Computing simulation method in orders based transparent parallelizing technology

Vitalij Pavlenko, Viktor Burdeinyi

Computer Control Systems
Odessa National Polytechnic University, Odessa
pavlenko_vitalij@mail.ru, vburdejny@gmail.com

Abstract

This paper is devoted to the problem of analysis of time characteristics of execution of parallel programs that use orders based transparent parallelizing technology. An approach that combines profiling and asymptotic execution time analysis is proposed. It is split into a set of stages, making it possible to repeat only some of the stages if the input data of the program, cluster configuration or scheduling algorithm changes. The proposed method can be used for estimating program execution time, finding and eliminating bottlenecks, estimating the power of cluster needed to execute the program within given time limit.

1. Introduction

The problem of estimation of time of execution of algorithms and their parts is a significant problem that is often solved by software developers. This problem is also a significant part of applied algorithms optimization. It is usually solved by profiling (test runs of the program with measuring of execution time of methods and their parts), finding asymptotic estimations of execution time of procedures and by amortized analysis of execution time [1].

In the case of parallel computing this problem is solved during algorithms optimization, estimating program execution time for a fixed cluster, estimating cluster computing power needed for the program to run within given time limit, finding and removing bottlenecks of parallel program. It becomes much more complex in the case of parallel computing because processors have no common time scale and communication between them should be taken into consideration [2]. The purpose of this paper is to propose execution characteristics analysis method for parallel applications based on the technology of orders based transparent parallelizing [3][4].

2. Technology of orders based transparent parallelizing

The technology of orders based transparent parallelizing is based on assumption that user has selected some procedures in the program. Each procedure should not modify any data during execution except values of parameters and/or 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. They can be loosened and work with global variables and I/O devices can be allowed under specific conditions, however, so strict assumptions make explaining and understanding the technology easier.

The example that will be used to illustrate the technology is shown on figure 1:

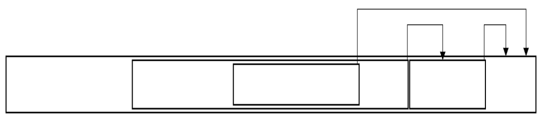

Fig. 1. Illustration of a part of program.

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

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

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

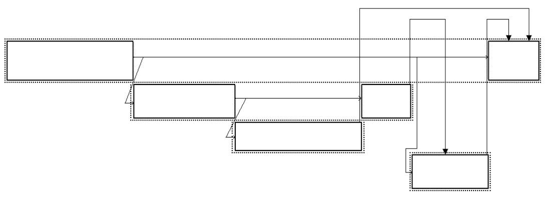

Fig. 2. An illustration of the first principle of offered technology.

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

This principle is illustrated on figure 3:

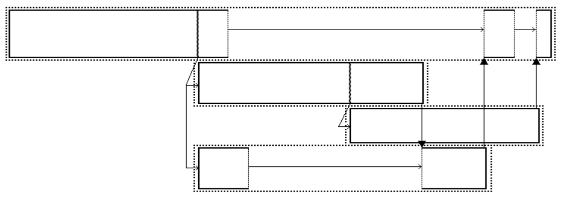

Fig. 3. *An illustration of the second principle of offered technology.*

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

An order means a unit of work but is based on parts of program source, marked as procedures in the terms of programming language, and programmers can have reasons to mark parts of source as procedures that can have nothing in common with getting high efficiency of parallel application. Theoretically there’s no problem about that because we can easily split a procedure with big execution time into a few smaller ones, and any unneeded splitting only changes order of computations and does not change efficiency of program. But from the practical point of view we can tell that having a lot of often called procedures with small execution time is a bad situation because of big overhead for selected procedures calls. So we should allow programmer to use standard way of calling the selected procedures.

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. So organization of computers interaction can be hidden from user. That can make parallel applications development much easier, but also means that a framework that implements the technology should take care or efficient usage of network. Note that a program in this technology is a set of instructions for a whole cluster (unlike programs in MPI technology that must be a set of instructions for each computer).

3. Computing simulation method

The main idea of the proposed method 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 user in any conditional operators.

