Heterogeneous Parallel Computations for Solving Global Optimization Problems

Ilja Lebedev and Victor Gergel
Lobachevsky State University of Nizhni Novgorod, Nizhni Novgorod, Russian Federation
lebedev.ilya.g@gmail.com, gergel@unn.ru

Abstract
This paper presents an integrated approach to parallel solution of global optimization time-consuming problems. This approach is based on combining several schemes for reducing multidimensional optimization problems to one-dimensional ones. The schemes include using Peano space-filling curves and the recursive nested reduction technique. Finally, both ways are combined in a new unified block recursive nested optimization scheme. Based on this integrated scheme extensive parallel computations can be set up by using computational nodes with distributed memory, multicore processors with shared memory, graphics processors, and various computational accelerators. To evaluate the efficiency of proposed approach the results of the numerical experiments on Lobachevsky supercomputer using thousands of GPU cores are presented.

Keywords: Global Lipschitzian optimization, parallel computations, speedup, GPU

1 Introduction

The problem of multidimensional multiextremal optimization can be defined as a problem of finding for the minimum value of a real function \( \varphi(y) \)

\[
\varphi(y^*) = \min\{\varphi(y) : y \in D\},
\]

\( D = \{y \in \mathbb{R}^n : a_i \leq y_i \leq b_i, 1 \leq i \leq N\}. \quad (1)
\]

where \( a, b \in \mathbb{R}^N \) are given vectors.

The numerical solution of problem (1) reduces to computing an estimate \( y_i^* \in D \), which belongs to some neighborhood of the point \( y^* \) (for example, \( \| y^* - y_i^* \| \leq \varepsilon \) where \( \varepsilon > 0 \) is a given accuracy) based on a finite number \( k \) of computations of the optimized function values.

With respect to the considered problems, the fulfillment of the following conditions is assumed.

* This research was supported by the Russian Science Foundation, project №15-11-30022 "Global optimization, supercomputing computations, and applications".
First, it is proposed that the optimized function \( \phi(y) \) can be defined not analytically but by some numerical procedure for computing its values at the points of the domain \( D \); and these computations (further called trials) are time-consuming process.

Second, let us assume that \( \phi(y) \) satisfies the Lipschitz condition

\[
|\phi(y_1) - \phi(y_2)| \leq L \|y_1 - y_2\|, \quad y_1, y_2 \in D, \quad 0 < L < \infty.
\] (2)

This condition corresponds to the limited variations of function values in accordance with limited variations of the function argument. This assumption can be considered (with respect to the applied problems) as the reflection of limited power initiating the changes in the optimized problems.

The multiextremal optimization problems are essentially more computation-costly as compared to other kinds of the optimization problems as the global optimum is an integral characteristic of the problem being solved and requires to investigate the whole search domain. As a result, the global optimum search reduces to generating some mesh (grid) in the search domain and selecting the best function value on this grid. The computational costs for solving global optimization problems increase exponentially with increasing dimensionality (so called «curse of dimensionality»).

The computational cost can be reduced by constructing a nonuniform mesh in the search domain: the grid should be dense enough in the neighborhood of the global optimum and sparse further away from the sought optimum point. Generating such kind of meshes can only be provided by increasing the complexity of the global search numerical methods themselves. These methods based on nonuniform mesh generating schemes allow to double or even triple the dimensionality of the global optimization problems being solved, which is crucial for many applications (see, for instance, Bastrakov et al., 2013).

The Lipschitzian optimization problem (1)-(2) has been investigated by many researchers having used different approaches for designing global search algorithms – see, e.g., numerous references given in monographs (Törn, Žilinskas, 1989), (Horst, Tuy, 1990), (Zhigljavsky, 1991), (Horst, Pardalos, 1995), (Pintér, 1996), (Strongin, Sergeyev, 2000). Parallel computations are widely used for solving the global optimization problems as well. Generally, the following approaches are used to parallelize optimization methods.

First, one can subdivide the search domain between the processors and solve the optimization subproblems in these subdomains in parallel. However, this approach is a low efficient one since in such division of the search domain a minor number of the processors (one processor only in the worst case) would be solving the problem in the subdomains that contain the sought global minimum. The most processors would run in the subdomains, where there are no optimum points of the optimized problem.

Second, one can parallelize the computations of the optimized function. This approach may result in a sufficient speedup, but requires a special implementation for each particular problem solved.

