ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 5, Issue 12, December 2018

Analysis of Algorithms and Presentation of Algorithm Graph in a Line-Parallel Form

D.M. Okhunov, M.H.Okhunov

Cand.Econ.Sci., the senior lecturer of the Fergana Branch of the Tashkent University of Information Technologies named after Mukhammad Al-Khorezmi, Fergana, Uzbekistan
The senior lecturer Fergana Polytechnic Institute, Fergana, 150100, Uzbekistan

ABSTRACT: The article considers the main approaches to the organization of multiprocessor computing systems, the development of parallel algorithms for numerical solution of problems and technologies of parallel programming.

KEYWORDS*:* Line-parallel form, equivalent algorithm transformation, algorithm graph, deterministic graph, nondeterministic graph, operators, operands, adjacency matrix, method of identical vertices.

I. INTRODUCTION

The cardinal difference between the execution of the program on a parallel architecture computer system from that of a serial computer is the possibility of simultaneously executing a whole group of operations that are independent of each other (on a sequential machine, only one operation is performed at each moment of time, others can only be on stage of preparation). On different parallel machines, these groups and the sequence of their execution will most likely be different. However, there is a (natural) requirement for the repeatability of results (an algorithm that leads to different end results even for identical input data, hardly anyone needs - modeling of statistical processes is not considered) [1].

In general, the development phase of a parallel program should be preceded by the process of identifying blocks (sequences of executable instructions) that can be executed independently of each other and then in the program in which synchronization of the execution of these blocks is necessary (for input or exchange of data). Only for the simplest algorithms this task can be performed "in the mind", in most cases (quite complex) analysis of the structure of the algorithm is required. In some cases, it is advisable to perform equivalent transformations of the algorithm (replacing this algorithm - or part thereof - with an algorithm that guarantees the same end result on all sets of input data, preferably without reducing the accuracy of calculations).

II. SIGNIFICANCE OF THE SYSTEM

The software user component of parallel computing technologies includes both the choice (and in some cases the independent development) of the problem solving algorithm and the rational (from the point of view of the architecture of the multiprocessor computer system - AIM) its program implementation (a convex example is the classical algorithm Matrix multiplication can be parallelized in a dozen ways, by orders of magnitude differing in the execution time of the problem with the same size of the original data).

The most important stage here is the identification (usually hidden) of parallelism in the algorithm (in fact, the identification of code segments that are independent of data - that is able to be executed independently, and therefore in parallel). One of the methods for revealing parallelism is the representation of the algorithm in the so-called line-parallel form (LPF), while on the separate layer there are operators' dependent (based on the initial data for execution - operands) only from the results of operations that are higher level. The operation of representing the algorithm in the JPL can be performed programmatically or in hardware (using a computer system with a specialized architecture).

ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 5, Issue 12, December 2018

III. LITERATURE SURVEY

Identifying parallelism in an arbitrary algorithm is a non-trivial task. The fact is that parallelism is usually hidden (for an untrained mind). However, there are formal procedures that allow us to reveal the hidden parallelism in the algorithm; one of them is the representation of the algorithm in a line-parallel form (LPF) [2].

IV. METHODOLOGY

For a long time we know and apply the method of representing the algorithm in the form of a graph structure. The graph G is usually denoted by G = (V, E), where V is the vertex set, E is the edge set, and the edge between the vertices i and j is denoted as e (i, j). In general, the vertices of a graph correspond to some actions of the program, and the edges to the relations between these actions.

