0}{0}$
e2367_o1397_t137 (2367/1397/137)	-	$\frac{496}{306}$	$\frac{613}{471}$	$\frac{969}{668}$	$\frac{1183}{883}$	$\frac{1733}{1126}$	$\frac{2339}{1260}$
	137	$\frac{209}{209}$	$\frac{237}{237}$	$\frac{338}{293}$	$\frac{425}{402}$	$\frac{791}{733}$	$\frac{1397}{1397}$
	0.483	$\frac{0.288}{0.474}$	$\frac{0.294}{0.39}$	$\frac{0.235}{0.356}$	$\frac{0.218}{0.291}$	$\frac{0.24}{0.153}$	$\frac{0}{0}$
e17039_o9858_t199 (17039/9858/199)	-	$\frac{14579}{7926}$	$\frac{16129}{8297}$	$\frac{17418}{8677}$	$\frac{20252}{9064}$	$\frac{23524}{9456}$	$\frac{27512}{9654}$
	199	$\frac{1387}{1074}$	$\frac{1730}{1323}$	$\frac{2108}{1724}$	$\frac{3291}{2541}$	$\frac{5865}{4981}$	$\frac{9853}{9853}$
	0.558	$\frac{0.231}{0.23}$	$\frac{0.194}{0.214}$	$\frac{0.201}{0.173}$	$\frac{0.219}{0.135}$	$\frac{0.278}{0.074}$	$\frac{0}{0}$

timal functioning of the entire parallel computing system in multi-tasking mode.

The results of this research are applicable to the analysis of algorithms (with regard to demonstrating the efficiency of parallel program execution) in development of the parallelizing compilers and in educating specialists in the field of parallel programming.

References

[1] Voevodin V. V., Voevodin Vl. V., Parallel'nye vychislenija (Parallel computing), St. Petersburg, BHV-Petersburg, 2002 (in Russian)
[2] Ierusalimschy R., Programming in Lua, 3rd Edition, PUC-Rio, Brasil, Rio de Janeiro, 2013
[3] AlgoWiki, Open Encyclopedia of Parallel Algorithmic Features, Available at: 18.10.2016, http://algowiki-project.org/en/Open_Encyclopedia_of_Parallel_Algorithmic_Features (reference date: 01.06.2018)
[4] Gary M., Johnson D., Vychislitel'nye mashiny i trudnoreshaemye zadachi (Computers and intractable problems), Moscow, Mir, 1982 (in Russian)
[5] Fisher J. A., Very long instruction word architectures and the ELI-512, In: Proceedings of the 10th annual International Symposium on Computer Architecture, New York, NY, USA: Association for Computing Machinery (ACM), 1983, 140–150
[6] McNairy C., Soltis D., Itanium 2 processor microarchitecture, IEEE Micro Magazine, 2003, 23(2), 44–55
[7] Bakanov V. M., Dynamics control computing in the processor data flow architecture for different types of algorithms, Programmnaya Ingeneria (Software Engineering), 2015, 9, 20–24 (in Russian)