ISSN 0005-1179, Automation and Remote Control, 2007, Vol. 68, No. 5, pp. 787–798. © Pleiades Publishing, Ltd., 2007.
Original Russian Text © Yu.G. Evtushenko, V.U. Malkova, A.A. Stanevichyus, 2007, published in Avtomatika i Telemekhanika, 2007, No. 5, pp. 46–58.

TOPICAL ISSUE

Parallelization of the Global Extremum Searching Process[1]

Yu. G. Evtushenko, V. U. Malkova, and A. A. Stanevichyus

Dorodnitsyn Computer Center, Russian Academy of Sciences, Moscow, Russia

Received September 18, 2006

Abstract—The parallel algorithm for searching the global extremum of the function of several variables is designed. The algorithm is based on the method of nonuniform coverings proposed by Yu.G. Evtushenko for functions that comply with the Lipschitz condition. The algorithm is realized in the language C and message passing interface (MPI) system. To speed up computations, auxiliary procedures for founding the local extremum are used. The operation of the algorithm is illustrated by the example of atomic cluster structure calculations.

PACS number: 89.20.Ff

DOI: 10.1134/S0005117907050062

1. INTRODUCTION

In the overwhelming majority of optimization problems, it is required to find the global maximum. However, due to high complexity of these problems, the task is usually limited to finding local solutions. Modern multiprocessor computers considerably broaden possibilities for solving global optimization problems; they also allow using parallel computing algorithms. Transfer of sequential algorithms and design of new parallel computing methods of global optimization have become more urgent. Review of numerous publications on the search for global extremum is given in [1]. Problems of computing parallelization are stated in detail in [2].

In 1971, the method of nonuniform coverings for finding the global extremum of Lipschitz functions was proposed and realized in ALGOL–60. This method was further developed in [4, 5]. In this paper, we consider the parallel variant of the method of nonuniform coverings, give information on its software implementation in C in MPI system. To speed up calculations, auxiliary procedures for finding the local extremum are used. The operation of the algorithm is illustrated by the example of atomic cluster structure calculations. Computing experiments are carried out on systems MVS 6000 and MVS 15 000 [6].

2. FORMULATION OF THE PROBLEM AND THE GENERAL IDEA OF THE METHOD OF COVERINGS

Let us consider the problem of finding the global minimum of the function f determined and continuous on the compact set $P \subset R^n$

$$f_* = \operatorname{glob}\min_{x \in P} f(x). \tag{1}$$

[1] This work was supported by Program no. 14 of basic research of the Presidium of the Russian Academy of Sciences "Section 2: High-Performance Computing and Multiprocessor Systems" in 2006 and Program no. 15 of the Presidium of the Russian Academy of Sciences "Investigation and Design of Scientific Applications in the Distributed Supercomputing Environment at the Computing Centre of the Russian Academy of Sciences" in 2006.

Let X_* is a set of solutions to problem (1): f_* is a minimum value of the objective function $f(x)$. Let us introduce the set of ε-optimal (approximate) solutions to problem (1):

$$X_*^\varepsilon = \{x \in P : f(x) \leqslant f_* + \varepsilon\}. \tag{2}$$

Obviously, $X_* \subset X_*^\varepsilon \subset P$. In the most of practical problems, it is sufficient to find at least one point $x_r \in X_*^\varepsilon$ and take the number $f_r = f(x_r)$ as an optimal value of f_*. In other words, it is necessary to determine the value of the global minimum of function with the prescribed accuracy ε and find at least one point x_r where this approximate value is achieved.

Let us introduce a number of sets $B_m = [P_1, P_2, \ldots, P_m]$ from R^n and a set of n-dimensional points $N_m = [c_1, c_2, \ldots, c_m]$ such that $c_i \in P_i \subset P$ for all $1 \leqslant i \leqslant m$. Let the union m of sets P_i is

$$U_m = \bigcup_{i=1}^{m} P_i.$$

We say that the union of sets P_i covers the set P, if

$$P = U_m. \tag{3}$$

At each point c_i is computed the value of the function f, and by the formula

$$R_m = \min_{1 \leqslant i \leqslant m} f(c_i) = f(c_r) \tag{4}$$

we determine the record point c_r and the record R_m.

Assume that the function f and the set P_i are so that for all $x \in P_i$ is met the condition

$$f(x) \geqslant R_m - \varepsilon. \tag{5}$$

The basis of the method of nonuniform coverings is the following easy-testable assertion

Assertion 1. *Let the number of sets B_m and the function f are such that $U_m = P$ and condition (5) is fulfilled for all $1 \leqslant i \leqslant m$. Then the record point is $c_r \in X_*^\varepsilon$.*

Indeed, if the number of sets P_i completely covers P, then each point x_* in X_* will belong at least to one of the sets P_i. Let $x_* \in P_s$, $s \leqslant m$, then taking into consideration (5), we obtain

$$f_* = f(x_*) \geqslant R_m - \varepsilon = f(c_r) - \varepsilon.$$

According to (2), the record point c_r belongs to the set of ε-solutions to problem (1).

