UDC 681.3.07

Vitaliy D. Pavlenko[1], Doctor of Technical Sciences, Professor, Professor of the Department of Computerized Control Systems, E-mail: pavlenko_vitalij@ukr.net, ORCID ID: 0000-0002-5655-4171
Sergey V. Pavlenko[1], Candidate of Technical Sciences, Senior Scientist at the Department of Computerized Control Systems, E-mail: psv85@yandex.ru, ORCID ID: 0000-0002-9721-136X
[1]Odessa National Polytechnic University, Shevchenko Avenue, 1, Odessa, Ukraine, 65044

ORGANIZATION OF COMPUTATIONS IN CLUSTERS USING TRANSPARENT PARALLELIZING PRINCIPLES

Abstract. *Methods of consructing of the systems identification and recognition requirements significant computational resources and therefore require usage of parallel systems, such as clusters or computers with multiple processors or processors with multiple cores. In this paper cluster computing organization principles based on transparent parallelizing are considered. Questions that arise while implementing this technology as a parallel calculations framework are described. Described technology has been implemented as a framework on Java programming language. Architecture of such framework is shown and functionality of its parts is described. The concept of a value ID and the concept of an unready value ID have been proposed to implement the proposed principles. The ID of a value is an ID that should be assigned to each value that is used as input or output parameter of procedure. These assignments are cluster-wide and are used to replace sending parameter value with sending its ID. The same values are often used in different calls in parallel programs, so using IDs allows the framework to save traffic. IDs of unready values are created each time during a procedure call and are assigned to the output parameters of the procedure. They are used to get the value of parameter in the moment of the first access. Also they are passed to the server as a part of information about an order. When the execution of an order is finished, value IDs are obtained for values of output parameters of the order and these IDs are assigned to the corresponding IDs of unready values. RMI technology has been used to implement communication between server and clients. Also JDBC has been used to implement storing of final and intermediate computations results to external database. In this paper is to propose method of execution time characteristics analysis for parallel applications that have been created using the technology of orders based transparent parallelizing. Its efficiency has been proven by solving the problem of determination of diagnostic value of formed features diagnostics on a cluster of 2, 3, 5 and 10 computers. The result of multiplication of execution time by number of processors has grown by not more than 1.13% when using 2, 3 or 5 computers instead of one, and by not more than 3.25% when using 10 computers instead of one during this experiment. The closest analogue of offered approach is T-system.*

Keywords: *Parallel computing; Cluster computing; Software development; Frameworks; Transparent parallelizing*

Introduction. Parallel computing is the subject of many researches nowadays because of its practical importance. The main reason for it is large volume of problems that cannot be solved in acceptable time without utilization of parallel computing [1-9]. For example, these are the problem of Volterra series based nonlinear dynamic systems models identification [10-14], problem of full scan based comparison of features diagnostic value, modeling problems and so on [15]. Despite the simplicity of the idea of parallel computing, this field of science has to solve many problems, including problems of creating efficient parallel algorithms, introducing all kinds of parallelism to hardware, making software and hardware fit each other [16]. Processors of modern PCs have multiple cores, so possibility of parallel data processing has to be taken into consideration by software developers more and more often.

One of the problems related to parallel computing is the problem of creating software development tools that allow developers to create efficient parallel applications without significant effort. There are many approaches for creating parallel software, including manual, automatic and semi-automatic parallelizing. The technology of orders based transparent parallelizing has been proposed in [17-26]. It is based on the idea of introducing large groups of algorithms that can be parallelized in similar way, creating skeletons of such algorithms and implementing efficient parallel implementations of these skeletons. Parallelizing has to be done only once and can later be used for many algorithms, so it's possible to implement complex logic of parallel execution, including fault tolerance logic, load balancing logic, logic for adding and removing nodes during computations and so on. The current implementation of this technology includes implementation of only one method of parallelizing that introduces "call-by-future" semantics to imperative programming language and was designed to run on clusters. The clusters were chosen as destination architecture because of their high scalability and relatively low cost (for example, 82% of computers in November 2011 release of Top500 list are clusters [27]).

This paper is devoted to implementation of technology of orders based transparent parallelizing [19-23] as a cluster computing framework. Decisions on architecture of the framework are described and questions of its usage are being discussed.

© V. Pavlenko; S. Pavlenko; 2019
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/deed.uk)

1. Existing technologies of parallel applications development. There is one general tendency about modern technologies and software for development. An attention is paid not only to traditional requirements (such as efficiency of created applications), but also to the requirement about high speed and low labor intensiveness of software development. It seems that this tendency is caused by low cost of computer work time and high cost of programmer work time. However, this tendency did not affect parallel computing technologies much. It seems to be caused by big cost of parallel computers work time. High cost of parallel computer work time seems also to be the reason of popularity of low-level technologies that give the programmer more control over the computer and allow programmer to minimize program execution time while the time and the labor intensiveness of program development are not so critical. A similar situation can be observed in area of distributed computing where mainly low-level tools are being developed nowadays.

Therefore, the purpose of this approach is creation of parallel computing technology that follows the requirements:

– High level of technology. It is a well-known situation in the history of programming when some features have been abandoned to get some advantages. For example, "go to" operator has been abandoned to make source understanding easier. So this technology should not provide low-level operations (such as sending and receiving messages) to user, but the set of provided high-level operations should be enough for development of efficient parallel applications. This requirement should make parallel applications development much faster and easier.

– Transparency of parallel architecture. It is much easier to think about writing a program for one processor, so the technology should hide parallel architecture from user where possible.

– Efficiency of the technology. Overhead, caused by the technology, must be minimal. In addition, the technology must enable user to create efficient parallel implementations for wide enough class of applications.

– High speed and low labor intensiveness of parallel applications development. It also means high speed and low labor intensiveness of porting existing applications.

2. **Technology of orders based transparent parallelizing.** Transparent parallelizing technology is based on splitting of parallel algorithms and the means of their parallelizing. Its main idea is finding large groups of algorithms that can be parallelized in similar way, introducing template of parallel algorithm and implementing parallel version of the template. It has to be done only once, so we can implement some advanced features in such parallel algorithm, such as handling communication failures, monitoring tools, tools for adding computers to cluster while computations are in progress and removing them and so on. Once such parallel template is implemented, we can easily run any algorithm that fits the template.

Only one algorithm template has been implemented for now. Let us assume that we have selected some set of procedures in the program. Each procedure should not modify any data during execution except values of parameters and temporary (and inaccessible outside the procedure) data structures. Each parameter of each selected procedure should be passed by value. Execution of program must mean execution of certain selected procedure. This assumption imposes some limits on program. For example, it forbids using global variables or I/O devices. However, it is shown below that these limits can be loosened. For example, work with global variables and I/O devices can be allowed is some specific requirements are met.

The example which will be used to illustrate the technology is shown on Fig. 1.

