Optimizing the concurrency level of network artificial intelligence computations

Anatoly Prihozhy
Information Technologies and Robotics Department
Belarusian National Technical University
Minsk, Republic of Belarus
Email: prihozhy@yahoo.com

***Abstract*—Recent achievements in development of artificial intelligence systems would be impossible without the use of high performance distributed computing platforms. This paper presents a graph model of concurrent network schedules and a technique that estimates the execution time and implementation cost over maximum weight cliques in task graphs. It proposes an algorithm for recalculating clique sets after changing the concurrency level of a schedule by adding an edge to the concurrency graph and removing the edge from the complement graph. Since the set of pairs of concurrent tasks has been found, it treats a schedule existence problem as solving a combined logical equation. The proposed model and technique are a basis for the development and implementation of network algorithms.**

***Keywords*—artificial intelligence systems; distributed computations; computing schedule; parallelization; optimization**

I. Introduction

Distributed artificial intelligence systems and semantic networks aim at solving complex knowledge acquisition, reasoning, learning, recognition, planning, decision-making and other problems in the case they process data on large-scale distributed computing platforms [1] - [3]. Distributed intelligence systems consist of numerous autonomous agents that perform learning, processing and communication tasks. Large-scale graph mining is an important technology in modeling, analyzing and solving artificial intelligence problems on large computer networks [4] - [6]. Processing such graphs is possible only by developing distributed algorithms for parallel systems. Deep neural networks are an important branch in artificial intelligence [1]. In deep machine learning, the neural networks trained by observing big data provide good solutions for problems thought to be unsolvable: autonomous driving, image classification, speech recognition, medical diagnosis, complex games and others. Parallelization strategies and concurrent algorithms for evaluation, implementations and training of deep neural networks make such networks the field of high performance parallel computing. In this paper, we extend the net scheduling techniques [7] - [9] for optimizing the concurrency level of solving intelligent problems.

II. Graph Model of Network Schedule

Network scheduling determines the precedence and concurrency relations between tasks, which conserve both time and resources [7] - [9]. Let $N = 1, ...n$ be a set of task numbers. A directed graph $G_H = (N, H)$ can describe the net schedule without constraints on resources, where H is the tasks direct precedence relation. If tasks $i_1, ..., i_k$ are direct predecessors of task j in the network schedule, then j may execute when all of its predecessors have finished executing.

A binary matrix Q (Figure 1) describes data dependences between the tasks, which element $q_{i,j}$ equals 1 if i is a predecessor of j, and equals 0 otherwise. In a triple matrix W, element $w_{i,j}$ equals 0 if the tasks i and j may not execute on the same processor, equals 1 if the tasks may execute on the same processor sequentially, and equals 2 if the tasks may execute on the same processor concurrently. The last case applies when the tasks $T1$ and $T2$ are mutually exclusive [7]; that is the tasks are under conditions $if\ c1\ then\ T1\ end\ if$ and $if\ c2\ then\ T2\ end\ if$, and test variables $c1$ and $c2$ are orthogonal, i.e. they cannot take value true simultaneously. We can equivalently transform the tasks behavior to the single basic block dataflow model presented in [9].

$$Q = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&1&1&0&0&1&0&0\\ 0&0&0&0&1&1&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&1&1&0&1\\ 0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \quad W = \begin{bmatrix} 0&0&1&0&0&1&0&1&1&1&1\\ 0&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&1&0&1&1&1&1\\ 0&0&0&0&1&0&1&0&0&0&0\\ 0&0&0&1&0&0&1&0&0&0&0\\ 1&0&1&0&0&0&0&1&1&1&1\\ 0&0&0&1&1&0&0&0&0&0&0\\ 1&0&1&0&0&1&0&0&1&1&1\\ 1&0&1&0&0&1&0&1&0&1&1\\ 1&0&1&0&0&1&0&1&1&0&1\\ 1&0&1&0&0&1&0&1&1&1&0 \end{bmatrix}$$

