Parallel hybrid Price-genetic algorithm for global optimization

MARGHERITA BRESCO AND GIANCARLO RAICONI
Department of Mathematics and Informatics (DMI)
University of Salerno
via Ponte Don Melillo
I-84084 Fisciano (SA)
ITALY

Abstract

A hybrid algoritm to find the absolute extreme point of a multimodal scalar function of many variables is presented. The algorithm is well suited to be implemeted in parallel over a distributed computer environment. The algorithm is suitable when the objective function is expensive to compute, the computation can be affected by noise and/or partial derivatives cannot be calculated. The method used is a genetic modification of a previous algorithm based on the Price's method. The genetic part of the algorithm is very effective to escape from local attractors of the algorithm and assures convergence in probability to the global optimum. The parallel implementation of the algorithm is based on a domain decomposition approach, fully exploiting available information over global behaviour of the function, reducing to a small amount the computer charge due to need for syncronization. The algorithm has been tested for optimizing several multimodal test function.

Key-words:- Global Optimization, Controlled Random Search, Genetic Algorithms, Parallel Computation.

1 Introduction and problem formulation

The problem of finding the minimum of a given function $f : \mathbb{R}^n \to \mathbb{R}$ was studied extensively and many efficient algorithms are available to solve it. Most algorithms developed for this purpose are devoted to find a local minimum. In recent years more attention was reserved to the problem of finding the global minimum of a multimodal function. Several approaches were proposed to solve this kind of problems, [6],[11]. Most studied classes of global optimization (GO) problems are: combinatorial problems with a linear or nonlinear function, defined over a set of solutions that is finite but very large [10]; general unconstrained problems, that have a nonlinear objective function defined over reals unconstrained or subject to simple bound constraints.

In dealing with problems of the last mentioned class, a variety of partitioning strategies have been proposed to solve this problem exactly. These methods typically rely on prior knowledge of how rapidly the function can vary (e.g. the Lipshitz constant) [14] or the availability of an analytic formulation of the objective function (e.g. interval methods). Statistical [8] methods also use partitioning to decompose the search space, but they use prior information (or assumptions) about how the objective function can be modeled. A wide variety of approximate methods have been proposed for solving these problems, including simulated annealing, genetic algorithms, clustering methods, and continuation methods. When the computation of the objective function is very expensive, derivatives of the objective function are not available and/or the computation of objective function is affected by noise the most adapted methods ([2], [7], [3]) are the controlled random search (CRS) ones. Such family of methods first examine, using some random search scheme, all areas of definition region of the function in order to locate the most *promising* to contain the global minimum. When one sufficiently small of such promising areas is found, a local search is started to refine the global minimum estimate. In CRS methods the global search is performed maintaining a population of candidate points that tends to cluster around most *interesting* areas, giving also information about the global behavior of the function. The first of CRS is that of Price [12], [13], the algorithm in [3] improves the local search phase of the CRS method without any significant increase in computational cost. A further improvement of the method in [3] was obtained by authors [4] introducing a global minimization step based on a Genetic Algorithm

(GA) [5] when the local optimization based on a quadratic model of the function fails to find a definite positive matrix.

A difficulty always present in GO problems, and much more in the presence of complex objective function, is the severe computational effort required. In fact in order to escape from local extrema the value of the objective function over a rich population of test points must be computed at any step of the evolution of algorithm. If a complex objective function must be optimized and results must be obtained in real time, the only available possibility is that of resorting to greater computational power. At present days one of most cost effective ways to obtain this result is to use parallel computation over a distributed computer environment, more precisely on cluster of general purpose processing units. A bottleneck of such an approach is that the synchronizing information exchanged between processing units was transmitted across a relatively low speed network. The CRS approach is very efficient from this point of view because it can be parallelized following a domain decomposition scheme. We use both domain decomposition and a tree parallelization approach that enable us to efficiently share relevant information about global behavior of the function, using very little synchronization work an obtaining very good results. In order to optimize the management of computing resources we use the MOSIX [1] parallel computing environment. In fact, in MOSIX the migration of the workload is allowed from a node to another in a preemptive and transparent way, accomplishing in this way an efficient load balancing and preventing the trashing in case of memory-swapping. Several adaptive resource sharing algorithms are used for the dynamic balancing of the workload on the CPUs using when necessary the PPM (Preemptive Process Migration) module for load reallocation among the CPUs. The migration take place automatically on the basis of the conditions reported by the optimization of resource sharing and load balancing algorithms and, in particular, the individual load of the single nodes and the network speed.