The simplest graph of this kind describes the information dependencies of the algorithm (the vertices of the graph correspond to individual operations of the algorithm, the presence of an edge between the vertices i, j indicates the necessity for the operation j to have the arguments (operands) generated by the operation i, in the case of independence of the operations i and j there is no arc between the vertices). Such graph is called the graph of the algorithm (the computational model "operators - operands"). Even in the absence of conditional statements (which is unlikely), the number of operations performed (and therefore the total number of vertices of the graph and, accordingly, the number of edges) depends on the size of the input data, that means the algorithm graph (AG) is parameterized in the size of the input data. The acyclivity of AG follows from the impossibility of determining any quantity in the algorithm through itself. AG is also oriented (all the edges are directed). Distinguish between deterministic AG (the program does not contain conditional operators) and nondeterministic AG (otherwise). For non-terminal AG there is no one-to-one correspondence between the operations of the program describing it and the vertices of the graph for all sets of input parameters; therefore, deterministic algorithms are most often considered. Having no input or output edge, vertices of AG are called input or output vertices, respectively. The construction of AG is not a labor-intensive operation (which cannot be said about graph analysis procedures) - any compiler (interpreter) constructs (explicitly or implicitly) it when analyzing each expression of a high-level programming language

V. EXPERIMENTAL RESULTS

The figure 1 shows the computation algorithm is given by the formula r=a*b+a/cin general. The initial data for the calculation are constants a, b, c, the result is d. In general, the transformation r ← a, b, c requires 3 actions (operators) - a * b, a / c and a * b + a / c. The initial data (operands) for the first statement are a, b; for the second - a, c; for the third one, the results of the computations a*b and a/c (see “the computation’s cloud”r ← a, b, c, shown in Figure 1.a).

ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 5, Issue 12, December 2018

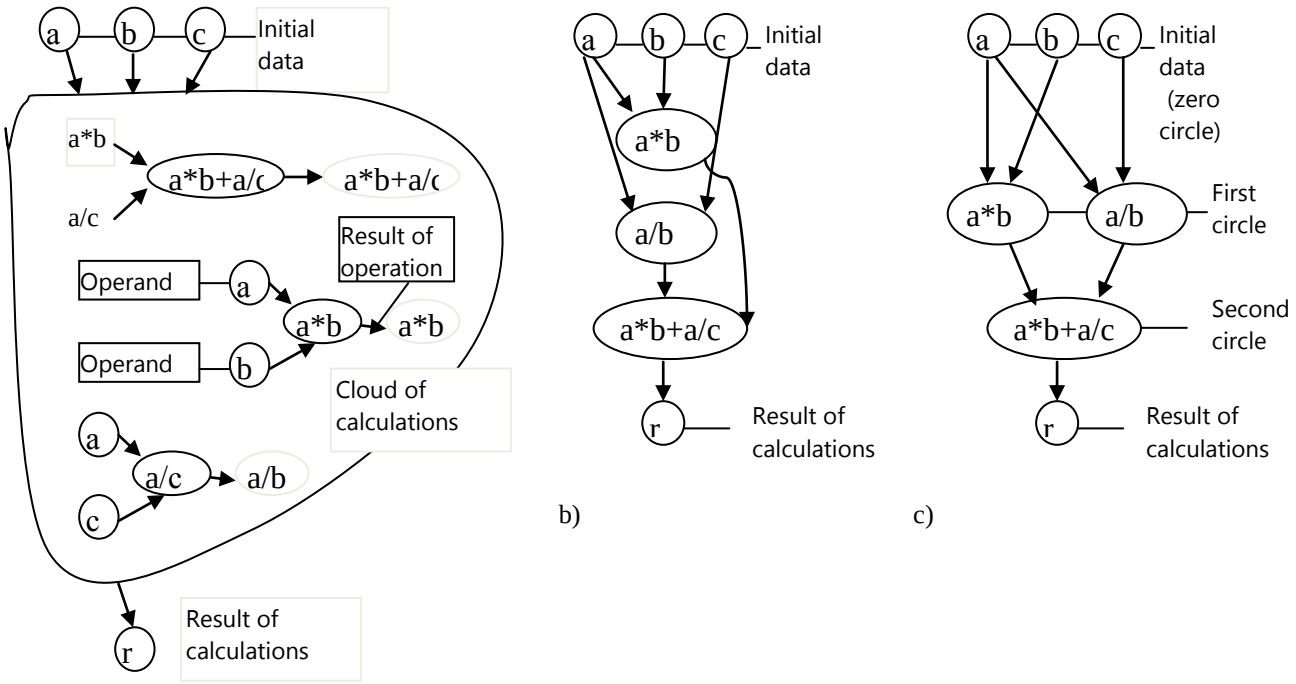

Fig. 1. Methods of transformation r ← a, b, c: a) - representation of the algorithm in the form of a "cloud of computations" (the order of execution is not defined), b) - sequential execution, c) - a leveled parallel form of the algorithm

