Adaptive Global Optimization Using Graphics Accelerators

Konstantin Barkalov[1(✉)], Ilya Lebedev[1], and Vassili Toropov[1,2]

[1] Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
{konstantin.barkalov,ilya.lebedev}@itmm.unn.ru
[2] Queen Mary University of London, London, UK
v.v.toropov@qmul.ac.uk

Abstract. Problems of multidimensional multiextremal optimization and numerical methods for their solution are considered. The general assumption is made about the function being optimized: it satisfies the Lipschitz condition with an a priori unknown constant. Many approaches to solving problems of this class are based on reducing the dimension of the problem; i.e. addressing a multidimensional problem by solving a family of problems with lower dimension. In this work, an adaptive dimensionality reduction scheme is investigated, and its implementation using graphic accelerators is proposed. Numerical experiments on several hundred test problems were carried out, and they confirmed acceleration in the developed GPU version of the algorithm.

Keywords: Global optimization · Multiextremal functions · Reduction of dimensionality · Peano space-filling curves · Recursive optimization · Graphics accelerators

1 Introduction

A promising direction in the field of parallel global optimization (which, indeed, is true in many areas related to the software implementation of time-consuming algorithms) is the use of graphics processing units (GPUs). In the past decade, graphics accelerators have rapidly increased performance to meet the ever-growing demands of graphics application developers. Additionally, in the past few years some principles for developing graphics hardware have changed, and as a result it has become more programmable. Today, a graphics accelerator is a flexibly programmable, massive parallel processor with high performance, which is in demand for solving a range of computationally time-consuming problems [14].

However, the potential for graphics accelerators to solve global optimization problems has not yet been fully realized. Using GPUs, they basically parallelize nature-inspired optimization algorithms, which are somehow based on the idea of random search (see, for example, [5,7,17]). By virtue of their stochastic nature, algorithms of this type guarantee convergence to the global minimum only in

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2020, CCIS 1331, pp. 150–161, 2020.
https://doi.org/10.1007/978-3-030-64616-5_13

the sense of probability, which differentiates them unfavorably from deterministic methods.

With regard to many deterministic algorithms of Lipschitzian global optimization with guaranteed convergence, parallel variants have been proposed [4,13,19]. However, these versions of algorithms are parallelized on CPU using shared and/or distributed memory; presently, no GPU implementations have been made. For example, [19] describes parallelization of an algorithm based on the ideas of the branch and boundary method using MPI and OpenMP.

Within the framework of this research, we consider the problems of capturing the optimum, which are characterized by a lengthy period for calculating the values of objective function in comparison with the time needed for processing them. For example, objective function can be specified using systems of linear algebraic equations, systems of ordinary differential equations, etc. Currently, graphics accelerators can be used to solve problems of this type. Moreover, an accelerator can solve several such problems at once [16]; i.e., using GPU, one can calculate multiple function values simultaneously.

Thus, calculating the optimization criterion can be implemented on GPU, and the role of the optimization algorithm (running on CPU) consists in the selection of points for conducting parallel trials. This scheme of working with the accelerator is fully consistent with the work of the parallel global search algorithm developed at the Lobachevsky State University of Nizhni Novgorod and presented in a series of papers [1–3,9–12].

2 Multidimensional Parallel Global Search Algorithm

Let's consider the problem of finding the global minimum of the N-dimensional function $\varphi(y)$ in the hyperinterval $D = \{y \in R^N : a_i \leqslant x_i \leqslant b_i, 1 \leqslant i \leqslant N\}$. We will assume that the function satisfies the Lipschitz condition with an a priori unknown constant L.

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

$$|\varphi(y_1) - \varphi(y_2)| \leqslant L\|y_1 - y_2\|, y_1, y_2 \in D, 0 < L < \infty. \tag{2}$$

In this work we will use an approach based on the idea of reducing dimensionality using the Peano space-filling curve $y(x)$, which continuously and unambiguously maps a segment of the real axis $[0, 1]$ onto an n-dimensional cube

$$\{y \in R^N : -2^{-1} \leqslant y_i \leqslant 2^{-1}, 1 \leqslant i \leqslant N\} = \{y(x) : 0 \leqslant x \leqslant 1\}. \tag{3}$$