2 The CRS-Genetic algorithm

In this section we give only the formal description and a brief description of relevant characteristics of the Genetic Price Algorithm (GPA), a full analysis of the properties of the algorithm can be found in [4] . The hybrid algorithm is described in the following pseudocode:

Algorithm GPA:

```
Step 0. (INITIALIZATION)
  Set $k = 0$; determine the initial set
      $S^k = \{x_1^k, x_2^k, ..., x_m^k\}$
    $x_i^k, i = 1, 2, ..., m$ chosen at random on $D$; evaluate $f$
    at each point $x_i^k, i = 1, 2, ..., m$.
Step 1. Determine $x_{\min}^k, x_{\max}^k, f_{\min}^k, f_{\max}^k$ such that:
      $f(x_{\min}^k) = f_{\min}^k = \min_{x \in S^k} \{f(x)\}$,
      $f(x_{\max}^k) = f_{\max}^k = \max_{x \in S^k} \{f(x)\}$.
    if stopping criterion is satisfied then STOP.
Step 2. (WEIGHTED CENTROID)
  Choose at random $2n + 1$ points $x_{i_0}^k$ ,$x_{i_1}^k$, ..., $x_{i_n}^k$ on
  $S^k$;determine the weighted centroid
      $c_w^k = \sum_{j=1}^{n} w_j^k x_{i_j}^k$,
    with weights $w_j^k$ are suitably chosen.
Step 3. (WEIGHTED REFLECTION)
  Determine new trial point $\tilde{x}^k$ by the weighted reflection:
      $\tilde{x}^k = \begin{cases} c_w^k - \alpha^k(x_{i_0}^k - c_w^k) & \text{if} \quad f_w^k \leq f(x_{i_0}^k) \\ c_w^k + \alpha^k(x_{i_0}^k - c_w^k) & \text{if} \quad f_w^k > f(x_{i_0}^k) \end{cases}$
    with $f_w^k = \sum_{i=1}^{n} w_j^k f\left(x_{i_j}^k\right)$, and
      $\alpha^k = \begin{cases} 1 - \frac{f\left(x_{i0}^k\right) - f_w^k}{f_{\max}^k - f_{\min}^k + \psi^k} & \text{if} \quad f_w^k \leq f\left(x_{i0}^k\right) \\ 1 - \frac{f_w^k - f\left(x_{i0}^k\right)}{f_{\max}^k - f_{\min}^k + \psi^k} & \text{if} \quad f_w^k > f\left(x_{i0}^k\right) \end{cases}$
  If $\tilde{x}^k \notin D$ then go to Step 2; otherwise compute $f(\tilde{x}^k)$.
Step 4. (RANDOM SAMPLING)
  If $f(\tilde{x}^k) \geq f_{\max}^k$ then choose a random point $\hat{x}$ in $D$ :
      If $f(\hat{x}) < f_{\max}^k$
    then set $S^{k+1} = S^k \cup \{\hat{x}\} - \{x_{\max}^k\}$ ;
          $k = k + 1$, and go to Step 1.
    Else set $S^{k+1} = S^k, k = k + 1$, and go to Step2.
Step 5. (SET UPDATING)
  If $f_{\min}^k < f(\tilde{x}^k) < f_{\max}^k$ then set
      $S^{k+1} = S^k \cup \{\tilde{x}_k\} - \{x_{\max}^k\}, k = k + 1$
    and go to Step 1.
Step 6. (QUADRATIC MODEL OF $f$)
  If $f(\tilde{x}^k) < f_{\max}^k$ then set
      $\tilde{S} = S^k \cup \{\tilde{x}_k\} - \{x_{\max}^k\}$
    and select the subset $S_{\min}$ constituted by the $2n + 1$
    points of $\tilde{S}$ corresponding to the smallest values of
    $f$. Determine the diagonal $n \times n$ matrix $Q$ =diag($q_i$)
    the vector $c$ and the scalar $d$ of the quadratic form
    interpolating $f$ on $\tilde{S}$.
      $f(x) = \frac{1}{2} x^T Q x + c^T x + d, \forall x \in \tilde{S}$.
```

