Algorithms and software used in selecting structure of machine-training cluster based on neurocomputers

V A Romanchuk, V V Lukashenko

Ryazan State University named after S.A. Yesenin, 46, Svobody Ave., Ryazan, 390000, Russia

E-mail: v.romanchuk@rsu.edu.ru

Abstract. The technique of functioning of a control system by a computing cluster based on neurocomputers is proposed. Particular attention is paid to the method of choosing the structure of the computing cluster due to the fact that the existing methods are not effective because of a specialized hardware base - neurocomputers, which are highly parallel computer devices with an architecture different from the von Neumann architecture. A developed algorithm for choosing the computational structure of a cloud cluster is described, starting from the direction of data transfer in the flow control graph of the program and its adjacency matrix.

1. Introduction

In the modern scientific and production sphere, the problem of using distributed data processing systems is quite common due to insufficient computational resources for solving a number of problems. At the heart of any system of distributed computing there is the model of computation. The model of computation creates the connection between the architecture and the programming model in computer systems, and also reflects the interaction of processes in them. But at the present time for the classified technologies of distributed computing, universal models are not built, which is explained by the strong dependence of the computation model on the architecture of the distributed system.

The paper considers a model of computation of a computerized cluster of machine learning based on neurocomputers, which has a software implementation in the form of a distributed computing cluster management system.

2. Methodology of the state of the cluster management system

For the sequential development of the described system, a technique for the operation of the control system of a computerized cluster of machine learning is proposed. Figure 1 shows the sequence of steps in this technique.

The input data are:
- Distributed computing cluster based on neurocomputers ($G$) [2, 3].
- The program code for execution on a cluster, theoretically regarded as an $A_{t(J)}$ algorithm [1].

Also at the second stage, the input data are:
- A set of computing neurocluster structures $Str = \{Str_1, Str_2, ..., Str_n\}$ [1].

Let us consider each stage of the developed technique:

1. At the first stage, algorithm $A_{t(J)}$ is considered as a set of operations...
\( Op = \langle Op_1, Op_2, ..., Op_p, ..., Op_p > \) that are related to each other by the relations of sequence and parallel operations. The relation of the sequence of operations is called the binary relation of two operations \( Op_1 \) and \( Op_2 \) when the data necessary for performing such operations are called by each of these sequentially, denoted by \( Pos \) [3, 4].

The relation of parallelism of operations is called the binary relation of two operations \( Op_1 \) and \( Op_2 \) when the data necessary for performing operations are available initially, do not depend on the results of operations, and are denoted as \( Par \).

2. At the second stage, the \( Al^{(j)} \) algorithm is put in correspondence with the \( PR^{(j)} \) program for its implementation and for the set of operations - subroutines-micro-programs \( PR^{(j)} = \langle MK_1, MK_2, ..., MK_i, ..., MK_f > \). Thus, a breakdown of the program code into atomic subroutines is described, which is realized in the algorithm for partitioning the tasks that were submitted for execution to the cluster for subtasks. Simultaneously, suitable computing neurocluster structure \( Str \) is selected from a variety of possible structures \( Str = \{ Str_1, Str_2, ..., Str_w, ..., Str_W \} \). The algorithm for selecting and forming the computational structure of a cluster of neurocomputers is responsible for the implementation of the described process.

3. At the third stage, subprograms \( PR^{(j)} = \langle MK_1, MK_2, ..., MK_i, ..., MK_f > \) are distributed among the nodes of the chosen computing structure of the cluster. The result of the distribution of subprograms \( \{ RO^{(j)} \} ; i = \overline{1,L} \) among the nodes of the selected computing structures \( Str_{Ri} \in Str; w = \overline{1,W} \) is the set of sets of parallelism relations and the sequence of subroutine execution: \( \bigcup_{i=1}^{N} Par_i \parallel Pos_i \).