The questions of numerical construction of approximations to Peano curve (*evolvents*) and the corresponding theory are discussed in detail in [21,23]. Using evolvents $y(x)$ reduces the multidimensional problem (1) to a one-dimensional problem

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

An important property is that the relative differences of the function remain limited: if the function $\varphi(y)$ in the region D satisfies the Lipschitz condition, then the function $\varphi(y(x))$ in the interval $[0,1]$ will satisfy a uniform Hölder condition

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

where the Hölder constant H is related to the Lipschitz constant L by the ratio $H = 2L\sqrt{N+3}$. Therefore, without limiting generality, we can consider minimizing the one-dimensional function $f(x) = \varphi(y(x)), x \in [0,1]$, which satisfies the Hölder condition.

The algorithm for solving this problem (Global Search Algorithm, GSA) involves constructing a sequence of points x_k, in which the values of objective function $z_k = f(x_k)$ are calculated. We will call the process of computing the value of a function at a single point a trial. Assume that we have $p \geqslant 1$ computational elements at our disposal and p trials which are performed simultaneously (synchronously) within a single iteration of the method. Let $k(n)$ denote the total number of trials performed after n parallel iterations.

At the first iteration of the method, the trial is carried out at an arbitrary internal point x^1 of the interval $[0,1]$. Let $n > 1$ iterations of the method be performed, during which trials were carried out at $k = k(n)$ points $x^i, 1 \leqslant i \leqslant k$. Then the trial points $x^{k+1}, \ldots, x^{k+p}$ of the next $(n+1)$th iteration are determined in accordance with the following rules.

Step 1. Renumber the points of the set $X_k = \{x^1, \ldots, x^k\} \cup \{0\} \cup \{1\}$, which includes the boundary points of the interval $[0,1]$, as well as the points of the previous trials, with the lower indices in the order of their increasing coordinate values, i.e.

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

Step 2. Assuming $z_i = f(y(x_i)), 1 \leqslant i \leqslant k$, calculate the values

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

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

Step 3. For each interval $(x_{i-1}, x_i), 1 \leqslant i \leqslant k+1$, calculate the characteristic in accordance with the formulas

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

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

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

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

and select p of the largest characteristics with interval numbers $t_j, 1 \leqslant j \leqslant p$.

Step 5. Carry out new trials at the points $x_{k+j}, 1 \leqslant j \leqslant p$, calculated using the formulas

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

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

The algorithm stops working if the condition $\Delta_{t_j} \leqslant \varepsilon$ is satisfied for at least one number $t_j, 1 \leqslant j \leqslant p$; here $\varepsilon > 0$ is the specified accuracy. As an estimate of the globally optimal solution to the problem (1), the values are selected

$$f_k^* = \min_{1 \leq i \leq k} f(x^i), x_k^* = \arg \min_{1 \leq i \leq k} f(x^i).$$

For the rationale in using this method of organizing parallel computing see [23].

3 Dimensionality Reduction Schemes in Global Optimization Problems

3.1 Dimensionality Reduction Using Multiple Mappings

Reducing multidimensional problems to one-dimensional ones through the use of evolvents has important properties such as continuity and preserving uniform bounding of function differences with limited argument variation. However, some information about the proximity of points in multidimensional space is lost, since the point $x \in [0,1]$ has only left and right neighbors, and the corresponding point $y(x) \in R^N$ has neighbors in 2^N directions. As a result, when using evolvents, the images y', y'' that are close in N-dimensional space can correspond to rather distant preimages x', x'' on the interval $[0,1]$. This property leads to redundant calculations, because several limit points x', x'' of the sequence of the trial points generated by the method on the segment $[0,1]$, can correspond to a single limit point y in N-dimensional space.

A possible way to overcome this disadvantage is to use a set of evolvents (*multiple mappings*)

$$Y_L(x) = \left\{y^0(x),\ y^1(x), ...,\ y^L(x)\right\}$$

instead of using a single Peano curve $y(x)$ (see [22,23]). For example, each Peano curve $y^i(x)$ from $Y_L(x)$ can be obtained as a result of some shifting $y(x)$ along the main diagonal of the hyperinterval D. Another way is to rotate the evolvent $y(x)$ around the origin. The set of evolvents that have been constructed allows us to obtain for any close images y', y'' close preimages x', x'' for some mapping $y^i(x)$.