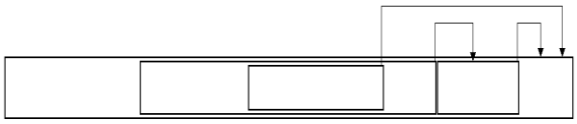


Fig. 1. Illustration of a part of program

We show procedure execution time with a rectangle (time goes from left to right). Called procedures are shown by nested rectangles. Lengths of rectangles and their parts are proportional to the execution time of corresponding program parts. It is considered that all input parameters are already known now of program execution start. Lines connect moments of getting some values and moments of their first usage.

The first principle of offered technology introduces the concept of an order as the minimal unit of work that should be executed on one computer and cannot be split into smaller parts. Such a unit of work is defined as execution of one procedure without execution of procedures it calls. Each procedure call creates a new order that should be executed by some computer of cluster (let's call such call making of order). One of selected procedures should be marked as main one to define program entry point.

This principle is illustrated on Fig. 2. It is considered that different processors execute four orders and their execution starts immediately after making corresponding order. Extra lines connect parts of one procedure.

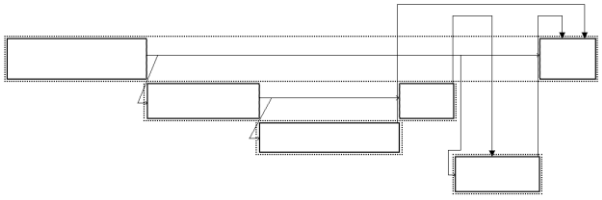


Fig. 2. An illustration of the first principle of offered technology

Many algorithms contain intervals of time between moments of getting some values computed and moments of first usage of these values; it is often possible to make such intervals bigger by some changes in computations order. If there are no such intervals in some algorithm that means that each operation should not be executed before previous one is over, so we cannot create parallel implementation of this algorithm at all. If we perform procedure call in common programming languages, caller procedure continues its execution only after called one is over. In other words, we can say that caller procedure starts waiting for output parameters of called procedure in the moment of call and stops waiting in the moment when called procedure finishes its execution. The second principle proposes to continue execution of caller procedure after a call and to start waiting only in the moment of first request to output parameters of called procedure. If called procedure execution is already over in the moment of such request, we should not start waiting at all.

This principle is illustrated on Fig. 3.

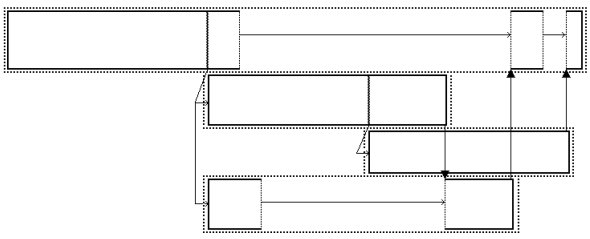


Fig. 3. An illustration of the second principle of offered technology

This diagram can be built from previous one by maximal possible left shift of all computations that keeps the following requirement met: each value is used only after it is got.

Please refer to [28-37] for more information about transparent parallelizing technology.

3. Computing simulation method. The main idea of the proposed method is described as follows: we can lower the time of test run of parallel application by skipping some computations that are going to take much time. Two conditions should be met for a block to make it possible to skip it. In the first place, estimation of execution time of such block should be known. In the second place, it should be possible to continue the test run without knowing the values that are normally computed in the block. We can warrant that by assuming that values, computed in the block, are not used in any conditional operators.

For instance, we can skip blocks with asymptotic execution time known. If we know asymptotic execution time for a block, it means that we know a function $f(\alpha)$ so that

$$\exists c_1, c_2 > 0, \forall \alpha \in A: c_1 f(\alpha) \le T(\alpha) \le c_2 f(\alpha),$$

where: $T(\alpha)$ is time of execution of the block, α is a value representing input parameters of the block, A is the set of all possible values of input parameters of the block. Function $f(\alpha)$ is usually a function of a few numerical characteristics of parameters of the block – such as length of an array or number of vertexes in a graph. After defining $c(\alpha) = \frac{T(\alpha)}{f(\alpha)}$ we can rewrite the definition of asymptotic estimation in the following form:

$$\exists c_1, c_2 > 0, \forall \alpha \in A: c_1 \le c(\alpha) \le c_2 .$$

If we run the block for a few times on one computer, we will be able to compute values $c_i(\alpha), i = \overline{1, N}$ (N is the number of runs). We can take $\bar{c}(\alpha) = M\{c(\alpha)\}$ as an estimation of $c(\alpha)$ and use it to compute the estimation of $T(\alpha)$. This estimation can be used only for the computer where it has been computed and for the ones with identical hardware and software. In order to get such estimations for other computers of the cluster, we can either repeat test runs of the block on other computers of the cluster or to use time of execution of some sample algorithm as the unit of time.

The proposed method consists of four stages. During the first stage user marks a set of blocks in the source of the program. Each marked block should meet the following requirements: estimation of block execution time should be known and it should be possible to compute it quickly; values, computed in blocks, should not be used in any conditional operators; blocks should not make any calls or requests for data; blocks should not be nested; user should implement alternative version of each block that works as quickly as possible and

makes all further computations work properly. Each marked block should be surrounded with sending notifications to computing support environment about block execution time and choosing the implementation of the block to be executed. For instance, if a block multiplies two *n*-by-*n* matrices by definition, it should declare execution time *n*n*n* and should have alternative implementation that creates a new *n*-by-*n* matrix without its initialization. In addition, programmer has to create a set of tests that run every marked block at least once.

During the second stage, the tests are run in order to compute values $\bar{c}(\alpha)$ for marked blocks. In order to minimize the influence of random variations of execution time, each test has to be repeated for a few times. A sample of result of execution of the second stage for a block is shown on Fig. 4:

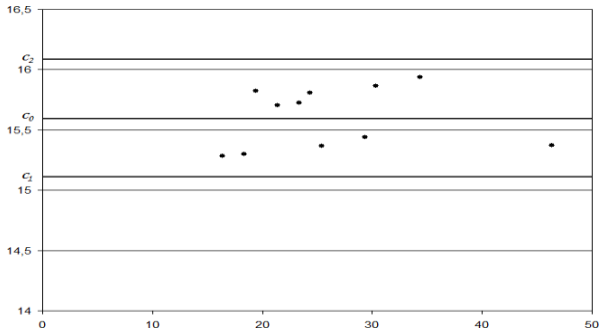


Fig. 4. Constant in asymptotic estimation of execution time

During the third stage a test run of the program is done. Alternative implementations of marked blocks are used, and execution time of blocks is considered to be equal to its estimation. If execution of marked blocks takes almost all the time of a real run of the program, we can make a test run fast enough. It has to be done either on every computer of the cluster, or only once if time of execution of a sample program is used as the unit of time. After the third stage we have the following information about each executed order: duration, moments of data requests, moments of providing data. A sample result of third state execution is shown on Fig. 5.