Step 7. (CROSSOVER)
If $\exists\, q_i \leq 0$, then consider the set

$$\check{S} = S_{\min} - \{\tilde{x}_{\max}\},$$

with $\tilde{x}_{\max}$ such that $f(\tilde{x}_{\max}) = \max_{x\in\check{S}} \{f(x)\}$. Using the $2n$ points of $\check{S}$ form n random chosen couples of points, for any couple compute two new vectors by single point crossover with random cutting point. Then define the set $\hat{S} = \{x^k_{T,1}, x^k_{T,2}, ..., x^k_{T,2n}\}$ of all new vectors then evaluate f on all points of $\hat{S}$. Set S^{k+1} as the set of m points of the set $S^k \cup \hat{S}$ corresponding to the smallest values of f, set $k = k + 1$ and go to Step 1.

Step 8. If $q_i > 0, i = 1, 2, ..., n.$ let $x^k_q = -Q^{-1}c$, if $x^k_q \notin D$ or $f(x^k_q) \geq f(\tilde{x}^k)$ then take$S^{k+1} = \tilde{S}$ else take $S^{k+1} = \tilde{S} \cup \{\tilde{x}^k_q\} - \{x^k_{\max}\}$; set $k = k + 1$ and go to Step 1.

The whole algorithm is equal to a version of that proposed in [3] except for the Step 7 of the algorithm that represents the genetic hybridization of the CRS. In that step both the selection and crossover operators are embedded, note that Step 7 and 8 are alternative. The improvements of the version in [3] with respect to the original Price's algorithm are the introduction of weights in the computation of centroid and of the reflection, and also the choice of a possibly better point based on the minimization of a quadratic model of the objective function (Step 6 and Step 8). Step 4 introduces a random choice in the population of candidate points, this is essential to assure that the algorithm generates a succession of function values converging in probability to the global minimum. For a discussion about such point see [15]. The algorithm can be viewed also as a GA characterized by the following particularities:

a) The selection operator chooses at any generation as candidate to reproduce only the $2n$ *best fitting* individuals of that generation, in classical GA the selection operator acts applying a probability to reproduce (proportional to fitness value) to all individuals of the generation and selecting individuals according to this probability. In our approach we select for reproduction exactly $2n$ individuals, those characterized by lower objective functions. This approach both helps to prevent genetic derive and preserve the same data structure of the modified Price algorithm, thus assuring that no further function evaluation are required.

b) The crossover operator is explicitly introduced in our algorithm. Because individuals in our application are $n-$dimensional real valued vectors, the crossover is performed by splitting at a certain, randomly chosen, index the two vector and recombining first subvector of one individual with the second of the other an viceversa. All recombined vector are scanned and if one is found with a better fitting whit respect to that present in S^k when this occurs the last scanned of new vectors is substituted in place of the worst fitting individual in S^k.

c) The mutation operator is not explicitly introduced, in fact the Step 4 of the algorithm introduce at any iteration a random chosen point in the population, this assures global convergence in probability to the algorithm and prevent the genetic derive risk.

From the computational point of view note that the new algorithm requires, with respect to the improved Price algorithm, n more function evaluations, only when the global improvement phase fails because it is not possible to find a definite positive matrix Q. The selection of $\tilde{S}$ and $\hat{S}$ has computational cost $O(0)$ if elements of S^k was maintained ordered, the most complex operation on the algorithm is the ordering of the set $S^k \cup \hat{S}$ which has a computational cost $O(m + n)$.

