Orders Based Transparent Parallelizing Technology

Vitaliy D. Pavlenko, Victor V. Burdeinyi

Department of Computer Control Systems
National Polytechnical University of Odesa, Odesa
pavlenko_vitalij@mail.ru
Department of Mathematical Software of Computer Systems
I.I. Mechnikov National University of Odesa, Odesa
vburdejny@gmail.com

Abstract

This paper proposes a new approach to parallel applications development, showing high speed and low labour intensiveness of creating new parallel applications and parallelizing existing ones. The basic principles of the proposed technology are described. The possible ways of implementation of these principles are proposed. Parallel execution characteristics analysis method is proposed. The result of experiment is given to show high efficiency of proposed technology.

1. Introduction

Parallel computing is the subject of a lot of researches nowadays [1]. It's caused by large amount of problems that cannot be solved fast enough on single modern computers. For example, that's the problem of Volterra series based nonlinear dynamic systems models identification [2, 3], problem of full scan based comparison of features diagnostic value [4, 5], modeling problems and so on [6–10].

Modern parallel architectures can be splitted into three large groups – parallel computers with shared memory, clusters and distributed systems. Each group has own advantages and disadvantages and own specific problems. In this approach we use clusters. Clusters are very popular nowadays because of comparatively low cost, high speed and high scalability. For example, 72% of computer systems of Top 500 list are clusters [11].

There are a lot of problems in the field of parallel computing that should be solved. One of not completely solved problems of parallel computing is the problem of development of tools for parallel programming. The main obstacle for creating such tools is the complicatedness of finding parts of program that can be executed in parallel. Modern programming technologies usually allow programmer to use popular imperative programming languages to make parallelizing of existing applications easier. It's hard to discover the potential parallelism in programs in these programming languages automatically. That's why modern parallel programming technologies use either to provide user some tools for declaring the parts of program that can be executed in parallel or to provide user only low–level tools for organizing computers communication. The second way is also much more popular nowadays (for instance, MPI among the processes for clusters, uses it). It makes parallel application development and debugging much harder, but technology, which is a de facto standard for communication gives programmer more control over the efficiency of created programs.

The purpose of this paper is to create a high level cluster computing technology that allows user to develop parallel applications fast enough for certain wide class of algorithms.

2. Existing technologies of parallel applications development

There's 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. But 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 labour 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. Also 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.

3. Technology of orders based transparent parallelizing

We assume that we have selected some set of procedures in the program. Each procedure should not work with any data during execution except 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 significant limits on program. For example, it forbids using global variables or I/O devices. But 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 that will be used to illustrate the technology is shown on Fig. 1:

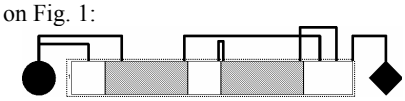

Figure 1: Illustration of a sample program.

We show procedure execution time with a rectangle (time goes from left to right). If one procedure calls another one, a part of rectangle is shaded to show called procedure execution time (there are no nested calls in this example). Lengths of rectangles and their parts are proportional to the time of execution of corresponding program parts. A circle is used to show input parameters and a rhombus is used to show output ones. It is considered that all input parameters are known already at the moment of program execution start. Lines connect moments of getting some values and moments of their first usage. Computations, performed by one processor, are shown with a dotted rectangle.

The first principle of offered technology introduces the concept of an order. An order is defined as the minimal unit of work that should be executed on one computer and cannot be splitted into smaller parts. Such a unit of work is defined as execution of one procedure without execution of procedures it calls. Each procedure call is replaced with creation of a new order that should be executed by some computer of cluster (we will call such procedure call "making an order"). One of selected procedures should be marked as main one to define program entry point (and all input and output data of program should be passed through parameters of this procedure).

This principle is illustrated on Fig. 2. It is considered that three orders are executed by different processors and their execution starts immediately after making corresponding orders. Vertical lines show the moments of time when the orders are made.

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

This principle is illustrated on Fig. 3.

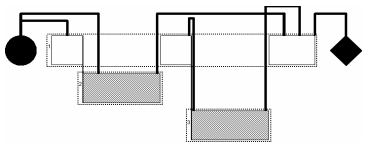

Figure 2: An illustration of the first principle of offered technology.

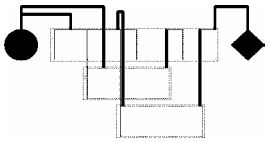

Figure 3: An illustration of the second principle of offered technology.