This assertion expresses the main idea of the method of nonuniform coverings: instead of the global minimum on P are sought global minimums on subsets, whose union coincides with P. The sets P_i can be various. For example, on each P_i, the function f can meet the Lipschitz condition with its constant L_i. The function can be convex or twice differentiable in some domains. It is reasonable to take into accounts all these peculiarities for speeding up computations.

At realizing the method, the sets B_m and N_m are sequentially constructed; after each computation of the value of the function f, refinement of the record is conducted. We suppose that the last computation of f was fulfilled at the point c_m; the record point was c_r; and the record, R_m. Determine the following Lebesgue set:

$$\mathcal{L}_m = \{x \in P : R_m - \varepsilon \leqslant f(x)\}. \tag{6}$$

Let the global minimum on $\mathcal{L}_m$ is achieved at the point x_l. Then $R_m - f(x_l) = f(c_r) - f(x_l) \leqslant \varepsilon$ and, hence, as a result of global minimization on $\mathcal{L}_m$, the record R_m can be precised no more than ε. Therefore, the set $\mathcal{L}_m$ is out of interest and can be excluded from the search domain of P.

If the condition $P = U_m$ is not fulfilled after the computations described, then the search for minimum is continued on the set $W_m = P \backslash U_m$. Let $c_{m+1} \in P_{m+1} \subseteq W_m$ are determined, $f(c_{m+1})$

and R_{m+1} are computed. If on P_{m+1}, the condition $f(x) \geqslant R_{m+1} - \varepsilon$ is fulfilled, then the set P_{m+1} can be excluded and the search for minimum is continued on $W_m \backslash P_{m+1}$. Otherwise, the set P_{m+1} is parted into several subsets; in each of them, values of the function f are computed, the record is recomputed, some subsets are excluded, another are parted, and etc. The process is finished when the whole set P will be completely covered. The sequences of records is monotonically decreasing. The sequence of Lebesgue sets $\mathcal{L}_m$ is, on the contrary, increasing, i.e., $\mathcal{L}_m \subseteq \mathcal{L}_{m+k}$ with $k \geqslant 0$. Thus, if with some $s \leqslant m$ from P was excluded the set P_s, then P_s can be excluded from P and with all $s \geqslant m$.

The Lebesgue set $\mathcal{L}$ will be the largest, if to let $R_m = f_*$ in (6). However, a priori the quantity f_* is unknown. Therefore, to decrease the record, it is reasonable to use the methods of local search for the minimum of the function f on the set P. At the points $c_i \in P$ where we obtained that $f(c_i) < R_m$, the record is bettered; there are reasons for supposing that solving the problem of local search

$$\operatorname{loc} \min_{x \in P} f(x),$$

and starting from the point c_i, we will be able to find another point $\overline{c} \in P$ where $f(\overline{c}) < f(c_i)$. This technique considerably speeds up computations; it allows bettering the record and thus broadening the domain that can be excluded in the process of covering the set P.

The most difficult stage of implementation of the extremum search algorithm is determination at least a part of the set $\mathcal{L}_m$. Here we have to impose on the function f supplementary requirements. Assume that we can construct a minorant $g(c_i, x)$ for $f(x)$ such that for all $x, c_i \in P$ are fulfilled the inequalities

$$g(c_i, x) \leqslant f(x), \quad g(c_i, c_i) = f(c_i). \tag{7}$$

Determine the set

$$N_i = \{x \in P : R_m - \varepsilon \leqslant g(c_i, x)\}. \tag{8}$$

From (7) it follows that $N_i \subseteq \mathcal{L}_m$; thus, the set N_i can be excluded from the search domain of P. Let us give two simplest cases when minorants that are satisfy (7) are easy to construct. Let the function f satisfy the Lipschitz condition with the constant L, i.e., for any x and c_i in P is fulfilled the inequality

$$|f(x) - f(c_i)| \leqslant L\|x - c_i\|_\infty. \tag{9}$$

Here and hereinafter we use the pth Holder norm

$$\|v\|_p = \left(\sum_{j=1}^{n} |v^j|^p \right)^{1/p},$$

that with $p = \infty$ coincides with the Chebyshev norm

$$\|v\|_\infty = \max_{1 \leqslant j \leqslant n} |v^j|.$$

Then for all $x \in P$ we have

$$f(x) \geqslant f(c_i) - L\|x - c_i\|_\infty.$$

Thus we can take the minorant that meets (7)

$$g(c_i, x) = f(c_i) - L\|x - c_i\|_\infty. \tag{10}$$

Let us introduce the set

$$K_i = \{x \in R^n : \|x - c_i\|_\infty \leqslant \rho_{im}\}, \tag{11}$$

where

$$\rho_{im} = (f(c_i) - R_m + \varepsilon)/L. \tag{12}$$

Here K_i is a cube with the center at the point c_i and the main diagonal $2\rho_{im}$ (or if $p = \infty$, the sphere with the center at the point c_i and the radius ρ_{im}).