On the fourth stage simulation of parallel application execution is performed. Only information about orders, gathered on the third stage, is used. After the fourth stage we have information about the load of the cluster and scheduling-related events that happened during the simulation. This information can be used to find program execution time and to find information about its bottlenecks.

Each stage uses only the results of previous stages and some specific information about the parallel application. First two steps use the source of the application; third one uses program input data and the fourth one uses information about the cluster and scheduling algorithm. Splitting the method into a set of stages makes results re-using in repeated experiments possible. For instance, user can repeat only the fourth stage to find the configuration of cluster if program execution time is limited.

4. Implementation of transparent parallelizing technology. Transparent parallelizing technology has been implemented as a cluster computing framework. This section describes architecture of the framework and main decisions made during implementation.

Architecture of the framework is shown on Fig.6.

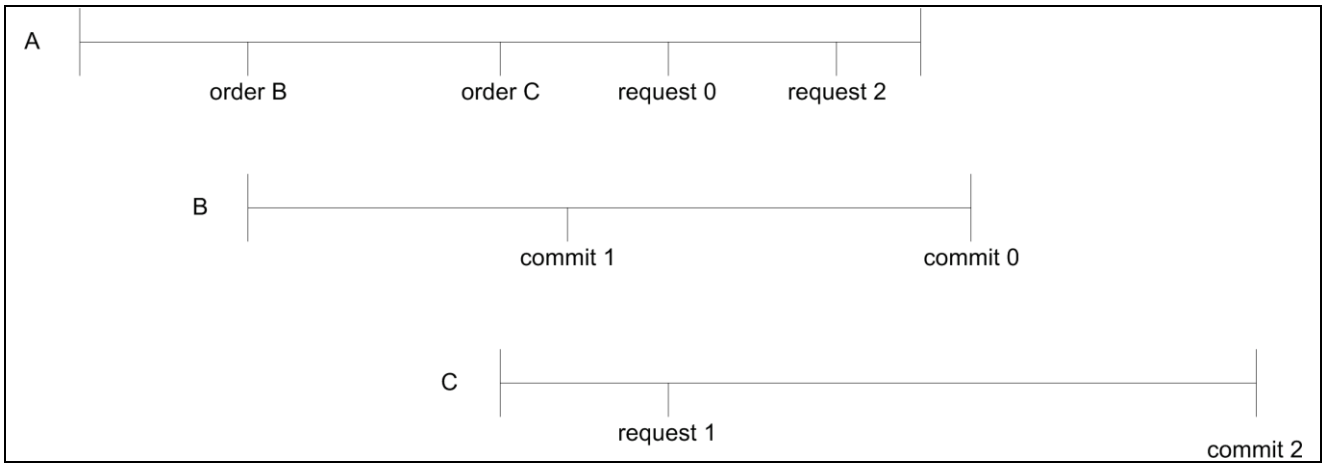


Fig. 5. Sample result of the third stage of method analysis of time characteristics parallel programs

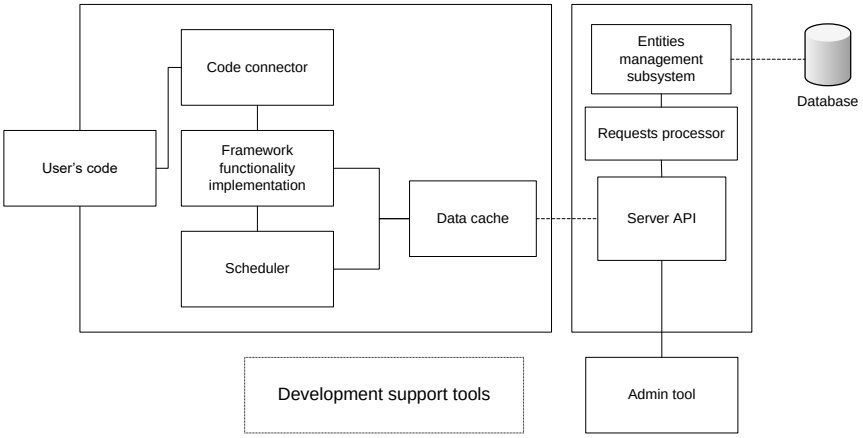


Fig. 6. Architecture of the framework

Described technology has been implemented as a framework on Java programming language. The framework consists of server part, responsible for storing intermediate data and providing tasks to clients and clients that are responsible for performing computations. Having a server in the cluster makes recovery after failures and managing cluster easier: if a client gets down its tasks can be sent to another computer and if the server gets down it can be restarted and its state can be recovered from the database.

Implementation of client and server is based on manipulations with the following entities:

– **Data identifier.** It identifies a value of input or output parameter of a method. The identifier can obtain actual value of the parameter. If some data identifier is used in multiple calls, we can avoid sending actual value of parameter multiple times as it can be cached on clients.

– **Unready data identifier.** Procedures can use it as a "future" value that will be bound with data identifier or other unready data identifier later. Each time when a new order is created an unready data identifier is created for each of its output parameters. Parent order puts these identifiers in its local parameters and child order binds these identifiers after its execution is over.

– **Order.** It is a formal description of an order that includes information about method that should be executed, information about input parameters (their data identifiers or unready data identifiers) and information about output parameters (their unready data identifiers).

Code, written by user, interacts only with the "code connector" layer. This layer implements semantics of orders and their parameters. On one hand, it provides user code with access to small set of methods of framework (creating an order, obtaining data by unready data identifier and some utility methods); on the other hand, it represents user code to the rest of framework as a "black box" that can execute orders. There may be different implementations of this block that provide functionality of framework to user code in different manner or use different agreements on about user code.

Code connector does not provide functionality for creating unready data identifiers and getting data for them to user code directly, but does it by the means of DataHandler class. Classes, used as parameters of orders, should meet the following requirements:

– They should extend DataHandler class.

– They should implement methods for saving data to stream and restoring data from stream. Standard mechanism of serialization cannot be used as it creates new objects during deserialization and cannot update existing objects in place.

– They should provide access to their content only by the means of getters and setters that should can inherited methods: preRead() should be called before reading data from fields, preWrite() should be called before changing some fields, preReplace() should be called before replacing all fields within one call.

In addition, this layer provides methods for reading data from files located on server and writing messages to cluster-wide log file.

Framework functionality implements logic of threads management and provides functionality for management of data identifiers, unready data identifiers and orders, remote logging, accessing remote files and reporting failures to code connector.

Scheduler is a block responsible for choosing orders to be loaded from server or a ready order to be woken up. Currently only one scheduler based on naive planning heuristic that prefers continuation of execution of ready order to getting a new one from server.