3 Parallel implementation

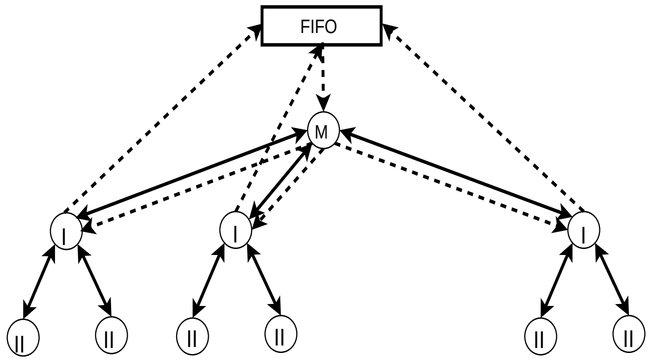
The parallel implementation of our algorithm follows a domain decomposition scheme, i.e. the whole domain is subdivided in a number of subdomains and embedded in a tree shaped control structure. This idea is based on the use of a three levels processes-tree. It realize the fusion between the domain splitting, and the parallelization of the genetic search enclosed in the algorithm. At level 0 (root) the master process works, performing the domain sharing between his sons (slave processes at level 1). These sons perform (each for his own count) the algorithm until the invocation of the crossover operator. At this point the process should do $2n$ function evaluations. We suppose that every function evaluation needs approximately time t. Each slave will spend approximately time $2n \cdot t$ to perform this step. If each slave (level 1) should have $2n$ sons (slave processes at level 2) performing the $2n$ computation of functions. The need for synchrony should involve, in this case, a wait of the 1^{st} level processes, of $t + \tau$, where τ is the time needed for the communication. If τ is much less than $(2n - 1) \cdot t$, the running time of each process at level 1 is substantial reduced. The drawback of this approach is that, if each process is mapped on a processor and the crossover operator is invoked rarely, the host processors for the 2^{nd} level processes might be often idle, with a waste of unused resources. We can avoid this drawback by mapping more 2^{nd} level slave processes on a single processor.

3.1 The problem of finding a better point out the assigned box

In domain decomposition approach to parallel global optimization the domain D is decomposed in a certain number p of number of subdomains $D_i, i = 1, 2, , ..., p$ and p independent processes are started, the aim of the i-th process is to find the global minimum of the function f on D_i let be $\hat{f}_i$ such minimum. When all processes are terminated the optimum solution is chosen as the best between these p solutions i.e. the smallest of $\hat{f}_i$. Following a CRS approach in any subdomain a population of candidate points is maintained, in the case of a well shaped function in subdomains characterized by high values of function, the candidate points tends to cluster in few iterations near the subdomain boundary, giving indication about the fastest decreasing directions. Our algorithm incorporate a *local optimization* idea to increase the convergence, more precisely the construction of a *quadratic model* of the objective function (Step 6). It can occur that Step 6 really finds a better candidate point but such a point is away from the subdomain assigned to such process. Following the *brute force* domain decomposition rules such point is discarded and the corresponding valuable information is lost. A possible remedy should consist in allowing that the processor continues to use that point and, therefore, it widens its own box. This solution should lead to a boxes overlap and to a data duplication that might result unwanted. We will give a better solution to this kind of problems. In this solution a point lying in another box doesn't throw away a priori without consider the eventuality that his value of objective function can be useful.

3.2 Management of *out of the box* points

When a process finds a point out of his box, this point will have to be communicated to the process that hold the box in which lies that point. This communication might take place in two ways: the process communicates the point to the process interested or the process communicates the point to the master process, which communicates it to the process interested. The former approach, apparently easier, needs that each process knows how the boxes are shared between all slaves of first level, so, the master should have to communicate all information concerning the sharing of boxes to all processes, involving a large communication overhead. The latter approach needs two data transmissions instead of one, but it is , because only the master has to know which process hold the box in question, therefore, it isn't necessary to communicate the box assignments to the 1^{st} level processes.