If $c_i \in P$ for all $1 \leqslant i \leqslant m$, then according to (5) $f(c_i) \geqslant f(c_r)$, thus the radius ρ_{im} prescribed by the formula (12) will be greater or equal to ε/L:

$$\min_{x \in K_i} g(c_i, x) = f(c_i) - L \max_{x \in K_i} \|x - c_i\|_\infty = f(c_i) - L\rho_{im} \geqslant R_m - \varepsilon.$$

Hence, the cube K_i can be excluded from the search domain of P. If the union of cubes K_i, $1 \leqslant i \leqslant m$ that satisfy (10) cover the set P, then $c_r \in X_*^\varepsilon$.

The radius of the sphere K_i will be large if $f(c_i) \gg f(c_r)$ and small in domains where $f(c_i) \approx f(c_r)$. Thanks to this, the nonuniform covering of the set P by different-radius spheres is possible.

If except (9) the condition of the form $\|f'(x) - f'(z)\|_1 \leqslant M\|x - z\|_\infty$ is fulfilled and σ there is a scalar product of the vectors $f'(c_i)$ and $x - c_i$, then

$$f(x) - f(c_i) \geqslant \sigma - M\|x - c_i\|_\infty \geqslant -\|x - c_i\| \times \|f'(c_i)\|_1 - M\|x - c_i\|_\infty^2.$$

Joining this with (10), we obtain that the minorant will be

$$g(c_i, x) = f(c_i) - \|x - c_i\| \times \min\left\{L, \|f'(c_i)\|_1 + M\|x - c_i\|_\infty\right\}.$$

There are other techniques for determining the lower estimate of the function f.

3. THE ALGORITHM OF SEQUENTIAL COVERING

In the process of program implementation of the method, supplementary simplifying assumptions are introduced. Assume that the feasible set P is a n-dimensional parallelepiped with the faces parallel to coordinate planes:

$$P = \{x \in R^n, a \leqslant x \leqslant b\}.$$

Here and hereinafter the inequality $a \leqslant x$ implies that $a^j \leqslant x^j$ for all $1 \leqslant j \leqslant n$.

During computing are used auxiliary vectors $a_i, b_i \in R^n$ and rectangular parallelepipeds generated by them with faces parallel to the coordinate planes: $P_i = \{x \in R^n : a_i \leqslant x \leqslant b_i\}$.

Let us assume that all vectors are $a_i \geqslant a$ and $b_i \leqslant b$. Thus all parallelepipeds are $P_i \subseteq P$. As points c_i, we take the centers of the parallelepipeds P_i:

$$c_i^j = (a_i^j + b_i^j)/2, \quad 1 \leqslant j \leqslant n.$$

The vector of the main diagonal d_i of the parallelepiped P_i has components $d_i^j = b_i^j - a_i^j$, $1 \leqslant j \leqslant n$.

We consider that the function f satisfies Lipschitz condition (9) everywhere on P with the same constant L. Let us use minorant (10). Let us set

$$y_i = \min_{x \in P_i} g(c_i, x) = f(c_i) - \frac{L}{2} \max_{1 \leqslant j \leqslant n} \|d_i^j\| = f(c_i) - \frac{L}{2}\|d_i\|_\infty. \tag{13}$$

If $y_i \geqslant R_m - \varepsilon$, then the parallelepiped P_i can be excluded as the global minimum on it does not improve the current record R_m more than ε and the search is continued on the set $P \backslash P_i$. Otherwise, if $y_i < R_m - \varepsilon$, then the parallelepiped P_i is halved along the maximal rib and the search is conducted in these two parts. If the main diagonals of one or two parts is smaller than $2\varepsilon/L$, these parts are excluded from the search domain. In the centers of the remained parts are computed values of $f(x)$.

And also, the record is bettered as far as possible and the condition for covering both parts is tested. The parts covered are excluded from the search domain. If both parts are excluded , then P_i is excluded; if no, then the further partition is continued till the parallelepiped P_i is not covered. Various techniques for covering the parallelepiped P are possible. In [3], e.g., layered covering was realized by recursive procedures in ALGOL–60. This variant was used to solve global optimization problems when $n \leqslant 15$. Below, following [4], the branch-and-bounds method is applicable for covering P. The similar approach for knapsack-type problems was used in [7].

Let us connect the set $S_i = (c_i, d_i, y_i)$ with each parallelepiped P_i. Let us identify the aggregate of sets S_i for the whole set of uncovered parallelepipeds B_m as a list of sets and denote it $S = \{S_1, S_2, \ldots, S_m\}$. The record $S = \varnothing$ implies that the list S is empty.

In the formulation of the algorithm, the letter k will denote the ordinal number of an iteration of the main cycle.

Starting operation. Let $k = 1$, $m = 1$, $P_1 = P$. Set $\varepsilon > 0$, L and some point $x_0 \in P$. Compute c_1, d_1, $f(c_1)$, $f(x_0)$, y_1, $z = \varepsilon/L$, $\widetilde{R} = \min\{f(c_1), f(x_0)\}$. Let $N_1^{(1)} = \{c_1\}$, $B_1^{(1)} = \{P_1\}$, $S_1 = (c_1, d_1, y_1)$, $S^{(1)} = \{S_1\}$. If $\widetilde{R} = f(c_1)$, then to take c_1 as a record point; otherwise, the point, x_0. If $y_1 \geqslant \widetilde{R} - \varepsilon$, then finish the work.