The diagram of states of an order is shown on Fig. 7:

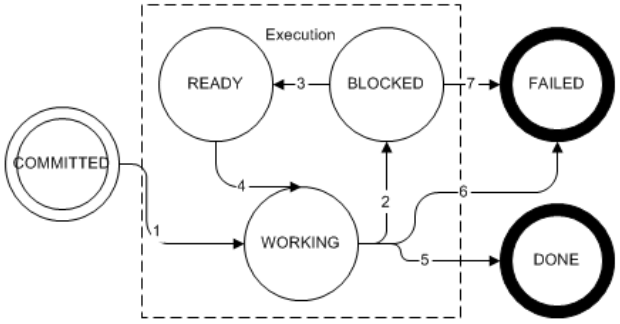


Fig. 7. Diagram of states of an order

The states mean the following:

– **COMMITTED** – the order has been made but it's execution haven't started yeta;

– **WORKING** – order is now being executed;

– **BLOCKED** – order execution has been stopped because of a request to not still computed data;

– **READY** – order execution has first been stopped because of a request to not computed data, but we already have the data we need;

– **DONE** – order execution has been successfully completed;

– **FAILED** – order execution has failed.

State changes can happen in the following situations:

1. Some computer of cluster gets the order.
2. Order tries to get not computed data.
3. Needed data has been computed.
4. Number of concurrently executed orders is less than number of processors of computer, so we can continue execution of that order.
5. Order execution has been successfully completed.
6. Order execution threw an exception.
7. Other method that had to compute data for current one has thrown an exception.

Data cache stores information about data identifiers and unready data identifiers to reduce traffic between client and server.

Communication between server and client is implemented by the means of RMI technology. Server exposes two interfaces – one for computations and one for administrative tool.

Administrative tool provides functionality for creating orders, viewing their status and accessing computations result. It is currently implemented as a Swing application.

Requests processor and entities management subsystem implement processing of requests from clients and administrative tool and provide methods that are used for scheduling. Data can be stored either in database or in memory.

Support tools developed that generates classes for sending orders according to description, provided by user [29, 35]. The description has to contain list of methods for sending orders and names of classes to generate. Four classes are generated: library class containing interfaces of methods for sending orders, its implementation that sends order, "local" implementation that just calls corresponding methods (it can be used for development) and executer class that is used by framework to call methods of the user's program [38, 39].

5. Practical usage of orders based transparent parallelizing framework. A distribution of the framework consists of the following files:

– parallel.jar – implementation of the framework;

– client.bat – sample file for running client part of the framework;

– server.bat – sample file for running server part of the framework;

– problem.bat – sample file for starting computations;

– sourcegen.bat – sample file for generating sources;

– serverconf.properties – server settings;

– lib\ – directory with libraries used by the framework;

– work\ – empty directory for input and output data of users' programs.

The distribution is used for both development of parallel applications and their execution. In order to parallelize an existing algorithm or to create a new parallel one, a developer has to do the following steps:

1. Find out whether the technology of order based transparent parallelizing is suitable for the specific algorithm. For instance, if a part of algorithm can be split into many independent parts, it can easily be parallelized unless the size of the parts is too small (and thus the overhead added by parallel computing support routines becomes significant).

2. Choose the methods that will be used for parallelizing. Such methods should meet the following requirements: they should have no side effects, they should only access data they received via parameters and should return data only by updating values of their output parameters, there should be routines in the program that call these methods and use the data computed by these methods only after doing some other computations. Refactoring may be required for such methods to appear. For instance, if there is a method that meets all requirements except for being static, then the fields that it uses may be added to its parameters.

3. Ensure that all chosen methods found above have return type “void”. If some method does not, then it should be changed by adding one more parameter to hold return value. This limitation has been introduced to avoid problems caused by lack of knowledge about exact class of an object that is returned by an order.

4. Ensure that chosen methods don’t use primitive types or arrays as parameters. If they do, create wrapper classes and replace usage of primitive types or arrays with usage of corresponding wrappers.

5. Make all classes used as types of parameters of chosen methods to be subclasses of DataHandler. All fields of these classes should be private; the classes should implement inherited abstract methods saveTo and loadFrom that should save values all fields to a stream and load values of all fields from a stream respectively; all methods that access their fields should call inherited method preRead, preChange or preReplace before performing such access; there should be a constructor calling inherited constructor with proper parameters.

6. Prepare an XML file with a list of all classes created on previous step and the list of chosen methods. For each method you should state the name of method for sending an order, name of the chosen method (including class name and package name), list of parameters with their names, types and directions (input / output / both), information on whether the chosen method accepts the library as a parameter.

7. Run source code generator. It will generate four files, including library, local library implementation, remote library implementation and executer. Library is an abstract class that includes methods for sending orders declared in the XML file. Local library implementation implements these methods by performing traditional execution of corresponding chosen methods (this class is useful for development and debugging) and remote library implementation implements sending orders. Executer provides information about methods and types to the framework.

8. Update the program by adding library parameter to the signatures of chosen methods where necessary and replacing calls of these methods with calls of methods of library. It is allowed not to change some calls if it will not speedup program execution.

9. Replace reading input data from console and/or local file system and writing output to console and/or local file system with reading and writing files located on the server respectively.

10. Create at least one static method that accepts library as a parameter and creates root order using it. These methods will be used as entry points of the program.

11. Compile the program and the generated files and pack the class files into a JAR file.

12. The list of methods of DataHandler class is given on Fig. 8.

Once a program has been created, a copy of the framework with the program should be created and distributed among the nodes of the cluster. The following changes have to be applied to the files of the framework:

– Classpath settings should be changed by adding JAR file of the problem in files problem.bat and client.bat.

– Fully qualified name of root method of the problem should be set in problem. bat.

– IP address or hostname of server should be set in client.bat.

Once framework has been distributed, the parallel application can be executed as follows:

– First the server has to be started using server.bat.

– After that the clients have to be started using client. bat. New clients may be added later at any time.

– Computations should be started using problem. bat.

One of advantages of described technology is high speed and low labor intensiveness of porting of existing non-parallel applications to that technology. Let us show that by parallelizing a program that solves the problem of features diagnostic value comparison based on full scan.

The initial program has the structure show on Fig. 9.

This algorithm does not contain significant intervals of time between computing of some values and their first usage. Let’s change the order of calculation in the following way: we will split all possible sets of features into some groups and find best set for each group independently. After that we will compare the results and find the best one. So we have to change the program as shown in Fig. 10.