Third, one can parallelize the optimization method procedures providing the selection of the points in the search domain to calculate next optimization trials. In this case, the parallelization techniques would also depend on the particular class of algorithms. Besides, these rules are often computationally simple enough, and there is no use to parallelize them (additional costs of parallel computations may eliminate the expected speedup).

As a result, it can be concluded that the most promising way to provide parallel computations for solving global optimization problems is the approach based on calculating several optimization trials in parallel. In this case, the major time-consuming part of computations is parallelized as calculating trials is computationally intensive. This approach is the most general one – it is applicable to a wide class of the global optimization methods. At the same time, this approach allows utilizing the heterogeneous computational resources of the most up-to-date supercomputers with the multicore processors, graphical units, computational accelerators efficiently.

The work continues studies which have been presented in (Gergel, Strongin, 2003), (Barkalov, Gergel, 2014), (Gergel, Grishagin and Israfilov, 2015)).
2 The Core Parallel Global Search Algorithm

In order to reduce the complexity of the global optimization algorithms that form a nonuniform mesh in the search domain, various approaches are used to reduce the multidimensional optimization problems to a family of the one-dimensional optimization problems (see section 3). Therefore, we will consider a one-dimensional multistart extremal optimization problem

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

where the objective function \( \varphi(x) \) satisfies the Lipschitz condition. Let us give a detailed description of the parallel global search algorithm (PGSA) developed to solve this problem.

Assume that \( p \geq 1 \) computational elements can be applied for computations. Then, one can execute \( p \) trials at given optimization iteration simultaneously. The total number of trials executed after \( n \) parallel iterations is \( k = pn \).

Assume \( n > 1 \) iterations of the method to be executed (as the trial points \( x^1, \ldots, x^n \) of the first iteration, different arbitrary points of the interval \([0,1]\) can be selected). Then, the trial points \( x^{k+1}, \ldots, x^{k+p} \) for current \((n+1)\)-th iteration are determined according to the following rules.

**Rule 1.** To renumber the points of the set

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

which includes the boundary points of the interval \([0,1]\) as well as the points of the preceding trials, by the lower indices in the order of increasing coordinate, i.e.

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

**Rule 2.** Assuming \( z_i = \varphi(x_i), 1 \leq i \leq k \), to calculate the values

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

where \( r > 1 \) is a given reliability parameter of the method and \( \Delta_i = x_i - x_{i-1} \).

**Rule 3.** To calculate the characteristics for each interval \((x_{i-1}, x_i), 1 \leq i \leq k + 1\) according to the formulas

$$R(1) = 2\Delta_1 - 4 \frac{z_1}{M}, \quad R(k+1) = 2\Delta_{k+1} - 4 \frac{z_{k+1}}{M};$$

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

**Rule 4.** To arrange the characteristics \( R(i), 1 \leq i \leq k + 1 \) in the descending order

$$R(t_1) \geq R(t_2) \geq \ldots \geq R(t_{k+1}) \geq R(t_{k+1})$$

and to select \( p \) highest characteristics with the indices of the intervals \( t_j, 1 \leq j \leq p \).

**Rule 5.** To execute the new trials at the points \( x^{k+j}, 1 \leq j \leq p \) calculated according to the formulas

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

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

The algorithm terminates the calculations if the condition \( \Delta_j \leq \varepsilon \) is satisfied at least for a single index \( t_j, 1 \leq j \leq p \); here \( \varepsilon > 0 \) is the given accuracy. The values
\[ \phi^*_i = \min_{i \in \mathbb{I}_k} \phi(x^i), \quad x^*_i = \arg \min_{i \in \mathbb{I}_k} \phi(x^i) \]

are taken as an estimate of the global optimal solution of the problem (1).

The mathematical validation of this parallel computational scheme is presented in (Strongin, Sergeyev, 2000). The most important point is that the interval characteristics \( R(i), 1 \leq i \leq k + 1 \) from (5) can be considered as some measures of the probability to find the global minimum point in the respective intervals. The inequality (6) arranges the intervals according to their characteristics. As a result, the trials are executed in parallel in the first \( p \) intervals with the highest probabilities of the global minimum point location.