The main cycle. Repeat the following steps 1–5 till $S^{(k)} \neq \varnothing$, i.e., till the set P is not covered completely.

(1) Choose the parallelepiped P_s from the current set of parallelepipeds $B_m^{(k)}$, for which

$$y_s = \min_{1 \leqslant i \leqslant m} y_i.$$

(2) Determine t, i.e., the number of the largest rib, in the parallelepiped P_s:

$$d_s^t = \max_{1 \leqslant j \leqslant n} d_s^j.$$

(3) Halve the parallelepiped P_s along the tth coordinate, thus generating two new parallelepipeds P' and P''. Let us denote their centers and main diagonals as c', d' and c'', d'' respectively. Exclude the parallelepiped P_s from the set $B_m^{(k)}$, remove the set S_s from the list $S^{(k)}$, and let $m = m - 1$.

(4) Compute

$$\widetilde{R} = \min\{f(c'), f(c'')\} \tag{14}$$

and using (12) determine $y' = f(c') - Ld'/2$, $y'' = f(c'') - Ld''/2$.

If $\widetilde{R} \geqslant R^{(k)}$, then let $R^{(k+1)} = R^{(k)}$. If $y' < R^{(k+1)} - \varepsilon$ and $d' \geqslant 2z$, then include $S' = (c', d', y')$ into the list of sets and let $m = m + 1$; if $y'' < R^{(k+1)} - \varepsilon$ and $d'' \geqslant z$, then include $S'' = (c'', d'', y'')$ into the list of sets and let $m = m + 1$.

If $\widetilde{R} < R^{(k)}$, then let $R^{(k+1)} = \widetilde{R}$. As a record point x_r, take the point c', c'' where is achieved minimum (14). Each set S_i in $\{S_i\}_{1 \leqslant i \leqslant m}$ that satisfies the condition $y_i \geqslant R^{(k+1)} - \varepsilon$ exclude from this list. Renumber a new list of sets $\{S_{i1}, S_{i2}, \ldots, S_{im}\}$ and denote as $S^{(k+1)} = \{S_j\}_{1 \leqslant j \leqslant m}$.

(5) Let $k = k + 1$.

It is to be noted that the main diagonal of the parallelepiped obtained by the described splitting is not smaller than $2\varepsilon/L$. Taking this fact into consideration, it is easy to prove that the following assertion is true

Assertion 2. *Let the function f in problem* (1) *complies with the Lipschitz condition with the constant L on the $n$dimensional parallelepiped P. Then the described algorithm determine the record point $x_r \in X_*^\varepsilon$ in the finite amount of computations of values of the function f.*

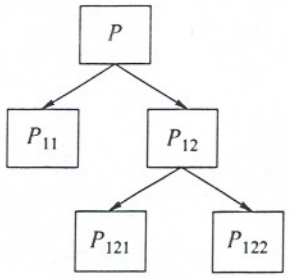

Fig. 1. The branching tree for the method of nonuniform coverings.

After finishing the computations, the set P will be completely covered by the parallelepipeds P_i, and the record point x_r will be ε-solution to problem (1). We can use other rules of selecting the parallelepiped from the current set that are differed from the indicated at step 1:

- select the parallelepiped P_s that was first included into the current amount of parallelepipeds;
- select the parallelepiped P_s that was last included into the current amount of parallelepipeds.

The operation of this algorithm is considerably increased by procedures of the local search for extremum. In the computing results below, the conjugate gradient method with the operation of projecting onto the parallelepiped P was used for the local search. The programming implementation was conducted by N.I. Grachev.

The indicated algorithm can be interpreted as a branch-and-bound method [7]:

- at step 1 is conducted the choice of the set for branching;
- at step 2 is realized the rule that defines the branching technique;
- at step 3 is conducted branching;
- at step 4 according to formula (13) are computed estimates of y', y'' that correspond to new set–candidates for branching, is recomputed the record R, and are screened the sets (if the record is not bettered) that can be excluded from the search domain.

One parallelepiped from the current amount is halved into two new parallelepipeds along the largest rib by the plane that is parallel to the coordinate plane at each step of the method described above. The process of half partition of parallelepipeds can be interpreted as growth of the binary tree (Fig. 1). Parallelepipeds obtained from the tree by partition correspond to its knots. Arcs connect this parallelepiped with the parallelepipeds obtained from it as a result of partition. Parallelepipeds that correspond to leaf knots (P_{11}, P_{121}, P_{122}) of the tree form a current amount of parallelepipeds. Some leaf knots can be excluded from this amount according to the screening rule (see step 4).

For short, let us further identify the current amount of parallelepipeds as a pool; actions of the main cycle described by steps 1–5, iteration.