This process is carried out by switching packets \( Pack_i = \{ Pack_1, ..., Pack_i \} , i = \overline{1,N} \) containing the necessary commands and data for computations; it also implements the algorithm for loading subtasks into the computational structure of a cluster of neurocomputers.

4. At the fourth stage, the calculation process itself takes place, i.e. the execution of subprograms on the computing cores of neurocomputers - neuroprocessors. This process is complicated by the fact that, unlike traditional architecture processors, neuroprocessors are capable of processing in a single clock cycle a number of operations, for example the NM640x family of neuroprocessors up to 64 operations [4]. The parameter responsible for this process \( BitVal \) (data bit rate) is fundamental for describing the features of architectural differences of neuroprocessors. It is also necessary to take into account the situation when the bit depth of the result exceeds the bit depth of the input data - overflow. Then the data with bit capacity \( BitVal = (\{ flag.n, ..., \{ flag.k \} ) \), where \( n, ..., k < 64 \) is the bit depth of the data and \( flag = 1 \cup 0 \) is the value of the over-flow flag, submitted for execution on the neuroprocessor must load it completely for efficient use of the entire computer system resources.

Control of the functioning of the calculation process is carried out by the cluster management system and implemented in the algorithm for managing calculations in the cluster.

5. At the fifth stage, the results of operation \( Rz_i = \{ Rz_1, ..., Rz_i \} , i = \overline{1,N} \) of the cluster computing structure are sequentially collected by the cluster manager and the cluster management system. This process should occur immediately after the 4th stage and with the direct control of the functioning of the calculations. To implement the operation of the cluster management system, the algorithm for collecting and returning the final computational result from the cluster structure operates at the fifth stage.

3. Choice of cluster structure

Let us consider the stage of choosing the structure of the computing cluster of machine learning in connection with the fact that existing methods are not effective because of a specialized hardware base - neurocomputers, which are highly parallel computing devices with an architecture different from the von Neumann architecture.
At the entrance of the stage, there is a program that can be represented from the point of view of compilation theory in form $Pr^{(j)} = (Gpu, \Omega)$, where $Gpu = (V, E, start, stop)$ is the control flow graph of the program, and $\Omega = \{w_1, \ldots, w_j\}$ is the alphabet of the operators.

The graph of the control flow is an oriented graph in which two start and stop vertices are identified, linked together by sets of vertices $V$ and arcs $E$, respectively, and satisfying the following requirements:
- no arcs are included in the start;
- no arcs leave the stop;
- any vertex is reachable from start;
- the vertex stop is accessible from any vertex.

Thus, after the presentation of the program is convenient for the computer, for example, in the form of an inverse Polish record, it is necessary to build a program control flow graph that will display a multitude of all ways of executing the program. The representation of the program in the extended reverse Polish record is stored and transmitted in the form of an abstract data type of the stack class. We will call such a stack a stack of program commands. Let us form the abstract syntactic tree (AST) of the program from the stack of program commands. An abstract syntax tree is a finite, labeled, oriented tree in which the inner vertices are labeled by the operators of the programming language, and the leaves - by the corresponding operands. The leaves of the lowest layer are input parameters, and the root of the tree is the result that the program returns as a result of its work. For this, let us use the following algorithm. Let us store the current readable symbol in $R$.

If $R$ is an identifier or a constant, its value is selected from the stack and becomes an AST sheet, then the next character is read.

If $R$ is a binary operator, then it is applied to the next two operands from the stack; thus, the AST sub-tree is formed, where the parent is a binary operator, and its children are operands. Then, the next character is read.

If $R$ is a unary operator, then it is applied to the upper consecutive character from the stack, thus, the AST sub-tree is formed, where the parent is a unary operator, and its sole successor is the operand.

It should be noted that all logical constructions of the language, such as "fork", "cycles", "conditional transitions", etc. in the expanded Polish record are described in the form of binary and unary operators, which allows making a transition from the program command stack to the AST.