We use dashed lines to show moments when an order stops to wait for some value. 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 computed.

An order means a unit of work but is based on parts of program source, marked as procedures. But procedures are terms of programming language, and programmers can have different reasons to mark parts of source as procedures. These reasons can have nothing in common with getting high efficiency of parallel application. Theoretically there's no problem about that: we can easily split a procedure with big execution time into a few smaller ones and any unneeded splitting only changes the order of computations and does affect the efficiency of the program. But from the practical point of view the procedures with small execution time and big number of calls cause big overhead. So we should allow programmer to call selected procedures in standard way.

Offered technology is based on task parallelism and MIMD model. It uses only four computers communication operations: getting an order, getting results of order execution, making an order and sharing results of order execution. There are only two operations that are accessible to user: making an order and getting a value, computed in another order. It means that we can make parallel applications development much easier by hiding computer communication operations from user. But that also means that a framework that implements the proposed technology should take care or efficient usage of network itself. Note that a program in this

technology is a set of instructions for a whole cluster (unlike programs in MPI technology that are a set of instructions for each computer).

4. Formal description of technology

Offered technology can be used to create parallel applications on many structural procedural or object–oriented programming languages. We will use terms of Java programming language in the following description.

Requirements about the selected set of procedures can be explained in the following way. There must be a set of static methods in the program. Each of them can only perform some computations and execute other selected methods using some mechanism, provided by the framework. Each selected method can work only with its parameters and some temporary data structures. We do not take care about traditional procedure calls because they do not differ from usual computations. It is impossible to pass all parameters by value in a lot of programming languages, including Java. So let's replace this requirement with the following one: if we replace a pointer to some data with a pointer to copy of that data, time and result of method execution should stay the same. If data contains some pointers inside, this should also be true for it.

We can make two conclusions from these requirements: two concurrently executed procedures do not affect each other and the procedures can pass data to each other only through parameters. We can also tell that if procedure A calls procedure B and we replace call of procedure B with applying the results of its execution to values, passed as parameters, we will not change result of execution of A and will make time of execution of A lower by the time of execution of B. So we are able to execute B on another computer as proposed in the first principle of offered technology.

However, the first principle does not allow us to get acceleration by using many computers instead of one. The second principle describes the way to allow more that one computer to work in the same time.

These two principles split the operators in the source of user code into three groups: operators of making orders, operators of data request and operators of computations. So we can describe algorithms we need.

Scheduler is a part of client that makes decisions about continuation of execution of previously suspended order or getting a new one from server after some processor is being freed. Depending on used algorithms the scheduler can either work on different clients independently or can use server for coordination.

The main question about implementing the proposed approach as a parallel computing framework is a question about the way of second principle implementation because the second principle means that the system should work in a little unusual mode. In order to implement the second principle we have to find each point of program that contains access to data that can be unknown and to add a verification of presence of that data and getting it from server if needed. There are a few ways to do that:

- We may ask user to add such verifications. A small advantage of this method is possibility of optimization of such verifications because user can known the places where such verifications are not needed and not to place them there. But the main disadvantage of this method is that is causes low speed and high labor intensiveness of parallel applications development. Also this method requires user to pay a great attention to such verifications because mistakes in them can cause bugs that are hard to reproduce, find and fix.
- We may analyze the source of the program and add verifications there needed. The main advantage of this method is hiding the verifications from user. The problem of this method is that it's hard to find places in program where we can be sure that checks are not needed.
- We can analyze compiled version of program. This variant makes sense only if program has been compiled to some kind of bytecode which is easy to analyze – for example, Java bytecode or MSIL in .NET Framework.
- If used programming language is object–oriented, we can ask user to implement a class for each data type, used as procedure parameter, and to implement data getting logic is the methods of these classes that provide access to encapsulated data.

The first principle means that we have to provide some mechanism of making orders to user. We can do that either with applying changes to language (by patching compiler or adding a preprocessor) or without applying any changes. In the second case we can simply generate a method for making an order for each selected method. These methods can be generated either as source or as binary code (second variant can be better in Java or .NET Framework). There are a few ways that can be used for declaring the selected set of procedures:

- User can declare procedures list in a separate file.
- User can mark selected procedures with specific comments.
- User can mark procedures with annotations (in Java 1.5.0 or above or C#).
- A modified variant of programming language can be used.

