To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–15

GLOBALIZER: A NOVEL SUPERCOMPUTER SOFTWARE SYSTEM FOR SOLVING TIME-CONSUMING GLOBAL OPTIMIZATION PROBLEMS

V. P. Gergel[a*] and K. A. Barkalov[b] and A. V. Sysoev

[a] *Taylor & Francis, 4 Park Square, Milton Park, Abingdon, UK;* [b] *State University of Nizhny Novgorod, Nizhny Novgorod, Russia*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

In this paper, we describe the Globalizer software system for solving global optimization problems. The system is designed to maximize the use of computational potential by modern high performance computer systems in order to solve the most time-consuming optimization problems. Highly parallel computations are facilitated using various distinctive computational schemes: processing several optimization iterations simultaneously, reducing multidimensional optimization problems using multiple Peano curve evolvents, multi-stage computing based on nested block reduction schemes. This novelty leverages supercomputer system capabilities with shared and distributed memory and with large numbers of processors to efficiently solve global optimization problems.

Keywords: global optimization, information-statistical theory, parallel computing, high-performance computer systems, supercomputer technologies

1. Introduction

Global (or multiextremal) optimization problems are among the most complex problems in both theory and practice for optimal decision making. In these kinds of problems, the criterion to be optimized has several local optima within the search domain, which have different values. The existence of several local optima essentially makes finding the global optimum difficult, since it requires examining the whole feasible search domain. The volume of computations for solving global optimization problems can increase exponentially with increasing number of varied parameters.

In addition, global optimization problem statements are used, as a rule, in the most complex decision making situations, for example, when a computer-aided design of complex technologies, products, and systems is conducted. In such problems, the efficiency criteria are nonlinear, the search domains may be non-contiguous, and, most importantly, the computational complexity of the functions which the optimized criteria and constraints are based on can be quite essential.

These global optimization problem features impose special requirements on the quality of the optimization methods and on the software to implement them. The global optimization methods should be highly efficient, and the software systems should be developed on a good professional basis. In general, global optimization problems can

*Corresponding author. Email: gergel@unn.ru be solved using modern supercomputer systems at a reasonable time and cost by only employing parallel global optimization algorithms.

The general state of the art in the field of global optimization has been presented in a number of key monographs [13], [6], [14], [10], [12], [7], [5], etc. The development of optimization methods that use high-performance computer systems to solve time-consuming global optimization problems is an area receiving extensive attention – see, for example, [1], [12], [2], [8], [11].

The theoretical results obtained provide efficient solutions to many applied global optimization problems in various fields of scientific and technological applications – see, for example, [4], [10], [8], [3], [7], [9], etc.

At the same time, the practical implementation of these algorithms for multiextremal optimization within the framework of industrial software systems is quite limited. In many cases, software implementations of global optimization algorithms are experimental in nature and are just used by the developers themselves to obtain results from computational experiments required for scientific publication. To a large extent, this situation originates from high development costs for professional software systems that can be used by numerous users. In addition, global optimization problems could rarely be solved in an automatic mode because of their complexity. As a rule, the user should actively control the global search process which implies an adequate level of qualification in the field of optimization (particularly, the user should know and understand global optimization methods well).

On the market for global optimization software, one can select from the following systems:

- LGO (Lipschitz Global Optimization) is designed to solve global optimization problems for which the criteria and constraints satisfy the Lipschitz condition (see, for example, [10]). The system is a commercial product based on diagonal extensions of one-dimensional multiextremal optimization algorithms.
- GlobSol (Kearfott, R. B., 2009) is oriented towards solving global optimization problems as well as systems of nonlinear equations. The system includes interval methods based on the branch and bound method. There are some extensions of the system for parallel computations, and it is available to use for free.
- LINDO (Lin, Y., Schrage.L., 2009) is features by a wide spectrum of problem solving mehtods which it can be used for – these include linear, integer, stochastic, nonlinear, and global optimization problems. The ability to interact with the Microsoft Excel software environment is a key feature of the system. The system is widely used in practical applications and is available to use for free.
- IOSO (Indirect Optimization on the basis of Self-Organization) is oriented toward solving of a wide class of the extremal problems including global optimization problems (see, for example, (Egorov, I. N., Kretinin, G. V., Leshchenko, I. A., Kuptzov, S. V., 2002)). The system is widely used to solve applied problems in various fields. There are versions of the system for parallel computational systems. The system is a commercial product, but is available for trial use.
- MATLAB Global Optimization Toolkit (see, for example, (Venkataraman, P., 2009)) includes a wide spectrum of methods for solving the global optimization problems, including multistart methods, global pattern search, simulated annealing methods, etc. The library is compatible to the TOMLAB system (see, for example, (Holmström, K., Edvall, M. M., 2004)), which is an additional extension the widely-used MATLAB. It is also worth noting that similar libraries for solving global optimization problems are available for MathCAD, Mathematica, and Maple systems as well.