The next step in constructing the control flow graph from the AST is the process of transmitting data over the AST. To do this, let us use the algorithm of the traversal of the tree. When the AST is reversed, a left sub-tree is traversed recursively, then the right sub-tree, and then the root, i.e. the node is visited after traversing all its sub-trees. At the AST nodes, there are binary and unary operators in the reverse Polish record, depending on their implementation, when transmitting data on the AST, the number of links in the sub-trees of the AST will change. Thus, the algorithm for converting the AST to the control flow graph of the program takes the form:

Let us read the next left sub-tree of the SDA, consisting of the lowermost left and right leaves, if any, and their roots, i.e. read the extreme operands of the lower layer and their operator.

If this is a binary arithmetic operator, a binary operator for specifying an array, a binary operator for accessing an array element, then the data operation is performed and the next right AST sheet (if any) and its root are read. The leaves and their root, connected by arcs, are entered into the adjacency matrix, which is represented by the program control flow graph.

If this is a unary unconditional junction operator, a binary conditional jump operator, then an arc is built from the operator to the target of the unconditional jump to the AST and the next right reading from the leaflet (if any) and its root is read. The sheet and the root, connected by arcs, are entered into the adjacency matrix, which is represented by the program control flow graph.

Thus, when new arcs arise, the AST actually loses the structure of the binary tree and is transformed into a graph. This is noteworthy in that the conversion occurs when data are fed into the AST. Therefore, it can be argued that the AST is transformed into a program control flow graph.
4. Cluster structure selection algorithm

To develop algorithmic support for the choice of the computational structure of the cluster, it is necessary to analyze the directions of data transfer in the flow control graph of the program by its contiguity matrix. The analysis of data flows in the control flow graph shows that the data can be distributed sequentially and in parallel. Those subprograms obtained as a result of the algorithm of partitioning the program for subprograms into a cluster can be located with each other in binary relations of sequence $Pos$ and parallel $Par$ operation with data.

The relation between the sequence of programs is called the binary relation of two subprograms $RO_1$ and $RO_2$ when the data necessary for the operation of subroutines appear and are required for each subprogram sequentially, one after another.

The relation of parallel execution of subprograms will be called the binary relation of two subprograms $RO_1$ and $RO_2$ when the data necessary for the operation of subroutines are available initially, do not depend on the results of subroutines and denote $RO_2$.

Step 1. Read the next row of the adjacency matrix of the graph of the control flow of the program and write it into an array.
- If only one element of the array is 1, then this is a string describing the next vertex of the graph that is an operand. Write the operand that marks the row of the adjacency matrix in the queue.
- If two elements of the array are equal to 1, then this is a string describing the next vertex of the graph that is a unary operator. Let us write a unary operator that marks the row of the adjacency matrix in the queue.
- If three elements of the array are equal to 1, then there is a string describing the next vertex of the graph that is a binary operator. Let us write a binary operator that marks the row of the adjacency matrix in the queue.

Step 2. Read the next element of the queue.
- If this element is an operand, then let us read the next element.
- If this element is a unary operator, then let us remove the previous operand and the unary operator from the queue. The resulting construction is called an independent operation, let us write it into a separate vector. Let us assign the next serial number to the generated vector, consisting of their two numbers $N_j \in N$, where $N_j = 1, N$ is the coefficient of the sequence of the operation.
- If this element is a binary operator, then delete the previous two operands and the binary operator from the queue. The resulting construction is called an independent operation, let us write it into a separate vector. Let us assign to the formed vector the next serial number, consisting of their two numbers $N_j \in N$.
- If there are no operands in front of the operator in the queue, let us write to the vector, instead of the operands, references to the previous ones, obtained as a result of the vector algorithm’s operation. Thus, the resulting construction is called a dependent operation, since its execution depends on the performance of previous independent operations. Let us assign the following number also including their two numbers: $N_j \in N$ to the formed vector.