5. Computing emulation

Computing emulation is a method for estimating program execution time and finding bottlenecks. Its main idea is the following: 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 this test run without knowing the values that are going to be computed in the block. We can warrant that by assuming that values, computed in the block, should not be used in any conditional operators, and we can use asymptotic estimation of blocks execution time in order to estimate block execution time for specific input data.

The proposed method consists of four stages. In the first one 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 the block, should not be used in any conditional operators; block 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. Also 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 estimate constant part of execution time asymptotic estimation 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.

During the third stage a test run of a 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 run the program 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 information about the orders of the parallel program that contains the following information about each order: duration, moments of data requests, moments of providing data.

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

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 its input data and the fourth one uses information about the cluster. Splitting the method into a set of stages makes it possible to re–use results of some stages if something changes in the parallel application and the cluster. For instance, user can repeat only the fourth stage to find the configuration of cluster if program execution time is limited.

6. Conclusions

This paper proposes a new parallel applications development technology, based on transparent replacement of calls of some methods with their execution on other computers of cluster. This approach enables user to develop new and port existing parallel applications of certain wide enough class fast enough, making development cost much lower without significant changes in applications efficiency. Offered technology has been implemented as a framework on Java programming language. 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.

A way of analysis of time characteristics of parallel program execution is proposed.

7. References

[1] Voyevodin, V.V. and Voyevodin, V.V., *Parallel computations*, BHV–Petersburg, Saint Petersburg, 2002 (in Russian).

[2] Pavlenko, V.D., "Estimation of the Volterra Kernels of a Nonlinear System Using Impulse Response Data", *Signal/Image Processing and Pattern Recognition: Proceedings the Eighth All–Ukrainian International Conference UkrOBRAZ'2006*, *August 28–31, 2006, Kyjiv*, *Ukraine*, pp. 191 – 194.

[3] Kolding, T.E. and Larsen, T. "High Order Volterra Series Analysis Using Parallel Computing", http://citeseer.ist.psu.edu/242948.html

[4] Pavlenko, V. and Fomin, O. "Reconstruction of the Parameters Space on the Base of Diagnostic Models of Object with using Volterra Models", *Proc. of 5th Middle Eastern Simulation and Modelling Conference (MESM'2004), September 14–16, 2004, Philadelphia University, Amman, Jordan*, pp. 30 – 40.

[5] Pavlenko V. and Fomin A. "Methods For Black–Box Diagnostics Using Volterra Kernels", *ICIM 2008*: *Proc. 2nd International Conference on Inductive Modelling*, September 15–19, 2008, Kyiv, Ukraine, pp.104–107, ISBN 978–966–02–4889–2.

[6] Afanasiev, A.P., Khutornoy, D.A., Posypkin, M.A., Sukhoroslov, O.V., and Voloshinov V.V., "Grid Technologies and Computing in Distributed Environment", *Proc. of the III International Conference* "*Parallel Computations and Control Problems*" *PACO'2006, V.A. Trapeznikov Institute of Control Sciences, Moscow, October 2–4, 2006,* pp. 19–40, CD ISBN 5–201–14990–1, www.paco.sicpro.org

[7] Pavlenko, V.D. and Burdejnyj, V.V. "Principles of Organization of Orders Based Cluster Calculations Using Implicit Parallelizing", *Proc. of the III International Conference* "*Parallel Computations and Control Problems*" *PACO'2006, V.A. Trapeznikov Institute of Control Sciences, Moscow, October 2–4, 2006,* pp. 670–690, CD ISBN 5–201–14990–1, www.paco.sicpro.org

[8] Pavlenko, V.D. and Burdejnyj, V.V. "Cluster Computing using Orders Based Transparent Parallelizing", *Proc. of IEEE East–West Design & Test International Symposium, EWDTS'07, Armenia, Yerevan, September 7–10, 2007*, pp. 83–88.

[9] Pavlenko, V. and Burdeinyi V. Computing Simulation in Orders Based Transparent Parallelizing. *ICIM 2008*: *Proc. 2nd International Conference on Inductive Modelling*, September 15–19, 2008, Kyiv, Ukraine, pp.168–171, ISBN 978–966–02–4889–2.

[10] 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/prom.pdf

[11] 28th TOP500 List. http://www.top500.org/lists/2006/11