Using a set of mappings leads to the formation of a corresponding set of one-dimensional multiextremal problems

$$\min\left\{\varphi(y^l(x)) : x \in [0,1],\ \right\},\ 0 \leqslant l \leqslant L.$$

Each problem from this set can be solved independently, and any calculated value $z = \varphi(y')$, $y' = y^i(x')$ of the function $\varphi(y)$ in the i-th problem can be interpreted as calculating the value $z = \varphi(y')$, $y' = y^s(x'')$ for any other s-th problem without repeated labor-intensive calculations of the function $\varphi(y)$. Such informational unity makes it possible to solve the entire set of problems in a parallel fashion. This approach was discussed in detail in [3].

3.2 Recursive Dimensionality Reduction Scheme

The recursive optimization scheme is based on the well-known relation

$$\min \varphi(y) : y \in D = \min_{a_1 \leqslant y_1 \leqslant b_1} \min_{a_2 \leqslant y_2 \leqslant b_2} \dots \min_{a_1 \leqslant y_N \leqslant b_N} \varphi(y), \tag{4}$$

which allows one to replace the solution of the multidimensional problem (1) with the solution of a family of one-dimensional subproblems recursively related to each other. Let's introduce a set of functions

$$\varphi_N(y_1, \dots, y_N) = \varphi(y_1, \dots, y_N), \tag{5}$$

$$\varphi_i(y_1, \dots, y_i) = \min_{a_{i+1} \leqslant y_{i+1} \leqslant b_{i+1}} \varphi_{i+1}(y_1, \dots, y_i, y_{i+1}), 1 \leqslant i \leqslant N-1. \tag{6}$$

Then, in accordance with the relation (4), the solution of the original problem (1) is reduced to the solution of a one-dimensional problem

$$\varphi_1(y_1^*) = \min\{\varphi_1(y_1), y_1 \in [a, b]\}. \tag{7}$$

However, each calculation of the value of the one-dimensional function $\varphi_1(y_1)$ at some fixed point corresponds to the solution of a one-dimensional minimization problem

$$\varphi_2(y_1, y_2^*) = \min\{\varphi(y_1, y_2) : y_2 \in [a_2, b_2]\}.$$

And so on, until the calculation of φ_N according to (5).

For the recursive scheme described above, a generalization (*block recursive scheme*) is proposed that combines the use of evolvents and a recursive scheme in order to efficiently parallelize computations.

Consider the vector y as a vector of block variables

$$y = (y_1, \dots, y_N) = (u_1, u_2, \dots, u_M),$$

where the i-th block variable u_i is a vector of sequentially taken components of the vector y, i.e. $u_1 = (y_1, y_2, \dots, y_{N_1})$,$u_2 = (y_{N_1+1}, y_{N_1+2}, \dots, y_{N_1+N_2})$,$\dots$, $u_M = (y_{N-N_M+1}, y_{N-N_M+2}, \dots, y_N)$, while $N_1 + N_2 + \dots + N_M = N$.

Using new variables, the main relation of the nested scheme (4) can be rewritten as

$$\min_{y \in D} \varphi(y) = \min_{u_1 \in D_1} \min_{u_2 \in D_2} \dots \min_{u_M \in D_M} \varphi(y), \tag{8}$$

where the subdomains $D_i, 1 \leqslant i \leqslant M$, are projections of the original search domain D onto the subspaces corresponding to the variables $u_1, 1 \leqslant i \leqslant M$.

The formulas that determine the method for solving problem (1) based on relations (8) generally coincide with the recursive scheme (5)–(7). One need only replace the original variables $y_i, 1 \leqslant i \leqslant N$, with block variables $u_1, 1 \leqslant i \leqslant M$. In this case, the fundamental difference from the original scheme is the fact that the block scheme has nested subproblems

$$\varphi_i(u_1, \ldots, u_i) = \min_{u_{i+1} \in D_{i+1}} \varphi_{i+1}(u_1, \ldots, u_i, u_{i+1}), 1 \leqslant i \leqslant M-1, \qquad (9)$$

which are multidimensional, and to solve them a method of reducing dimensionality based on Peano curves can be applied.

