Research and Selection of Rational Methods for Obtaining Framework of Schedules for the Parallel Programs Execution

Valery Bakanov

MIREA (MIREA Russian Technological University), HSE University (National Research University Higher School of Economics), Doctor of Engineering, Professor. Moscow.
http://vbakanov.ru/left_1.htm, 499-122-1328, 915-053-5469
e881e@mail.ru

Abstract. TheThis paper deals with the creation of rational methods for developing a schedule (execution plan) of programs for parallel computing systems of a given (homo- and heterogeneous) architecture for arbitrary algorithms. To solve this task, a software system is proposed that allows the implementation of various (heuristic) methods for building up an execution schedule for parallel programs and to quantitatively evaluate and compare the computational complexity of these methods. The quantitative results gained from the application of some methods for constructing rational parallel execution plans for classical data processing algorithms are presented. #COMESYSO1120

Keywords: Analysis of the program information structure, Graph representations of the algorithm, Purposeful reorganization of the stacked-parallel form of the data dependence graph, Rational parameters of parallel programs execution, Parallel program execution framework, Procedure for constructing a rational parallel program execution plan.

Introduction

With Currently, the data processing parallelization is used mainly to reduce computational time. Data are simultaneously processed in parts on many different computing devices with the subsequent combination of results. Parallel execution allows evading the fundamental law formulated by Lord Rayleigh in 1871. According to this law (as applied to the CPU heat dissipation), their heat dissipation power is proportional to the fourth power of CPU clock speed (speed doubling increases the heat dissipation by 16 times) and, in fact, to replace it with a linear one in the number of parallel computers (while maintaining the CPU clock speed). Nothing is given for free. The task of identifying (usually hidden from the uninitiated observer) the parallelism potential in algorithms is plain to see, and even more so the efficiency of its (parallelism) use.

To obtain a plan (schedule) for the parallel program execution (hereinafter PPE) becomes a problem when organizing parallel computations. The fact is that this sequential program can be represented in parallel form in many ways (while maintain-

ing the invariant in the form of a set of operations), and each way will have a different execution efficiency on a computer of a given architecture.

The starting point for work is algorithms. The interest in their intrinsic properties has recently been growing again [1].

1 Methods

A specialized software system of the instrumental level was created to solve the issue of determining the methods for calculating the rational PPE plans. The generalized chart and data flow diagram of this system are shown in the following figure.

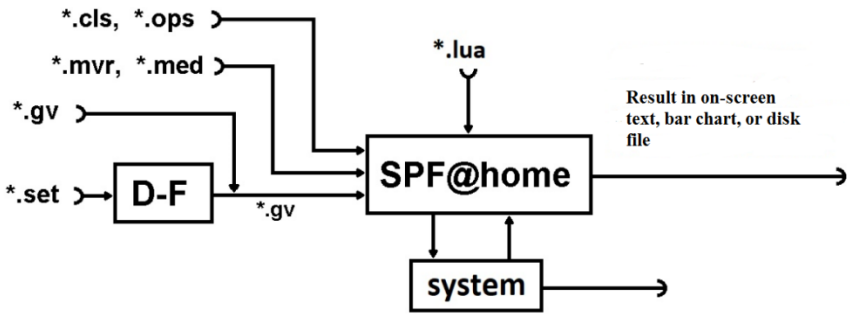

Fig. 1. Instrument complex diagram (*.set and *.gv are the program file and the analyzed program information graph file respectively, *.mvr, *.med are the files of metrics for the algorithm graph vertices and arcs respectively, *.cls, *.ops are the files of computer and program operator parameters respectively,*.lua is a text file in the Lua language that contains the methods of SPF reorganization).

The complex input (diagram in Fig. 1) receives a description of the analyzed algorithm in the form of a traditional program or a formal description in the form of an oriented acyclic information graph of the algorithm (hereinafter IGA) – an "operators - operands" dependence. The graph vertices are associated with program operators (groups of operators), and arcs are associated with data transmission lines.