## 3 Dimension Reduction of Global Optimization Problems

The reduction of the multidimensional global optimization problems to the one-dimensional ones and using the efficient one-dimensional global search algorithms to the reduced problems is one of the approaches to solving such problems. In this section, two well-known approaches for the dimension reduction as well as a generalization of these ones will be presented briefly.

### 3.1 The Dimension Reduction Using Peano Curves

The multidimensional optimization problems can be reduced to the one-dimensional ones using Peano curves mapping single-valued interval \([0,1]\) of the real axis onto the search domain ((Butz, 1968), (Goertzel, 1999), (Strongin, Sergeyev, 2000), (Hime, Oliveira and Petraglia, 2011), (Sergeyev, Strongin and Lera, 2013), (Barkalov, Gergel, 2014)). The issues of the numerical construction of the Peano curve-like mappings (evolvents) and the relevant theory were considered in details in (Strongin, Sergeyev, 2000). Here we note that a evolvent constructed numerically is an approximation to the theoretical Peano curve with the accuracy of the order of \( 2^{-m} \), where \( m \) is the density parameter.

Using this kind of mappings allows reducing the multidimensional problem (1) to a one-dimensional one

\[ \phi(y^i) = \phi(y(x^i)) = \min \{ \phi(y(x)) : x \in [0,1] \}. \]

It should be noted, that this transformation preserves the limitations on the relative differences of the optimized function: if the function \( \phi(y) \) satisfies the Lipschitz condition (2) with the constant \( L \) within the domain \( D \), then the function \( \phi(y(x)) \) will satisfy the uniform Hölder condition

\[ |\phi(y(x_i)) - \phi(y(x_j))| \leq H|x_i - x_j|^\beta \], \( x_i, x_j \in [0,1] \),

within the interval \([0,1]\). The Hölder constant \( H \) is related to the Lipschitz constant \( L \) by the relation

\[ H = 4Ld\sqrt{N}, \quad d = \max \{ b_i - a_i : 1 \leq i \leq N \}. \]

Relation (8) allows expanding the algorithm for solving the one-dimensional problems described in Section 2 for solving the multidimensional problems reduced to the one-dimensional ones. To do so, the interval lengths \( \Delta_i \) used in the expressions (3)–(5) of the algorithm, are substituted by the lengths in a new metrics

\[ \Delta_i = (x_i - x_{i-1})^\beta, \]

and the expression

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

should be applied instead of formula (7).
3.2 The Recursive Nested Optimization Scheme

The scheme of the recursive nested optimization is based on the well-known relation (see (Shi, Ólafsson, 2000), (Strongin, Sergeyev, 2000), (Sergeyev, Grishagin, 2001))

\[
\min \{ \varphi(y) : y \in D \} = \min_{a_1 \leq y_1 \leq b_1} \min_{a_2 \leq y_2 \leq b_2} ... \min_{a_N \leq y_N \leq b_N} \varphi(y),
\]

which allows the replacement of solving the multidimensional problem (1) by the solving a family of one-dimensional subproblems nested recursively.

Let us introduce into consideration a set of functions

\[
\varphi_i(y_1, ..., y_N) = \varphi(y_1, ..., y_N), \quad (10)
\]

\[
\varphi_i(y_1, ..., y_i, y_{i+1}, ..., y_N) = \min_{y_i \in [a_i, b_i]} \varphi_i(y_1, ..., y_i, y_{i+1}, ..., y_N), \quad 1 \leq i \leq N - 1.
\]

Then, according to relation (9), the solving of the initial problem (1) is reduced to the solving of a one-dimensional problem

\[
\varphi_i(y^*_i) = \min \{ \varphi_i(y_i) : y_i \in [a_i, b_i] \}. \quad (12)
\]

However, at that each calculation of the one-dimensional function value \( \varphi_i(y_i) \) at some fixed point implies the solving of a one-dimensional minimization problem

\[
\varphi_i(y_i) = \min \{ \varphi_i(y_i, y_{i+1}, ..., y_N) : y_i \in [a_i, b_i] \}, \quad \text{etc. up to the calculation of } \varphi_N \text{ according to (10)}.
\]

3.3 The Block Recursive Nested Optimization Scheme

The block recursive nested optimization scheme combines both reduction approaches described above to improve the efficiency of parallel computations for solving high dimensional global optimization problems.