- BARON (Branch-And-Reduce Optimization Navigator) is designed to solve continuous integer programming and global optimization problems using the branch and bound method (Sahinidis, N. V., 1996). BARON is included in the GAMS (General Algebraic Modeling System) system used extensively – see (Bussieck, M. R., Meeraus, A., 2004).
- Global Optimization Library in R is a large collection of optimization methods implemented in the R language (see, for example, (Mullen, K. M., 2014)). Among these methods, are stochastic and deterministic global optimization algorithms, the branch and bound method, etc.

The list provided above is certainly not exhaustive – additional information on software systems for a wider spectrum of optimization problems can be obtained, for example, in (Rios, L. M., Sahinidis, N. V., 2013), (Mongeau, M., Karsenty, H., Rouzé, V., Hiriart-Urruty, J. B., 2000), (Pintér, J. D., 2009), etc. Nevertheless, even from such a short list the following conclusions can be drawn (see also Liberti, L., 2006):

- the collection of available global optimization software systems for practical use is insufficient,
- the availability of numerous methods through these systems allows complex optimization problems to be solved in a number of cases, however, it requires a rather high level of user knowledge and understanding in the field of global optimization,
- the use of the parallel computing to increase the efficiency in solving complex time-consuming problems is limited, therefore, the computational potential of modern supercomputer systems is very poorly utilized.

In this paper, a novel Globalizer software system is considered. The development of the system was conducted based on the information-statistical theory of multiextremal optimization aimed at developing efficient parallel algorithms for global search – see, for example, (Strongin, R. G., 1978), [12], ([11]). The advantage of the Globalizer is that the system is designed to solve time-consuming multiextremal optimization problems. In order to obtain global optimized solutions within a reasonable time and cost, the system efficiently uses modern high-performance computer systems.

This paper is further structured as follows. In Section 2, a statement of the multidimensional global optimization problem is presented, and an approach to reducing these to one-dimensional optimization problems is described. In Section 3, parallel computation schemes are presented, and parallel optimization methods are described. In Section 4, the Globalizer architecture is examined. In Section 5, the results are presented from computational experiments that confirm the system's high level of efficiency. Finally, Section 6 presents the conclusions and some ideas for future research.

2. Multidimensional Global Optimization Problems and Dimension Reduction

In this paper, the core class[1] of optimization problems which can be solved using the Globalizer is examined. This involves multidimensional global optimization problems

[1]In general, the Globalizer can be applied for solving multicriterial multiextremal multidimensional optimization problems with nonlinear constraints.

without constraints, which can be defined in the following way:

$$\begin{gathered}\varphi(y^*) = \min\{\varphi(y) : y \in D\} \\ D = \{y \in \mathbf{R}^N : a_i \leqslant x_i \leqslant b_i, 1 \leqslant i \leqslant N\}\end{gathered} \tag{1}$$

where the objective function $\varphi(y)$ satisfies the Lipschitz condition

$$|\varphi(y_1) - \varphi(y_2)| \leqslant L\|y_1 - y_2\|, y_1, y_2 \in D, \tag{2}$$

where $L > 0$ is the Lipschitz constant, and $|| \cdot ||$ denotes the norm in $\mathbf{R}^N$ space.

Let us further assume that the minimized function $\varphi(y)$ is defined as a computational procedure, according to which the value $\varphi(y)$ can be calculated for any value of vector $y \in D$ (let us further call the process of obtaining a value for the minimized function *a trial*). As a rule, this procedure is computational-costly i.e. the overall costs of solving optimization problem (1) are determined, first of all, by the number of trials executed. It is also worth noting the essence of the assumption on satisfying the Lipschitz condition, since one can construct an estimate of the global minimum based on a finite number of computed values from the optimized function in this case only.

As has been previously shown by many researchers, finding numerical estimates of globally optimal extrema implies constructing coverage of the search domain D. As a result, the computational costs of solving global optimization problems are already very high even for a small number of varied parameters (the dimensionality of the problem). A notable reduction in the volume of computations can be achieved when the coverages of the search domain being obtained are non-uniform, i.e. the series of trial points is dense only in terms of its nearness to the sought-after globally optimized variants. The generation of such non-uniform coverages could only be provided in an adaptive way when the selection of the next trial points is determined by the search information (the preceding trial points and the values of the minimized function at these points) obtained in the course of computation. This necessary condition considerably complicates the computational schemes for global optimization methods, since it implies a complex analysis of a large amount of multidimensional search information. As a result, many optimization algorithms use, to some extent, various methods of dimensional reduction.

Within the framework of the information-statistical approach, Peano *curves* (or *evolvents*) $y(x)$ mapping the interval $[0, 1]$ onto an N-dimensional hypercube D are unambiguously used for dimensional reduction (see, for example, [12]).