The actor model imitation (Data-Flow module) and construction of IGA special sections in the form of its stacked parallel form (hereinafter SPF, SPF@home module) are implemented to reveal and analyse the internal logical parallelism in algorithms, [2]. Both of the mentioned modules are developed using the C/C ++ in the GUI style for the Win'32 model (the command line mode is additionally implemented for mass calculations). Both modules are completely OpenSource and can be downloaded for free use from the following addresses: http://vbakanov.ru/dataflow/content//install_df.exe and http://vbakanov.ru/spf@home/content/install_spf.exe (installation file format). The D-F module builds up the schedules according to the asynchronism model both at the beginning and at the end of the operator execution. SPF@home style schedules correspond more to synchronism at the beginning of execution.

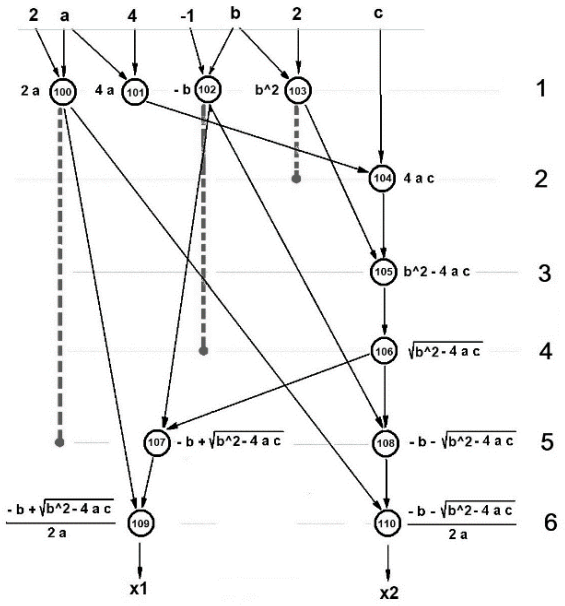

Fig. 2. A stacked-parallel form of the algorithm for detecting the real roots of a complete quadratic equation in the upper version (the tier numbers are shown on the right, the dotted line shows the valid positions of operators along the SPF tiers).

The SPF@home background is the IGA file itself (always present, because it is built by the translator when analyzing the program source code) and additional files that determine the measures of graph arcs and vertices (for taking into account, for example, the temporal characteristics of transfers and operator execution) and parameters of individual computers of a parallel (including heterogeneous) computational field. The SPF@home module also allows to determine the local data lifetime between SPF tiers, which is necessary for determining / optimizing the parameters of temporary data storage devices (high-speed shared memory, for example, general-purpose registers).

Due to *NP*-completeness of the scheduling problem [7], it can be said that the only rational methods for its solution, which involves the use of heuristic methods for SPF transformation is developed. The actual transform methods are described using the Lua scripting language [8]. Lua calls serve as a wrapper for API functions of SPF@home; calculation of various graph parameters and careful modeling process registration are provided. Herewith an iterative approach to gradually improve the quality of heuristic methods while moving towards the stated goal is assumed.

The system is mainly aimed at analyzing programs developed using the high-level programming languages without explicitly specifying parallelism and in systems with the ILP (*Instruction-Level Parallelism*) concept [5]. Although the SPF@home module capabilities make it possible to use indivisible blocks of a commands sequence of any size. The concerned system builds up exactly the *framework* of programs for parallel execution. A specific parallel programming technology, methods to call for subrou-

tines, interrupts processing, cache usage, etc. are solved for each specific case separately.

Taking into consideration the long-known fact about the movement of an active (where at a given period of time there is a vigorous machine instructions execution) operators domain along the front of the program being executed, it is advisable to process in blocks (within individual files, subroutines) sequentially from the beginning of the program to its end. In this case, the output vertices of the previous block are input for the next one.

The internal data implementation, of course, is not obliged at all to provide for the explicit building of the SPF in a 2D array. It can be any convenient for computer implementation one. For example, in the naive case, it establishes a one-to-one correspondence between the IGA in the form of a set of directed arcs {*k, l*} (adjacency matrix) and twos of numbers of the vertices i_k,j_k and i_l,j_l, where *i, j* are the numbers of rows and columns in the SPF.