Let us consider the vector \( y \) as a vector of block variables

\[
y = (y_1, y_2, ..., y_N) = (u_1, u_2, ..., u_M),
\]

where the \( i \)-th block variable \( u_i \) is a vector of the dimensionality \( N_i \) from the components of the vector \( y \) taken sequentially, i.e.

\[
u_i = (y_{i_1}, y_{i_2}, ..., y_{i_{N_i}}),
\]

\[
u_2 = (y_{i_2 + 1}, y_{i_2 + 2}, ..., y_{i_2 + N_{i_2}}), \quad \ldots,
\]

at that \( N_1 + N_2 + \ldots + N_M = N \).

Using the new block variables, the basic relation of the recursive nested scheme (9) can be rewritten in the form

\[
\min_{y \in D} \varphi(y) = \min_{a_1 \leq y_1 \leq b_1} \min_{a_2 \leq y_2 \leq b_2} \ldots \min_{a_M \leq y_M \leq b_M} \varphi(y),
\]

where the subdomains \( D_i, 1 \leq i \leq M \) are the projections of the initial search domain \( D \) onto the subspaces, corresponding to the block variables \( u_i, 1 \leq i \leq M \).

The algorithmic scheme, determining the method for solving the problem (1) based on the relation (13), generally, is the same as in the recursive nested scheme (10)–(12). It is required to substitute the initial variables \( y_i, 1 \leq i \leq N \) by the block variables \( u_i, 1 \leq i \leq M \) only.

At that, the fact that the nested subproblems

\[
\varphi_i(u_1, ..., u_M) = \min_{a_{i_1} \leq u_{i_1} \leq b_{i_1}} \varphi_i(u_1, ..., u_M), \quad 1 \leq i \leq M - 1,
\]

in the block recursive nested scheme are the multidimensional ones is the essential difference from the initial recursive nested scheme. The dimensionality reduction method based on Peano curves can be applied for solving these problems.
The number of block variables and the size of each block are the parameters of the block recursive nested scheme. These ones can be used for the forming of the subproblems with the necessary properties. For example, if \( M = N \) i.e. \( u_i = y_i, 1 \leq i \leq N \), then the block scheme is identical to the initial one; and each nested subproblem is a one-dimensional one. If \( M = 1 \) i.e. \( u = u_1 = y \), then solving the optimization problem is an equivalent to its solving without using nested optimization scheme.

4 Parallel Global Optimizations on Heterogeneous Clusters

In order to organize the parallel computations we will use a small number of the nesting levels (2-3), at which the initial problem of large dimensionality is subdivided into 2-3 nested subproblems of less dimensionality. Then, applying the parallel global optimization methods to solve the nested subproblems (14) in the block recursive scheme (13), we obtain a parallel algorithm with a wide degree of variability.

For example, one can vary the number of processors at different levels of optimization (i.e. in solving the subproblems with respect to different variables \( u_i \)), apply various parallel optimization methods at different levels, etc.

To describe the parallelism of the recursive nested optimization scheme, let us introduce the parallelization vector

\[
\pi = (\pi_1, \pi_2, \ldots, \pi_M),
\]

where \( \pi_i, 1 \leq i \leq M \) denotes the number of the subproblems of the \((i+1)\)-th nesting level arising as a result of the execution of the parallel iterations at the \(i\)-th level, being solved in parallel. For the \(M\)-th level the value \( \pi_M \) means the number of the parallel trials in the process of the minimization of the function \( \phi(u_1, \ldots, u_M) = \phi(y_1, \ldots, y_M) \) with respect to the variable \( u_M \) at fixed values \( u_1, \ldots, u_{M-1} \), i.e. the number of the values of the objective function \( \phi(y) \) computed in parallel.

In the general case, the values \( \pi_i, 1 \leq i \leq M \) may depend on various parameters and may vary in the course of optimization. Implementing the parallel optimization algorithms according to the block recursive nested scheme with the parallelization vector (15) allows using

\[
\Pi = 1 + \sum_{i=1}^{M} \prod_{j=1}^{i} \pi_j,
\]

processors/cores running in parallel for solving the problem (1).

Using various parameters of the parallelization vector, one can adopt the algorithm for running on a heterogeneous computer system. At that, the parallel computation of multiple values of the objective function simultaneously is the calculation procedure, which can be executed by a graphics processor efficiently. The data transfer from CPU to GPU will be the minor one: it is required to send the coordinates of the trial points to GPU and to receive back the function values at these points only. The processing of the trial results according to the algorithm requires calculations with a large size of the accumulated optimization data and should be run on CPU.