As a result of the reduction, the initial multidimensional global optimization problem (1) is reduced to the following one-dimensional problem:

$$\varphi(y(x^*)) = \min\{\varphi(y(x)) : x \in [0, 1]\} \tag{3}$$

The dimensional reduction scheme considered above superimposes a multidimensional problem with a minimized Lipschitz function to a one-dimensional problem with a corresponding minimized function to satisfy the uniform Hölder condition (see Strongin, R. G., 1978, [12]) i.e.

$$|\varphi(y(x_1)) - \varphi(y(x_2))| \leqslant H|x_1 - x_2|^{\frac{1}{N}}, x_1, x_2 \in [0, 1] \tag{4}$$

where the constant H is defined by the relationship $H = 4L\sqrt{N}$, L is the Lipschitz constant from (2), and N is the dimensionality of the optimization problem (1).

The algorithms for the numerical construction of the Peano curve approximations are presented in [12]. As an illustration, an approximation of a Peano curve for the third level of density is shown in Figure 1. The curve shown in Figure 1 demonstrates the winding order of a two-dimensional domain; the precision of the Peano curve approximation is determined by the level of density used in the construction.

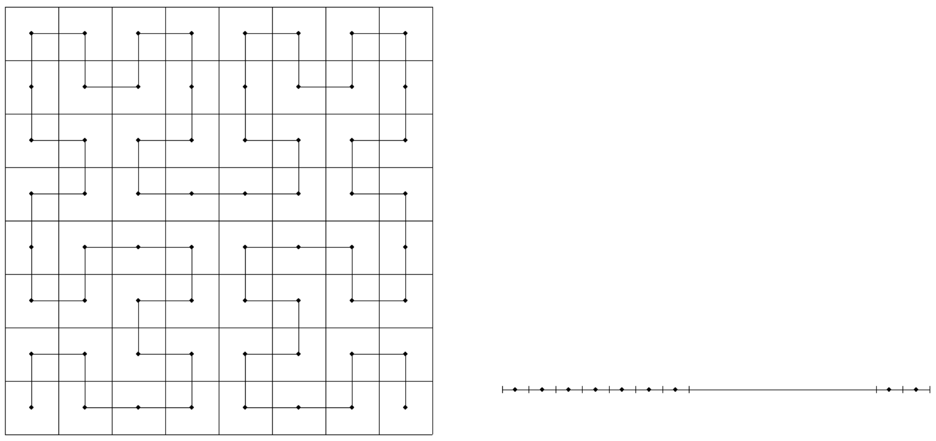

Figure 1. A Peano curve approximation for the third level of

The computational scheme obtained as a result of the dimensional reduction consists of the following (see Figure 2):

- The optimization algorithm performs the minimization of the reduced one-dimensional function $\varphi(y(x)$,
- After determining the next trial point x, a multidimensional image y in the mapping $y(x)$ is calculated,
- The value of the initial multidimensional function $\varphi(y)$ is calculated at the multidimensional point $y \in D$,
- The calculated value $z = \varphi(y)$ is used further as the value of the reduced one-dimensional function $\varphi(y(x))$ at the point x.

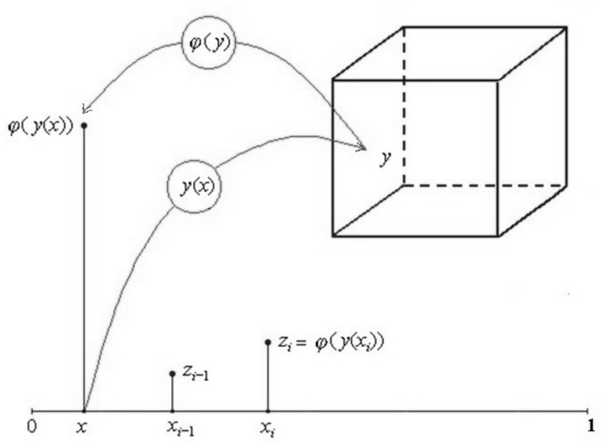

Figure 2. The computational scheme for obtaining the value of the reduced one-dimensional function $\varphi(y(x))$

3. The Parallelization Approach and Global Optimization Algorithms

Let us consider the parallelization methods used widely in the theoretical and practical application of parallel computing within the context of global optimization problems:

- The distribution of the search domain D among the available computing units (data parallelization scheme). In the case of optimization problems, this approach is insufficient, since in organizing such computations the subdomain, which contain the sought global minimum, will be processed by only one processor, and, therefore, all of the remaining processors would perform the excess computations.
- The parallelization of the computational schemes for optimization algorithms (task parallelization scheme). This approach is also insufficient since the direct computational costs for executing optimization algorithms are relatively low (the majority of computations in the course of a global search are represented by calculations of the optimized function values due to the initial assumption of considerable computational costs for such calculations).
- The parallelization of the computations executed in order to obtain values for the optimized function. This approach can be applied since the most computationally costly part of the global search process will be parallelized. However, this method is not featured by the generality (the development of parallelization methods is to be performed every time from scratch while solving each particular optimization problem).

Within the framework of information-statistical theory, a general approach to parallelization computations when solving global optimization problems has been proposed — *the parallelism of computations is provided by means of simultaneously computing the values of the minimized function $\varphi(y)$ at several different points within the search domain* D – see, for example, [12], [11]. This approach provides parallelizaion for the most costly part of computations in the global search process.

