Technology of Orders Based Transparent Parallelizing

Vitaliy Pavlenko
Victor Burdejnyj

Odessa National Polytechnic University
Shevchenko 1
Odessa, Ukraine

Odessa I.I. Mechnikov National University
Dvoryanskaya 2
Odessa, Ukraine

E-mail: pavlenko_vitalij@mail.ru, vburdejny@gmail.com

INTRODUCTION

There are a lot of problems today that cannot be solved fast enough on traditional computers. That is one of the reasons why parallel computing (including cluster computing) is the subject of a lot of researches nowadays (Voyevodin and Voyevodin 2002). For example, that's the problem of Volterra series based nonlinear dynamic systems models identification (Pavlenko and Cherevatiy 2006 or Kolding and Larsen or Pavlenko and Fomin 2004), problem of full scan based comparison of features diagnostic value, modeling problems and so on (Afanasiev, Khutornoy, Posypkin, Sukhoroslov, Voloshinov 2006 or Pavlenko and Burdejnyj 2006 or Fissgus 2001).

There is a set of problems in the field of parallel computing that should be solved. Those are problems of hardware support of parallel computing, problems of parallel algorithms development and problems of development of parallel applications for concrete parallel architecture. One of not completely solved problems of parallel computing is the is the problem of creating tools for parallel applications development. Main obstacle for creating such tools is the complicatedness of finding parts of program that can be executed in parallel. That's the reason why almost all modern technologies of parallel programming abandon this work to programmer. Implementation of data sharing on parallel architectures without shared memory is usually abandoned to programmer, too. That moves accent from implementing the algorithm of applied problem to using tools, offered by some parallel computing technology. Also that makes parallel applications development much harder.

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.

EXISTING TECHNOLOGIES OF PARALLEL APPLICATIONS DEVELOPMENT

Used parallel architecture should be always taken into consideration for efficient parallel applications development. It this approach we use clusters.

There's one general tendency about modern development tools and technologies. Except traditional requirements (such as efficiency of created applications) an attention is paid to requirements of 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 while the cost of programmers work time is not higher than it is in other areas. 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 time and 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 these 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 understanding source 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. The overhead, caused by the technology, must be minimal. Also the technology must allow used to create efficient implementations of wide enough class of applications.
- High speed and low labor intensiveness of parallel applications development. That also means high speed and low labor intensiveness of porting existing applications.

TECHNOLOGY OF ORDERS BASED TRANSPARENT PARALLELIZING

Basic principles

We assume that we have selected some set of procedures in the program. Each procedure should not work 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 some 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:

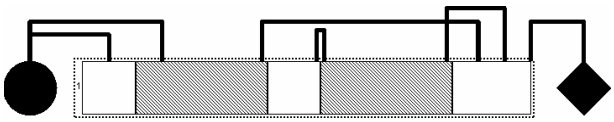

Fig. 1. Illustration of a sample program.

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 by 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 creates a new order that should be executed by some computer of cluster (let's 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 order. Vertical lines show moments of time when orders are made.

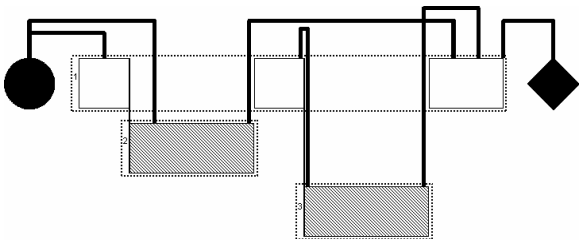

Fig. 2. An illustration of the first principle of offered technology.

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 order of computations. If there are no such intervals in some algorithm, we can say 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 the 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 first request, we should not start waiting at all.
This principle is illustrated on fig. 3:

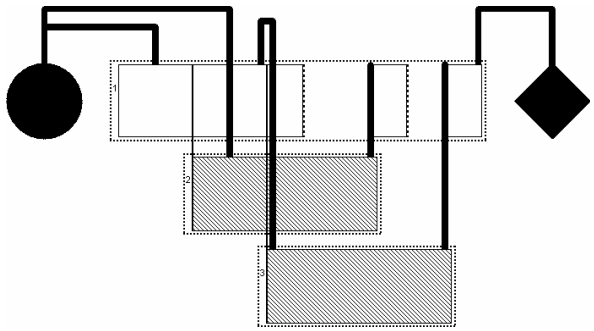

Fig. 3. An illustration of the second principle of offered technology.

We use dashed lines to show moments when an order stops execution 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 got.
So 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). That also means that the best algorithms to be parallelized with offered technology are algorithms with task parallelism.

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 during execution and can work only with its parameters and some temporary data structures. It can also execute other selected methods using some mechanism, provided by the framework. 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 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 them.
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 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 some 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 an access to some data and 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 an object-oriented language, we can ask user to implement a class for each data type, used as procedure parameter, and to implement data getting logic is 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 set of methods with the same signature as selected ones that will perform making orders. 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 figure 4:

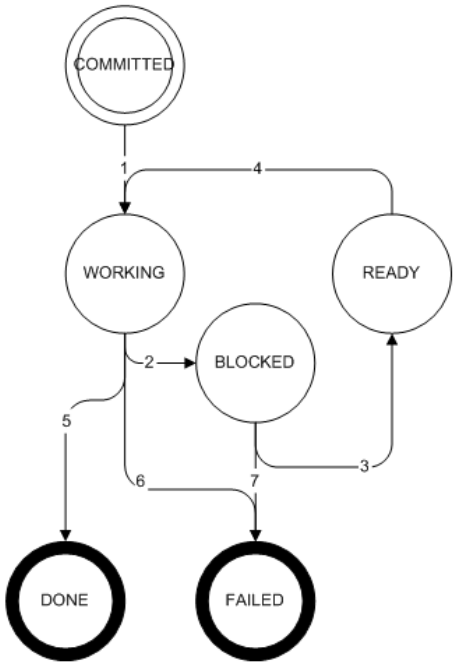

Fig. 4. Diagram of states of an order.

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.

Comparison of the offered technology and the nearest analogues

The closest analogue of the offered technology is the T-system that is being developed in Institute of the programmatic systems of the Russian academy of sciences. Although the offered technology has been developed independently, its main principles are close to the main principles of the T-system (Abramov and Adamovich and Inyukhin and Moskovsky and Roganov and Shevchuk and Shevchuk and Vodomerov 2005). 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 getting 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. Also other methods of method calls optimization have been proposed to prevent getting big overhead.

TESTING OF EFFICIENCY OF OFFERED TECHNOLOGY

Described technology has been implemented as a framework on Java programming language. RMI has been used to implement communication between server and clients. JDBC has been used to implement storing of final and intermediate computations results to external database. 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:

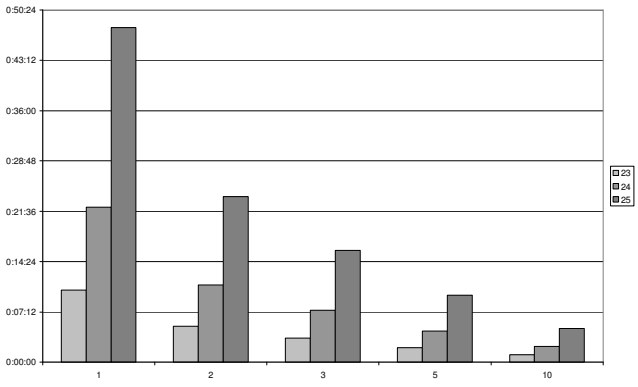

Fig. 5. Results of experimental testing of efficiency of framework.

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.

REFERENCES

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

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.

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

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.

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.

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.

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)

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.