For instance, we can skip blocks with asymptotic execution time known. If we know asymptotic execution time for a block, it means that we know a function $f(\alpha)$ that $\exists c_1, c_2 > 0, \forall \alpha \in A : c_1 f(\alpha) \le T(\alpha) \le c_2 f(\alpha)$, where $T(\alpha)$ is time of execution of the block, α is a value representing input parameters of the block, A is the set of all possible values of input parameters of the block. Function $f(\alpha)$ is usually a function of a few numerical characteristics of parameters of the block — such as length of an array or number of vertexes in a graph. After defining $c(\alpha) = \dfrac{T(\alpha)}{f(\alpha)}$ we can re-write the definition of asymptotic estimation in the following form: $\exists c_1, c_2 > 0, \forall \alpha \in A : c_1 \le c(\alpha) \le c_2$.

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

The proposed method consists of four stages. 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 compute values $\overline{c}(\alpha)$ for marked blocks. In order to minimize the influence of random variations of execution time, each test has to be repeated for a few times. A sample of result of execution of the second stage for a block is shown on figure 4:

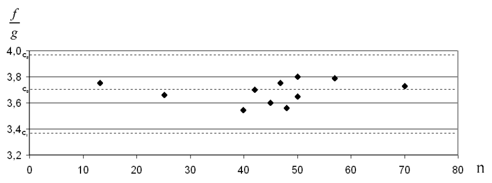

Fig. 4. Constant in asymptotic estimation of execution time.

For example, execution time and $c(\alpha)$ for algorithm that computes multiplication of two square matrixes is shown on figures 5 and 6:

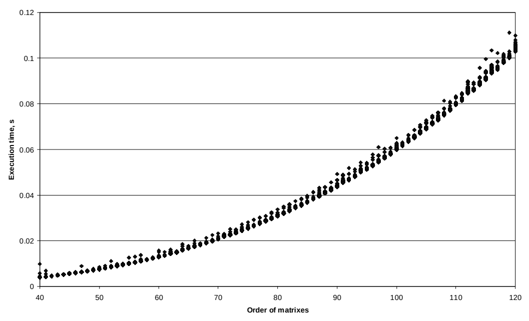

Fig. 5. Execution time of multiplication of square matrixes.

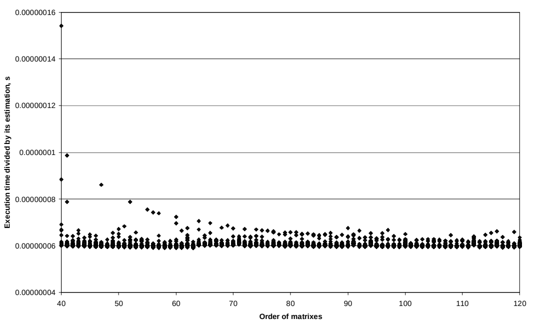

Fig. 6. Execution time of multiplication of square matrixes divided by its estimation.

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. A sample result of execution of the third stage is shown on figure 7:

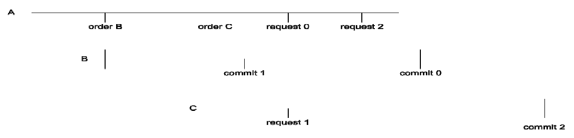

Fig. 7. Sample result of the third stage of method.

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.

4. Conclusions

The method, proposed in this paper, can be used for estimating the time of parallel program execution, finding information about bottlenecks, estimating the power of cluster, needed to solve problem within given time limit. This method combines profiling and algorithms complexity analysis.

5. References

[1] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, Charles E. Leiserson, “Introduction to Algorithms”, *McGraw-Hill Higher Education, 2001.*

[2] Voyevodin V.V., Voyevodin Vl.V., “Parallel Computations” [in Russian], *BHV, St. Petersburg, 2002.*

[3] Pavlenko V.D., Burdejnyj V.V. “Cluster Computing Using Orders Based Transparent Parallelizing”, *Information Systems Technology and its Applications, Proceedings 6th International Conference ISTA’2007, May 23–25, 2007, Kharkiv, Ukraine. – Lecture Notes in Informatics (LNI), Series of the Gesellschaft fur Informatik (GI), Vol. P–107: 152-163, Bonn 2007*

[4] Pavlenko V., Burdeinyi V., “Orders Based Transparent Parallelizing Technology”, *Signal/Image Processing and Pattern Recognition: Proceedings the Ninth All–Ukrainian International Conference UkrOBRAZ’2008, November 3-7, 2008, Kyiv, Ukraine.*