Use of Xeon Phi Coprocessor for Solving Global Optimization Problems

Konstantin Barkalov(✉), Victor Gergel, and Ilya Lebedev

Lobachevsky State University of Nizhni Novgorod, Nizhni Novgorod, Russia
{barkalov,lebedev}@vmk.unn.ru, gergel@unn.ru

Abstract. This work considers a parallel algorithm for solving multidimensional multiextremal optimization problems. The issue of implementation of the algorithm on state-of-the-art computing systems using Intel Xeon Phi coprocessor is considered. Speed up of the algorithm using Xeon Phi compared to using only CPU is experimentally confirmed. Computational experiments are carried out using a set of a several hundred of multidimensional multiextremal problems.

Keywords: Global optimization · Dimension reduction · Parallel algorithms · Speedup · Intel Xeon Phi

1 Introduction

Optimization problems are of great practical importance. Almost each problem of design of new devices, products or systems includes a stage where optimal variants are selected. Among the most complex optimization problems are problems of global optimization, where the criterion of optimality is multiextremal. While validation of local optimality of a solution requires only analysis of its local neighborhood, global minimum is an integral characteristic of the optimization problem solved and it requires analysis of the whole search domain. As a result, search of a global optimum is reduced to construction of a grid in the parameter domain. It leads to exponential growth of computational effort with more dimensions (the so-called "curse of dimensionality").

A decrease in computational effort can be provided through construction of a non-uniform grid in the search domain: it has to be quite dense in the neighborhood of the global optimum and more sparse farther from the required solution. There is a number of methods allowing to build non-uniform grids of such kind (see, for example, [1–4]). Among those, we note the global search algorithm and its modifications developed within the framework of the information-statistical approach [5–9]. It is experimentally confirmed in [8], that this algorithm is more effective than other known methods of the same purpose.

Use of parallel computing systems significantly expands capabilities for solving global optimization problems. Parallel versions are proposed for almost all existing algorithms (see, for example, [10, 11]). However, the provided versions of algorithms are parallelized in a CPU using MPI and/or OpenMP technologies, whereas currently the main tendency in the field of parallel computing is use of accelerators. Of special

V. Malyshkin (Ed.): PaCT 2015, LNCS 9251, pp. 307–318, 2015.
DOI: 10.1007/978-3-319-21909-7_31

interest in this regard is the Intel Xeon Phi coprocessor. It is based on x86 architecture and standard technologies and libraries can be used in programming for Xeon Phi (unlike specialized and, as a rule, more complex programming technologies for GPU).

The present work contains the results of an analysis of parallel global search algorithm developed within the framework of the information-statistical approach [8], and its implementation using Xeon Phi.

2 Global Search Algorithm with Parallel Trials

Let us consider the problem of global minimum search of an N-dimensional function $\varphi(y)$ in hyperinterval D

$$\varphi(y^*) = \min\{\varphi(y) : y \in D\}, \tag{1}$$

$$D = \{y \in R^N : a_i \le y_i \le b_i,\ 1 \le i \le N\}.$$

Let us assume that objective function $\varphi(y)$ satisfies Lipschitz condition

$$|\varphi(y_1) - \varphi(y_2)| \le L\|y_1 - y_2\|,\ \ y_1, y_2 \in D,$$

with constant L, which in the general case is unknown.

The considered approach reduces solving multidimensional problems to solving equivalent one-dimensional problems (*reduction of the dimension*). Thus, use of continuous single-valued mapping like the Peano curve

$$\{y \in R^N : -2^{-1} \le y_i \le 2^{-1},\ 1 \le i \le N\} = \{y(x) :\ 0 \le x \le 1\}$$

allows reduction of the problem of minimization in domain D to a problem of minimization on interval [0,1]

