Cluster Computing Using Orders Based Transparent Parallelizing

Vitaliy D. Pavlenko[1], Victor V. Burdejnyj[2]

[1]Computer Control Systems Department
Odessa National Polytechnic University, Odessa, Ukraine
pavlenko_vitalij@mail.ru

[2]Applied Mathematics Department
Odessa I.I. Mechnikov National University, Odessa, Ukraine
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. Usage of the technology is shown with a sample algorithm. The result of experiment is given to show high efficiency of proposed technology.

Key words: cluster computing, parallel computing, task parallelism, parallel computing frameworks

1 Introduction

Parallel computing is the subject of a lot of researches nowadays [VV02]. 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 [PC06, KL98, PF04], problem of full scan based comparison of features diagnostic value, modeling problems and so on [Af06, PB06, Fi01].

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 [To06].

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 technology, which is a de facto standard for communication among the processes for clusters, uses it). It makes parallel application development and debugging much harder, but 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

3.1 Basic principles

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:

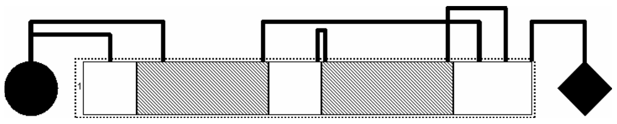

Fig. 1. Illustration of a sample program.

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.

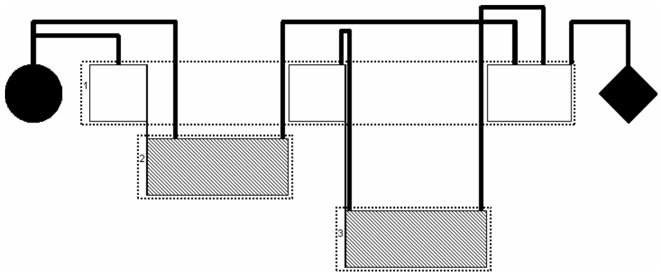

Fig. 2. An illustration of the first principle of offered technology.

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:

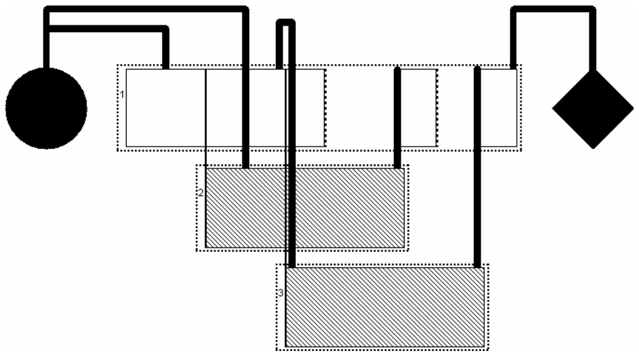

Fig. 3. An illustration of the second principle of offered technology.

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. And there are only two operations that are accessible to user: making an order and getting a value, computed in another order. That 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).

3.2 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.

Order execution algorithm (should be executed on each client computer):

```
get an order from the server;
for (every parameter of the procedure) {
  if (parameter is input or input/output) {
    if (parameter value is known) {
      set the value of parameter according to order data;
    } else {
      bind the parameter to the identifier from order data;
    }
  } else {
    set default value to parameter;
  }
}
execute needed selected static method;
for (every output or input/output parameter of the procedure) {
  send parameter value to server;
}
notify the scheduler about a new free processor;
```

Algorithm of making an order:

```
send ID of procedure that should be executed to server;
for (every input or input/output parameter) {
  if (parameter value is known) {
    send parameter value to the server;
  } else {
    send the identifier bound to this parameter to the server;
  }
}
get the set of identifiers from the server;
for (every output or input/output parameter) {
  bind the parameter to the next identifier;
}
notify the scheduler about a new order;
```

Algorithm of getting a value:

```
get an identifier, bound to the value;
get a value from server according to identifier;
if (the value is not yet computed) {
  tell the identifier to the scheduler;
  notify the scheduler about a new free processor;
  stop execution and wait for a notification from the scheduler;
  get the value from the server according to identifier;
```

```
    }
    unbind the identifier from the value;
```

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.