The global optimization algorithms implemented in Globalizer will be described step-by-step below. In Subsection 3.1, the core sequential multidimensional algorithm of global search (MAGS) will be presented. In Subsection 3.2, a parallel generalization of the MAGS algorithm for parallel computations on computer systems with shared memory will be described. In Subsection 3.3, the scheme for parallel computations by multiprocessor systems with distributed memory will be provided.

3.1 *Core Multidimensional Generalized Algorithm of Global Search*

The information-statistical theory of global search formulated in (Strongin, R. G., 1978), [12] has served as a basis for the development of a large number of efficient multiextremal optimization methods — see, for example, (Gergel, V. P., 1996, 1997), (Gergel, V. P., Strongin, R. G., 2003, 2005), (Grishagin, V. A., Strongin, R. G., 1984), (Sergeyev Y. D., 1995, 1999), (Sergeyev, Y. D., Grishagin V. A., 1994, 2001), (Sergeyev, Y. D., Strongin, R. G., Lera, D., 2013), (Barkalov K. A., Gergel V. P., 2014), etc.

Multidimensional Algorithm of Global Search (MAGS) established the basis for the methods applied in Globalizer. The general computational scheme of MAGS can be presented as follows — see (Strongin, R. G., 1978), [12].

Let us introduce a simpler notation for the problem being solved on a computing node

$$f(x) = \varphi(y(x)) : x \in [0, 1]\} \tag{5}$$

The initial iteration of the algorithm is performed at an arbitrary point $x^1(0, 1)$. Let us assume further k, $k > 1$, iterations of a global search are to be completed. The selection of the trial point $k+1$ for the next iteration is performed according to the following rules.

Rule 1. To renumber the points for the preceding trials by the lower indices in order of increasing value of coordinates

$$0 = x_0 < x_1 < \ldots < x_{k+1} = 1 \tag{6}$$

the points x_0, x_k were introduced additionally for convenience in further explanation, the values of the minimized function z_0, z_k at these points are undefined.

Rule 2. To compute a current estimate of the Hölder constant H from (4)

$$m = \max_{1 \leqslant i \leqslant k} \frac{|z_i - z_{i-1}|}{\rho_i}, M = \begin{cases} r \cdot m, m > 0 \\ 1, m = 0 \end{cases} \tag{7}$$

as the maximum of the relative differences of the minimized function $f(x)$ from (5) on the set of executed trial points $x_i, 1 \leqslant i \leqslant k$ from (6). Further, $\rho_i = (x_i - x_{i-1})^{\frac{1}{N}}, 1 \leqslant i \leqslant k+1$. The constant r, $r > 1$, is the *parameter* for the algorithm.

Rule 3. For each interval $(x_{i-1}, x_i), 1 \leqslant i \leqslant k+1$, compute the characteristics $R(i)$ where

$$\begin{gathered} R(i) = \rho_i + \frac{(z_i - z_{i-1})^2}{M^2 \rho_i} - 2\frac{z_i + z_{i-1}}{M}, \quad 1 < i < k+1 \\ R(i) = 2\rho_i - 4\frac{z_i}{M}, \quad i = 1 \\ R(i) = 2\rho_i - 4\frac{z_{i-1}}{M}, \quad i = k+1 \end{gathered} \tag{8}$$

Rule 4. To determine the interval with the maximum characteristic

$$R(t) = \max_{1 \leqslant i \leqslant k+1} R(i) \tag{9}$$

Rule 5. To execute a new trial (computation of the minimized function value $f(x)$) at the point x^{k+1} located within the interval with the maximum characteristic from (9)

$$\begin{gathered} x_{k+1} = \frac{x_t + x_{t-1}}{2} - \operatorname{sign}(z_t - z_{t-1}) \frac{1}{2r} \left[\frac{|z_t - z_{t-1}|}{M} \right]^N, \quad 1 < t < k+1 \\ x_{k+1} = \frac{x_t + x_{t-1}}{2}, \quad t = k+1 \end{gathered} \tag{10}$$

The stop condition by which the trials are terminated, is defined by the inequality

$$\rho_t < \varepsilon \tag{11}$$

for the interval with the maximum characteristic from (9) and $\varepsilon > 0$ is the predefined accuracy of the problem solution. If the stop condition is not satisfied, the index k is incremented by unity, and a new global search iteration is executed.

In order to explain the algorithm presented above, let us note the following. The characteristics $R(i)$, $1 \leqslant i \leqslant k+1$, calculated according to (8) could be interpreted as some measures of importance to the intervals with respect to location of the global minimum point. Thus, the scheme for selecting the interval for executing the next trial described in (9-10) becomes more clear — the point of every next global search iteration is selected within the interval where the global minimum point can most likely be found.

The conditions for the algorithm's convergence described above were examined, for example, in [12].