$$\varphi(y^*) = \varphi(y(x^*)) = \min\{\varphi(y(x)) : x \in [0,1]\}$$

Problems of numerical construction of Peano-type space filling curves and the corresponding theory are considered in detail in [8, 13]. Here we will note that a numerically constructed curve (*evolvent*) is an approximation to a theoretical Peano curve with accuracy 2^{-m}, where m is an evolvent construction parameter. An important property is preservation of boundedness of function relative differences: if function $\varphi(y)$ in domain D satisfies Lipschitz condition, then function $\varphi(y(x))$ on interval [0,1] will satisfy a uniform Hölder condition

$$|\varphi(y(x_1)) - \varphi(y(x_2))| \le H|x_1 - x_2|^{1/N},\ \ x_1, x_2 \in [0,1],$$

where Hölder constant H is linked to Lipschitz constant L by the relation

$$H = 2L\sqrt{N+3}.$$

Therefore, it is possible, without limitation of generality, to consider minimization of one-dimensional function

$$f(x) = \varphi(y(x)),\ x \in [0,1],$$

satisfying Hölder condition.

An algorithm for solving problem (1) (let us formulate it here according to [12]) involves constructing a sequence of points x^i, where the values of the minimized function $z^i=f(x^i)=\varphi(y(x^i))$ converging to the solution of the problem are calculated. Let us call the function value calculation process (including construction of image of $y^i=y(x^i)$) the trial, and pair (x^i, z^i) – the result of the trial. At each iteration of the method p of trials is carried out in parallel, and the set of pairs $\{(x^i, z^i)\}$, $1 \le i \le k = np$, make up the search information collected by the method after carrying out of n steps. The rules that define the work of a parallel global search algorithm (PGSA) are as follows.

At the first iteration of the method p of arbitrary points $x^1, \ldots, x^p$ in interval [0,1] (for example, these points can be uniformly located), and in these points trials are carried out in parallel. The results of trials $\{(x^i, z^i)\}$, $1 \le i \le p$, are saved in the search base of the algorithm.

Suppose, now, that $n \ge 1$ iterations of the method have already been executed. The trial points of the next $(n+1)$-th iteration are then chosen by using the following rules.

Step 1. Renumber points of the set

$$X_k = \{x^1, \ldots, x^k\} \cup \{0\} \cup \{1\}$$

which includes boundary points of interval [0,1], and points $\{x^1, \ldots, x^k\}$ of the previous $k = k(n) = np$ trials, with subscripts in increasing order of coordinate values, i.e.

$$0 = x_0 < x_1 < \ldots < x_{k+1} = 1.$$

Step 2. Supposing that $z_i = f(x_i),\ 1 \le i \le k$, calculate values

$$\mu = \max_{1 \le i \le k} \frac{|z_i - z_{i-1}|}{\Delta_i},\ M = \begin{cases} r\mu, \mu > 0, \\ 1,\ \mu = 0, \end{cases} \tag{2}$$

where $r > 1$ is a preset reliability parameter of the method, and $\Delta_i = (x_i - x_{i-1})^{1/N}$.

Step 3. Calculate characteristic for every interval $(x_{i-1}, x_i),\ 1 \le i \le k+1$, according to the following formulas

$$R(1) = 2\Delta_1 - 4\frac{z_1}{M},$$

$$R(k+1) = 2\Delta_{k+1} - 4\frac{z_k}{M},$$

$$R(i) = \Delta_i + \frac{(z_i - z_{i-1})^2}{M^2 \Delta_i} - 2\frac{z_i + z_{i-1}}{M}, \quad 1 < i < k+1.$$

Step 4. Arrange characteristics $R(i)$, $1 \le i \le k+1$, in decreasing order

$$R(t_1) \ge R(t_2) \ge \ldots \ge R(t_k) \ge R(t_{k+1}) \tag{3}$$

and select p maximum characteristics with interval numbers t_j, $1 \le j \le p$.