Methods analyses Group and main meet the requirements described above (except for main reading data from file system and writing output to file system and can be used for parallelizing. Once the program is parallelized according to steps above, it can be run on cluster. Communication between nodes of three-node cluster during execution of this program is shown on Fig. 11.

```
public abstract class DataHandler {

    /**
     * Constructor
     *
     * @param canChangeSuddenly must be <code>true</code> if it may happen that some data in object is changed without
     * {@link #preChange} or {@link #preReplace} method call. Some optimizations are turned off in this case.
     */
    protected DataHandler(boolean canChangeSuddenly) {
                    ....
    }

    /**
     * This method must be called from methods of subclasses that want to get the real data from
     * this DataHandler. This method simply gets sure that this class already contains real data
     * inside (if it does not, the thread hangs up and waits for real data from server)
     */
    protected final void preRead() {
                    ....
    }

    /**
     * This method has to be called before replacing ALL real data
     */
    protected final void preReplace() {
                    ....
    }

    /**
     * This method has to be called before changing SOME real data
     */
    protected final void preChange() {
                    ....
    }

    /**
     * This abstract method has to de-serialize real data
     *
     * @param source DataInputStream to load data from
     * @throws IOException if I/O exception occured while loading
     */
    abstract protected void loadFrom(DataInputStream source) throws IOException;

    /**
     * This abstract method has to serialize real data
     *
     * @param dest DataOutputStream to save data to
     * @throws IOException it can't be thrown, but is declared to allow users not to catch I/O
     *                     exceptions from OutputStream's methods
     */
    abstract protected void saveTo(DataOutputStream dest) throws IOException;

}
```

Fig. 8. The list of methods of DataHandler class

```
class Algorithm {
    public static void main(String args[]) {
        //Read input data from file
        //Find averages of distributions of features for each class and build
        //      covariation matrixes
        //For each non-empty set of features:
        //  Build quadratic decision rule and calculate it's values using provided
        //        values  of features of objects of both classes
        //  Find maximal probability of right recognition for this rule.
        //  Save the result as the best one on the first iteration or compare it with
        //        current best result on any other iteration
        //Write result into file
    }
}
```

Fig. 9. The structure of initial program

```
class Algorithm {
   public static void analyseGroup(... /*data about group of sets, etc.*/,
                                    SetWrapper bestSet) {
      //Find best set and put it into bestSet variable
   }
   public static void main() {
      //Get input data from the file, placed on server
      //Find averages of distributions of features for each class and build
      //      covariation matrixes
      //For each group of sets:
      //  Analyze it using analyseGroup() method which should be executed in a
      //        separate order
      //For each group of sets:
      //  Get the result for the group and save it as the best one on the first
      //        iteration or compare it with current best result on any other
      //        iteration
      //Write result into the file, placed on the server
   }
}
```

Fig. 10. The structure of initial program

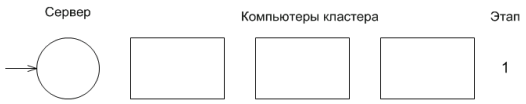
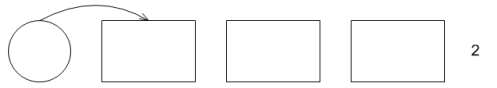
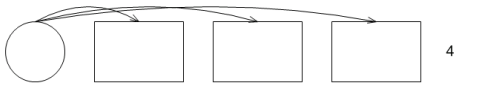
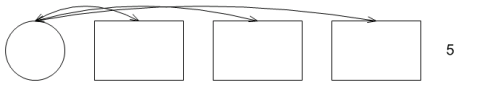
	Server gets an order for execution of *main* from user
	The root order is passed to one of the clients (let it be the first one)
	The client executes *main* and sends orders for execution of *analyzeGroup* to server.
	First computer finishes sending orders while other computers execute them. Finally, the first computer attempts to access result of analysis of the first group which hasn't been computed yet, so it has to suspend computations and start request a new order from server.
	All three computers execute *analyzeGroup* orders. Time to time the first computer may proceed with execution of *main* but stops again after a request for new data that is not computed yet.
	First computer ends execution of *main*.
	A file with computations result is created on server. Execution of *main* stops.
	User gets results of computation from server.

Fig. 11. Stages of execution of parallel application on cluster with 3 nodes

6. Testing of efficiency of offered technology

The problem has been tested on a computer with Intel Core i5 CPU M430 running at 2.27GHz in different configurations (this processor has 2 cores and supports Hyper-Threading) [36-37]. Ten runs were done for each configuration for the problem of dimension 24, their results are given below:

– Serial execution: 288 seconds min, 290 seconds max, 289 seconds in average.

– Parallel execution, 1 core: 292 seconds min, 296 seconds max, 293 seconds in average.

– Parallel execution, 2 cores: 180 seconds min, 185 seconds max, 182 seconds in average.

– Parallel execution, 3 cores: 142 seconds min, 145 seconds max, 144 seconds in average.

– Parallel execution, 4 cores: 134 seconds min, 140 seconds max, 137 seconds in average.

– Parallel execution, 2 cores, Hyper-Threading disabled: 153 seconds min, 157 seconds max, 154 seconds in average.

Another experiment has been conducted with a network of 1; 2; 3; 5 and 10 computers with Intel Pentium 4 1.7 GHz processors, connected with Fast Ethernet for problems with dimensions 23; 24 and 25. The dependence of execution time from the problem dimension and the number of computers is shown on Fig. 12.

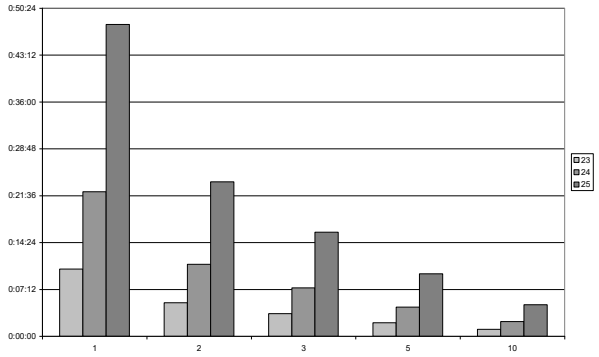


Fig. 12. Results of experimental testing of efficiency of framework

Result of multiplication of execution time by number of processors grows by not more than 1.13% when using 2; 3 or 5 computers instead of one, and by not more than 3,25 % when using 10 computers instead of one.

Conclusion. This paper describes implementing transparent parallelizing technology as a cluster computing framework on Java. The technology has been implemented as a framework that consists of server part, a client part, administrative tool and tools for development support. The efficiency of framework has been shown by solving the problem of determination of diagnostic value of formed features diagnostics on a cluster of 2; 3; 5 and 10 computers.

Conclusion. This paper describes implementing transparent parallelizing technology as a cluster computing framework on Java. The technology has been implemented as a framework that consists of server part, a client part, administrative tool and tools for development support. The efficiency of framework has been shown by solving the problem of determination of diagnostic value of formed features diagnostics on a cluster of 2; 3; 5 and 10 computers.

The closest analogue of the offered technology is the T-system that is being developed in Program Systems Institute of the Russian Academy of Sciences [40, 41]. Although the offered technology has been developed independently, its main principles are close to the main principles of the T-system [8]. The base concepts of the T-system are T-functions and unready values. A T-function is defined as some clean function. Any call of T-function is transparently replaced with a network call that means execution of method on another computer of cluster. An unready value is a variable which value is not currently known. Such values appear because of T-functions calls, and any attempt to get value of such variable causes getting its value from another computer with (possibly) waiting.