4. DESCRIPTION OF THE PARALLEL ALGORITHM

The main idea of the parallel algorithm is to conduct iterations simultaneously, with the periodic exchange of records and redistribution of search domains among processors. For parallel performance of iterations, it is necessary to have a technique for distributing parallelepipeds from pools among processors.

Let us first consider the variant of the algorithm where parallelepipeds are not transferred from one pool to another. Assume that there are p processors for computations. The simplest parallelization procedure is as follows: let us divide P into p parts; we obtain an amount of parallelepipeds P_i that satisfies condition (3). We transfer each ith parallelepiped to the ith processor $1 \leqslant i \leqslant p$. We

give a task to each ith processor: using the sequential algorithm described in the previous section, we find the global minimum of the function f on the set P_i. When all processors fulfill their tasks, select the best record from the obtained p. This record will be ε-solution to the original problem. This algorithm is attractive by its simplicity. At the same time, it has several disadvantages. We indicate them:

• Computing time on different processors can considerably differ. As a result, the situation when one processors quickly finishes their computations and then stand idle whereas others, on the contrary, are always busy with computing is typical. Computation time is determined by the processor that operates longer than others.

• Due to the absence of information interchange, the obtained records are not transferred to other processors; computations cannot be sped up.

• There is no possibility of joining idle processors and unload occupied.

Let us consider the variant of the parallel algorithm, in which there is periodically conducted the exchange of records and redistribution of search domains among processors during computing.

Let there is an initial sequence $B_m = \{P_1, P_2, \ldots, P_m\}$ of parallelepipeds P_i,, which belong to P constructed, e.g., in the process of operation of the sequential algorithm. Let $N_m = \{c_1, c_2, \ldots, c_m\}$ is a sequence of the centers of these parallelepiped. Let the current record R_m and the current record point x_r are computed over (4).

The technique of work distribution among the processors is closely connected with the choice of the unit of work. Let us define the computing work connected with Q-divisible ($Q \geqslant 1$) execution of iteration of the sequential algorithm as a unit of work.

Let besides $p = m$ working processors there is a processor called a dispatcher. As previously, let us distribute parallelepipeds of the initial amount B_m among the working processors.

In the execution process of iterations of the sequential algorithm, each working processor sustains its individual pool. We say that the processor is finishing its work, if it has carried out Q cycles of the sequential algorithm (that Q half divisions are executed) or if its individual pool is empty. The processor that has finished its work reported to the dispatcher the following quantities: the number of parallelepipeds in the individual pool, minimum estimate y for each of them, and its individual record (i.e., the best value of the function f obtained in the process of computing). This processor is enqueued by the dispatcher in the line of waiting processors. The waiting processor is called free if its individual pool is empty.

Let us consider the method of communication between the processors. The exchange of information would be ideal if each processor reported to the rest processors the best obtained record, the number of available parallelepipeds, and values of the lower estimate for parallelepipeds in the current individual amount. However, too frequent exchanges between the processors may result in excessive increase in communication cost and decrease in efficiency of the parallel algorithm on the whole.

In the proposed variant of the algorithm, each processor sends the dispatcher the data stated above in asynchronous mode, i.e., independently of the other processors. Basing on the data received, the dispatcher betters the record R and reports it to all waiting processors that will use the new record for screening parallelepipeds from their amounts. The dispatcher reveals those processors among the waiting that have only one parallelepiped in its individual amount and orders them to repeat the main cycle Q times taking new R as a record. The same order is obtained by the other waiting processors if at the moment there are no free processors.

As soon as the dispatcher reveals the free processor, it selects the free processor among the waiting for transferring the parallelepiped with the minimum lower estimate. Selection of such a pair is accomplished by two orders: the free processor is ordered to accept the parallelepiped

from the waiting processor; the waiting processor is ordered to pass the parallelepiped to this free processor.

The dispatcher finishes the algorithm if all its working processors are free.

In the proposed algorithm, reference frequency of processors to the dispatcher depends on the parameter Q. As soon as a free processor is found, redistribution of computing work among the processors by transferring parallelepipeds is carried out. With this scheme of work organization, the processors fulfill the algorithm in asynchronous mode and exchange information only while addressing the dispatcher. As a result, the process of algorithm realization becomes "nondeterministic," i.e., the sequence of generated parallelepipeds and the obtained solution may be different for different launches of the same problem (even with the same amount of processors).

Let us describe the parallel algorithm for the scheme of processors interaction stated above. To exchange data between the parallel processors, we propose to use the explicit message passing. The message sending from the working processor to the dispatcher, contains information on values of the following quantities:

- the number of parallelepipeds in the current pool;
- minimum estimates of y for all parallelepipeds of the current pool;
- the individual record $R = f(c)$.

The dispatcher sends the working processors the following message-commands:

- $work(Q, R)$ is to carry out the main cycle Q times considering R a current record;
- $take(i)$ is to accept the parallelepiped from the processor i;
- $give(j)$ is to pass the parallelepiped to the processor j;
- $finish$ is to finish the work.