3.3 Adaptive Dimensionality Reduction Scheme

Solving the resulting set of subproblems (9) can be organized in various ways. The obvious method (elaborated in detail in [11] for the nested optimization scheme and in [1] for the block nested optimization scheme) is based on solving subproblems in accordance with the recursive order of their generation. However, in this case there is a significant loss of information about the target function.

Another approach is an adaptive scheme in which all subproblems are solved simultaneously, which makes it possible to more fully take into account information about a multidimensional problem and thereby to speed up the process of solving it. In the case of one-dimensional subproblems, this approach was theoretically substantiated and tested in [10,12], and in the paper [2] a generalization of the adaptive scheme for multidimensional subproblems was proposed.

The adaptive dimensionality reduction scheme changes the order in which subproblems are solved: they will be solved not one by one (in accordance with their hierarchy in the task tree), but simultaneously, i.e., there will be a number of subproblems that are in the process of being solved. Under the new scheme:

- to calculate the value of the function of the i-th level from (9), a new $(i+1)$th level problem is generated in which trials are carried out, after which a new generated problem is included in the set of existing problems to be solved;
- the iteration of the global search consists in choosing p (the most promising) problems from the set of existing problems in which trials are carried out; points for new trials are determined in accordance with the parallel global search algorithm from Section 2;
- the minimum values of functions from (9) are their current estimates based on the accumulated search information.

A brief description of the main steps of a block adaptive dimensionality reduction scheme is as follows. Let the nested subproblems in the form (9) be solved using the global search algorithm described in Section 2. Then each subproblem (9) can be assigned a numerical value called the characteristic of this problem. As such, we can take the maximum characteristic $R(t)$ of the intervals formed in this problem. In accordance with the rule for calculating characteristics, the higher the value of the characteristic, the more promising the subproblem is in the continued search for the global minimum of the original problem (1).

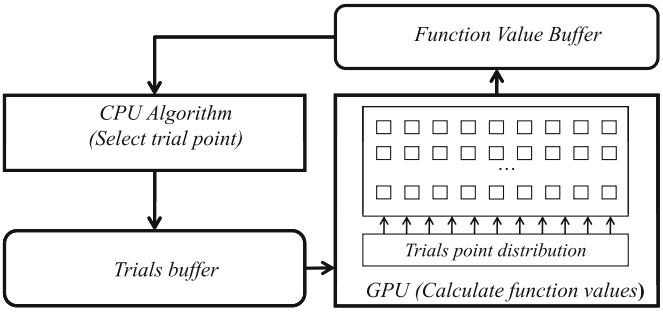

Fig. 1. Scheme of information exchanges in the GPU algorithm

Therefore, at each iteration, subproblems with the maximum characteristic are selected for conducting the next trial. The trial either leads to the calculation of the value of the objective function $\varphi_1(y)$ (if the selected subproblem belonged to the level $j = M$), or generates new subproblems according to (9) for $j \leqslant M-1$. In the latter case, the newly generated problems are added to the current set of problems, their characteristics are calculated, and the process is repeated. The optimization process is completed when the root problem satisfies the condition for stopping the algorithm that solves this problem. Some results pointing in this direction are presented in [2].

4 GPU Implementation

4.1 General Scheme

In relation to global optimization methods, an operation that can be efficiently implemented on GPU is the parallel calculation of many values of the objective function at once. Naturally, this requires implementing a procedure for calculating the value of a function on GPU. Data transfers from CPU to GPU will be minimal: one need only transfer the coordinates of the trial points to GPU, and get back the function values at these points. Functions that determine the processing of trial results in accordance with the algorithm, and require work with large amount of accumulated search information, can be efficiently implemented on CPU.

The general scheme for organizing calculations using GPU is shown in Fig. 1. In accordance with this scheme, steps 1–4 of the parallel global search algorithm are performed on CPU. The coordinates of the p trial points calculated in step 4 of the algorithm are accumulated in the intermediate buffer and then transmitted to GPU. On GPU the function values are calculated at these points, after which the trial results (again through the intermediate buffer) are transferred to CPU.