2 Results

The results of research carried out using the SPF@home module as one characterized by more flexibility to achieve the set goals are given below. Experiments on the computation intensity management using the D-F module were published earlier (see, for example, paper [4]).

Three typical subtasks were defined reasoning from the fact that the SPF is already in fact (possibly requiring improvement) a plan for parallel program execution (as blocks of statements are executed sequentially in tiers from top to bottom along the SPF). In real cases, it is often required to execute several of these subtasks simultaneously. These three subtasks are:

- Balancing the number of operators over all the tiers of a given SPF without increasing its height. Balancing means obtaining the maximum evenness of operator distribution over the tiers while maximizing the use of hardware capabilities and minimizing the number of parallel computers required to solve the problem.
- Obtaining a program execution schedule on a given number of parallel computers with a possible increase in the SPF height (program execution time).
- Obtaining a program execution schedule on a heterogeneous field of parallel computers.

For the SPF@home module, the actual sequence of obtaining a rational schedule for executing a parallel task will be as follows (each and many other actions are performed by corresponding Lua calls):

- Obtaining the original SPF (which have no restrictions on the width of tiers).
- This SPF modification in the needed direction by purposeful rearrangement of operators from tier to tier while preserving the original connections in the algorithm information graph.

1. Programs execution schedule on the minimum number of parallel computers while maintaining the SPF height

Such an SPF organization allows obtaining a schedule with the maximum code density. Computers of the VLIW architecture (*Very Long Instruction Word*, machine instruction with a very long word) often lack such density, [9]. It should be under-

stood that this task has more of a methodological value, because the number of physical parallel computers is usually much less than the width of SPF for real tasks.

The 1-01_bulldozer empirical method is aimed at the evenest operators distribution over the SPF tiers without increasing in its height (saving the program execution time). The operators are moved down only (initially, the SPF is built in the upper version). To do this, the method tries to move the operators from the tiers of higher than average width to the tiers with the smallest width.

The 1-02_bulldozer method is a modification of the previous one with adaptation. We will call the range of possible operators' placement over the SPF tiers without changing the information connections in the algorithm graph as the variability maximum. For an operator with a variability maximum, the upper and lower limits of its possible placement along the tiers are calculated within the tier.

In many cases, it is possible to significantly (up to 1.5-2 times) reduce the width of the SPF but almost never to the minimum value (the arithmetic mean of the widths of the tiers). On the whole, the 1-02_bulldozer method is somewhat more efficient but loses in terms of computational complexity. In most cases, an increase in the processed data size increases the efficiency of balancing. Obviously, this is connected to an increase in the number of SPF degrees of freedom.

2. Programs execution schedules on a fixed number of parallel computers with the possibility of increasing the SPF height

Methods with an increase in the SPF height are of practical interest. Two methods were used with the metaphorical names - Dichotomy and WidthByWidth. At the same time, we have to accept the increase in program execution time (previously described features typical to the VLIW architecture are also present here). The finite width of the transformed SPF was set during the computational experiments. The coefficient of variation characterizing the non-evenness of the distribution of tier widths (the value that inversely characterizes the code density) was calculated as $CV = \sigma/\bar{W}$, where σ is the mean-square deviation of the number of operators for all SPF tiers, $\bar{W}$ is the arithmetic mean number of operators in the tiers.

Dichotomy method assumes the transfer of half of the operators to a newly developed tier below the current one to unload the unnecessarily wide tiers. WidthByWidth method implements a gradual transfer of operators to the newly developed SPF tiers. In most cases, WidthByWidth method leads to better results than Dichotomy. For example, the transformed SPF height is significantly smaller and this corresponds to a reduction in the parallel program execution time with a smaller number of operator transfers).

3. Program execution schedule on a fixed number of heterogeneous parallel computers

Modern multi-core processors are increasingly being developed with computing cores of various capabilities. Therefore, it is practically useful to be able to build a parallel program execution schedule for such systems (with a heterogeneous field of parallel computers).