3.2 *Parallel Computations for Systems with Shared Memory*

Modern supercomputer computational systems consist of plenty of computer nodes, which include several multicore processors. At that, random access memory is shared for the computing nodes — the content of any memory element can be read (written) by any computer cores available at any arbitrary moment. In most cases, shared memory is uniform — the time characteristics for accessing memory are the same for all computer cores and for all memory elements.

The following speculations can form the basis for selecting parallel computation organization methods. As was mentioned above when describing the MAGS algorithm, the characteristics $R(i), 1 \leqslant i \leqslant k+1$, calculated according to (8) can be interpreted as some measures of the importance of intervals with respect to the location of the global minimum point. Following this understanding, each successive global search iteration is executed within the most important interval with the maximum value of the characteristic. So far, in this case it is obvious how to select the other intervals for organizing simultaneous computations of the minimized function values at several different points within the search domain as well — these should be the intervals with the next magnitudes of characteristics.

The computational scheme of the Parallel Multidimensional Algorithm of Global Search for computer systems with shared memory (PMAGS) was developed based the approach considered above and is practically identical to the MAGS scheme — the differences consist just in the following set of rules — see [12], [11].

Rule 4'. To arrange the characteristics of the intervals obtained according to (8) in decreasing order

$$R(t_1) \geqslant R(t_2) \geqslant \cdots \geqslant R(t_k) \geqslant R(t_{k+1}) \tag{12}$$

and to select p intervals with the indices $t_j, 1 \leqslant j \leqslant p$, with the highest values of characteristics (p is the number of processors/cores used for the parallel computations).

Rule 5'. To execute new trials (computations of values for the minimized function $f(x)$) at the points $x_{k+j}, 1 \leqslant j \leqslant p$, located in the intervals with the highest characteristics from (12)

$$\begin{gathered} x_{k+j} = \frac{x_{t_j} + x_{t_j-1}}{2} - \mathrm{sign}(z_{t_j} - z_{t_j-1}) \frac{1}{2r} \left[\frac{|z_{t_j} - z_{t_j-1}|}{M} \right]^N, \quad 1 < t_j < k+1 \\ x_{k+j} = \frac{x_{t_j} + x_{t_j-1}}{2}, \quad t_j = 1, t_j = k+1 \end{gathered} \tag{13}$$

The stop condition for the algorithm (14) should be checked for all intervals from (12), in which the scheduled trials are executed

$$\rho_{t_j} < \varepsilon, 1 \leqslant j \leqslant p \tag{14}$$

As before, if the stop condition is not fulfilled, the index k is incremented by p, and a new global search iteration is executed.

The conditions for the developed parallel algorithm's convergence and for the non-redundancy of parallel computations have been examined in [12]. Thus, in particular, when the conditions for convergence are satisfied, the parallel computations are non-redundant as compared to the sequential method when using up to 2^N processors/cores (where N is the dimensionality of the global optimization problem being solved).

3.3 *Parallel Computations for Systems with Distributed Memory*

The next level of parallel computations in high-performance computer systems consists of using several computational nodes. For that, each computational node has its own separate memory and, in this case, the interaction between different nodes can be provided by means of data transfer only via the computer communication network.

To organize parallel computations for multiprocessor systems with distributed memory, using a set of mappings

$$Y_s(x) = \{y^1(x), \dots, y^s(x)\} \tag{15}$$

instead of applying a single Peano curve $y(x)$ has been proposed in (Strongin, R. G., 1992, [12]). In order to build the set $Y_s(x)$, several different approaches can be applied. Thus, for example, in (Strongin, R. G., 1992) a scheme has been applied by which each mapping $y_i(x)$ from $Y_s(x)$ is built as the result of some shift along the main diagonal of the hypercube D. This way, the set of constructed Peano curves enables the close prototypes x', x'' to be obtained from the interval $[0, 1]$ for any close multidimensional images y', y'' from D differing in one coordinate, only for some mapping $y_k(x), 1 \leqslant k \leqslant s$.

Some other methods for constructing multiple mappings have been examined in [11].

The set of mappings $Y_s(x)$ from (15), for multidimensional problem (1) generates s subsidiary information-linked one-dimensional problems of the kind (3):

$$\varphi(y^l(x^*)) = \min\{\varphi(y^l(x)) : x \in [0, 1]\}, 1 \leqslant l \leqslant s \tag{16}$$

It is important to note that the family of one-dimensional problems $\varphi(y^l(x)), 1 \leqslant l \leqslant s$, obtained as a result of the dimensional reduction is information-linked — the function values computed for any problem $\varphi(y^l(x))$ from the family (16) can be used for all of the remaining problems of this family.

The information compatibility of the problems from the family (16) allows the parallel computations to be organized in an obvious enough way. Thus, each particular problem can be solved by a separate processor in the computing system; the exchange of search information between the processors should be performed during the course of the computations. As a result of using such a parallelization method, a unified approach for organizing parallel computations for a multiprocessor computer systems with distributed, as well as shared memory can be developed. This method consists of the following.