The general scheme of the parallel computations using several computational nodes and several GPU is given in Fig. 1. The parallel computations are arranged in a tree corresponding to the levels of the nested subproblems. According to this scheme, the nested subproblems

\[
\phi_i(u_1, \ldots, u_i) = \min_{u_{i+1}, \ldots, u_n} \phi_{i+1}(u_{i+1}, \ldots, u_n)
\]

at \( i = 1, \ldots, M-2 \) are solved using CPU only. In these subproblems, the optimized function values are not computed: the computation of the value of the function \( \phi_i(u_1, \ldots, u_i) \) is the solving of the optimized

58
problem of the next level. Each subproblem is solved in separate process; the exchange of the computed values is performed using MPI.

The subproblem of the last \((M-1)\)-th level

\[
\varphi_{M-1}(u_1, \ldots, u_{M-1}) = \min_{u_M = \varphi_M} \varphi_M(u_1, \ldots, u_M)
\]
differs from all preceding subproblems. In this subproblem, the values of the optimized function are computed since \(\varphi_M(u_1, \ldots, u_M) = \varphi(y_1, \ldots, y_M)\). This subproblem is solved by CPU as well, however, the function values are computed by the graphics processors. At that, the rules 1 – 4 of the parallel global optimization algorithm are executed using CPU. The trial points computed at Step 4 of the algorithm are accumulated in a memory buffer and then are transferred to the graphics processor. The computations of the function values at these points are executed using GPU. After that, the trial results are transferred to CPU (again, via the memory buffer).

The basic implementation of this scheme will correspond to a two-component parallelization vector \(\pi = (\pi_1, \pi_2)\). Here \(\pi_1 + 1\) will correspond to the number of the MPI-processes and \(\pi_2\) will correspond to the number of the GPU cores applied. Therefore, the total number of the cores applied (in both CPU and GPU) will be defined as \(1 + \pi_1 \pi_2\).

Note also, that in the case if it is impossible to compute the optimized function values at the last parallelization level using GPU efficiently, one can use the cores of CPU or the ones of the Xeon Phi coprocessor in the hyperthread mode.

![Parallel computation scheme based on block recursive nested approach](image)

**Figure 1:** Parallel computation scheme based on block recursive nested approach

5 The Results of the Computational Experiments

The computational experiments have been carried out on the supercomputer of the Lobachevsky State University of Nizhni Novgorod (the operating system CentOS 6.4, the management system SLURM). A supercomputer node had two Intel Sandy Bridge E5-2660 2.2 GHz processors, 64 Gb RAM, and two NVIDIA Kepler K20X GPUs. Each CPU had 8 cores (i.e. there were total 16 CPU
cores per a node). In the graphics processors, there were 14 stream multiprocessors (2688 CUDA cores). Intel C++ 14.0.2 compiler and CUDA Toolkit 6.0 were used.

The optimization problems generated by the GKLS generator (Gaviano, Lera, Kvasov, and Sergeyev, 2003) have been selected as the test problems. This generator allows obtaining the global optimization problems with prior known properties: the number of local minima, the sizes of their neighborhoods, the global minimum point, the function value at this point, etc. In order to simulate the computation costs featuring the applied optimization problems, the computations of the objective function in all the experiments carried out were complicated by additional computation loads (the summation of a series of 20 thousand elements).

To examine the efficiency of the proposed approach the parallel global search algorithm described in section 2 executed in the sequential mode i. e. at $p=1$ is compared with two well-known algorithms: DIRECT (Jones, Perttunen and Stuckman, 1993) and DIRECTl (Gablonsky, Kelley, 2001) (the results of numerical experiments for these algorithms are given in (Sergeyev, Kvasov, 2006). The numerical comparison has been carried out based on the Simple and Hard function classes of the dimensionality equal to 4 and 5, since the solving of the problems of the dimensionality equal to 2 and 3 required a small number of iterations and, hence, there is no use in application of GPU to solving these problems. The global minimum $y^*$ was considered to be found if the algorithm has generated a trial point $y_k$ in the $G$-nearness of the global minimum, i. e. $\|y^* - y_k\| \leq \delta$. At that, the magnitude of the proximity was selected as $\delta = \|b - a\| \sqrt{N}$. Here $N$ is the dimensionality of the problem being solved, $a$ and $b$ are the boundaries of the search domain $D$, the parameter $\Delta=10^{-6}$ at $N=4$ and $\Delta=10^{-7}$ at $N=5$. When using the GSA method, the parameter $r$ was set to 4.5 and 5.6 for the Simple and Hard function classes, respectively; the evolvent density parameter was fixed to $m = 10$. The maximum allowed number of iterations was $K_{max} = 1 000 000$.