Structure of interprocess comunication

The figure 1 show the structure of tree-processes in object. The node labelled with M is the master, the nodes labelled with I are the slaves of 1^{st} level and those labelled with II are the slaves of 2^{nd} level. Solid lines depict bidirectional communication between parent and children. On these channels (pipes) the parent send the parameters to his sons (i.e the master process communicates to the 1^{st} level processes the respective subdomain and the 1^{st} level processes assign to the 1^{st} level ones the values where function must be evaluated). On the same pipes children processes return the computation results to their parent. Another communication channel, using a FIFO, has been created to communicate to the parent the points that lie in another box, such channel is represented by dotted lines. Dotted line from 1^{st} level processes to FIFO indicate the flow of out of box points to master (the master doesn't need to know who is the process sender). Dotted lines from master to sons show the communication of new points to interested child. Such information are in fact sent on the pipe because after the initial parameter communication links between master to 1^{st} level processes become idle, then each 1^{st} level process knows that all further arriving information is surely a new point coordinates.

4 Numerical Results

In this section we report few numerical results obtained applying the parallel method to some multimodal functions used to test the efficiency of global optimization algorithms. About the performance of the sequential Genetic-Price algorithm in [4] results obtained in a wide selection of test problems are reported, demonstrating as the new algorithm overcomes both classical CRS and GA approach. Over all examples considered the new algorithm gives improvements in the value of function at minimum *and* saving of computer time in the 30% to 60% range over the best performing method.

For such reason in the following experiments the parallel implementation is compared only with the sequential version of the same algorithm, in fact the computer code used is highly scalable and the sequential version is generated simply putting the input parameter that represents the number of processor used to one. Moreover the dynamic process allocation of MOSIX allows to allocate a number of processes greater than the number of actual processors. In our tests the number of concurrent processes was limited by the number of physical processors available and the migration utility of MOSIX was used to assure that all concurrent processes are scheduled to run on different processors. Note that, in spite of the scalability of the code, the parallel algorithm is really different from the sequential one. It isn't a parallelization of parallelizable part of the sequential code. One can say that there are several parallel processes cooperating to the solution of the problem, any of that can expect aid from the other, this characteristic gives us the hope for achieving a superlinear speedup.

All presented results are obtained on a computing cluster consisting of 10 computing units any of that is equipped by two AMD Athlon MP2400 processors, the operating system installed was Red Hat Linux version 9.0 with SMP kernel 2.4.21 with the kernel extension OPENMOSIX ver.2.0. In all the experiments reported we represent results on tables, the numerical results are averaged over ten independent runs of the algorithm with different random seed initialization. The total number of points maintained by the CRS algorithm (i.e. number of individuals of the population in the GA) was 3000. The symbols reported on numerical tables have the following meaning:

NP= processors number,
FV= mean function value,
FE= mean number of function evaluation,
T= mean CPU time.

Experiment 1: the Shubert function [9]

$$f(x_1, x_2) = -\sum_{j=1}^{5} j\left[\sin\left((j+1)\,x_1 + j\right) + \sin\left((j+1)\,x_2 + j\right)\right]$$

$$f_{\min} = -24.062499,$$

400 local minima, 9 global minima

$$X_{\min} = \left\{ \begin{bmatrix} a \\ a \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} a \\ c \end{bmatrix}, \begin{bmatrix} b \\ a \end{bmatrix}, \begin{bmatrix} b \\ b \end{bmatrix}, \begin{bmatrix} b \\ c \end{bmatrix}, \begin{bmatrix} c \\ a \end{bmatrix}, \begin{bmatrix} c \\ b \end{bmatrix}, \begin{bmatrix} c \\ c \end{bmatrix} \right\}$$

$$a = -6.774576, b = -0.491391, c = 5.791794$$

$n = 2$, $D = [-100, 100]^2$ results are reported on table 1

N.P	FV	FE	T
1	-24.060575	831418	2449
4	-24.062499	4017163	156
9	-24.062499	1278108	23.7
16	-24.062499	1039006	16.2

(table 1)

