© 2012 V.D. PAVLENKO, PhD
V.V. BURDEJNYJ
S.V. PAVLENKO
(Odessa National Polytechnic University, Odessa, Ukraine)

APPLICATION OF PARALLEL COMPUTING WITH USING OF ORDERS BASED TRANSPARENT PARALLELIZING TECHNOLOGY FOR THE MODELING AND SIMULATION

APPLICATION OF PARALLEL COMPUTING WITH USING OF ORDERS BASED TRANSPARENT PARALLELIZING TECHNOLOGY FOR THE MODELING AND SIMULATION / V.D. Pavlenko, V.V. Burdejnyj, S.V. Pavlenko (Odessa National Polytechnic University, Shevchenko Av. 1, Odessa, 65044, Ukraine, E-mail: pavlenko_vitalij@mail.ru), Practical usage of the implementation of orders based transparent parallelizing technology is described. The steps required to parallelize arbitrary programs and execute them on cluster are explained. A sample problem (the problem of features diagnostic value comparison based on full scan) was solved in cluster environment and on single multicore computer. Both experiments have shown high efficiency of the technology of orders based transparent parallelizing.

1. Introduction

Parallel computing is the subject of a lot of 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. For example, these are the problem of Volterra series based nonlinear dynamic systems models identification [1, 2], problem of full scan based comparison of features diagnostic value, modeling problems and so on [3, 4]. Despite the simplicity of the idea of parallel computing, this field of science has to solve a lot of problems, including problems of creating efficient parallel algorithms, introducing all kinds of parallelism to hardware, making software and hardware fit each other [5]. 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 [6]. 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 [7]).

2. Practical usage of orders based transparent parallelizing technology

This article is devoted to usage of this implementation and parallelizing applied algorithms using it.

A distribution of the framework consists of the following files:

parallel.jar – implementation of the framework;

client.bat, server.bat, problem.bat, sourcegen.bat – sample files for running client. server, starting computations and generating utility classes respectively;

serverconf.properties – server settings;

lib\ – directory with libraries used by the framewor;

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.
3) Ensure that all chosen methods found above have return type "void". The ones that do not 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 library, its local and remote implementations 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 filesystem and writing output to console and/or local filesystem 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 pack the class files into a JAR file.

The list of methods of DataHandler class is given below:

```
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;
}
```

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 is 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's show that by parallelizing a program that solves the problem of features diagnostic value comparison based on full scan.

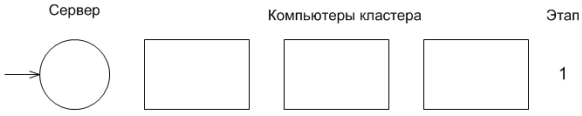
The structure of the initial program is shown:

```
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
        }
         }
```

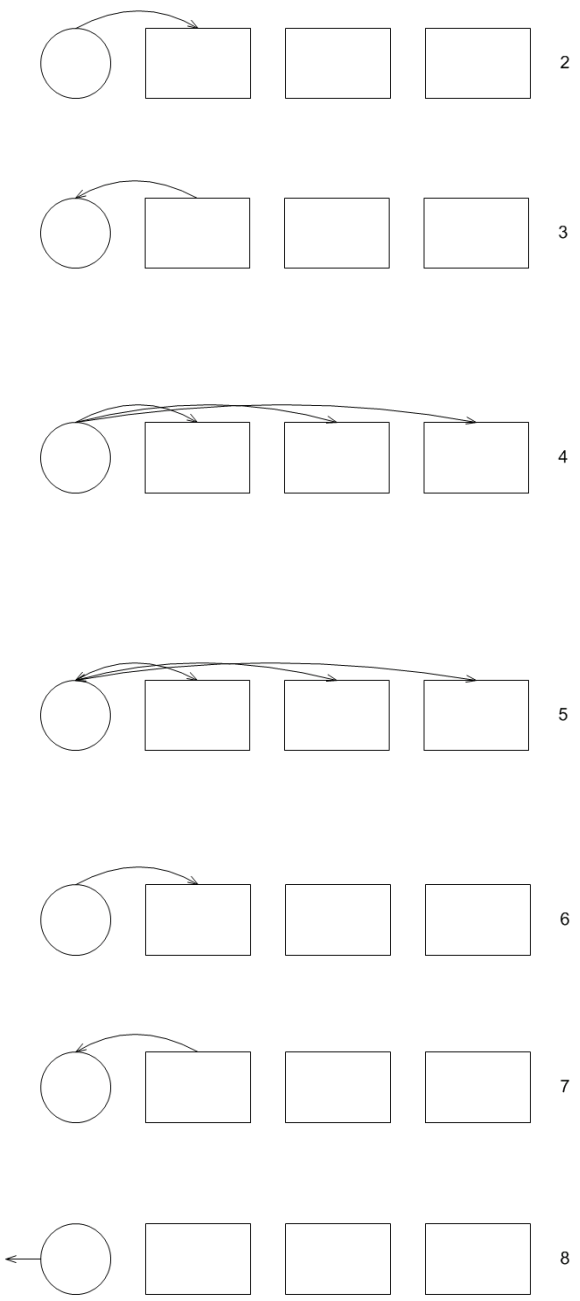
This algorithm does not contain significant intervals of time between computing of some values and their first usage. Let's change the order of calculation in the following way: we will split all possible sets of features into some groups and find best set for each group independently. After that we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

```
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
        }
        }
```

Methods analyseGroup 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. 1.