The Algorithm for the Processor-Dispatcher

(1) Announce all working processors free except one, e.g., p_1. Announce the processor p_1 waiting and indicate that the only parallelepiped in its pool is initial; accept the record $R = f(x_0)$ where $x_0 \in P$ is a prescribed initial point; let the estimate y is equal to the estimate of the initial parallelepiped; and prescribe the parameter $Q \geqslant 1$.

(2) Meanwhile all processors are not free, repeat following actions 3–6.

(3) If there are no free processors, then give the command $work$ to each processor and make the list of waiting processors empty.

(4) Give the command $work(Q, R)$ to each waiting processor with the single parallelepiped in the pool and remove it from the list of waiting processors.

(5) (The cycle of selecting the pairs of processors for parallelepipeds transferring.) Meanwhile the lists of waiting and free processors are not empty, select the processor i with the minimum estimate among waiting; and some processor j, among free; delete all selected processors from the corresponding lists; send the first processor the command $give(j)$; the second, the command $take(i)$.

(6) Accept messages coming from processors; put the processor that sent the message on the list of waiting or free processors depending on the number of parallelepipeds in its pool. Recompute the current record taking into account the data obtained from the processors.

(7) Give each processor (they all are free) the command $finish$.

The Algorithm for the Working Processor

(1) Till the command $finish$ has not been received, fulfill actions 2–4.

(2) Accept the following command from the dispatcher.

(3) If the command $give(j)$ or $take(i)$ is obtained, then fulfill it and send the dispatcher a message about the number of its parallelepipeds and the minimum estimates of y.

(4) If the command $work(Q, R)$ is obtained, then carry out all iterations till the individual pool is not empty and less than Q iterations are fulfilled; send the dispatcher a message about the results of the work.

(5) If the command $finish$ is obtained, then finish the work.

5. RESULTS OF NUMERICAL EXPERIMENTS

For numerical experiments was used the function that defines energy of Morse atomic clusters which consist of n atoms:

$$f(x_1, x_2, \ldots, x_n, \rho) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\left(e^{\rho(1-\|x_i - x_j\|)} - 1 \right)^2 - 1 \right),$$

where ρ is a scalar parameter, $x_i \in R^3$ and $x_j \in R^3$ are three-dimensional vectors of coordinates of the atom centers i and j respectively.

The problem is in defining the Cartesian coordinates of n atoms so that the value of the function f achieves the global minimum with the fixed value of the parameter ρ. The quantity ρ was accepted equal to 3, 6, and 8; the number of atoms of the cluster amounted to 80; and the number of sought desired variables was 240. Assumed that the function f complies with the Lipschitz condition with some constant L. Since the value of L is a priori unknown, then several computations were carried out: first with the small value of L; and after finding some record, the value of L gradually increased so that the computing time was acceptable.

For computing, the computer system MVS15000 of the Interdepartmental Supercomputer Center of the Russian Academy of Sciences was used [6]. The results of numerical experiments were compared to the computations carried out at Cambridge Cluster Database.

The structure of the cluster of 25 atoms is represented in Fig. 2. The values of ρ were taken equal to 3 and 6. The results of computations with great accuracy coincided with the results represented in the database above. The maximum value of the Lipschitz constant was 26.3; accuracy, $\varepsilon = 0.1$. In the determination of the feasible set P, all elements of the vector a were taken equal to -2.1; of the vector b, to 1.5. The conjugate gradient method was used for the local search. The computations started from the same initial structure of the cluster in all cases.

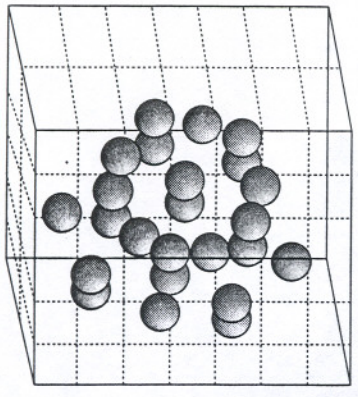

Fig. 2. The structure of the cluster ($n = 25$, $\rho = 6$).

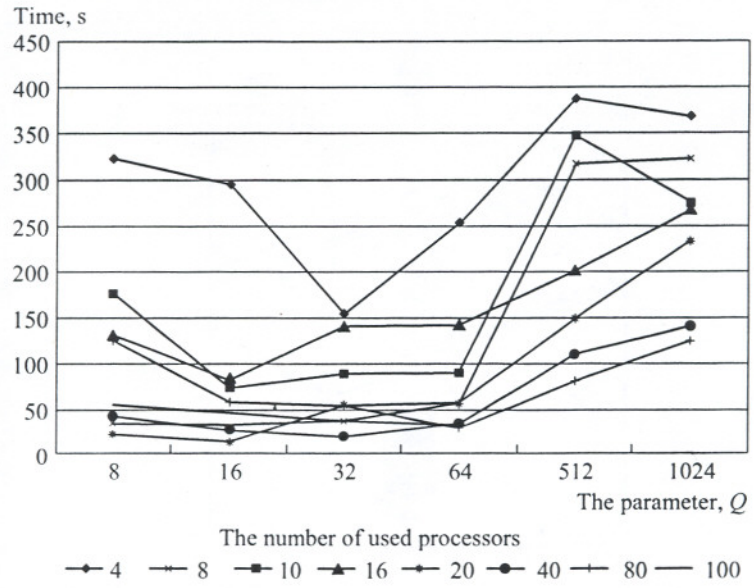

Fig. 3. Dependence of the computation time on the parameter Q.