Step 5. Carry out new trials in points x^{k+j}, $1 \le j \le p$, calculated using formulas

$$x^{k+j} = \frac{x_{t_j} + x_{t_j - 1}}{2}, \quad t_j = 1, \, t_j = k+1,$$

$$x^{k+j} = \frac{x_{t_j} + x_{t_j - 1}}{2} - \mathrm{sign}(z_{t_j} - z_{t_j - 1}) \frac{1}{2r} \left[\frac{|z_{t_j} - z_{t_j - 1}|}{\mu} \right]^N, \quad 1 < t_j < k+1.$$

The algorithm terminates if the condition $\Delta_{t_j} \le \varepsilon$ is satisfied at least for one number t_j, $1 \le j \le p$; $\varepsilon > 0$ is the preset accuracy.

This method of organizing parallel computing has the following justification [8]. The characteristic $R(i)$ used in the global search algorithm can be considered as probability measure of the global minimum point location in the interval (x_{i-1}, x_i). Inequalities (3) arrange intervals according to their characteristics, and trials are carried out in parallel in p intervals with the largest probabilities.

3 Convergence and Speedup of the Parallel Algorithm

The following theorem form [8] identifies convergence conditions for the algorithm.

Theorem 1. Let $\bar{x}$ be the limit point of the sequence $\{x^k\}$ generated by PGSA during minimization of the Hölder with constant H, $0 < H < \infty$, function $f(x)$, $x \in [0, 1]$, and number p of parallel trials is fixed, $1 \le p < \infty$, and $\varepsilon = 0$ in the stop condition of the algorithm. Then

- convergence to the internal point $\bar{x} \in (0, 1)$ is bilateral;
- the point $\bar{x}$ is locally optimal if the function $f(x)$ has a finite number of local extremums;
- if, together with $\bar{x}$, another limit point $\hat{x}$ exists then $f(\bar{x}) = f(\hat{x})$;
- for all $k \ge 1$ it follows than $f(x^k) \ge f(\bar{x})$;
- if, at some step, for M from (2) the condition $M > 2^{2-1/N} H$ holds, then the set of limit points of the sequence $\{x^k\}$ will coincide with the set of global minimizers of the function $f(x)$.

More general variants of parallel global search algorithm and corresponding convergence theory are presented in [8].

Let us describe theoretical properties of a parallel algorithm, which characterize its speedup. One of the main indicators of efficiency of parallel algorithms (in any domain, not only in global optimization) is speedup in time

$$S(p) = T(1)/T(p)$$

where $T(1)$ is the time required for solving the problem by a sequential algorithm, and $T(p)$ is the time for solving the same problem by a parallel algorithm in a system with p computing elements. The characteristic of efficiency of parallel algorithms (in relation to algorithms of optimization) is also speedup in number of iterations

$$s(p) = n(1)p/n(p),$$

where $n(1)$ is the number of the trials carried out using the sequential method, and $n(p)$ is the number of the trials carried out using the parallel method with p processors. This characteristic is especially important since in applied problems the time of carrying out of a trial exceeds the time of processing of its results.

It is obvious that number of trials $n(p)$ for sequential and parallel algorithms will differ. Actually, the sequential algorithm when selecting point x^{k+1} of the next $(k+1)$ trial possesses complete information received at the previous k iterations. The parallel algorithm selects not one, but p points x^{k+j}, $1 \le j \le p$, at iteration $(k+1)$ based on the same information. It means that selection of point x^{k+j} is carried out in absence of information on the results of the trial in points x^{k+i}, $1 \le i < j$. Only the first point x^{k+1} will coincide with the point selected by the sequential algorithm. Points of the other trials, generally speaking, can not coincide with the points generated by the sequential algorithm. Carrying out of such trials can reduce efficiency of use of parallel processors. Therefore, let us consider such trials as "redundant", and the value

$$\lambda(p) = \begin{cases} (n(p) - n(1))/n(p), n(p) > n(1) \\ 0, n(p) \le n(1) \end{cases}$$

as "method redundancy".

Let us set the series of trials $\{x^k\}$ and $\{y^m\}$ generated correspondingly by sequential and parallel algorithms for solving the same problem with $\varepsilon = 0$ in the condition of stopping. The following theorems from [8] determine the number of computing elements p, which can be involved for non-redundant parallelization.