(1) The family of reduced one-dimensional information-linked problems (16) is distributed among the computational nodes of a multiprocessor system. One problem, as well as several others from within the family, can be allocated to each particular computational node.
(2) The Parallel Multidimensional Algorithm of Global Search (PMAGS, see Subsection 3.2) is applied to solve the allocated problems from the family (16) at each

computational node, supplemented by the following rules of information interaction.

(a) Prior to beginning a new trial at any point $x' \in [0,1]$ for any problem $\varphi(y_l(x)), 1 \leqslant l \leqslant s$, the following should be performed:
- compute the image $y' \in D$ for the point $x' \in [0,1]$ according to the mapping $y_l(x)$,
- determine the prototypes $x'_i, 1 \leqslant i \leqslant s$, for each problem from the family (16),
- send the prototypes $x'_i, 1 \leqslant i \leqslant s$ to all computational nodes employed in order to exclude the repeated selection of the intervals in which the prototypes fall, for using them to determine the points for new trials. To organize the data transfer, a queue can be formed at each computational node for sending the trial points and receiving the minimized function values at these points.

(b) After completing any trial for any problem $\varphi(y_l(x)), 1 \leqslant l \leqslant s$, at any point $x' \in [0,1]$, it is necessary:
- to determine all prototypes $x'_i, 1 \leqslant i \leqslant s$, for the point of the completed trial for each problem from the family (16),
- to send the prototypes $x'_i, 1 \leqslant i \leqslant s$, and the result of the trial $z' = \varphi(y_l(x'))$ to all computational nodes employed to include the obtained data into the search information processed within the rules of the parallel global search algorithm.

(c) Prior to starting the next global search iteration, the algorithm should check the queue of the received data; if there are some data in the queue, they should be included in the search information.

The possibility of asynchronous data transfer (the computational nodes process received data only upon receipt) is a meaningful feature of such a scheme for organizing parallel computations.

In addition, there is no single managing node within this scheme. The number of computational nodes can vary during the course of the global search, and excluding any node does not result in the loss of the sought global minimum of the minimized multiextremal function.

This computational scheme generates the Generalized Parallel Multidimensional Algorithm of Global Search (GPMAGS) for high-performance computing systems, which may include many multiprocessor (multicore) computational nodes with distributed memory as well as accelerators for computations based on Intel Xeon Phi processors and on general purpose graphic processors.

Additional information on organizing such parallel computation schemes is provided in (Gergel, V. P., Sidorov, S., 2015).

4. Globalizer System Architecture

As previously mentioned in Section 1, the development of software systems for global optimization is complex professional work. The optimization systems should include advanced optimization methods, provide for the accumulation and efficient processing of search information, include various forms of visualization for the results, provide tools for interacting with users, etc. All the features pointed out above become much more complicated when organizing parallel computations using high-performance multiprocessor systems of varying architectures.

The Globalizer considered in this paper expands the family of global optimization software systems successively developed by the authors during the past several years. One of the first developments was the SYMOP multiextremal optimization system (see Gergel,

V. P., 1993), which has been successfully applied for solving many optimization problems. A special place is occupied by the ExaMin system (see, for example, (Barkalov, K., Gergel V., 2015)), which was developed and used extensively to investigate the application of novel parallel algorithms to solve global optimization problems using high-performance multiprocessor computing systems.

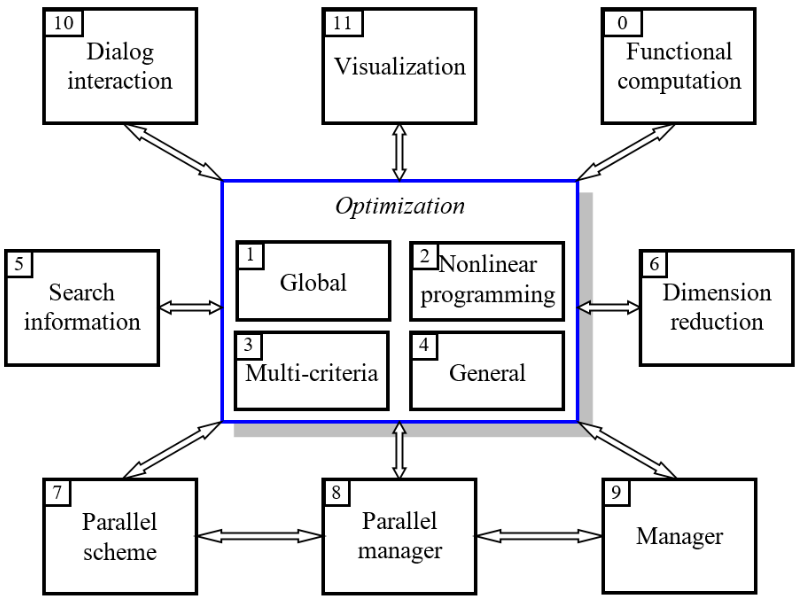

Figure 3. The Globalizer system architecture (Blocks 1-2, 5-7 have been implemented; Blocks 3-4 and 8-11 are under development)

The Globalizer system architecture is presented in Figure 3. The structural components are:

- Block 0 consists of the procedures for computing the function values (criteria and constraints) for the optimization problem being solved; this is an external block with respect to the Globalizer system and is provided by the user per a predefined interface.
- Blocks 1-4 form the optimization subsystem and solve the global optimization problems (Block 1), nonlinear programming (Block 2), multicriterial optimization (Block 3), and general decision making problems (Block 4). It is worth noting the sequential scheme of interaction between these components — the decision making problems are solved using the multicriterial optimization block, which, in turn, uses the nonlinear programming block, etc.
- Block 5 is a subsystem for accumulating and processing the search information; this is one of the basic subsystems — the amount of search information for complex optimization problems may appear to be quite large on the one hand, but, on the other hand, the efficiency of the global optimization methods depends to a great extent on how completely all of the available search data is utilized.
- Block 6 contains the dimensional reduction procedures based on Peano evolvents; the optimization blocks solve the reduced (one-dimensional) optimization problems; Block 6 provides interaction between the optimization blocks and the initial multidimensional optimization problem.
- Block 7 organizes the choice of parallel computation schemes in the Globalizer system subject to the computing system architecture employed (the numbers of cores in the processors, the availability of shared and distributed memory, the availability of

accelerators for computations, etc.) and the global optimization methods applied.

- Block 8 is responsible for managing the parallel processes when performing the global search (determining the optimal configuration of parallel processes, distributing the processes between computer elements, balancing computational loads, etc.).
- Block 9 is a common managing subsystem, which fully controls the whole computational process when solving global optimization problems.
- Block 10 is responsible for organizing the dialog interaction with users for stating the optimization problem, adjusting system parameters (if necessary), and visualizing and presenting the global search results.
- Block 11 is a set of tools for visualizing and presenting the global search results; the availability of tools for visually presenting the computational results enables the user to provide efficient control over the global optimization process.

The Globalizer system comes in two different options:

- Globalizer Pro is a research version of the system providing a full set of capabilities that are oriented toward solving of the most complicated global optimization problems.
- Globalizer Lite is a limited version of the Globalizer system accessible for free for solving global optimization problems with a medium level of difficulty, and is accessible free of charge.

5. Results of Numerical Experiments

In order to confirm the operability of the Globalizer system, results from two series of computational experiments are presented in this section.

The computational experiments were conducted using the Lobachevsky supercomputer at the State University of Nizhny Novgorod (operation system — CentOS 6.4, management system — SLURM). Each supercomputer node had 2 Intel Sandy Bridge E5-2660 processors, 2.2 GHz, with 64 GB RAM. Each processor had 8 cores (i.e. a total of 16 CPU cores were available at each node). In order to generate the executable program code, the Intel C++ 14.0.2 compiler was used.

A number of nodes in the computer system utilized Intel Xeon Phi 5110P processors, each containing 60 cores (a total of 240 threads). In addition, the supercomputer also included computer nodes equipped with two GPU NVIDIA Kepler K20X, each providing 14 stream multiprocessors (a total of 2688 CUDA-cores). In order to generate the executable code, the CUDA Toolkit 6.0 was used.

The problems generated by the GKLS-generator (Gaviano, M., Lera, D., Kvasov, D. E., Sergeyev, Y. D., 2003) were selected for the test problems. This generator allowed for obtaining the multiextremal optimization problems with *a priori* known properties: the number of local minima, the sizes of their attractors, the global minimum point, the function value at this point, etc. In order to simulate the computation costs inherent in the applied optimization problems (see, for example, Bastrakov, S., Gergel, V., Meyerov, I., et al, 2013), in all of the experiments computing the objective function was complicated by auxiliary computations, not altering the form of the function and the position of its minima.

We considered the global minimum y^* in the test problem to be found if the algorithm generated a trial point y^k in the δ-nearness of the global minimum i. e. $\|y^k - y^*\| \leqslant \delta$. The size of the nearness was selected as $\delta = \|b-a\| \sqrt[N]{\Delta}$ where a and b are boundaries of the search domain D, N is the dimensionality of the optimization problem. The maximum allowed number of parallel iterations was $K_{max} = 10000$, and the evolvent density parameter was $m = 10$.The computational experiments considered below were first carried out using the ExaMin system (see (Barkalov, K., Gergel V., 2015), (Barkalov, K., Gergel, V., Lebedev, I., 2015), (Barkalov, K., Gergel, V., Lebedev, I., Sysoev, A., 2015), (Gergel, V., Lebedev, I., 2015)) and then were reproduced using the Globalizer system.

5.1 *Parallel Computations Using Intel Xeon Phi processors*

The results of the numerical experiments with GPMAGS on an Intel Xeon Phi are provided in Table 5.1. The computations were performed using the Simple and Hard function classes with the dimensions equal to 4 and 5. The parameters $\Delta = 10^{-6}$ at $N = 4$ and $\Delta = 10^{-6}$ at $N = 5$ were used. When using the GPMAGS method, the parameter $r = 4.5$ was selected for the Simple class, and $r = 5.6$ was selected for the Hard class.