SPF@home module supports this feature by comparing the information from two files - for operators and computers (*.ops and *.cls, respectively). It is possible to set a match for a variety of freely assignable attributes in relation to any range of operators/computers. The condition for this operator to be satisfied on a given calculator is

minVal_1≤Val_1≤maxVal_1 for the same parameter. *Val_1, minVal_1, maxVal_1* are the numerical values of this parameter for the operator and the computer, respectively.

Development of a schedule for a program executing on a heterogeneous field of parallel computers is a more complicated procedure than the ones described above, and the emphasis here is on programming in Lua (API functions of the SPF@home system provide the minimally necessary support). As the operators that require different computers can be on the same SPF tier, the concept of splitting the SPF tiers into families of sub-tiers can be useful. Each family corresponds to a block of calculators with certain capabilities. Since all given operators of the tier have the GKV-property, the sequence of their execution within the tier/sub-tier in the first approximation is arbitrary. The diagram below shows the splitting of operators on one of the SPF tiers in the case of six parallel computers of three types.

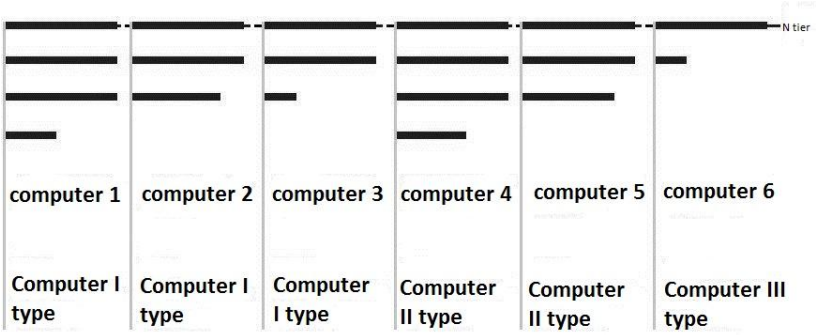

Fig. 3. The scheme for SPF tiers splitting into families of sub-tiers when solving the problem of determining the schedule for heterogeneous field of parallel computers.

In this case, the total problem solving time *T* is determined by the sum over all the tiers of maximum operator execution time values on the sub-tiers of the given 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 sub-tiers on a given tier, k_j are computer types on *j*-th tier, t_{ik} is the execution time of *i* type operator on a *k* type computer.

If the maximum performance is a task, it is quite possible to determine the number of computers of a particular type that minimizes *T*. The task to minimize the total *T* solution time becomes more complicated if it is possible to execute each operator on several computers due to the t_{ik} ambiguity in the expression given above. The additional balancing by sub-tiers is required here.

As the developed techniques are supposed to be used as part of parallelizing translators (possibly virtual machines) generating programs for execution in the EPIC style (*Explicitly Parallel Instruction Computing*, a set of instructions with explicit parallelism) [9] so the issue of performance is important. More precisely, the balance be-

tween the performance of the resulting program and the complexity of work on developing a plan for its implementation. As it was said, the computational complexity of SPF transformation will be estimated in units of operator transferring from level to level, and the quality – in the statistical parameters of width evenness for the SPF tiers. When obtaining the results presented below, the ability to build a schedule with a minimum amount of local data was not used to minimize the amount of temporary data storage during computations.

Maintaining the original SFP height is equal to executing the algorithm (program) in the shortest possible time. The evenest load on all the parallel system computers will be with the same width of all SPF tiers (arithmetic mean). In the SPF of real algorithms and the span of processed data , the arithmetic mean is quite large (almost always this number is much larger than the number of individual computers in a parallel system). So the concerned case is rare in practice but still needs to be analyzed.

We selected algorithms that are commonly used in data processing (mainly, linear algebra) and two heuristic methods for purposeful SPF transformation, 1-01_bulldozer methods and 1-02_bulldozer, for the comparison. The span of processed data (put along the abscissa) was a parameter in the calculations.

The results of applying these methods are shown in Fig. 4-6. Key for these figures:

- numbering along the ordinate axes on the graphs a), b) and c) is the SPF width, coefficient of variation (CV) of the SPF tier widths, the number of operator transfers from tier to tier (characteristic of the computational complexity), respectively;
- solid, dotted, and dash-dotted lines are the initial data, the result of applying the heuristic methods of 1-01_bulldozer and 1-02_bulldozer, respectively (they coincide in Fig. 6a).