Theorem 2. Suppose x^* is the point of global minimum, x' is the point of the local minimum of function $f(x)$, and the following conditions are fulfilled:

1. Inequality

$$f(x') - f(x^*) \le \delta,\ \delta > 0, \tag{4}$$

holds.

2. The initial $q(l)$ trials of the sequential and parallel methods coincide, i.e.

$$\{x^1, \ldots, x^{q(l)}\} = \{y^1, \ldots, y^{q(l)}\},$$

Where

$$\left\{x^1, \ldots, x^{q(l)}\right\} \subset \{x^k\}, \left\{y^1, \ldots, y^{q(l)}\right\} \subset \{y^m\}.$$

3. There exists a point $y^n \in \{y^m\}$, $n < q(l)$, such that $x' \le y^n \le x^*$ or $x^* \le y^n \le x'$.
4. For value M from (2) the following inequality

$$M > 2^{2-1/N} H$$

holds, where H is Hölder constant of the minimized function.

Then a parallel algorithm of global search using two processors will be non-redundant (i.e. $s(2) = 2,\ \lambda(2) = 0$), while the following condition is satisfied

$$(x_{t_j} - x_{t_j-1})^{1/N} > \frac{4\delta}{M - 2^{2-1/N} H},\ j = 1,\ 2, \tag{5}$$

where t_j are determined according to (3).

Corollary. Let the objective function $f(x)$ have Q local minimum points $\left\{x_1', \ldots, x_Q'\right\}$, for which condition (4) is fulfilled, and let there exist trial points y^{n_i}, $1 \le i \le Q$, such as

$$y^{n_i} \in \{y^1, \ldots, y^{q(l)}\},$$

$$\alpha_i \le y^{n_i} \le \alpha_{i+1},\ \ \alpha_i, \alpha_{i+1} \in \{x^*, x_1', \ldots, x_Q'\},\ 1 \le i \le Q.$$

Then, if the theorem conditions are satisfied, the parallel algorithm of global search with $Q + 1$ processors will be non-redundant (i.e. $s(Q + 1) = Q + 1,\ \lambda(Q + 1) = 0$), while condition (5) is satisfied.

The theorem conclusion plays a special role for solving multidimensional problems reduced to one-dimensional problems by means of Peano-like evolvent $y(x)$. Evolvent $y(x)$, which is approximation to Peano curve, has the effect of "splitting" of a point of the global minimum $y^* \in D$ to several preimages in interval [0,1]. If function $\varphi(y)$ has the only global minimum in D, the "reduced" function $f(x)$ can have up to 2^N local extremum points close (by value) to a global extremum point (see [8]). In the case of applying a parallel global search algorithm for minimization of a similar function it is possible receive non-redundancy when using up to 2^N+1 computing elements.

4 Implementation on Xeon Phi

At the end of 2012 Intel presented Xeon Phi processor with MIC (Many Integral Core) architecture. The basis of MIC is using a large number of x86 computing cores in one processor. As a result, standard technologies, including OpenMP and MPI, can be used for parallel programming. Moreover, a vast number of tools and libraries has been developed for x86 architecture. It is a significant advantage as compared to other accelerators, for which special (usually more complex) technologies of parallel programming (CUDA, OpenCL) are used.

Intel Xeon Phi supports a few modes of coprocessor use, which can be combined to achieve maximum performance depending on characteristics of the solved problem. The process can be started both in the basic operating system or in coprocessor OS. Depending on the mode of use the computing capacity of either basic system processors or coprocessor or basic system processors and coprocessor combined can be used.

In the MPI mode the basic system and each Intel Xeon Phi coprocessor are considered as separate nodes, and MPI processes can be carried out on basic system processors and Xeon Phi coprocessors in any combination.

During operation in the offload mode MPI processes are carried out only on basic system processors, uploading and execution of functions on the coprocessor is used for implementation of Xeon Phi computing capabilities.