Next, in all cases, let us read the next element of the queue.

Step 3.
- Independent operations can be performed in parallel. In this case, let us say that the subroutines corresponding to operations are in relation to parallelism and choose the vector structure of which these subroutines will be members.
- Dependent operations can be performed only sequentially, passing the results from the vector to the vector. In this case, one can say that the subroutines corresponding to the operations are in relation to the sequence and choose the pipeline structure, of which these subroutines will be members.

Successively moving from a vector to a vector, realizing a combination of them depending on the relationship they are in, let us obtain the computational structure of the cluster necessary for the task to be performed on a cluster of neurocomputers.
Practical studies are implemented according to the developed methodology and algorithmic support and can be represented as a subsystem "Logical design of neuroprocessor systems" based on the "NP Studio" platform.

5. Conclusion
The procedure for the functioning of the computer cluster management system is based on a conceptually new generation of computer technology - neurocomputers and the algorithmic software developed to select the computational structure of a cloud cluster based on neurocomputers. It is based on the direction of data transfer in the control flow graph of the program and its contiguity matrix.

6. Acknowledgments
The research was carried out within the framework of the assignment for the performance of public works in the sphere of scientific activity within the framework of the initiative scientific project of the state task of the Ministry of Education and Science of the Russian Federation, No. 2.9519.2017 / BC on the topic "Technologies for parallel processing of data in neurocomputer devices and systems".

References
[1] Pande S, Morgan F, Cawley S, Bruintjes T, Smit G, McGinley D, Carrillo S, Harkin J and McDaid L 2013 Modular neural tile architecture for compact embedded hardware spiking neural network Neural Process Letters 38 131–153
[2] Izeboudjen N, Larbes C and Farah A 2014 A new classification approach for neural networks hardware: from standards chips to embedded systems on chip Artif Intell Rev 41 491–534
[3] Kolinummo P, Pulkkinen P, Hämäläinen T and Saarinen J 2000 Parallel implementation of self-organizing map on the partial tree shape neurocomputer Neural Processing Letters 12 171-82
[4] Ruchkin V, Romanchuk V and Sulitsa R 2013 Clustering, restorability and designing of embedded computer system based on neuroprocessors Proceedings of the 2nd mediterranean conference on embedded computing (Montenegro: Budva) pp 58-62
[5] Ruchkin V, Fulin V, Kostrov B, Romanchuk V and Ruchkina E R 2015 Parallelism in embedded microprocessor system based on clustering Proceedings of the 4nd mediterranean conference on embedded computing ed D Jurisic, R Stojanovic, L Jozwiak (Montenegro: Budva) pp 45-50
[6] Avetisyan A, Grushin D, Ryzhov A 2002 Systems of cluster management Proceedings of the Institute for System Programming of Russian Academy of Sciences 140-147
[7] Ivutin A, Larkin E 2014 Estimation of latency in embedded real-time systems Proceedings of the 3rd mediterranean conference on embedded computing ed L Jozwiak, D Jurisic, R Stojanovic (Montenegro: Budva) pp 236-239
[8] Diken E, Jordans R, Corvino R, Jozwiak L, Corporaal H and Chies F 2014 Construction and exploitation of VLIW ASIPs with heterogeneous vector-widths Microprocessors and Microsystems 38 947–959
[9] Jordans R, Jozwiak L, Corporaal H 2014 Instruction set architecture exploration of VLIW ASIPs Using a genetic algorithm Proceedings of the 3rd mediterranean conference on embedded computing ed L Jozwiak, D Jurisic, R Stojanovic (Montenegro: Budva) pp 32-35
[10] Bekhtin Y, Gurov V, Guryeva M 2014 Algorithmic supply of IR sensors with FPN using texture homogeneity levels Proceedings of the 3rd mediterranean conference on embedded computing ed L Jozwiak, D Jurisic, R Stojanovic (Montenegro: Budva) pp 252-255