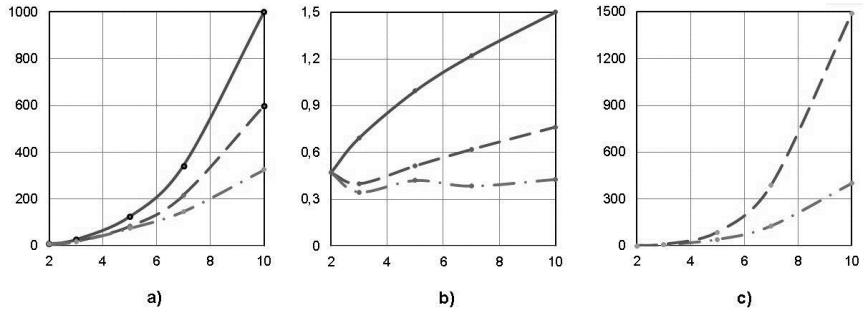

Fig. 4. Parameters of the parallel execution plan while maintaining the SPF height for the algorithm of classical multiplication of square matrices of the 2, 3, 5, 7, 10th orders (corresponds to the numbering along the abscissa).

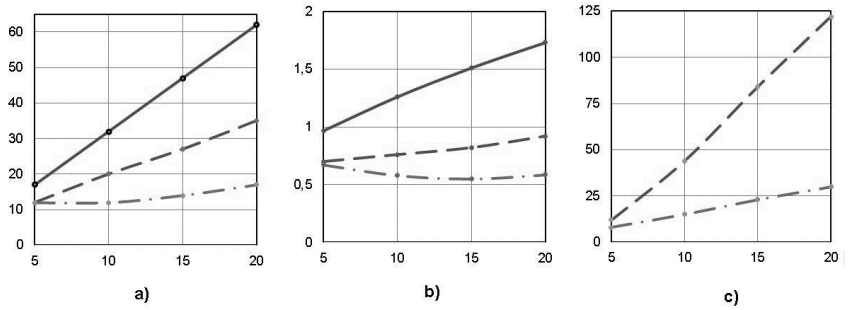

Fig. 5. Parameters of the parallel execution plan while maintaining the height of SPF for the algorithm for computing the pair correlation coefficient by 5, 10, 15, 20 points (corresponds to the numbering along the abscissa axes).

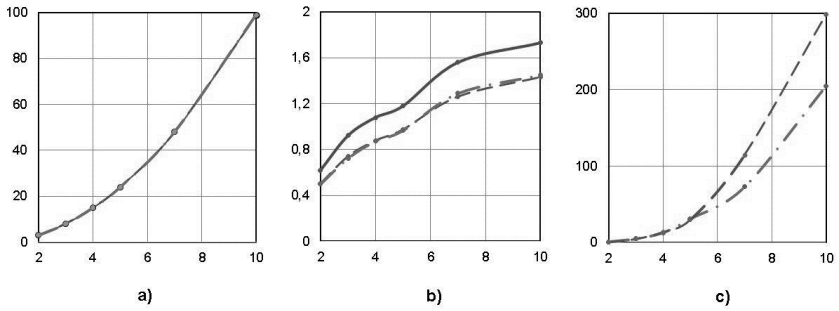

Fig. 6. Parameters of the parallel execution plan while maintaining the height of SPF for the algorithm for solving the linear algebraic equations system (LAES) for the 2, 3, 4, 5, 7, 10th order (corresponds to the numbering along the abscissa axes) by the direct (non-iterative) Gauss algorithm

Fig. 4-6 show that in many cases it is possible to get closer to the specified goal. For example, Fig. 4a) illustrates a decrease in the SPF width up to 1.7 times (1-01_bulldozer method) and up to 3 times (1-02_bulldozer method) when multiplying matrices of the 10th order.

Coefficient of variation of the SPF tier widths (Fig. 4b) approximates to 0.3 (the data set homogeneity border) when using the 1-02_bulldozer empiric and, which is important, is stable over the entire range of data dimensions.