Taking into account that peak performance of one Xeon Phi core is comparable to peak CPU core performance (difference can make 5 – 10% depending on exact processor type), it is preferable to use an accelerator to carry out complex operations not requiring transfer of large amounts of data between the CPU and Phi. With regard to the considered parallel global optimization algorithm such a complex operation is parallel calculation of many objective function values. Transfers of data from the CPU to Phi will be minimal: it is only required to transfer to Phi the trial points coordinates, and to receive function values in these points. The functions that determine the trial results processing according to the algorithm and requiring operation with a large volume of collected search information can be efficiently implemented on the CPU. The described organization scheme corresponds well to the accelerator offload mode.

The general scheme of organization of calculations using Xeon Phi is shown in Fig. 1. According to this scheme steps 1–4 of the parallel global search algorithm are performed on the CPU. Coordinates of the p trial points calculated at step 4 of the algorithm are transferred to the accelerator. Calculation of function values in these points is carried out on Xeon Phi, and then the trial results are transferred to the CPU. We use Xeon Phi offload mode for synchronous computing p function values at each iteration. Current implementation of the parallel algorithm supports only one coprocessor.

We note here that parallel calculation of function values in several tens or hundreds of points (up to 240 threads can be launched on Xeon Phi) not always gives speedup of the search process by a factor of tens or hundreds. In this case, the conditions of the theorem of non-redundant parallelization can be violated: the number of local extremums will be less, than the number of computing cores. Then (according to the

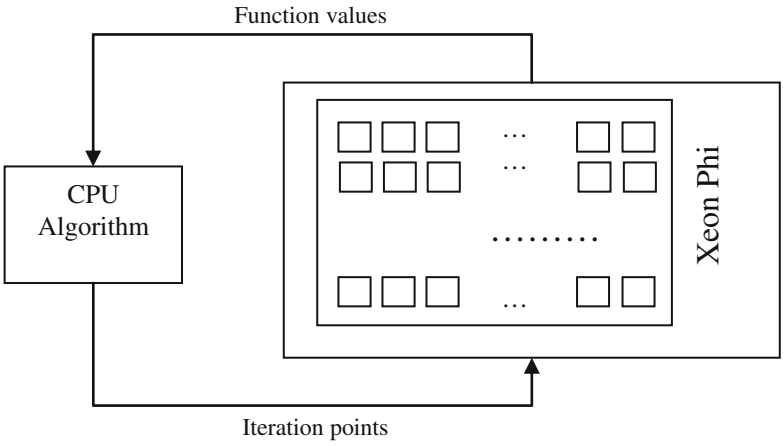

Fig. 1. Scheme of information exchanges

theorem and its corollary) the parallel global search algorithm will generate redundant trial points. Nevertheless, despite some redundancy, use of Xeon Phi will reduce overall algorithm operating time. It is confirmed by computational experiments, results of which are given in Sect. 5.

5 Results of Numerical Experiments

Computing experiments were carried out on one of the nodes of a high-performance cluster of the Nizhny Novgorod State University. The cluster node includes Intel Sandy Bridge E5-2660 2.2 GHz CPUs, 64 Gb RAM, and Intel Xeon Phi 5110P. For implementation of the parallel algorithm Intel C++ Compiler 14.0.2 under CentOS 6.4 was used.

It is significant, that widely known test problems from the field of multidimensional global optimization are characterized by small time of objective function values calculation. Usually, such a calculation is reduced to summation of several (according to problem dimension) values of elementary functions. Therefore, for the purpose of imitation of the computational complexity inherent to applied problems of optimization [14], calculation of the objective function in all performed experiments was complicated by additional calculations without changing the type of function and arrangement of its minimums (series summation from 20 thousand elements).

In work [15] a GKLS generator allowing generation of multiextremal optimization problems with known properties (number of local minimums, size of their domains of attraction, global minimum point, etc.) is described.