Figure 1. Matrices Q and W of tasks precedence and compatibility

III. Schedule Execution Time and Cost

Let t_j and s_j be the execution time and implementation cost respectively of a task on processor of type j. Time t_j can be constant or variable. If $type(i)$ denotes the type of processor that executes task i, the network schedule execution time is

$$T = \max_{u \in U_{\overline{D}}} \sum_{i \in u} t_{type(i)} \tag{1}$$

where $U_{\overline{D}}$ is the set of cliques of graph $G_{\overline{D}} = (N, \overline{D})$ constructed on a set N of the nodes that represent tasks, and on a set $\overline{D}$ of edges that represent precedence of the tasks. The clique of $G_{\overline{D}}$ that gives the maximum sum of the tasks execution time defines the schedule execution time. The network schedule implementation cost is

$$S = \sum_{j=u}^{Types} S_j \times (\max_{v \in V_D} m_{jv}) \tag{2}$$

where $Types$ is the number of processor types; V_D is the set of cliques of graph $G_D = (N, D)$ constructed on the set D

of edges that represent concurrency of the tasks. The number of processors of type j needed to execute the tasks of clique v concurrently is m_{jv}. The sum of costs of all type processors defines the overall cost. Clique set V_D provides the number of all type processors. Sets $U_{\overline{D}}$ and V_D determine the path and section of maximum weight in graph G_H.

IV. OPTIMIZATION OF NETWORK SCHEDULES

The optimization of a network schedule aims at: (1) minimizing the schedule execution time, given constraints on resources; (2) minimizing the resources, given constraints on the execution time. While set D_M determines the most concurrent (and thus fastest) network schedule, a subset D of D_M determines a network schedule of less concurrency, yet lower system cost. Set D also defines execution time T and cost S. We can find up to 2^r different network schedules, where r is the cardinality of set D_M. As the tasks in any pair of set D_O of orthogonal tasks execute concurrently and does not require additional resources, we can always include D_O in D. For instance, the sample matrix Q (Figure 1) is potentially a source for generating 2^{39} network schedules. Synthesizing a net schedule involves solving one of two optimization problems, depending on the optimization criteria selected:

$$\min_{D \in D_M} \{T_D | S_D \leq S_O\} \tag{3}$$

or

$$\min_{D \in D_M} \{S_D | T_D \leq T_O\} \tag{4}$$

where T_O and S_O are constraints on execution time and implementation cost. We account for the estimates of execution time (1) and implementation cost (2) to calculate the value of T_O and S_O .

Two techniques let us generate D while solving problems (3) and (4) by consecutively adding pairs to D and by consecutively removing pairs from D. The first technique solves problem (3) and starts with set $D = D_O$. The second technique solves problem (4), starts with set D_M, and never removes orthogonal pairs of D_O from D. Because of the concurrency of orthogonal tasks, the pairs of D_O do not require additional execution time and implementation cost.

Both techniques select a pair for including in or removing from D by analyzing the maximum-weight cliques of sets $U_{\overline{D}}$ and V_D. First of all, they select pairs that decrease the execution time and not increase the implementation cost.

V. CALCULATION OF GRAPH CLIQUE SETS

Adding or removing a pair from D changes the clique set according to four rules. Two rules transform $U_{\overline{D}}$ into $U_{\overline{D"}}$ when we add pair $d = (i, j) \in D_M$ to set D creating new set $D" = D \cup \{d\}$. The first rule splits a clique containing tasks i and j into two new cliques of less cardinality; the second rule allows the removal of cliques from the new set $D"$:

- Rule 1 (splitting) - If element $u \in U_{\overline{D}}$ satisfies the condition that $\{i, j\} \in u$, then the elements $u\backslash\{i\}$ and $u\backslash\{j\}$ are added to set $U_{\overline{D"}}$; otherwise element u is
- Rule 2 (absorbing) - If in set $U_{\overline{D"}}$ two elements u' and $u"$ exist for which $u' \supseteq u"$, then element $u"$ is removed from the set

Two additional rules recalculate set V_D as new set $V_{D"}$. The third rule combines two cliques containing both tasks i and j into a new clique that is included into set $V_{D"}$. The fourth rule removes the absorbed cliques from the set:

- Rule 3 (merging) - If $v' \cup v" \supseteq \{i, j\}$ is true for $v', v" \in V_D$ then element $v = (v' \cap v") \cup \{i, j\}$ is added to $V_{D"}$. All elements of V_D are also included in $V_{D"}$
- Rule 4 (absorbing) - If in set $V_{D"}$ two elements v' and $v"$ exist for which $v' \supseteq v"$ then element $v"$ is removed from the set

If we remove pair d from set D and $D' = D\backslash\{d\}$ is the new set, then rules 1 and 2 transform set V_D into the set $V_{D"}$, and rules 3 and 4 transform set $U_{\overline{D}}$ into the set $U_{\overline{D"}}$.

Solving problem (3) to minimize the execution time for matrix Q with one processor of type $p1$ and two processors of type $p2$ ($t_{p1} = 100ms, t_{p2} = 40ms$) produces set D, which contains 31 pairs, as described by the zero elements of the top right part of matrix Q^x_D (Figure 2). The markings along the column heads indicate the tasks, executed on processors of type $p1, p2$ and control type c. No pair is added to D without increasing the number of processors (Figure 3). For each clique of set $U_{\overline{D}}$, the execution time is the sum of the clique tasks' execution time. The overall time is 340 ms (Figure 4).

$$Q^x_D = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & x & 0 & x & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & x \\
0 & 0 & 0 & 0 & 1 & 1 & x & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & x & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x \\
0 & 0 & 0 & \overline{x} & \overline{x} & 0 & 0 & 1 & 1 & 0 & 1 \\
\overline{x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\overline{x} & 0 & 0 & 0 & 0 & 0 & 0 & \overline{x} & 0 & 0 & x \\
0 & 0 & \overline{x} & 0 & 0 & \overline{x} & 0 & 0 & 0 & \overline{x} & 0
\end{bmatrix}$$

Figure 2. Matrix Q^x_D

No	1	c	1	2	2	1	2	1	1	1	1	p1	p2
1	0	1	0	0	0	0	0	0	1	1	0	2	0
2	0	1	0	0	0	1	0	1	0	0	0	2	0
3	1	0	0	0	0	0	0	0	1	0	0	2	0
4	0	1	1	0	0	0	1	0	0	1	0	2	1
5	0	1	1	1	0	0	0	0	0	1	0	2	1
6	0	1	1	1	0	0	0	1	0	0	0	2	1
7	1	0	0	0	0	1	1	0	0	0	0	2	1
8	0	0	0	0	0	0	0	0	1	0	1	2	0
9	0	1	0	0	0	1	1	0	0	1	0	2	1
10	1	0	1	1	0	0	0	0	0	0	0	2	1
11	1	0	1	0	0	0	1	0	0	0	0	2	1
12	0	0	0	1	0	0	0	1	0	0	1	2	1
13	0	0	0	0	1	0	0	1	0	0	1	2	1
14	0	1	0	0	1	0	0	0	0	1	0	1	1
15	0	1	0	0	1	0	0	1	0	0	0	1	1
16	1	0	0	0	1	0	0	0	0	0	0	1	1
											max=	2	1

Figure 3. Clique set V_D for matrix Q^x_D

If we add pair (i, j) to set D, tasks i and j are concurrent; if (i, j) is not included in set D_M, task i precedes task j. For pair (i, j) of set D_M not included in set D, we know that tasks i and j are not concurrent, but do not know whether i should precede j or j should precede i.

Introducing Boolean variable x_{ij} into matrix Q^x_D for pair (i, j) and its negation $\overline{x}_{ij}$ for pair (j, i) solves this problem. If x_{ij} equals 1, task i precedes task j. If the value equals 0, j precedes i. Thus, while many net schedules possible for a given D, for some set D no net schedule exists.