The complexity of achieving the result (Fig. 4c) with 1-02_bulldozer is significantly lower (up to 3.7 times with the 10 order of matrices) than with 1-01_bulldozer. It is important that the method efficiency increases with the growth of the processed data dimension.

The 1-02_bulldozer method proved to be no less effective on the algorithm for calculating the pair correlation coefficient (Fig. 5).

An attempt to reorganize the SPF of the algorithm for solving LAES of order up to 10 by both methods (Fig. 6) turned out to be of little use. It was impossible to reduce the SPF width at all (Fig. 4a). The decrease in CV is very small (Fig. 6b). However, the 1-02_bulldozer method gains slightly in labor intensity (Fig. 6c).

The research scope should be expanded to make the conclusions more complete. However, it is already clear that not every method for SPF reorganizing is equally effective for transforming (in the sense of constructing a rational parallel execution schedule) the various algorithms. As the general approach adopted in this paper presupposes an iterative approximation to the best solution of the set task it is necessary to continue the development of more efficient heuristic methods.

The formulation of the task in the form of programs' execution on a given number of parallel computers is as close as possible to the real case of scheduling a VLIW machine with a given number of processor instructions in a machine word (bundle size) of a very long instruction word). At this, the task is posed of obtaining a schedule with the minimum execution time (minimum SPF height) at the maximum code density (the SPF width is as close as possible to the number of operators in the bundle).

Below, we consider the case of program execution on a given homogeneous field of W parallel computers (from $W = W_0$ to $W = 1$, where W_0 is the SPF width, and the lower bound corresponds to completely sequential execution). We compare two methods of SPF reorganization – Dichotomy and WidthByWidtn:

- Dichotomy. The goal is to obtain an SPF variant of a width not exceeding a given W with an increase in height by transferring operators from a tier to a newly developed tier below the given one. If the width of the tier is more than W, exactly half of the operators from it are transferred to the tier newly created below, and so on, until the width is no higher than the specified W. The method works very quickly, but roughly. The SPF height is clearly excessive and the unevenness of the tiers widths is high).
- WidthByWidtn. Only those operators of the tier with the number of operators higher than the specified $N>W$ are subject to transfer by creating under this tier the M tiers with the number equal to:

$$M = \begin{array}{l} Quot=N/W \text{ for integer division (on each of the lower tiers there will be exactly } Quot \text{ operators.} \\ \hline N/W+1 \text{ in the case of } N/W \text{ division with the remainder (then on the last tier we have exactly } N\%W \text{ operators)} \end{array}$$

An important feature of this method is the rule for selecting the specific operators to be transferred to the newly created tiers.

Figs. 7, 8 show the results of applying these methods to two common linear algebra algorithms – multiplication of square matrices by the classical method and the solution of linear system algebraic equations by the direct (non-iterative) Gauss algorithm. Curves 1 and 2 in these and the following figures correspond to the heuristic methods of WidthByWidtn and Dichotomy, respectively. The reformed SPF width here corresponds to the number of instructions in the bundle of a very long machine word.

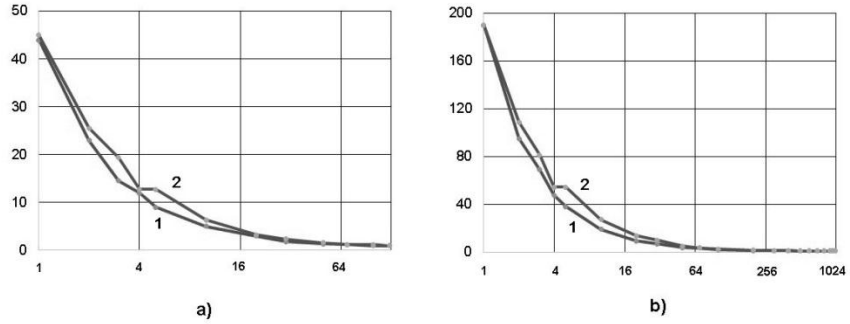

Fig. 7. Increase in height (ordinate) when limiting the SPF width (abscissa), times; the algorithm for multiplying square matrices by classical method of the 5th and 10th orders – Fig. a) and b), respectively