The results of numerical comparison of three sequential algorithms – DIRECT [1], DIRECT*l* [2] and global search algorithm (PGSA from Sect. 2 with p=1) – are provided below (results of work of the first two algorithms are given in [3]). Numerical comparison was carried out on function classes *Simple* and *Hard* of dimension 4 and 5 from [3] since solving problems of dimension 2 and 3 requires a small number of

iterations and use of accelerator for solving these problems is impractical. Global minimum y^* was considered as found, if the algorithm generated trial point y^k in δ-vicinity of the global minimum, i.e. $\left\|y^k - y^*\right\| \leq \delta$. The size of the vicinity was selected (according to [3]) as $\delta = \|b - a\| \sqrt[N]{\Delta}$, N – problem dimension, a and b – borders of search domain D, parameter $\Delta = 10^{-6}$ at $N = 4$ and $\Delta = 10^{-7}$ at $N = 5$. When using the PGSA method for class *Simple* $r = 4.5$ parameter was selected, for class *Hard* $r = 5.6$ was selected; evolvent construction parameter was fixed as $m = 10$. The maximum allowable number of iterations was $K_{\max} = 10^6$.

The average number of iterations k_{av} performed by the method for solving a series of problems from these classes is shown in Table 1. Symbol ">" reflects a situation, when not all problems of a class were solved by a method. It means that the algorithm was stopped as the maximum allowable number of iterations $K_{\max}$ was achieved. In this case, $K_{\max}$ value was used for calculation of the average value of number of iterations k_{av} that corresponds to the lower estimate of this average value. The number of unsolved problems is specified in brackets.

Table 1. Average number of iterations

N	Problem class	DIRECT	DIRECT*l*	PGSA
4	Simple	>47282(4)	18983	11953
	Hard	>95708(7)	68754	25263
5	Simple	>16057 (1)	16758	15920
	Hard	>217215 (16)	>269064 (4)	>148342 (4)

Table 1 shows that PGSA outperforms DIRECT and DIRECT*l* methods on all classes of problems by average number of iterations. And in class 5-*Hard* each of the methods solved not all problems: DIRECT did not solve 16 problems, DIRECT*l* and PGSA – 4 problems.

Let us estimate now the speedup when using PGSA implemented on CPU depending on number of used cores p. Tables 2 and 3 show time speedup $S(p)$ and

Table 2. Time speedup $S(p)$ on CPU

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
2	2.45	2.20	1.15	1.32
4	4.66	3.90	2.82	2.59
8	7.13	7.35	3.47	5.34

Table 3. Iteration speedup $s(p)$ on CPU

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
2	2.51	2.26	1.19	1.36
4	5.04	4.23	3.06	2.86
8	8.58	8.79	4.22	6.56

Table 4. Redundancy $\lambda(p)$ on CPU

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
2	0.00	0.00	0.23	0.29
4	0.00	0.00	0.47	0.18
8	0.00	0.00	0.00	0.27

iteration speedup $s(p)$ respectively; speedup of parallel algorithm was measured in relation to the sequential one (p=1). Table 4 shows the average redundancy $\lambda(p)$ of the method during solving a problem series.

The results of the experiments show considerable acceleration and low redundancy of PGSA when using CPU.

Now let us perform a series of experiments using Xeon Phi. We will measure acceleration and redundancy of an algorithm that uses Phi as compared to a CPU algorithm that fully uses an eight-core CPU. In the experiments we will vary the number of threads p on Xeon Phi. All other parameters of the method will not vary.

Table 5. Time speedup $S(p)$ on Phi

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
60	0.54	1.02	1.07	1.61
120	0.55	1.17	1.05	2.61
240	0.51	1.06	1.07	4.17

Table 6. Iteration speedup $s(p)$ on Phi

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
60	8.13	7.32	9.87	6.55
120	16.33	15.82	15.15	17.31
240	33.07	27.79	38.80	59.31

Table 7. Redundancy $\lambda(p)$ on Phi

p	N = 4		N = 5	
	Simple	Hard	Simple	Hard
60	0.00	0.02	0.00	0.13
120	0.00	0.00	0.00	0.00
240	0.00	0.07	0.00	0.00

The results of the experiments (Table 5) show that in most cases an algorithm using Phi is not less fast than a CPU algorithm: acceleration approximately 1.05 is observed. Slowing down with class 4-*Simple* is explained by relative simplicity of the problems

solved: the computational load on the coprocessor is not sufficient, the additional costs of transmission of the problem parameters from CPU to Phi produce a significant effect. With class 5-*Hard*, which is characterized by a high computational effort, a quadruple acceleration is observed; additional costs produce no decisive impact here.