4.2 Organization of Parallel Computing

To organize parallel calculations, we will use a set of evolvents and a block adaptive dimensionality reduction scheme. We take a small number of nesting

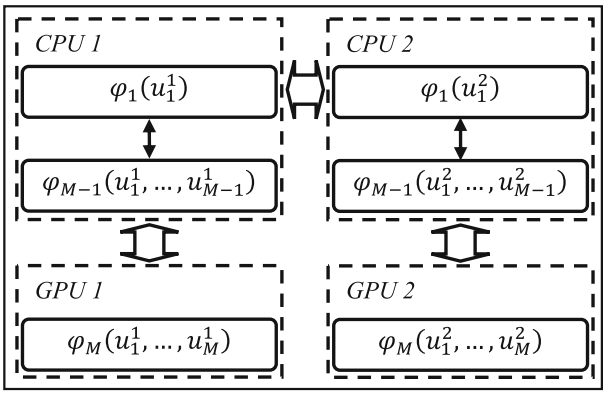

Fig. 2. Diagram of parallel computing on a cluster

levels, in which the original large dimensionality problem is divided into 2–3 nested lower dimensional subproblems. We will use multiple evolvents only at the upper level of nesting, corresponding to the variable u_1. This subproblem will be reduced to a set of one-dimensional problems that will be solved in parallel, each in a separate process. Trial results at point x obtained for the problem solved by a specific processor are interpreted as the trial results in the remaining problems (at the corresponding points $u_1^1, ..., u_1^s$). Then, applying an adaptive scheme to solve the nested subproblems (9), we get a parallel algorithm with a wide degree of variability.

Figure 2 shows the general scheme of organizing computations using several cluster nodes and several GPUs. In accordance with this scheme, nested subproblems $\varphi_i(u_1, ..., u_i) = \min\limits_{u_{i+1} \in D_{i+1}} \varphi_{i+1}(u_1, ..., u_i, u_{i+1})$ with $i = 1, \ldots, M-2$ are solved using CPU only. The values of the function are not calculated directly in these subproblems: the calculation the function value $\varphi_i(u_1, ..., u_i)$ is a solution to the minimization problem at the next level. The subproblem of the last $(M-1)$-th level

$$\varphi_{M-1}(u_1, ..., u_{M-1}) = \min_{u_M \in D_M} \varphi_M(u_1, ..., u_M)$$

differs from all the previous subproblems; it calculates the values of the objective function, since $\varphi_M(u_1, ..., u_M) = \varphi(y_1, ..., y_N)$. This subproblem transfers data between CPU and GPU.

5 Numerical Experiments

The numerical experiments were carried out using the Lomonosov supercomputer (Lomonosov Moscow State University). Each supercomputer node included two quad-core processors Intel Xeon X5570, two NVIDIA Tesla X2070 and 12

Table 1. Average number of iterations k_{av}

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

Gb RAM. To build the program for running on the Lomonosov supercomputer, the GCC 4.3.0 compiler, CUDA 6.5 and Intel MPI 2017 were used.

Note that well-known test problems from the field of multidimensional global optimization are characterized by a short time of calculating the values of the objective function. Therefore, in order to simulate the computational complexity inherent in applied optimization problems [18], the calculation of the objective function in all experiments was complicated by additional calculations that do not change the form of the function and the location of its minima (summing the segment of the Taylor series). In the experiments carried out the average time for calculating the function values was 0.01 s, which exceeds the latency of the network or the data transfer time between CPU and GPU.

In the paper [8] a GKLS generator is described that allows one to generate multiextremal optimization problems with previously known properties: the number of local minima, the size of their regions of attraction, the global minimum point, the value of the function in it, etc.

Below are the results of a numerical comparison of three sequential algorithms: DIRECT [15], DIRECTl [6] and Global Search Algorithm (GSA) from Section 2. A numerical comparison was carried out on the Simple and Hard function classes of dimension 4 and 5 from [8]. The global minimum y^* was considered found if the algorithm generated a trial point y^k in the δ-neighborhood of the global minimum, i.e. $\|y^k - y^*\| \leqslant \delta$. The size of the neighborhood was chosen (in accordance with [20]) as $\delta = \|b - a\| \sqrt[N]{\Delta}$, here N is the dimension of the problem to be solved, a and b are the boundaries of the search domain D, the parameter $\Delta = 10^{-6}$ for $N = 4$ and $\Delta = 10^{-7}$ for $N = 5$. When using the GSA method for the Simple class, the parameter $r = 4.5$ was selected, for the Hard class $r = 5.6$; the parameter for constructing the Peano curve was $m = 10$. The maximum number of iterations allowed was $K_{max} = 10^6$.