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. 1. Stages of execution of parallel application on cluster with 3 nodes

3. Experiments

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). 10 runs were done for each configuration for the problem of dimension 24. Their results are given in table. Their results are given in table.

Table

Results of experimental testing of efficiency of framework on a multi-core computer

	Execution time			Speedup	Utilization of cores,%
	Min	Aveage	Max		
Serial execution, framework not used	288	289	291	1.000	100.0
Parallel execution on 1 core (in fact it was serial execution, but with the framework used)	292	293	297	0.985	98.5
Parallel execution on 2 cores	180	183	186	1.580	79.0
Parallel execution on 3 cores	143	144	146	2.006	66.9
Parallel execution on 4 cores	134	138	141	2.101	52.5
Serial execution, framework not used, 2 instances of program are being executed at the same time	319	334	341	0.866	86.6
Parallel execution on 2 cores without Hyper-Threading	153	154	158	1.875	93.7

Speedup was computed by dividing average execution time by average time of serial execution and utilization was computed by dividing speedup by number of cores used in the computations. The experiment shows good utilization of processor in the case of parallel execution on 1 core and execution on 2 cores without Hyper-Threading. Utilization is worse in other cases and the speedup is almost the same for 3 cores and for 4 cores. Possible explanation for it is low speedup of computations assigned to two virtual cores in one physical core. Utilization shown in the last test is probably caused by effect of non-computational threads of Java application (such as finalizer thread).

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 using 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. 2.

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.

4. Conclusion

The method of practical usage of technology of orders based transparent parallelizing has been shown by explaining the steps for parallelizing arbitrary programs, implementing and testing a sample program. Steps for parallelizing include initial analysis of the algorithm and its serial implementation, choosing methods to be replaced by making orders, applying changes to them according to the requirements of the technology, creating data structures to be used as parameters and replacing calls with making orders. An experiment conducted on multi-core computer shows that the overhead

introduced by the technology is small and utilization of computer cores is high if Hyper-Threading is not used while increasing number of virtual cores using Hyper-Threading does not decrease execution time significantly for the sample algorithm. An experiment conducted on cluster also demonstrates high utilization and low overhead introduced by the technology/

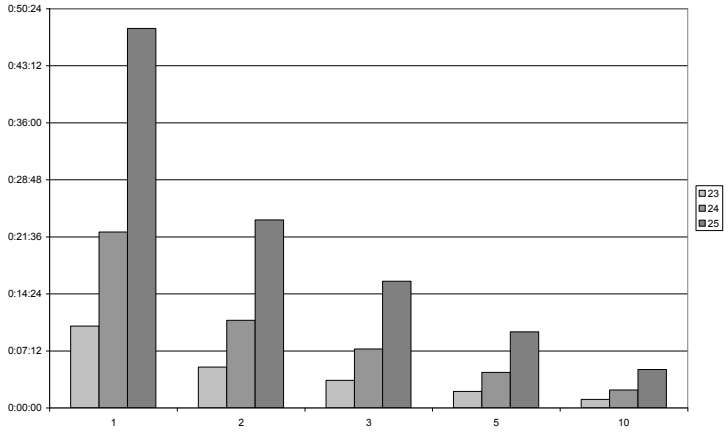


Fig. 2. Results of experimental testing of efficiency of framework on clusters

REFERENCES

1. *Павленко В.Д.* Идентификация нелинейных динамических систем в виде ядер Вольтерры на основе данных измерений импульсных откликов // Электрон. моделирование. 2010. Т. 32, № 3. С. 3–18.
2. *Kolding T.E., Larsen T.* High Order Volterra Series Analysis Using Parallel Computing // International Journal of Circuit Theory and Applications. 1997. V. 25. No. 2. P. 107–114.
3. Грид–технологии и вычисления в распределенной среде / *А.П. Афанасьев, В.В. Волошинов, М.А. Посыпкин, О.В. Сухорослов, Д.А. Хуторной* // Труды III Межд. конф. “Параллельные вычисления и задачи управления”, PACO’2006, Москва, 2–4 октября 2006 г. М.: Институт проблем управления им. В.А.Трапезникова РАН, 2006. С.19–40. CD ISBN 5-201-14990-1.
4. *Fissgus U.* A Tool for Generating Programs with Mixed Task and Data Parallelism // Dissertation. University Halle-Wittenberg. 2001. (http://sundoc.bibliothek.uni-halle.de/diss-online/01/01H119)
5. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ–Петербург, 2002.
6. *Павленко В.Д., Бурдейный В.В.* Принципы организации кластерных вычислений с помощью неявного распараллеливания, основанного на заказах // Труды III Межд. конф. “Параллельные вычисления и задачи управления”, PACO’2006, Москва, 2–4 октября 2006 г. — М.: Институт проблем управления им. В.А.Трапезникова РАН, 2006. С.670-690.— CD ISBN 5-201-14990-1.
7. TOP500 Supercomputing Sites / *H. Meuer, E. Strohmaier, J. Dongarra, H. Simon* // Site Promoted by: Prometeus GmbH Fliederstr. 2, D-74915 Waibstadt-Daisbach, Germany, info at prometeus.de, +49 7261 913 160 — November, 2011. (http://i.top500.org/stats).