No	1	c	1	2	2	1	2	1	1	1	1	T
1	1	0	0	0	0	0	**0**	1	0	1	0	120
2	**0**	0	1	0	0	1	**0**	0	0	0	1	120
3	1	0	0	0	0	0	**0**	0	0	1	1	120
4	**0**	0	0	0	0	0	1	1	1	0	0	180
5	**0**	0	0	1	**1**	1	**0**	0	1	0	0	280
6	**0**	0	0	1	**1**	0	1	0	1	0	0	340
7	**0**	0	1	0	**1**	1	**0**	0	1	0	0	220
8	1	1	0	0	0	0	**0**	0	0	0	1	280
9	**0**	0	0	0	0	0	**0**	0	0	0	1	140
										max=		340

Figure 4. Clique set $U_{\overline{D}}$ for matrix Q_D^x

VI. SOLVING THE EXISTENCE PROBLEM

For set D and the given values of variables x_{ij}, a net schedule exists if the matrix derived from the matrix Q_D^x by substituting the variable values describes a transitive relation. This transitivity condition expresses the requirement that the net schedule must have the level of concurrency the set D defines. The relation is transitive if the logical equations (5)-(7) have at least one solution for x_{ij}. In the equations, variables z_{ij} are intermediate. Equation (5) describes the transitivity condition for the elements of set D, and equation (6) describes the transitivity condition for the elements out of D. One algorithm effectively solves the equations (5)-(7) by constructing a graph G_D^x and searching for its non-conflicting labeling (Figure 5).

$$\sum_{\substack{i,j,k\in N,(i,j)\in D\\(i,k)\notin D,(k,j)\notin D}} [(z_{ik}\wedge z_{kj})\vee(z_{jk}\wedge z_{ki})] \tag{5}$$

$$\sum_{\substack{i,j,k\in N,(i,j)\notin D\\(i,k)\notin D,(k,j)\notin D}} [(z_{ij}\wedge z_{ik}\wedge z_{kj})\vee(z_{ji}\wedge z_{jk}\wedge z_{ki})] \tag{6}$$

$$z_{ij}=\begin{cases}1, & \text{if } (i,j)\notin D_M \text{ and } i<j\\ 0, & \text{if } (i,j)\notin D_M \text{ and } j<i\\ x_{ij}, & \text{if } (i,j)\in D_M\backslash D \text{ and } i<j\\ \overline{x_{ij}}, & \text{if } (i,j)\in D_M\backslash D \text{ and } j<i\end{cases} \tag{7}$$

The graph nodes are variables x_{ij} that correspond to non-concurrent pairs of tasks. The algorithm introduces edge (x_{ij}, x_{ik}) if tasks j and k are concurrent and pair (j,k) belongs to set D. It labels the graph nodes with 0 and 1. The initial label 1 is assigned to the nodes which belong to set $\{x_{ij}|(i,j)\notin D_M, i,j=1,...,n, i<j\}$ of the Boolean variables that correspond to the non-concurrent task pairs not included in set D_M. If an edge connects two variables x_{ij} and x_{jk} that satisfy constraint $i<j<k$, it is labeled +, otherwise it's labeled -. Labeling two variables and the edge connecting them creates one type of conflict if the variable labels are the same and the edge label is +, or the variable labels are different and the edge label is -. If the graph has at least one of this first type of conflict, equation (5) has no solution. For variables x_{ij}, x_{ik}, and x_{kj} where $i<k<j$, a second type of conflict occurs if variable x_{ij}'s value equals 0 (1) and the values of x_{ik} and x_{kj} equal 1 (0). If the graph has at least one such conflict, equation (6) has no solution. To generate a net schedule, the algorithm must label the nodes in such a way as to avoid the conflicts of both types.

Removing nine pairs from D updates matrix Q_D^x (Figure 6) in such a way as to satisfy equations (5)-(7). Figures 7 and 8 show the updated clique sets V_D and $U_{\overline{D}}$. The concurrent schedule execution time has increased from 340 ms to 380 ms.