Experiment 2: The exponential function [3]

$$f(x_1, ..., x_n) = e^{-\frac{\sum_{i=1}^{n} x_i^2}{2}}$$

$$f_{\min} = \frac{1}{e^{\frac{n}{2}}},$$

2^n local (and global) minima

$$x_{\min} = [\pm 1, \pm 1, ..., \pm 1]^T$$

$n = 4, D = [-1, 1]^n$ results are reported on table 2 (row marked 16 report data obtained with 16 processors splitting the domain across two dimensions, and row 16* domain splitted across all 4 dimension).

N.P	FV	FE	T
1	0.14834654018	212433	2431
4	0.1353352832	435060	44.1
9	0.1353352832	366777	36.6
16	0.1353352832	288622	27.1
16*	0.1353352832	186801	18.6

(table 2)

Experiment 3: The Hansen function. [9]

$$f(x_1, x_2) = \sum_{i=1}^{5} i\cos\left((i-1)\,x_1 + i\right) \sum_{i=1}^{5} j\cos\left((j-1)\,x_2 + j\right)$$

$$f_{\min} = -176.541793,$$

176 local minima 9 global minima

$$X_{\min} = \left\{ \begin{bmatrix} a \\ d \end{bmatrix}, \begin{bmatrix} a \\ e \end{bmatrix}, \begin{bmatrix} a \\ f \end{bmatrix}, \begin{bmatrix} b \\ d \end{bmatrix}, \begin{bmatrix} b \\ e \end{bmatrix}, \begin{bmatrix} b \\ f \end{bmatrix}, \begin{bmatrix} c \\ d \end{bmatrix}, \begin{bmatrix} c \\ e \end{bmatrix}, \begin{bmatrix} c \\ f \end{bmatrix} \right\}$$

$$a = -7.589893, b = -1.306708, c = 4.976478$$

$$d = -7.708314, e = -1.425128, f = 4.858057$$

$n = 2$, and is considered on $D = [-100, 100]^2$, results for such function are reported in table table 3

N.P	FV	FE	T
1	-176.53385885374	815906	2435.4
4	-176.5417931367	3540380	55.9
9	-176.5417931367	1466601	22.2
16	-176.5417931367	880913	13.9

(table 3)

Experiment 4: The extended Rosenbrock function [16] (unimodal)

$$f(x) = \sum_{i=1}^{n-1} \left\{ (x_i - 1)^2 + 100(x_i^2 - x_{i+1})^2 \right\}$$

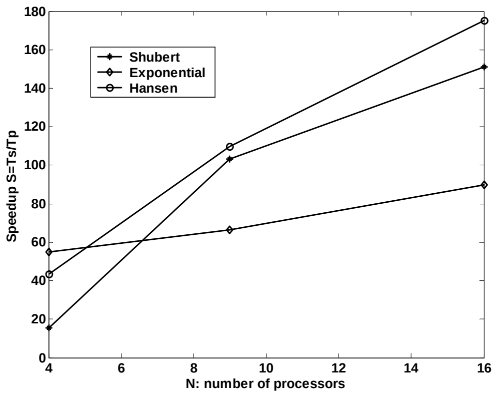
$$f_{\min} = 0,$$

$$x_{\min} = [1, 1, ..., 1]^T,$$

table 4 report results with $n = 8, D = [-1000, 1000]^n$

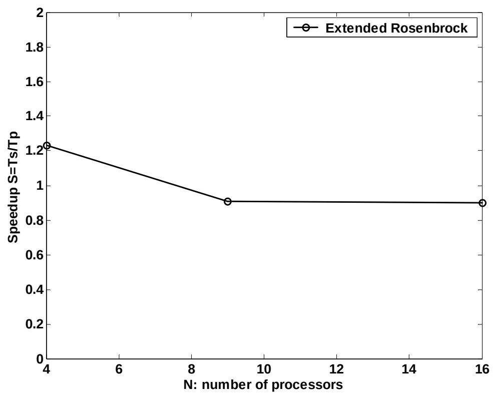
N.P	FV	FE	T
1	0	415690	23.15
4	0	222069	18.79
9	0	264830	25.43
16	0	241743	25.67

(table 4)



Speedups for experiments 1,2 and 3.