Main principles of these two technologies are close, so their problems should be close, too. Their main problems are caused by using of existing program splitting into procedures to find parts of code that should be executed in parallel. That may cause either get a lot of small orders (and big overhead for their management) or small number of big orders that cannot utilize whole cluster. The problem of small orders is partially solved in the offered technology by allowing user to call selected procedures locally. In addition, other methods of method calls optimization have been proposed to prevent getting big overhead.

References

1. Antonov, A. S. (2004). Parallelnoe programmirovanie s ispolzovaniem tehnologii MPI. [Parallel programming using MPI technology]. Moscow, Russian Federation, *Publishing House of Moscow State University*, 71p. (in Russian)

2. (2012). Osnovnyie klassyi sovremennyih parallelnyih kompyuterov. [The main classes of modern parallel computers]. [Electronic resource] – Available at : http://parallel.ru/computers/classes.html, Information analytical center on parallel computing. http://parallel.ru admin – 07.01.2012 – 15:53 (in Russian).

3. Roganov, V., Moskovsky, A., & Abramov, S. (2006). "The Open TS parallel programming system". *The Twelfth International Conference on Parallel and Distributed Systems, Minneapolis*, USA (ICPADS, July 12-15, 2006). – http://skif.pereslavl.ru/skif/index.cgi?module=chap&action=getpage&data=publications\pub2006\opents_ext.doc.

4. Abramov, S. M., Moskovskiy, A. A., Roganov, V. A., Shevchuk, Yu. V., Shevchuk, E. V., Paramonov, N. N., & Chizh, O. P. (September 19-24, 2005). Open TS: arhitektura i realizatsiya sredyi dlya dinamicheskogo rasparallelivaniya vyichisleniy. [Open TS: Architecture and Implementation environments for dynamic parallelization of computations], *Scientific service on the Internet: distributed computing technologies: Proceedings of the All-Russian Scientific Conference*, Novo-rossiysk, Moscow, Russian Federation, *Publishing House of Moscow State University*, pp. 79-81 (in Russian).

5. Afanasyev, A. P., Voloshinov, V. V., Posypkin, M. A., Sukhoroslov, O. V. & Khutornoy, D. A. (2006). Grid-tehnologii i vyichisleniya v raspredelennoy srede. [Grid-technology and computing in a distributed environment], *Proceedings III Int. Conf. "Parallel computations and control problems", PACO'2006,* Moscow, Russian Federation, October 2-4, 2006. V. A. Trapeznikov Institute of Management Problems, pp. 19-40. CD ISBN 5-201-14990-1 (in Russian).

6. Khoroshevsky, V. G., Mamoilenko, S. N., Maidanov, Y. S., & Sedelnikov, M. S. (2006). "Space-distributed multicluster computer system for parallel multiprogramme regimes modeling", *Proceedings of the conference "MODELING-2006",* May 16-18, 2006. Kiev, Ukraine, IPME them. G. E. Pukhov NANU, pp. 67-69.

7. Feldman, L. P. & Nazarova, I. A. (2006). Parallelnyie algoritmyi chislennogo resheniya zadachi Koshi dlya multiprotsessorov s raspredelennoy pamyatyu. [Parallel algorithms for the numerical solution of the Cauchy problem for distributed memory multiprocessors], *Proceedings of the III International Conference "Parallel Computing and Control Problems"* PACO '2006 in memory of I.V. Prangishvili. Moscow, Russian Federation, October 2-4, 2006. Institute of Management Problems. V.A. Trapeznikova RAS. Moscow, Russian Federation, Institute of Management Problems. V. A. Trapeznikova, 2006, pp. 19-40. CD ISBN 55-201-14990-1 (in Russian).

8. Thanh Phuong Nguyen, & Shelestov A. Yu. (2005). Parallelnaya realizatsiya algoritmov filtratsii kosmicheskih izobrazheniy. [Parallel implementation of space image filtering algorithms], *Problems of control and informatics*, No. 5, pp. 121-132 (in Russian).

9. Giang Le Truong, & Nghiem Trinh Huu (2018), "Multidimensional Laplace approximation via trotter operator". *Applied Aspects of Information Technology*, Vol.1 No.1, pp. 59-65. DOI://10.15276/aait.01.2018.4.

10. Pavlenko, Vitaliy, & Pavlenko, Sergey. (2018), "Deterministic Identification Methods for Nonlinear Dynamical Systems based on the Volterra Model". *Applied Aspects of Information Technology*, Vol.1 No.1, pp. 11-32. DOI: 10.15276/aait 01.2018.1.

11. Pavlenko, V., Pavlenko S., & Speranskyy, V. (2014). "Chapter 10: Identification of systems using Volterra model in time and frequency domaino", In book: "Advanced Data Acquisition and Intelligent Data Processing". Chapter 10. V. Haasz & K. Madani (Eds.). *River Publishers*, pp. 233-270. ISBN 978-87-93102-73-6.

12. Lomovoy, V., & Pavlenko, V. (2019). "Methods and Tools for Identification of Nonlinear Dynamical Systems based on Volterra Models in Frequency Domain", Scientific notes of the Tavrichesky National University named after V. I. Vernadsky, Series: Engineering. Vol. 30 (69). Part. 1, No. 1, pp. 78-96.

13. Kolding, T. E., & Larsen, T. "High Order Volterra", Series Analysis Using Parallel Computting – http://citeseer.ist.psu.edu/242948.html.

14. Pavlenko, V. D., Cherevaty, V. V., & Burdeyny, V. V. (2006). Postroenie modeley nelineynyih sistem v vide yader Volterra s ispolzovaniem tehnologii klasternyih vyichisleniy. [Building models of nonlinear systems in the form of Volterra kernels using cluster computing technology]. The Second International Scientific Conference "Intellectual Decision Making Systems and Applied Aspects of Information Technologies" (ISDMIT'2006), Eupatorium (Crimea, Ukraine), May 15-18, 2006 Collection of scientific papers in 4 Volumes. V. 1, pp. 159-162 (in Russian).

15. Pavlenko, V. D., Fomin, O. O. (2015). "Intelligent Information Technology Building Systems Diagnostics Using Nuclear Moments Volterra". Herald of the National Technical University "KhPI". Subject issue: Information Science and Modelling. Kharkov, Ukraine, NTU "KhPI". No. 33 (1142), pp. 106-119.

16. Voevodin, V. V., & Voevodin, V. B. (2002). Parallelnyie vyichisleniya. [Parallel

computing]. SPb,m., Russian Federation, *BHV*-Petersburg, 608 p. (in Russian).

17. Burdeyniy, V. V. (2006). Printsip organizatsii klasternyih vyichisleniy s pomoschyu neyavnogo rasparallelivaniya, osnovannogo na zakazah. [The principle of organizing cluster computing using implicit parallelization based on orders]. *Proceedings of the XIII International Conference of Students, Postgraduates and Young Scientists "Lomonosov – 2006",* section "Computational Mathematics and Cybernetics". Moscow, Russian Federation, *Publishing Department of MSU Faculty of* "Computational Mathematics and Cybernetics", 2006. pp. 11-12 (in Russian).