An important additional feature is also acceleration on number of iterations, which goes up to a several dozens, if Phi is fully used (see Table 6). For example, solving a problem from class 5-*Hard* required on average only 633 parallel iterations on Phi, whereas when using all computing cores of the CPU the number of iterations was more than 37 thousand. At the same time, an algorithm using Phi is almost non-redundant in comparison to a CPU algorithm (see Table 7).

6 Conclusions

The work considers a parallel algorithm of global search developed within the framework of the information statistical approach to multiextremal optimization. This algorithm can be used for solving time-consuming optimization problems on state-of-the-art multiprocessor systems as it allows efficient implementation through use of Intel Xeon Phi coprocessor. The results of computational experiments confirm a high efficiency and low redundancy of the parallel algorithm. This very well correlates with the theoretical statements provided above.

Acknowledgements. The research is supported by the grant of the Ministry of education and science of the Russian Federation (the agreement of August 27, 2013, № 02.B.49.21.0003).

References

1. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
2. Gablonsky, J.M., Kelley, C.T.: A locally-biased form of the DIRECT algorithm. J. of Glob. Optim. **21**(1), 27–37 (2001)
3. Sergeyev, Y.D., Kvasov, D.E.: Global search based on efficient diagonal partitions and a set of Lipschitz constants. SIAM J. Optim. **16**(3), 910–937 (2006)
4. Žilinskas, J.: Branch and bound with simplicial partitions for global optimization. Math. Model. Anal. **13**(1), 145–159 (2008)
5. Gergel, V.P.: A method of using derivatives in the minimization of multiextremum functions. Comput. Math. Math. Phys. **36**(6), 729–742 (1996)
6. Gergel, V.P.: A global optimization algorithm for multivariate functions with lipschitzian first derivatives. J. Glob. Optim. **10**(3), 257–281 (1997)
7. Gergel, V.P., Sergeyev, Y.D.: Sequential and parallel algorithms for global minimizing functions with lipschitzian derivatives. Comput. Math Appl. **37**(4–5), 163–179 (1999)
8. Strongin, R.G., Sergeyev, Y.D.: Global optimization with non-convex constraints. Sequential and Parallel Algorithms. Kluwer Academic Publishers, Dordrecht (2000)
9. Barkalov, K.A., Strongin, R.G.: A global optimization technique with an adaptive order of checking for constraints. Comput. Math. Math. Phys. **42**(9), 1289–1300 (2002)

10. Evtushenko, Y., Malkova, V.U., Stanevichyus, A.A.: Parallel global optimization of functions of several variables. Comput. Math. Math. Phys. **49**(2), 246–260 (2009)
11. Paulavicius, R., Zilinskas, J., Grothey, A.: Parallel branch and bound for global optimization with combination of Lipschitz bounds. Optim. Meth. Softw. **26**(3), 487–498 (2011)
12. Grishagin, V.A., Sergeyev, Y.D., Strongin, R.G.: Parallel characteristical algorithms for solving problems of global optimization. J. Glob. Optim. **10**(2), 185–206 (1997)
13. Sergeyev, Y.D., Strongin, R.G., Lera, D.: Introduction to global optimization exploiting space-filling curves. Springer, Heidelberg (2013)
14. Barkalov, K., Polovinkin, A., Meyerov, I., Sidorov, S., Zolotykh, N.: SVM regression parameters optimization using parallel global search algorithm. In: Malyshkin, V. (ed.) PaCT 2013. LNCS, vol. 7979, pp. 154–166. Springer, Heidelberg (2013)
15. Gaviano, M., Lera, D., Kvasov, D.E., Sergeyev, Y.D.: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans. Math. Softw. **29**, 469–480 (2003)