For a sequential calculator, the order in which operators are executed on the surface is first calculated by a * b and a / c (and in any sequence), then a * b + a / c (Figure 1. b). Because the execution of a * b and a / c does not depend on each other (they say that they are orthogonal with respect to the operands), it is easy to get the LPF for this case - figure 1. c); often from the numbering of levels, the very first and last are included (the initial data and the results are relevant, since they do not actually compute). As a result, it becomes clear that this algorithm can be calculated in parallel processing in 2 steps (a * b and a / c simultaneously, then a * b + a / c sequentially) instead of 3 at successive (one by one a * b, a / c and a * b + a / c); and on the first level you need the work of two arithmetic processors (action multiplication and division), on the second - one (action - addition). In the case of sequential processing, the total execution time of the algorithm will be equal to the sum of 3 actions $t_{a*b} + t_{a/c} + t_{a*b+a/c}$, in the case of parallel - $\max(t_{a*b}, t_{a/c}) + t_{a*b+a/c}$(in the case of $t_{a*b} = t_{a/c} = t_{a*b+a/c}$, we get a AGin speed by one and a half times).

In fact, when the graph of an algorithm is transformed into the LPF, an analogy of the internal structure of the algorithm is carried out in order to find groups of operators that can be executed in parallel. Note that the number of levels determines the length of the critical path.

Generally speaking, operators can be considered complete actions of any complexity in the transformation of data - from a single expression (string) or a group of strings in a high-level programming language to a single machine (processor) instruction. However, there is a significant difference in the number of operands between these extreme cases (for a high-level language operator, the number of operands can reach tens / hundreds, and for a processor instruction, usually one / two). Of course, it is much easier to implement transformations similar to the above described operations with operations having 1-2 operands, than with dozens of operands.

Consider a slightly more complicated example - the solution of the roots x1, x2 of the complete quadratic equation a * x2 + b * x + c = 0 by means of an algorithm (suggested by the Indian mathematician Brahmagupta in the 7th century AD) in the form of an algebraic formula x1,2 = (-b ± sqr (b2-4 * a * c)) / (2 * a), where a, b, c are constants, sqr is the square root extraction operation. The sequence of calculations (one of the variants) for finding the roots is given in

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 5, Issue 12, December 2018

Table 1. and requires about a dozen operations (addition, subtraction, multiplication, division, change of sign, calculation of the square root).

Table 1. Sequence of calculating root values of the complete quadratic equation

№	*Action*	*The note*
	Input a, b, c	Input operations are not numbered
0	a2← 2 * a	a2 - working variable
1	a4 — 4 * a	a4 - working variable
2	b_neg←neg(b)	b_neg - working variable; neg - operation sign change (‘ a monadic minus ’)
3	bb←b * b	bb - working variable
4	ac4 ← a4 * c	ac4 - working variable
5	p_ sqr←bb-a4	p_ sqr - working variable
6	sq←sqrt(p_sqr)	sq - working variable, sqrt - operation of calculation of a square root
7	w1← b_neg+sq	w1 - working variable
8	w2←b_neg-sq	w2 - working variable
9	root_1←w1/a2	root_1 - the first root of the equation
10	root_1←w2/a2	root_ 2 - the second root of the equation

When the algorithm graph is sequentially computed (Fig. 2), the sequence of actions in Table 1 is completely copied; has 3 input vertices (corresponding to the input of the coefficients a, b and c), two output vertices (calculated roots x1, x2 of the original equation) and 11 vertices corresponding to the operators of the algorithm.