18. Pavlenko, V. D., & Burdeyny, V. V. (2006). Printsipyi organizatsii klasternyih vyichisleniy s pomoschyu neyavnogo rasparallelivaniya, osnovannogo na zakazah. [Principles of the organization of cluster computing using implicit paralleling based on orders], *Proceedings of the III International Conference "Parallel Computing and Control Problems'" PACO '2006 in memory of I.V.* Prangishvili. Moscow, October 2-4, 2006, Institute of Management Problems V.A. Trapeznikova RAS. Moscow, Russian Federation, V. A. Trapeznikov Institute of Management Problems, 2006, pp. 670-690, CD ISBN 5-201-14990-1 (in Russian).

19. Pavlenko, V. D., & Burdeyny, V. V. (2007). Tehnologiya klasternyih vyichisleniy s ispolzovaniem transparentnogo rasparallelivaniya, osnovannogo na zakazah. [Cluster computing technology using transparent order-based parallelization]. Bulletin of the National Technical University "Kharkiv Polytechnic Institute": Collection of scientific papers. Topical Issue "System Analysis, Management and Information Technology". Kharkiv, Ukraine, NTU "KhPI", No. 6, pp. 84-107 (in Russian).

20. Burdeyny, V. V., & Pavlenko, V. D. (2007). Organizatsiya klasternyih vyichisleniy s pomoschyu neyavnogo rasparallelivaniya, osnovannogo na zakazah. [Cluster computing using implicit parallelization based on orders], Bulletin of Young Scientists "Lomonosov": a collection of the best reports of the *XIII International Scientific Conference of Students, Postgraduates and Young Scientists "Lomonosov",* Moscow, Russian Federation, *MAKS Press,* Vol. III, pp. 47-53 (in Russian).

21. Pavlenko, V. D., & Burdeyny, V. V. (2007). Organizatsiya klasternyih vyichisleniy na osnove printsipov transparentnogo rasparallelivaniya. [Organization of cluster computing based on the principles of transparent parallelization]. *Second International Conference "Systems Analysis and Information Technologies" SAIT-2007* (September 10-14, 2007, Obninsk : Russia), Conference proceedings. In 2 vol. Vol. 2. Moscow, Russian Federation, *LKI Publ*., 2007, pp. 225-226 (in Russian).

22. Pavlenko, V. D., & Burdeyny, V. V. (2007). Tehnologiya transparentnogo rasparallelivaniya v klasternyih sistemah, osnovannogo na zakazah. [Technology of transparent parallelization in cluster systems based on orders], *Works of 12-th International Scientific Conference "System Analysis,* Control and Navigation", July 1-8, 2007, Evpatoria, Crimea, pp. 134-136 (in Russian).

23. Pavlenko, V. D., & Burdeyny, V. V. (2008). Klasternyie vyichisleniya s ispolzovaniem tehnologii transparentnogo rasparallelivaniya na osnove zakazov. [Cluster computations using the technology of transparent parallelization based on orders]. Modeling and controlling the state of the ecological-economic systems of the region. Collection of works. Kyiv, Ukraine, MNNTSITS, No. 4, pp. 170-179 (in Russian).

24. Pavlenko, V. D., Burdeinyi, V.V., & Pavlenko, S. V. (2011). "Organization of intelligent computations in clusters using transparent parallelizing principles". *Computational intelligence (results, problems, prospects): Materials of the 1-st International Scientific and Technical Conference* (May 10-13 2011, Cherkasy), Cherkasy, Ukraine, Maklaut, 2011, pp.71-72.

25. Burdeinyi, V. V. & Pavlenko, V. D. (2012). "Orderly method of parallelizing technology". Works Odessa National Polytechnic University, one Odessa : Vol. 1 (37), pp. 227-233.

26. Pavlenko, V. D. (2014). Tehnologiya programmirovaniya parallelnyih vyichisleniy na klasterah. [The technology of programming parallel computing on clusters]. *Informatics and mathematical methods in modeling*, Vol. 4, No. 2, pp. 105-110 (in Russian).

27. TOP500 List, (June 2010). http://www.top500.org/lists/2010/06.

28. Burdeyny, V. V., & Pavlenko, V. D. (2006). Instrumentalnyie programmnyie sredstva podderzhki metoda neyavnogo rasparallelivaniya klasternyih vyichisleniy, osnovannogo na zakazah. [Instrumental software support for the method of implicit parallelization of cluster computing, based on orders]. Systematic analysis and information technology technologies: *Materials of the VIII International Scientific Practical Conference of*

Students, Students and Young People (13–16 February 2006, Kyiv). Kyiv, Ukraine, NTUU "KPI", 2006, pp. 161-165 (in Russian).

29. Pavlenko, V. D., & Burdeyny, V. V. (2007). Method i instrumentalnyie sredstva transparentnogo rasparallelivaniya v klasternyih sistemah. [Method and tools for transparent paralleling in cluster systems]. In the collection of theses of III International Scientific Conference "Intelligent Decision-Making Systems and Applied Aspects of Information Technologies" (ISDMIT'2007), May 15-18, 2007, Evpatoria : Crimea, pp. 206-208 (in Russian).

30. Pavlenko, V. D., & Burdeyny, V. V. (2007). Sredstva otsenki parametrov parallelnyih vyichisleniy v klasternyih sistemah pri ispolzovanii printsipov transparentnogo rasparallelivaniya. [Means for estimating parameters of parallel computing in cluster systems using the principles of transparent paralleling]. XV International Conference on Computational Mechanics and Modern Applied Software Systems, VMSPPS'2007, May 25-31 2007, Alushta : Crimea. Moscow, Russian Federation, MAI, 2007, pp. 402-404 (in Russian).

31. Pavlenko, V. & Burdeinyi, V. (2008). Computing Simulation in Orders Based Transparent Parallelizing, ICIM' 2008: Proceedings 2-nd International Conference on Inductive Modelling, September 15-19, 2008, Kyiv, Ukraine, pp.168-171.

32. Pavlenko, V. D., & Burdeyny, V. V. (2008). Otsenka vremennyih harakteristik vyichislitelnogo protsessa v klasterah pri ispolzovanii tehnologii transparentnogo rasparallelivaniya. [Evaluation of the temporal characteristics of the computational process in clusters using transparent paralleling technology]. In the collection of the International Scientific Conference "Introductive Systems and Resolving the Problems of the Comparable Intertect", (ISDMCI'2008), April 19-23, 2008, city of Europe, Ukraine, in 3 Volumes, V. 3. Part 2 - "Theoretical and applied aspects and systems of solution". Evpatoria : Crimea. KNTU, 2008, pp. 46-49 (in Russian).