Let's talk about creating a framework that implements the offered technology. The main question is about the way of second principle implementation because the second principle means that the system should work in a little unusual mode. 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#).

The next question is the question about possible ways of scheduler implementation. It is impossible to create a scheduler that minimizes time of execution of any algorithm, so we have to use some heuristics. The following heuristics have been offered:

- Naive planning heuristic prefers continuation of execution of ready order to getting a new one from server.
- Greedy scheduling heuristic also prefers continuation to getting a new order. If there are a few different orders to continue or to get from server, it prefers the one that blocks execution of the biggest number of orders.

The diagram of states of an order is shown on fig. 4:

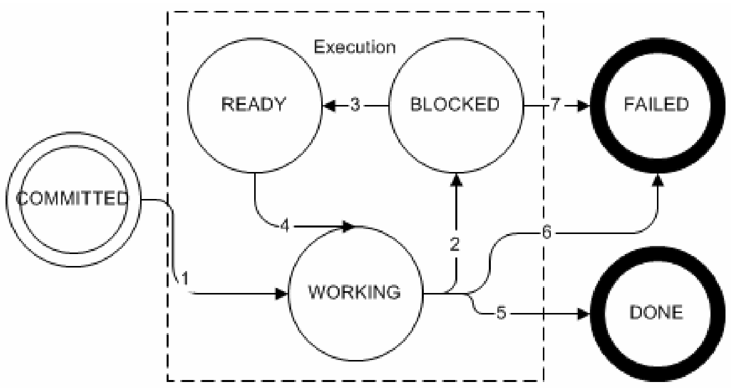

Fig. 4. Diagram of states of an order.

The states mean the following:

- COMMITTED – the order has been made but it's execution haven't started yet
- 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.

Also the idea of using computing emulation results for scheduling has been proposed, but no efficient algorithms have been proposed for now. The computing emulation is a method for estimating program execution time and finding bottlenecks. It consists of four steps.

On the first step user finds some parts of source that meet the following conditions: each such part of code must not contain requests to not ready values or sending orders, the time complexity of each such part of code must be known and the result of its execution must not be significant for subsequent program flow (just for the result). User has to surround such parts of code with operators that notify the framework about the time complexity of code part and a check if the part of code has to be executed.

On the second step the program is executed once. This step is used to find nearly values of constants for time complexity. It's better to change input data of program to make the execution time not too large. We can tell that the source is fixed on this step.

On the third step the program is executed again for the correct input data. The marked blocks are not executed on this step. After this step we get the "picture of computing" which contains the information about all executed orders. Information about each order contains its duration and the moments of requests and making orders. We can tell that the input data is fixed on this step.

On the fourth step the program execution is emulated for concrete cluster configuration and concrete scheduling algorithm to find total execution time and to find the bottlenecks.

4 Using the technology of orders based transparent parallelizing

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 initial program has the following structure:

```
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 in the following way:

```
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
    }
}
```

Also we have to declare classes to be used as types of parameters of analyseGroup method. Each of them should be written is the way, declared by the framework, and should implement serialization and in-place deserialization. Also they should call some methods of framework before accessing their content. Method analyseGroup should be declared in the way it's required by the framework. After that the program is ready to be executed on cluster.

5 Testing of efficiency of offered technology

Described technology has been implemented as a framework on Java programming language. 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. External database makes recovery after errors much easier and allows the framework to continue work after failure of any client and to restart calculations efficiently after server failure. Also this database can be used to save work time by re-using results of recently executed procedures. That can be useful, for example, if some problem is being solved several times for different input data and some parts of work are common.

A solution of the problem of determination of diagnostic value of formed diagnostic features has been implemented for testing of efficiency of created framework. It has been run on clusters of 1, 2, 3, 5 and 10 computers with Intel Pentium 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. 5:

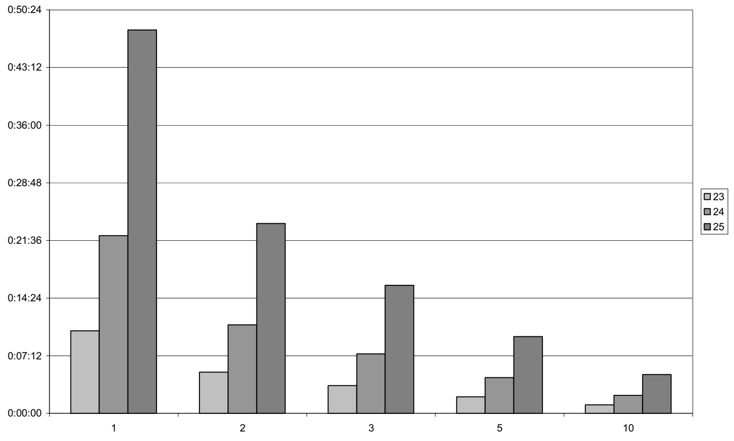

Fig. 5. Results of experimental testing of efficiency of framework.

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.

6 Conclusion

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.

The closest analogue of offered approach is T-system [Ab05].

Bibliography

[Ab05] Abramov S., Adamovich A., Inyukhin A., Moskovsky A., Roganov V., Shevchuk E., Shevchuk Y., Vodomerov A.. 2005. "OpenTS: An Outline of Dynamic Parallelization Approach. Parallel Computing Technologies: 8th International Conference", PaCT 2005, Krasnoyarsk, Russia, September 5-9, 2005. Proceedings. Editors: Victor Malyshkin - Berlin etc. Springer, 2005. - Lecture Notes in Computer Science: Volume 3606, pp. 303-312.

[Af06] Afanasiev A.P., Khutornoy D.A., Posypkin M.A., Sukhoroslov O.V., Voloshinov V.V., "Grid Technologies and Computing in Distributed Environment". Proceedings of the III International Conference "Parallel Computations and Control Problems" PACO'2006. Moscow, October 2 - 4, 2006. V.A.Trapeznikov Institute of Control Sciences, 2006, pp. 19–40, CD ISBN 5-201-14990-1.

[Fi01] 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)

[KL98] Kolding T. E., Larsen T. "High Order Volterra Series Analysis Using Parallel Computing", http://citeseer.ist.psu.edu/242948.html

[PB06] Pavlenko V.D., Burdejnyj V.V., "Principles of Organization of Orders Based Cluster Calculations Using Implicit Parallelizing". Proceedings of the III International Conference "Parallel Computations and Control Problems" PACO'2006. Moscow, October 2 - 4, 2006. V.A.Trapeznikov Institute of Control Sciences, 2006, pp. 670 – 690, CD ISBN 5-201-14990-1.

[PC06] Pavlenko V.D., Cherevatiy V.V., "Identification of Nonlinear Systems as Volterra Kernels with the Help of Differentiation of Responses on Amplitude of Test Signals", Proceedings of the V International Conference "System Identification and Control Problems" SICPRO'06, Institute of Control Sciences, Moscow, Jan 30 - Febr. 2, 2006, pp. 203—216. CD ISBN 5-201-14984-7, www.sicpro.org.

[PF04] Pavlenko V.D., Fomin A.A., "Parameters Space Construction on Base of the Models of Diagnostic Object Using the Volterra Series", Proceedings of the III International Conference "System Identification and Control Problems" SICPRO'04, Institute of Control Sciences, Moscow, January 28 – 30, 2004, pp. 899—918. CD ISBN 5-201-14966-9, www.sicpro.org.

[To06] The 28th TOP500 List. http://www.top500.org/lists/2006/11

[VV02] Voyevodin V.V., Voyevodin Vl.V., "Parallel computations", Saint Petersburg: BHV-Petersburg, 2002 (in Russian)