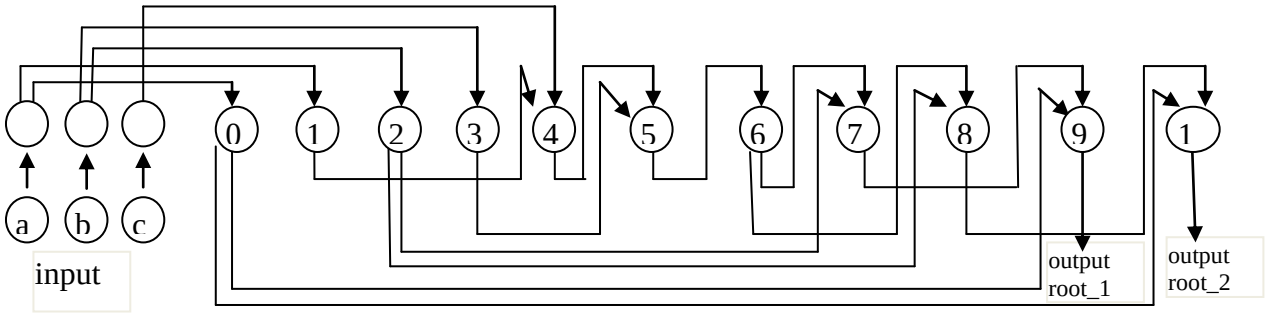

Fig. 2. Graph of algorithm (dependence of 'operation - operands') finding the roots of the complete quadratic equation for sequential execution

The same graph is shown in figure 3. in the JFP - in each level operators are collected, requiring for their performance values (operands) calculated only on the previous levels (in total, in Figure 3. 6 levels are allocated); t.o. parallel processing of this algorithm requires a sequential 6-parallel execution of blocks of parallel operations (in each of which 4,1,1,1,2,2 non-dependent processes are launched respectively, and lines 2,3,4 degenerate into follow-up implementation).

Analysis graph drawing. 3 allows you to make some specific conclusions about alternatives to parallelization. Note that level 1 is not loaded with operations (3 multiplications and 1 change of sign), some of them (except operation 2) can be transferred to the lower levels (options: operation 4 to level 2, operation 3 to levels 2 , 3 or 4, operation 1 on lines 2,3,4

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 5, Issue 12, December 2018

or 5); Specific variant should be selected based on additional data (for example, the time of execution of specific operations, the number of involved computing modules, minimizing the time of data exchanges between modules). Optimization of the calculation graph is a very labor-intensive operation (comparable to the complexity of the calculations themselves); some of the post-problems of the problem are given in [3].

The identification of these levels is one of the levels of analysis of the internal structure of the algorithm. With a large number of operators, the above transformation of the AG into an LPF is not easy, for this purpose special software is used.

Below is a simplified sequence of actions (Table 2) to identify possible graphs of algorithms that can be executed in parallel levels.