The new parallel implementation seems to give very high acceleration to solve complex optimization of multimodal functions. The speedup values: $S(N) = \frac{Ts}{Tp(N)}$ where (Ts =CPU time of sequential alg and $Tp(N)$ =CPU time of parallel alg. whit N processors) evaluated for experiments 1,2 and 3, and reported in fig 2 are quite impressive. The behavior, similar in all cases, shows that the saving in computational effort is dramatic just passing for the sequential to the parallel approach (values with $N = 4$), then increase in a substantially linear fashion with the increasing the processor number. This means that we can look forward for good performances even on small computing clusters.



Speedups for experiment 4.

The gain in performances of parallel algorithm over the sequential one is more evident as more complex is the problem, in the case of the *simple* Rosenbrock problem the gain using the parallel algorithm is negligible, moreover when the number of processors increases one can experience even a performance degradation. From the analysis of the speedup values for the experiment 4 it is clear that there is no advantage of the parallel algorithm. But in this case there are several local optimization methods more effective to solve the problem.

5. Conclusions

We presented a parallel implementation of a novel algorithm for global optimization. The algorithm in its sequential formulation is very attractive because retains good features of both the CRS and the GA approaches from which it is derived. In the case of very complex objective functions and very stringent due time requirements, as in real time data analysis problems, the computer time needed to find the global minimum can be a serious problem, computing time can be reduced using special purpose supercomputers but such an approach is very expensive. A cost effective approach to such problem is the parallel solution by means of a cluster of general purpose computers in a distributed computing environment. The algorithm proposed is well suited to be adapted to such type of computing environment, in fact the combination of an intelligent structure of interprocess data communication and of a *state of the art* load scheduling and balancing tool as MOSIX give extremely good experimental results.

References

[1] A. Barak, O. La'addan, A. Shiloh, *Scalable Cluster Computing with MOSIX for LINUX,* http://www.mosix.cs.huji.ac.il

[2] F. Barone, L. Milano and G. Russo, in *Active Close Binaries*, ed. C. Ibanoglu, (Kluwer, Dordrecht 1990) 161-188.

[3] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo and S. Lucidi, *Journal of Global Optimization,* **10,** (1997) 165-184

[4] M. Bresco, G. Raiconi, F. Barone, R. De Rosa, L. Milano, *Genetic approach helps to speed classical Price's algorithm for global optimization,* in Soft Computing Journal, (to appear), 2004..

[5] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison Welsey, New York 1989).

[6] R. Horst and P.M. Pardalos (eds.), *Handbook of GlobalOptimization* , (Kluwer, Dordrecht 1995)

[7] L. Milano, F. Barone, M. Milano, *Phys. Review D*, **55**, (1997), 4537-4554

[8] J.Mockus, *Bayesian Approach to Global Optimization*, (Kluwer, Dordrecht 1989)

[9] http://www.imm.dtu.dk/~km/GlobOpt/testex /testproblems.htm

[10] P. M Pardalos, A. Migdalas, R. E. Burkard (eds.), *Combinatorial and Global Optimization,* (World Scientific Pub. Co. Inc. 2002)

[11] J. D. Pinter, *Global Optimization in Action*, (Kluwer, Dordrecht 1995)

[12] W.L. Price, *Computer Jour.*, **20,** (1979) 367-370.

[13] W.L. Price, *Jouranal of Optimization Theory and Applications,* **40,** (1983) 333-348.

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

[15] F. Shoen, *Journal of Global Optimization*, **1,** (1991) 207-228.

[16] M. A. Wolfe, *Numerical Methods for Unconstrained Optimization*, (Van Nostrand Rehinold Company, N.Y. 1978).