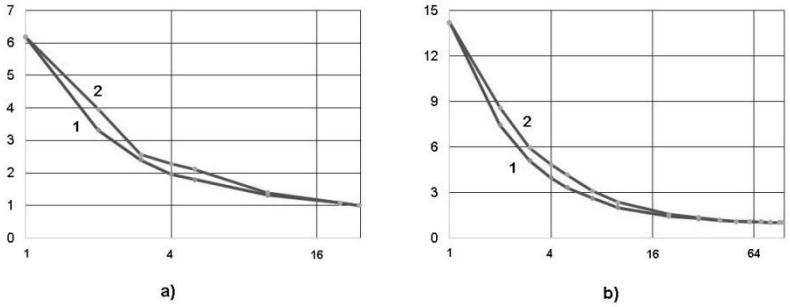

Fig. 8. Increase in height (ordinate) when limiting the SPF width (abscissa), times; an algorithm for solving a linear system algebraic equations by the direct (non-iterative) Gauss algorithm of the 5th and 10th orders – Fig. a) and b), respectively.

As can be seen from Figs. 7 and 8, both methods on the indicated algorithms lead to similar results (from considerations of representing the SPF by a flat table and the invariance of the total number of operators in the algorithm, this is, of course, a hyperbola!).

However, when comparing, the curves corresponding to WidthByWidtn are located *lower* than those in the Dichotomy method. This corresponds to higher running speed. The results obtained by WidthByWidtn practically coincide with the ideal SPF height equal to $N_{tot}/W_{av.}$, where $N_{tot.}$ is the total number of operators, $W_{av.}$ is the arithmetic mean number of operators over the SPF tiers at a given width.

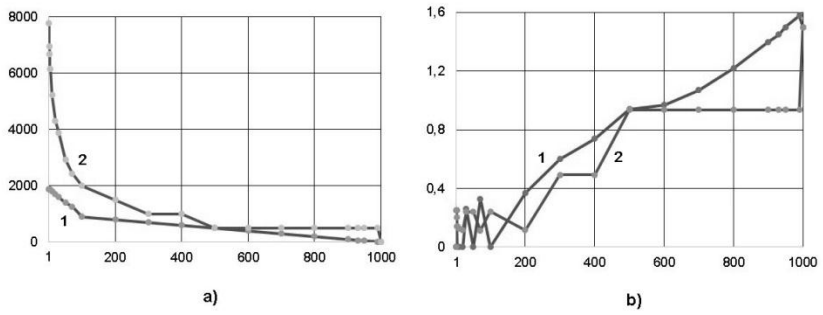

Fig. 9. The number of operator transfers between the tiers - a) and the CV coefficient of variation - b) with decreasing the SPF width for the algorithm of multiplying square matrices of the 10th order by the classical method (abscissa axis is the SPF width after reforming).

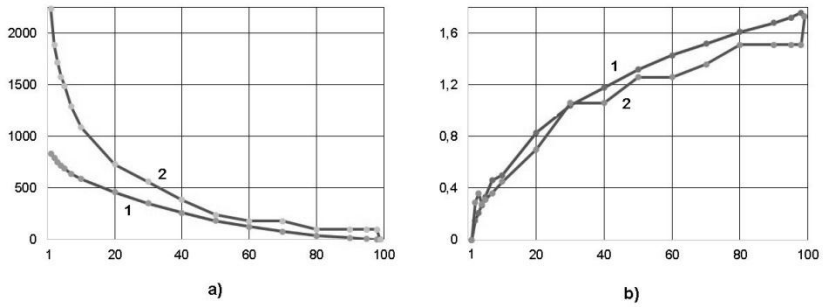

Fig. 10. The number of operator transfers between the tiers - a) and the CV coefficient of variation - b) with decreasing the SPF width for the algorithm for solving the linear system algebraic equations of the 10th order by the direct (non-iterative) Gauss algorithm (abscissa axis is the SPF width after reforming)

The results analysis shown in Figs. 9 and 10 is of greater interest (at least because it has a purely practical interest - computational complexity of the SPF transformation). As can be seen from Figs. 9 and 10, for the cases considered, WidthByWidtn has a lower (approximately 3-4 times) computational complexity (in units of the number of operators transferring from tier to tier) compared to Dichotomy (although the opposite is expected at first glance). However, WidthByWidtn has more complex internal logic compared to Dichotomy (in the latter case, it is primitive). It is logical to use the Dichotomy method in parallelizing translators for fast (but rather rough) construction of parallel program execution plans and the WidthByWidtn method for those construction in optimization mode.