The averaged numbers of iterations $k_{av}$ executed by each method for the solving the series of 100 problems are presented in Table 1. The symbol «>» denotes the situation when not all the problems of a class have been solved by the method. This means that the algorithm has been terminated upon achieving the maximum allowed number of iterations $K_{max}$. In this case, the value $K_{max} = 1 000 000$ was used in the calculation of the averaged number of iterations $k_{av}$ that corresponds to the lower estimate of this averaged value. The number of the unsolved problems is given in the brackets.

<table>
<thead>
<tr>
<th>N</th>
<th>Problem class</th>
<th>DIRECT</th>
<th>DIRECTl</th>
<th>GSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Simple</td>
<td>&gt;47282 (4)</td>
<td>18983</td>
<td>11953</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>&gt;95708 (7)</td>
<td>68754</td>
<td>25263</td>
</tr>
<tr>
<td>5</td>
<td>Simple</td>
<td>&gt;16057 (1)</td>
<td>16758</td>
<td>15920</td>
</tr>
<tr>
<td></td>
<td>Hard</td>
<td>&gt;217215 (16)</td>
<td>&gt;269064 (4)</td>
<td>&gt;148342 (4)</td>
</tr>
</tbody>
</table>

Table 1. The averaged number of iterations for the compared methods

As one can see from Table 1, GSA excels DIRECT and DIRECTl methods in the averaged number of iterations for all problem classes. At that, in the 5-Hard class all methods failed to solve some problems: DIRECT hasn’t solved 16 problems, DIRECTl and GSA 4 problems each.

The experiment with using one graphics processor has been carried out by solving a series of 100 six-dimensional problems from the Simple class. The magnitude of the nearness was $\delta=0.0 \|b - a\|$. The reliability parameter of the method was selected to be $r = 4.5$; the evolvent density parameter was set to $m = 10$. The maximum allowed number of the parallel iterations was $K_{max} = 10 000$. Since the problems have been solved using a single graphics processor, the recursive nested scheme has not been applied.
The averaged time of solving of a problem using GPU was 10.78 sec. while the averaged time of solving the problems using all 16 CPU cores available in a node was 53.8 sec.; so almost 5-fold speedup has been observed.

A larger-scale computational experiment has been carried out for the problems of the dimensionality $N = 8$ and $N = 10$: 12 cluster nodes with 36 graphics processors (three graphics processors per a node) have been used. So far, 96 768 CUDA cores have been employed.

According to the block recursive scheme (13), two levels of subproblems with the dimensionalities $N_1 = N_2 = 4$ for the 8-dimensional problem and $N_1 = N_2 = 5$ for the 10-dimensional one have been used. The parallelization vector (15) was selected as $\pi = (36, 2688)$ according to the total number of the GPU (the component $\pi_1$) and the CUDA cores per a graphics processor (the components $\pi_2$) employed. The time of solving the 8-dimensional problem was 405.6 sec., the speedup was 5.9 times as compared to the CPU version of the algorithm. The time of solving the 10-dimensional problem was 2055.8 sec. The speedup for this case is not computed because the 10-dimensional problem has not been solved using CPU due to large complexity of computations.

6 Conclusions

The results of the numerical experiments carried out on the basis of a series of test optimization problems of various dimensionality have demonstrated that the proposed block recursive nested optimization scheme of the dimensionality reduction combined with the parallel global search algorithm can be implemented using up-to-date computer systems efficiently. This scheme is featured by:

- high parallelization capabilities (about $10^5$ cores have been applied in the numerical experiments);
- low mutual data dependencies of the computations run in parallel (the trial points and results obtained within current iteration of the method should be transferred between the parallel processes only).

The obtained results allow assuming that the proposed scheme of the parallel computations will be efficient when using more cores (approximately $10^6$) as well. Further developments will consist also in taking into account the local information on the optimized function behavior, in extending the proposed approach for solving the constrained and multicriterial optimization problems.

References