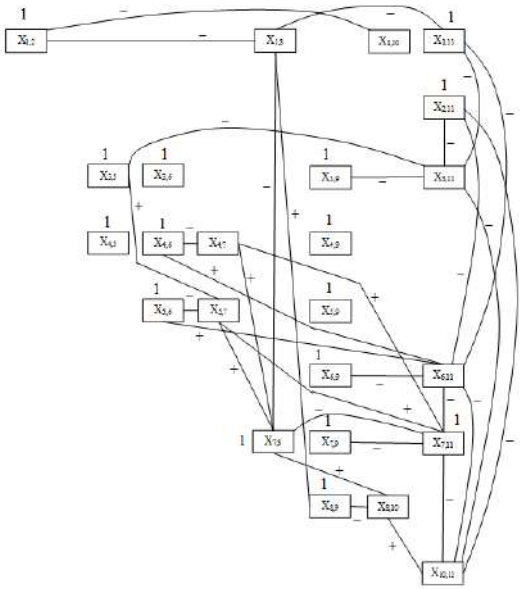

Figure 5. Conflicts graph G_D^x for logical equation $L1$

$$Q_D^x=\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & x & 0 & x & x & x & 1\\
0 & 0 & x & 0 & 0 & x & 0 & 0 & x & 0 & 1\\
0 & \overline{x} & 0 & 0 & 1 & 1 & x & 0 & 1 & 0 & x\\
0 & 0 & 0 & 0 & 1 & 1 & x & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & x & 0 & 1 & 0 & 0\\
\overline{x} & \overline{x} & 0 & 0 & 0 & 0 & x & 0 & 1 & x & x\\
0 & 0 & 0 & \overline{x} & \overline{x} & \overline{x} & 0 & 1 & 1 & 0 & 1\\
\overline{x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x & 0\\
\overline{x} & \overline{x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x\\
\overline{x} & 0 & 0 & 0 & 0 & \overline{x} & 0 & \overline{x} & \overline{x} & 0 & x\\
0 & 0 & \overline{x} & 0 & 0 & \overline{x} & 0 & 0 & \overline{x} & \overline{x} & 0
\end{bmatrix}$$

Figure 6. Updated matrix Q_D^x

VII. RESULTS

We have used the proposed model and technique of optimizing the concurrency level of network schedules as the basis of a graph language and a tool for development and execution of network algorithms. The graph vertices correspond to tasks. The graph edges correspond to data dependences between the tasks, and correspond to a control flow that implements constraints on resources. Tokens mark edges of the graph. The graph operates over firing vertices and moving tokens from input to output edges. The vertex firing can be conditional and cyclic, thus representing the behavior of all type of control structures in concurrent algorithms. The optimization of the concurrency level of a network schedule changes algorithm graph by adding or removing edges and tokens. Such a procedure of transformation is capable of taking into account the amount of available computational resources, while preserving the mapping of the input data onto the output data.

VIII. CONCLUSION

This paper has represented the concurrency level of a computing network schedule in an artificial intelligence system with

No	1	c	1	2	2	1	2	1	1	1	1	p1	p2
1	0	1	0	0	0	0	0	0	1	1	0	1	1
2	0	1	0	0	0	1	0	1	0	0	0	1	1
3	1	0	0	0	0	0	0	0	1	0	0	1	1
4	0	1	1	0	0	0	1	0	0	1	0	2	0
5	0	1	1	1	0	0	0	0	0	1	0	2	1
6	0	1	1	1	0	0	0	1	0	0	0	2	1
7	1	0	0	0	0	1	1	0	0	0	0	2	1
8	0	0	0	0	0	0	0	0	1	0	1	1	0
9	0	1	0	0	0	1	1	0	0	1	0	2	1
10	1	0	1	1	0	0	0	0	0	0	0	2	1
11	1	0	1	0	0	0	1	0	0	0	0	2	1
12	0	0	0	1	0	0	0	1	0	0	1	2	1
13	0	0	0	0	1	0	0	1	0	0	1	1	1
14	0	1	0	0	1	0	0	0	0	1	0	1	1
15	0	1	0	0	1	0	0	1	0	0	0	1	1
											max=	2	1