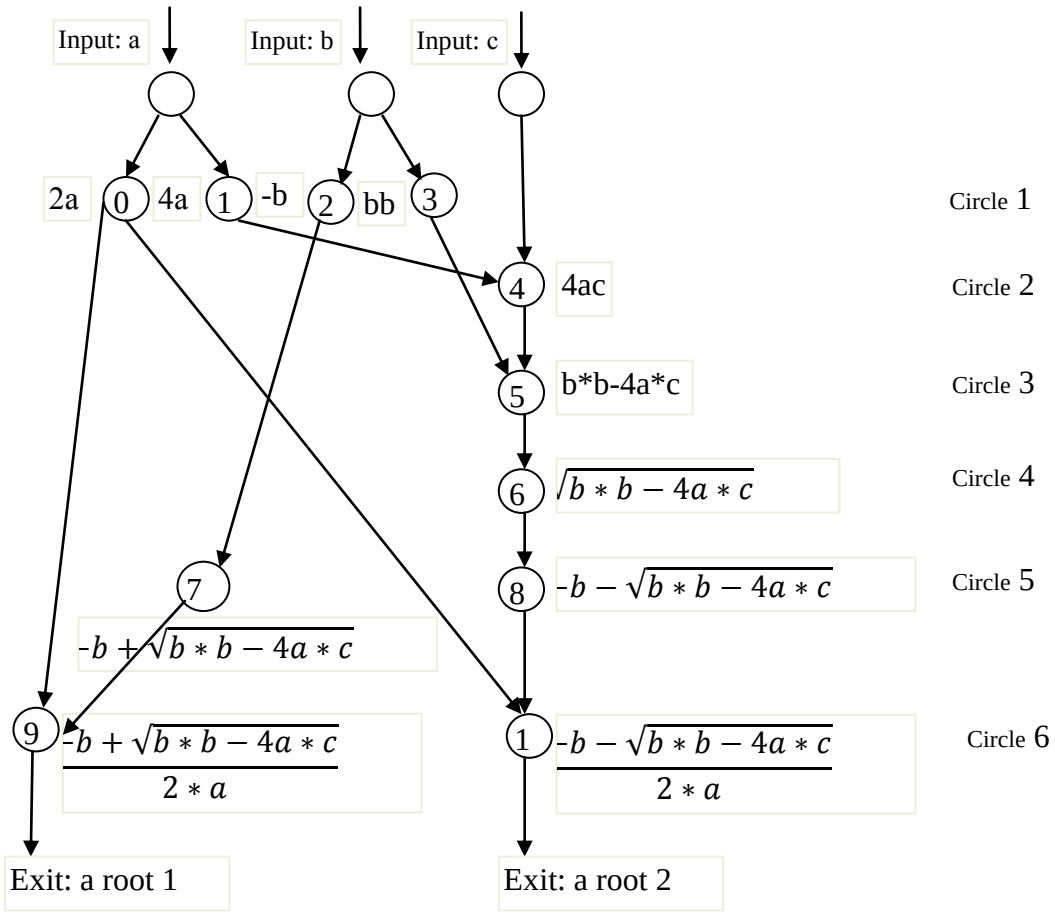

Fig. 3. The graph of the algorithm (the dependence of 'operation - operands') for finding the roots of the full quadratic equation with the grouping of operations over the longlines (in the LPF).

ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 5, Issue 12, December 2018

Table 2. A simplified sequence of actions to identify the graph of the algorithm that can execute in parallel lines

Item	*Action*	*Note*
a)	Define a list of vertices that depend only on the input data; put it in the list 1	The initial data for the algorithm's work
b)	Find vertices dependent on edges from vertices entering only the list 1 and previous lists (if any); put them in the list 2	The most time-consuming item that requires viewing the contiguity matrix for each element of the list
c)	If list 2 is not empty, copy it to list 1 and go to step b); Otherwise, finish the work	Loop over the levels, while the data can be detected

The implementation of the algorithm in C ++ is given below (the original data is a square Boolean contiguity matrix MS [] [] of dimension N_MS, one-dimensional integer arrays LIST_1 [] and LIST_2 [] of length N_L1 and N_L2 respectively):

```
001   do {                          // by levels how many will be detected
002   N_L2=0;
003   for (ii=0; ii<N_L1; ii++) {   //   loop through the vertices in the list LIST_1
004    i_ii=LIST_1[ii];             // i_ii - the number of the vertex from the list LIST_1
005    for (j=0; j<N_MS; j++)       // loop through the columns of MS (j is the number of
                                    // the vertex to which the arc is directed)
006   if (MS[i_ii][j]) {            // found some arc i_ii → j
007   j1 = j;                       // remember the vertex to which the arc from i_ii
008   for (i1=0; i1<N_MS; i1++)     // by the MS lines = outgoing vertices
009   if (MS[i1][j1]) {             // found some arc i1 → j1
010  flag=false;
011  for (k=0; k<N_L1; k++)         // loop on list LIST_1
012  if (LIST_1[k] == i1)           // if vertex j1 is included in LIST_1 ...
013  flag=true;                     // for flag = true, the vertex i1 enters the list LIST_1
014  }                              // the block end if (MS[i1][j1])
015  if (flag)                      // ... if i1 is in the list LIST_1

016  LIST_2[N_L2++] = j1;           // add vertex j1 to list LIST_2
017  }                              // the block end if (MS[i_ii][j])
018  }                              // the block end for (ii=0; ii<N_L1; ii++)
019  for(i=0; i<N_L2; i++)          // copy LIST_2 to LIST_1
020 LIST_1[i] = LIST_2[i];
021 N_L1 = N_L2;
022 } while (N_L2);                // ... while the list of LIST_2 is not empty
```