Discussion

In general, the developed software package has confirmed its effectiveness in detecting hidden parallelism of algorithms and researching the possibilities of its rational use in data processing. The approach of using a scripting language for the development of heuristic methods of purposeful transformation of the algorithm information graph form has shown the greater flexibility and transparency for the Researcher. At the same time, flexibility is achieved by using an interpreted scripting language, and processing speed is achieved by executable code of the parent application compiled language.

In the course of the research, a significant dispersion it was revealed in properties of the algorithms (represented by information graphs) according to the possibility of forming the frameworks for parallel execution plans with the maximum code density (or, which is practically the same - with the maximum use of the resources of a parallel computing system).

Moreover, different algorithms require different methods for their effective transformation. It seems to be an important task of a priori (even before the SPF reorganization, at the time of its receipt) determination of algorithm parameters for the selection of effective procedures for their purposeful reformation. Creation of algorithms classifying system on some parameters that determine the effective methods of their transformation should be a tool here. According to the author of this paper, the use of artificial intelligence formal methods for solving this problem may be promising.

The limited amount of processed data caused exclusively by the complexity of manual algorithms compilation may seem to be a certain drawback of the study. Indeed few people are practically interested in the tasks of solving the LAES or multiplying matrices of the 10th order. However, all experiments (including those carried out on algorithms artificially synthesized in accordance with the specified parameters) show that the revealed trends only intensify as the dimension of the processed data increases. And this gives confidence that this trend will continue in the future.

In accordance with the iterative principle of gradual approximation to the best solution of the problem under consideration, inherent in the heuristic approach, the author is confident in the possibility of a quantitative improvement (with respect to the above parameters) in the methods for determining program execution schedules on a given field of parallel computers.

The target consumers of the developed methods for generating parallel program execution schedules are, first of all, the translators and virtual machines developers, researchers of the algorithms' properties (in the direction of finding and using the potential of their latent parallelism). The developed system can be recommended to teachers and those who study the basics of system programming for parallel computing architectures.

References

1. AlgoWiki. Open Encyclopedia of Algorithm Properties. URL: http://algowiki-project.org, accessed on 30/06/2021.

2. Fedotov, I.E.: Parallel programming. Models and techniques. M.: SOLON-Press, pp. 390 (2018).
3. Dennis, J.B., Misunas, D.P.: A Preliminary Architecture for a Basic Data-Flow Processor. In Proc. Second Annual Symp. Computer Architecture, pp. 126-132, Houston, Texas (January, 1975).
4. Bakanov, V.M.: Controlling the dynamics of computations in processors of streaming architecture for various types of algorithms. Software Engineering Journal 9, 20-24 (2015).
5. Tanenbaum, E., Austin, T.: Computer architecture. SPb.: Peter Publishing House, pp. 816 (2019).
6. Bakanov, V.M.: Software complex for modeling and optimization of program implementation on parallel calculation systems. Open Computer Science, vol. 8, Issue 1, pp. 228–234, ISSN (Online) 2299-1093, DOI: https://doi.org/10.1515/comp-2018-0019.
7. Gehry, M., Johnson, D.: Computing machines and intractable tasks. Mir, Book on Demand, pp. 420, (2012).
8. Roberto Ierusalimschy. Programming in Lua. Third Edition. PUC-Rio, Brasil, Rio de Janeiro, pp. 348 (2013).
9. Fisher, Joseph A.: Very Long Instruction Word architectures and the ELI-512. International Symposium on Computer Architecture. Proceedings of the 10th annual international symposium on Computer architecture. New York, NY, USA: Association for Computing Machinery (ACM), pp. 140-150 (1983).