Table 1 shows the average number of iterations, k_{av}, that the method performed when solving a series of problems from these classes. The symbol " > " reflects a situation where not all problems of the class were solved by any method whatsoever. This means that the algorithm was halted because the maximum allowed number of K_{max} iterations was reached. In this case, the value $K_{max} = 10^6$ was used to calculate the average value of the number of iterations, k_{av}, which corresponds to the lower estimate of this average value. The number of unsolved problems is indicated in parentheses.

Table 2. Speedup on CPU

	Iteration speedup				Time speedup			
	$N = 4$		$N = 5$		$N = 4$		$N = 5$	
	Simple	Hard	Simple	Hard	Simple	Hard	Simple	Hard
$p = 2$	6,6	3,2	2,1	6,6	3,2	1,5	0,7	2,1
$p = 4$	21,5	10,0	6,9	19,2	5,2	2,4	0,9	2,3

Table 3. Speedup on one GPU

	Iteration speedup				Time speedup			
	$N = 4$		$N = 5$		$N = 4$		$N = 5$	
	Simple	Hard	Simple	Hard	Simple	Hard	Simple	Hard
$p = 64$	5,1	3,9	1,2	9,3	2,2	1,9	0,5	4,0
$p = 256$	19,9	15,0	11,9	39,6	8,3	6,9	3,4	11,0
$p = 1024$	15,6	52,9	22,1	105,7	5,2	20,0	2,2	10,2

Table 4. Speedup on two GPUs

	Iteration speedup				Time speedup			
	$N = 4$		$N = 5$		$N = 4$		$N = 5$	
	Simple	Hard	Simple	Hard	Simple	Hard	Simple	Hard
$p = 64$	11,3	6,8	8,2	19,8	2,4	1,6	1,3	3,2
$p = 256$	39,0	31,5	25,4	57,1	8,0	7,0	2,3	2,4
$p = 1024$	128,4	83,5	98,4	267,9	20,4	11,0	2,8	5,2

As can be seen from Table 1, the sequential GSA surpasses DIRECT and DIRECTl methods in all classes of problems in terms of the average number of iterations. At the same time, in the 5-Hard class, none of the methods solved all the problems: DIRECT failed to solve 16 problems, DIRECTl and GSA – 4 problems each.

Let us now evaluate the acceleration achieved using parallel GSA using an adaptive dimensionality reduction scheme based on the number p of cores used. Table 2 shows the speedup of the algorithm that combines multiple evolvents and an adaptive scheme for solving a series of problems on CPU, compared to the sequential launch of the GSA method. Two evolvents and, accordingly, two processes were used; each process used p threads, and calculations were performed on a single cluster node.

Table 3 shows the speedup obtained when solving a series of problems on one GPU using an adaptive scheme compared to a similar launch on CPU using 4 threads. Table 4 shows the speedup of the algorithm combining multiple evolvents and an adaptive scheme when solving a series of problems on two GPUs

Table 5. Speedup on six GPUs

	Iteration speedup	Time speedup
$p = 64$	30,8	1,9
$p = 256$	92,7	1,5
$p = 1024$	597,0	2,5

compared to an adaptive scheme on CPU using 4 threads. Two evolvents and, accordingly, two processes were used; each process used p threads on each GPU; all computations were performed on a single cluster node.

The last series of experiments has been carried out on 20 six-dimensional problems from the GKLS Simple class. Table 5 shows the speedup of the algorithm combining multiple evolvents and the adaptive scheme when solving the problems on 3 cluster nodes (using 2 GPUs per node, 6144 GPU threads in all) compared to the adaptive scheme on CPU using 4 threads.

6 Conclusion

In summary, we observe that the use of graphics processors to solve global optimization problems shows noteworthy promise, because high performance in modern supercomputers is achieved (mainly) through the use of accelerators.

In this paper, we consider a parallel algorithm for solving multidimensional multiextremal optimization problems and its implementation on GPU. In order to experimentally confirm the theoretical properties of the parallel algorithm under consideration, computational experiments were carried out on a series of several hundred test problems of different dimensions. The parallel algorithm demonstrates good speedup, both in the GPU and CPU versions.

Acknowledgments. This study was supported by the Russian Science Foundation, project No. 16-11-10150.