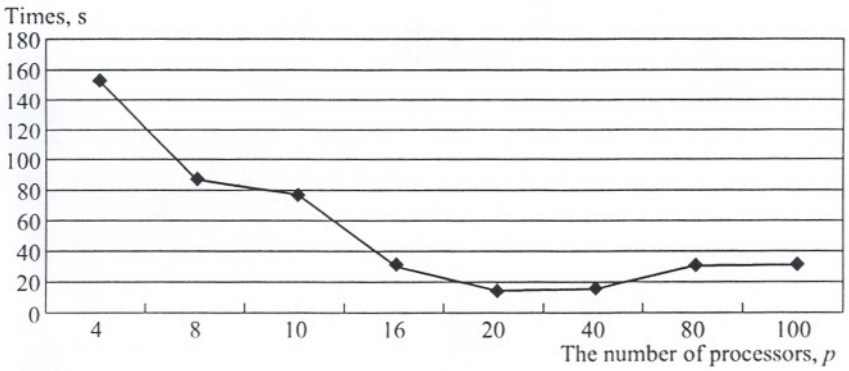

Fig. 4. Dependence of the computational time of the number of processors.

Dependence of the computation time on the parameter Q is represented in Fig. 3 at using the different numbers of processors. Computations demonstrated that the smallest computation time is achieved if $20 \leqslant Q \leqslant 40$.

Dependence of the computational time of the number of processors p is demonstrated in Fig. 4. The best value of the parameter Q determined in Fig. 3 was taken for each number of processors. It follows from this figure that the computation time decreases with the increase in the number of processors up to 20. With the further increase in the number of processors, the computation time is not improved due to the increase in the idle time, unbalanced load of processors, and overheads connected with communication exchanges. With $p = 4$, the computation time is $t = 154$ s; with, $p = 20$, is $t = 15$ s.

Figure 5 demonstrates by how many times the computations accelerate in the increase in the number of processors p compared to 4 processors. With $p = 10$ is achieved double acceleration; with $p = 20$, tenfold. This acceleration is obtained due to the considerable reduction of the number of parallelepipeds in pools and frequent exchange of information on the current record between

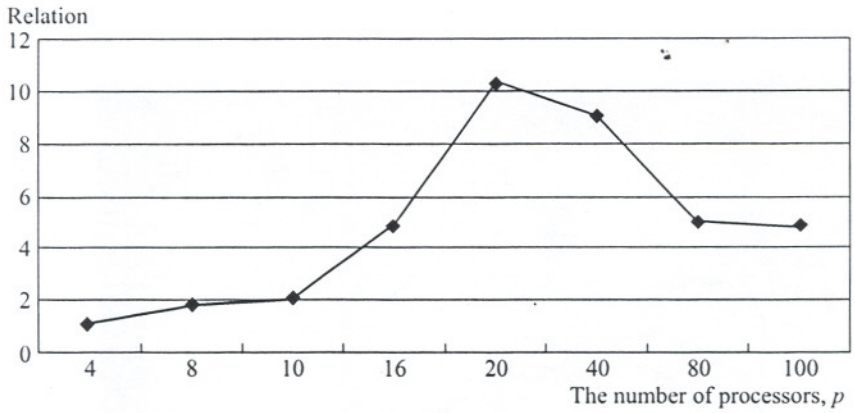

Fig. 5. Dependence of the computation rate on the number of processors.

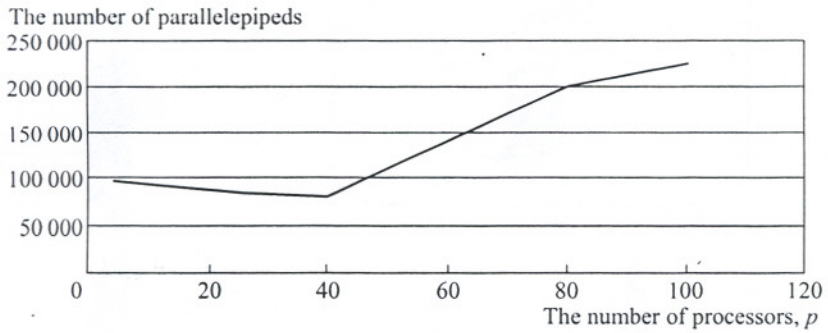

Fig. 6. Dependence of the number of parallelepipeds on the number of processors.