Figure 7. Updated clique set V_D for matrix Q_D^x

No	1	c	1	2	2	1	2	1	1	1	1	T
1	0	0	0	0	0	1	1	0	1	0	1	220
2	1	0	0	0	0	1	0	0	1	1	1	200
3	1	1	0	0	0	1	0	0	1	0	1	160
4	0	1	1	0	0	1	0	0	1	0	1	160
5	1	0	0	0	0	0	0	1	1	1	0	160
6	0	0	0	1	1	1	1	0	1	0	0	380
7	0	0	0	0	0	0	1	1	1	0	0	180
8	0	0	1	0	1	1	0	0	1	0	0	220
											max=	380

Figure 8. Updated clique set $U_{\overline{D}}$ for matrix Q_D^x

a set of pairs of tasks executed in parallel. We have estimated the execution time and implementation cost of the schedule over the clique set of the concurrency graph and over the clique set of the complement graph. Our optimization algorithm minimizes either the schedule execution time or resources. It finds the optimal concurrency level of network computations by solving the schedule existence problem at each step of updating the set of pairs of concurrent tasks.

REFERENCES

[1] Ben-Nun, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis / T. Ben-Nun, T. Hoefler // ETH Zurich, Department of Computer Science, Zürich. – 2018, 8006, Switzerland, pp. 1-47.

[2] Lee, W-P. Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment / W-P. Lee, Y-T. Hsiao, W-C. Hwang // BMC Syst Biol. – 2014, Vol. 8, No. 5, doi:10.1186/1752-0509-8-5.

[3] Qiao, S. A. Fast Parallel Community Discovery Model on Complex Networks Through Approximate Optimization / S. Qiao, N. Han, Y. Gao, R-H. Li, J. Huang, J. Guo, L. A. Gutierrez, X. Wu // IEEE Trans. On Knowledge and Data Eng. – 2018, Vol.30, No. 9, pp. 1638-1651.

[4] Lattanzi, S. Distributed Graph Algorithmics: Theory and Practice / S. Lattanzi, V. Mirrokni. // Proc. Eighth ACM Int. Conf. on Web Search and Data Mining – WSDM'15, Shanghai, China, 2015, pp. 419-420.

[5] Sakr, S. Large-Scale graph processing using Apache Giraph / S. Sakr, F. M. Orakzai, I. Abdelaziz, Z. Khayyat. // Springer, 2016. – 197 p.

[6] Khayyat, Z. Mizan: Optimizing Graph Mining in Large Parallel Systems / Z. Khayyat, K. Awara, H. Jamjoom, A. Alonazi, D. Williams, P. Kalnis // Proc. 8th ACM European Conference on Computer Systems. – ACM, 2013 pp. 169-182.

[7] Prihozhy, A. A. Analysis, transformation and optimization for high performance parallel computing / A. A. Prihozhy. – Minsk : BNTU, 2019. – 229 p.

[8] Prihozhy, A. Techniques for optimization of net algorithms / A. Prihozhy, D. Mlynek, M. Solomennik, M. Mattavelli // Proc. Int. Conf. PARELEC'2002. – IEEE CS, Los Alamitos, California, 2002, pp.211-216.

[9] Prihozhy, A. Net scheduling in High-Level Synthesis / A. Prihozhy // IEEE Design and Test of Computers, 1996, No.1, pp.26-35.

Оптимизация уровня параллелизма сетевых вычислений в системах искусственного интеллекта

Прихожий А. А.

Последние достижения в системах искусственного интеллекта невозможны без использования распределенных вычислительных платформ. В статье рассматривается графовая модель сетевых планов, а параметры планов оценивается посредством клик на графах предшествования и распараллеленности. Предлагается алгоритм пересчета клик при изменении уровня параллелизма вычислений посредством добавления или удаления ребра в графе. Для найденного множества пар распараллеленных задач решается проблема существования плана. Алгоритм распараллеливания минимизирует либо время выполнения плана, либо используемые ресурсы. Предлагаемые модель и алгоритм реализованы в программном обеспечении.

Received 23.12.2019