In the first series of experiments, serial computations using MAGS were executed. The average number of iterations performed by the method for solving a series of problems for each of these classes is shown in row I. The symbol ">" reflects the situation where not all problems of a given class were solved by a given method. It means that the algorithm was stopped once the maximum allowable number of iterations K_{max} was achieved. In this case, the K_{max} value was used for calculating the average number of iterations corresponding to the lower estimate of this average value. The number of unsolved problems is specified in brackets.

In the second series of experiments, parallel computations were executed on a CPU. The relative "speedup" in iterations achieved is shown in row II; the speedup of parallel computations was measured in relation to the sequential computations ($p = 1$).

Table 1. Results of numerical experiments

		p	$N = 4$			$N = 5$	
			Simple			Hard	
I	**Serial computations.** *Average number of iterations*	1	11953	25263		15920	>148342 (4)
II	**Parallel computations** of CPU. *Speedup*	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
III	**Parallel computations** of Xeon Phi. *Speedup*	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

The final series of experiments was executed using a Xeon Phi. The results of these computations are shown in row III; in this case, the speedup factor is calculated in relation to the PMAGS results on a CPU using eight cores ($p = 8$). It can be noted that relatively high speedup was achieved. For example, solving a problem from the 5-Hard class required on average only 633 parallel iterations on the Xeon Phi, whereas the number of iterations was more than 37 thousand when using all of the CPU's computing cores.

5.2 *Parallel Computations Using General Purpose Graphic Processors*

In the first series of experiments, only one graphic accelerator was used for solving 100 6-diminsional problems. The method's reliability parameter $r = 4.5$ was selected.

The average problem solving time using a GPU was 10.78 sec. whereas the averaged problem solving time using all of the node's 16 CPU cores was 53.8 sec.; an almost 5-fold improvement was observed.

A larger scale computational experiment was carried out for the problems with dimensions of $N = 8$ and $N = 10$: a total of 12 nodes within a cluster with 36 graphic accelerators (three accelerators per node) were employed, therefore, a total of 96768 CUDA-cores were employed.

The average time to solve the 8-dimensional problem was 405.6 seconds, 5.9 times faster than the CPU version of the algorithm. The average time to solve the 10-dimensional problem was 2,055.8 sec. The CPU was not used to solve the 10-dimensional problem because of the high degree of complexity in the computations.

6. Conclusions

In this paper, the Globalizer global optimization software system was examined for implementing a general scheme for the parallel solution of globally-optimized decision making using a wide spectrum of computer systems: with shared and distributed memory, NVIDIA graphic processors and Intel Xeon Phi coprocessors. This approach enables the efficient use of modern supercomputer systems to solve the most computational costly multiextremal optimization problems. The computational experiments have confirmed the prospects for this proposed approach.

Acknowledgements

This research was supported by the Russian Science Foundation, project No 16-11-10150 "Novel efficient methods and software tools for time-consuming decision making problems using supercomputers of superior performance."

References

[1] Censor, Y., Zenios, S. A., *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, 1998.

[2] Ciegis, R., Henty, D., Kågström, B., Zilinskas, J., *Parallel Scientific Computing and Optimization: Advances and Applications*, Springer, 2009.

[3] Fasano, G., Pintér, J. D., *Modeling and Optimization in Space Engineering*, Springer, 2013.

[4] Floudas, C. A., Pardalos, M. P., *State of the Art in Global Optimization: Computational Methods and Applications*, Kluwer Academic Publishers, Dordrecht, 1996.

[5] Floudas, C. A., Pardalos, M. P., *Recent Advances in Global Optimization*, Princeton University Press, 2016.

[6] Horst, R., Tuy, H., *Global Optimization: Deterministic Approaches*, Springer-Verlag, Berlin, 1990.

[7] Locatelli, M., Schoen, F., *Global Optimization: Theory, Algorithms, and Applications*, SIAM, 2013.

[8] Luque, G., Alba, E., *Parallel Genetic Algorithms. Theory and Real World Applications*, Springer-Verlag, Berlin, 2011.

[9] Pardalos, M. P., Zhigljavsky, A. A., Zilinskas, J., *Advances in Stochastic and Deterministic Global Optimization*, Springer, 2016.

[10] Pintér, J. D., *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*, Kluwer Academic Publishers, Dordrecht, 1996.

[11] Strongin, R. G., Gergel, V. P., Grishagin, V. A., Barkalov, K. A., *Parallel Computations for Global Optimization Problems*, Moscow State University (In Russian), Moscow, 2013.

[12] Strongin R.G., Sergeyev Ya.D., *Global optimization with non-convex constraints. Sequential and parallel algorithms*, Kluwer Academic Publishers, Dordrecht, 2000.

[13] Törn, A., Žilinskas, A., *Global optimization*, Lecture Notes in Computer Science 350 (1989).

[14] Zhigljavsky, A. A., *Theory of Global Random Search*, Kluwer Academic Publishers, Dordrecht, 1991.