References

1. Barkalov, K., Gergel, V.: Multilevel scheme of dimensionality reduction for parallel global search algorithms. In: OPT-i 2014–1st International Conference on Engineering and Applied Sciences Optimization, Proceedings, pp. 2111–2124 (2014)
2. Barkalov, K., Lebedev, I.: Adaptive global optimization based on nested dimensionality reduction. Adv. Intel. Syst. Comput. **991**, 48–57 (2020)
3. Barkalov, K., Lebedev, I., Sovrasov, V.: Comparison of dimensionality reduction schemes for parallel global optimization algorithms. Commun. Comput. Inform. Sci. **965**, 50–62 (2019)
4. Evtushenko, Y., Malkova, V., Stanevichyus, A.A.: Parallel global optimization of functions of several variables. Comput. Math. Math. Phys. **49**(2), 246–260 (2009)

5. Ferreiro, A., Garcia, J., Lopez-Salas, J., Vazquez, C.: An efficient implementation of parallel simulated annealing algorithm in GPUs. J. Glob. Optim. **57**(3), 863–890 (2013)
6. Gablonsky, J.M., Kelley, C.T.: A locally-biased form of the DIRECT algorithm. J. Glob. Optim. **21**(1), 27–37 (2001)
7. Garcia-Martinez, J., Garzon, E., Ortigosa, P.: A GPU implementation of a hybrid evolutionary algorithm: GPuEGO. J. Supercomput. **70**(2), 684–695 (2014)
8. Gaviano, M., Kvasov, D., Lera, D., Sergeyev, Y.: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans. Math. Softw. **29**(4), 469–480 (2003)
9. Gergel, V., Barkalov, K., Sysoyev, A.: A novel supercomputer software system for solving time-consuming global optimization problems. Numer. Algebra Control Optim. **8**(1), 47–62 (2018)
10. Gergel, V., Grishagin, V., Gergel, A.: Adaptive nested optimization scheme for multidimensional global search. J. Glob. Optim. **66**(1), 35–51 (2016)
11. Gergel, V., Grishagin, V., Israfilov, R.: Local tuning in nested scheme of global optimization. Procedia Comput. Sci. **51**(1), 865–874 (2015)
12. Grishagin, V., Israfilov, R., Sergeyev, Y.: Convergence conditions and numerical comparison of global optimization methods based on dimensionality reduction schemes. Appl. Math. Comput. **318**, 270–280 (2018)
13. He, J., Verstak, A., Watson, L., Sosonkina, M.: Design and implementation of a massively parallel version of DIRECT. Comput. Optim. Appl. **40**(2), 217–245 (2008)
14. Hwu, W.: GPU Computing Gems, Emerald edn. Morgan Kaufmann, San Francisco (2011)
15. Jones, D.R.: The DIRECT global optimization algorithm. In: The Encyclopedia of Optimization, pp. 725–735. Springer, Heidelberg (2009)
16. Kindratenko, V. (ed.): Numerical Computations with GPUs. Springer, New York (2014)
17. Langdon, W.: Graphics processing units and genetic programming: an overview. Soft Comput. **15**(8), 1657–1669 (2011)
18. Modorskii, V., Gaynutdinova, D., Gergel, V., Barkalov, K.: Optimization in design of scientific products for purposes of cavitation problems. In: AIP Conference Proceedings **1738** (2016)
19. Paulavičius, R., Žilinskas, J., Grothey, A.: Parallel branch and bound for global optimization with combination of lipschitz bounds. Optim. Method. Softw. **26**(3), 487–498 (2011)
20. Sergeyev, Y., Kvasov, D.: Global search based on efficient diagonal partitions and a set of Lipschitz constants. SIAM J. Optim. **16**(3), 910–937 (2006)
21. Sergeyev, Y.D., Strongin, R.G., Lera, D.: Introduction to Global Optimization Exploiting Space-filling Curves. Springer Briefs in Optimization, Springer, New York (2013)
22. Strongin, R.: Algorithms for multiextremal mathematical programming problems employing the set of joint space-filling curves. J. Glob. Optim. **2**, 357–378 (1992)
23. Strongin, R.G., Sergeyev, Y.D.: Global Optimization with Non-convex Constraints Sequential and Parallel Algorithms. Kluwer Academic Publishers, Dordrecht (2000)