The time t_j of the operations of each level is determined by the execution time of the longest operation from located on this level ($t_j=\max(t_{ji})$, where j is the level number, i is the operator number in this level). When planning the execution of a parallel program, it is necessary to take into account the limited number of processors ($P \le P_{max}$), so it is rational to transfer a part of (usually quickly executed) operators to lower (less full) levels.

VI. CONCLUSION AND FUTURE WORK

When analyzing the algorithm and identifying the levels of parallelism, computer-convenient methods for representing the graph in computer memory are used [4,5]. One of the (not most memory-conservative) representations of the graph G = (V, E) is the square adjacency matrix (the numbering of rows and columns corresponds to the numbering of the operators, '1' in the (i, j) - cell corresponds to the presence of the edge e (i , j), '0' - its absence), see the figure.

ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 5, Issue 12, December 2018

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & & & & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ & & & & & 0 & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & & & & & & & & & \end{pmatrix}$$

Fig. 4. The adjacency matrix for representing the algorithm graph

Classically, the adjacency matrix is Boolean; however, if it is necessary to more subtly model its elements can be integers or even real; for example, to take into account the value of the metric (as a time delay for the delivery of information, for example) of a data channel.

REFERENCES

[1]. Voevodin V.V. Parallelnye calculations. «BHV – Petersburg», Sank-Peterburg 2004, p.608.
[2]. *Garvi M.Deytel.* Introduction in operational systems (the lane from English L.A.Teplitskogo, A.B.Hoduleva, Bc. S.Shtarkmana under edition Bc. S.Shtarkmana). «The World», Moscow 1987.
[3]. *Gergel V.P., Strongin R.G.* Bases of parallel calculations for multiprocessing computing systems (the manual, pub. 2, added). «NNSU of N.I.Lobachevsky», N.Novgorod 2003 (the electronic version <http://pilger.mgapi.edu/metods/1441/basic_pr.zip>).
[4]. Korneyev V.V. Computing systems. «Gelios APB», Moscow 2004, p.512.
[5]. *Kryukov V.A.* Working out of parallel programs for computing *klasterov* and networks. «Information technologies and computing systems», Moscow 2003, № 1-2, p.42-61.

AUTHOR'S BIOGRAPHY

Dilshod Mamatjonovich Okhunov - Cand.Econ.Sci., the senior lecturer of the Department "Software Engineering" at the Fergana Branch of the Tashkent University of Information Technologies named after Mukhammad Al-Khorezmi. He graduated from the Faculty of "Information technologies" at the Tashkent state economic university. The expert in the field of designing and development of quality of information systems and technologies, calculus mathematics.
Conducts teaching activity on preparation of experts in the field of Software Engineering, information management. Teaching work combines with scientific activity. Takes part in scientific projects of the foreign organisations.
He has more than 150 scientific papers published on international and national journals and conference materials, including monographies, manuals and patents on problems of creation and management of information systems and technologies.

Mamatjon Hamidovich Okhunov - candidate of the physicist-mathematical of sciences, the senior lecturer of the Department "Computer science and information technologies" at the Fergana Polytechnic Institute. He graduated from the Faculty of "Calculus mathematics" at the Tashkent state university. The expert in the field of information technologies, calculus mathematics.
He has more than 200 scientific papers published on international and national journals and conference materials.