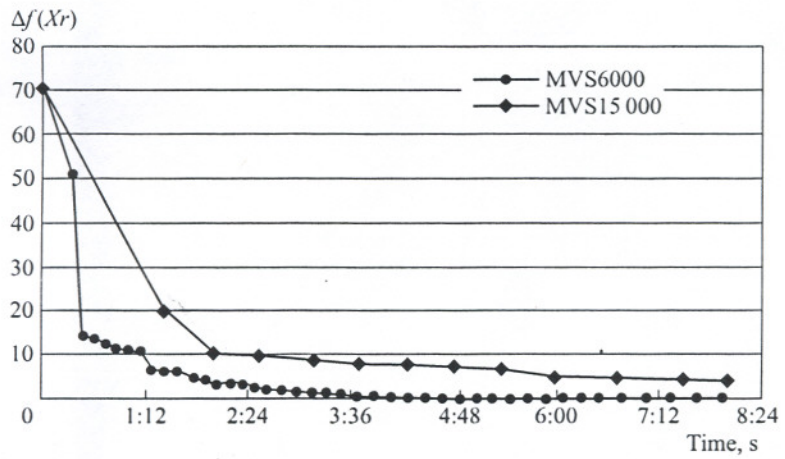

Fig. 7. The computation time on two computer systems.

processors. However, when the number of processors is more than 40, computations become slower due to the overheads mentioned above.

Figure 6 represents dependence of the number of all parallelepipeds generated in the process of their computing on the number of processors. With $p = 100$, this number exceeded 200 000.

To compare efficiency of two different computer systems MVS6000 and MVS15 000, each of them was used for computing the optimal structure of the cluster that contained 80 atoms. In both cases,

50 processors were ordered for computing. The maximum computation time was 8 h. The identical set of parameters of the algorithm was used: $\varepsilon = 0.01$, $n = 240$, $\rho = 3$, $L = 1.28$. On the computer system MVS6000 were constructed 70 499 parallelepipeds and obtained $f_* = -690.578$, i.e., was achieved the quantity indicated by Cambridge Cluster Database as optimal. This quantity was obtained for 5 h 20 min. On the second computer system MVS15 000, the achieved record for 8 h amounted to -686.306, the system was less efficient as it constructed only 28 653 parallelepipeds. Thus, the amount of computing work carried out by the first system was by 2.5 larger than by the second; hence, we obtained a more precise answer. Dependence of the quantity $\Delta f(x_r) = |f(x_r) - f_*|$ on the computation time for both systems is represented in Fig. 7.

6. CONCLUSIONS

The parallel variant of the method of nonuniform coverings described in this paper allows conducting global optimization of Lipschitz functions for the case with low dimension (less than 200 variables). Efficiency of parallelization is affected not only by the choice of values of the Lipschitz constant L and the parameter Q (see Section 4) but also the strategy of search for the global optimum.

The proposed method of the parallel global search can be applied to solving problems of multicriterion optimization. The simplest technique is in using the results of the work [8]. This method is also transferred to solving problems of search for the global minimax similarly to [9]. Moreover, it can be used for minimization of Lipschtz functions with the supplementary requirement for all components of the vector x to be integer.

We are grateful to A.Ya. Belyankov, Researcher of the Computing Center of the Russian Academy of Sciences for careful reading of the manuscript and helpful remarks.

REFERENCES

1. Strongin, R.G. and Sergeyev, Ya.D, *Global Optimization with Non-Convex Constraints. Sequential and Parallel Algorithms*, Dordrecht: Academic, 2000.
2. Voevodin, V.V. and Voevodin, Vl.V., *Parallel'nye vychisleniya* (Parallel Computing), St. Petersburg: BKhV-Peterburg, 2002.
3. Evtushenko, Yu.G., Numerical Method for Finding the Global Extremum of a Function (Enumeration of a Nonuniform Grid), *Zh. Vychisl. Mat. Mat. Fiz.*, 1971, vol. 11, no. 6, pp. 1390–1403.
4. Evtushenko, Yu.G. and Rat'kin, V.A., The Half-Division Method for the Global Function of Several Variables, *Tekh. Kibern.*, 1987, no. 1, pp. 119–127.
5. Belyankov, A.Ya., Numerical Efficiency Increase for Methods of Nonuniform Covering in Global Optimization, in *Metody matematicheskogo programmirovaniya i programmnoe obespechenie. Tezisy dokl.* (Mathematical Programming Methods and Software. Theses Rep.), Sverdlovsk: Akad. Nauk SSSR, 1989, pp. 21–22.
6. *Interdepartmental Supercomputer Center of the Russian Academy of Sciences*, website: http://www.jscc.ru.
7. Posypkin, M.A. and Sigal, I.Kh., Investigation of Algorithms for Parallel Computations in Knapsack-Type Discrete Optimization Problems, *Zh. Vychisl. Mat. Mat. Fiz.*, 2005, vol. 45, no. 10, pp. 1801–1809.
8. Evtushenko, Yu.G. and Potapov, M.A., Solution Methods for Multicreiterion Problems, *Dokl. Akad. Nauk SSSR*, 1986, vol. 291, no. 1, pp. 25–29.
9. Evtushenko, Yu.G., Numerical Method for Finding the Best Guaranteed Estimates, *Zh. Vychisl. Mat. Mat. Fiz.*, 1972, vol. 12, no. 1, pp. 89–104.

This paper was recommended for publication by A.I. Kibzun, a member of the Editorial Board