33. Pavlenko, V. D., & Burdeyny, V. V. (2008). Opredelenie vremennyih harakteristik vyichislitelnogo protsessa v klasternyih sistemah pri ispolzovanii printsipov transparentnogo rasparallelivaniya. [Determination of the temporal characteristics of the computational process in cluster systems using the principles of transparent parallelization]. Avtomatika – 2008: reports of the XV International Conference with automatic control, 23–26 February 2008, Odesa. In 3 t., Vol. 2. Odesa, Ukraine, ONMA, 2008, pp. 7-10 (in Russian).

34. Pavlenko, V. D., & Burdeyny, V. V. (2008). Identifikatsiya vremennyih harakteristik vyichislitelnogo protsessa v tehnologii transparentnogo rasparallelivaniya, osnovannogo na zakazah. [Identification of the temporal characteristics of the computational process in the technology of transparent paralleling based on orders]. *Proceedings of the IV International Conference "Parallel Computing and Control Problems", PACO'2008*. Moscow : October 27-29, V. A. Trapeznikov Institute of Management Problems, pp.719-748. CD ISBN 978-5-91450-016-7 (in Russian).

35. Pavlenko, V. D., & Burdeyny, V. V. (2009). Instrumentalnyie sredstva parallelnyih vyichisleniy na osnove tehnologii transparentnogo rasparallelivaniya. [Tools for parallel computing based on transparent paralleling technology], *Proceedings of the XVI International Conference on Computational Mechanics and Modern Application Software Systems (VMSPPS'2009),* May 25-31, 2009, Alushta: Moscow: *Publishing house MAI-PRINT*, 2009, pp. 560-563 (in Russian).

36. Pavlenko,, V. D., & Pavlenko, S. V. (2011). Vyichislitelnyiy intellekt i informatsionnaya optimizatsiya sistem diagnostirovaniya sostoyaniy nepreryivnyih ob'ektov [Computational intelligence and informational optimization of systems for diagnosing states of continuous objects]. *Materials of the 1-st International Scientific and Technical conferences* (May 10-13, 2011, Cherkasy). Cherkasy : Maklaut, 2011. pp. 113-114 (in Russian).

37. Pavlenko, V. D, Burdejnyj, V. V, & Pavlenko, S.. (2012). Application of parallel computing with using of orders based transparent parallelizing technology for the modeling and simulation, VI International Conference "Parallel Computing and Control Problems", 24-26 October 2012, Russia, Moscow : Institute of Management Problems V. A. Trapeznikova RAS, pp. 224-230.

38. Cormen T., Leiserson C., & Rivest R. (2002). Algoritmyi: postroenie i analiz [Algorithms: construction and analysis], Trans. from English by ed. A. Shen. Moscow, Russian Federation, MTSNMO, 2002, 960 p. (1999). T-system is a programming environment with support for automatic dynamic program parallelization. Institute of Software Systems RAS, Pereslavl-Zalessky : October, 20, 1999ю http://www.botik.ru/~t-system/ (in Russian).

Received 15.01.2019

[1]**Павленко, Віталій Данилович,** д-р техніч. наук, професор, професор каф. комп'ютеризованих систем управління, pavlenko_vitalij@ukr.net, ORCID ID: 0000-0002-5655-4171
[1]**Павлєнко, Сергій Віталійович,** канд. техніч. наук, старший науковий співробітник каф. комп'ютеризованих систем управління, E-mail: psv85@yandex.ru, ORCID ID: 0000-0002-9721-136X
[1]ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ**,** просп. Шевченка, 1, Україна, Одеса, 65044

ОРГАНІЗАЦІЯ ОБЧИСЛЕНЬ В КЛАСТЕРАХ З ВИКОРИСТАННЯМ ПРИНЦИПІВ ТРАНСПАРЕНТНОГО РОЗПАРАЛЕЛЮВАННЯ

Анотація. *Розглядаються принципи організації кластерних обчислень на основі технології транспарентного розпаралелювання, яка дозволяє для алгоритмів, реалізованих з використанням паралелізму завдань, достатньо легко переходити від існуючих послідовних програм до паралельних реалізацій, вносячи незначні зміни в код, так і в логіку роботи алгоритму прикладної задачі. Запропонована технологія реалізована у вигляді фреймворка на мові програмування Java. Наведено архітектуру фреймворку і описана функціональність його частин Розглянуто основні питання, що виникають при розробці інструментарію, так і при його практичному використанні. Пропонується метод аналізу часових характеристик виконання довільної прикладної задачі на однорідному досить великому кластері при використанні технології транспарентного розпаралелювання на онові замовлень. Ефективність технологіїї підтверджується вирішенням задачі визначення діагностичної цінності формованих ознак на кластері з 2; 3; 5 і 10 комп'ютерів. Наведено порівняння пропонованої технології з найближчим аналогом –Т-системою.*

Ключові слова*: нелінійні динамічні системи; ідентифікація; модель Вольтерра; ядра Вольтерра; вейвлет-перетворення*

[1]**Павленко, Виталий Данилович,** д-р технич. наук, профессор, профессор каф. компьютеризированных систем управления, E-mail: pavlenko_vitalij@ukr.net, ORCID ID: 0000-0002-5655-4171
[1]**Павленко, Сергей Витальевич,** канд. технич. наук, старший научный сотрудник каф. компьютеризированных систем управления, E-mail: psv85@yandex.ru, ORCID ID: 0000-0002-9721-136X
[1]Одесский национальный политехнический университет, просп. Шевченко, 1, Украина, 65044, Одесса,

ОРГАНИЗАЦИЯ ВЫЧИСЛЕНИЙ В КЛАСТЕРАХ С ИСПОЛЬЗОВАНИЕМ ПРИНЦИПОВ ТРАНПАРЕНТНОГО РАСПАРАЛЛЕЛИВАНИЯ

Аннотация. *Рассматриваются принципы организации кластерных вычислений на основе технологии транспарентного распараллеливания, которая позволяет для алгоритмов, реализованных с использованием параллелизма заданий, достаточно легко переходить от существующих последовательных программ к параллельным реализациям, внося незначительные изменения как в код, так и в логику работы алгоритма прикладной задачи. Предложенная технология реализована в виде фреймворка на языке программирования Java. Приведены архитектура фреймворка и описана функциональность его частей. Рассмотрены основные вопросы, возникающие при разработке инструментария, так и при его практическом использовании. Предлагается метод анализа временных характеристик выполнения произвольной прикладной задачи на однородном достаточно большом кластере при использовании технологии транспарентного распараллеливания создана на основе заказов. Эффективность технологии подтверждается решением задачи определения диагностической ценности формируемых признаков на кластере из 2; 3; 5 и 10 компьютеров. Приводится сравнение предлагаемой технологии с ближайшим аналогом – Т-системой.*

Ключевые слова: *параллельные вычисления; кластерные вычисления; разработка программного обеспечения; фреймворки, транспарентное распараллеливание*

;