Applied Mathematics, 2012, 3, 1380-1387
http://dx.doi.org/10.4236/am.2012.330194 Published Online October 2012 (http://www.SciRP.org/journal/am)

A Parallel Algorithm for Global Optimization Problems in a Distribuited Computing Environment

Marco Gaviano, Daniela Lera, Elisabetta Mereu
Department of Mathematics and Informatics, University of Cagliari, Cagliari, Italy
Email: gaviano@unica.it, lera@unica.it, elisabetta.mereu@hotmail.it

Received July 4, 2012; revised August 4, 2012; accepted August 11, 2012

ABSTRACT

The problem of finding a global minimum of a real function on a set $S \subseteq R^n$ occurs in many real world problems. Since its computational complexity is exponential, its solution can be a very expensive computational task. In this paper, we introduce a parallel algorithm that exploits the latest computers in the market equipped with more than one processor, and used in clusters of computers. The algorithm belongs to the improvement of local minima algorithm family, and carries on local minimum searches iteratively but trying not to find an already found local optimizer. Numerical experiments have been carried out on two computers equipped with four and six processors; fourteen configurations of the computing resources have been investigated. To evaluate the algorithm performances the *speedup* and the *efficiency* are reported for each configuration.

Keywords: Random Search; Global Optimization; Parallel Computing

1. Introduction

In this paper we consider the following global optimization problem.

Problem 1

$$\text{find } x^* \subseteq S, \text{ such that } f\left(x^*\right) \le f\left(x\right), \forall x \in S,$$

where f: $S \to R$ is a function defined on a set $S \subseteq R^n$.

In order to solve Problem 1 a very large variety of algorithms has been proposed; several books that describe the research trends from different points of view, have appeared in the literature [1-8]. Numerical techniques for finding solutions to such problems by using parallel schemes have been discussed in the literature (see, e.g. [9-13]. To generalize the investigation of the properties of the algorithms, these are classified in families whose components share common strategies or techniques. In [14] five basic families are defined: partition and search, approximation and search, global decrease, improvement of local minima, enumeration of local minima. Nemirovsky and Yudin [15], and Vavasis [16] have proved, under suitable assumptions, that the computational complexity of the global optimization problem is exponential; hence, the number of function evaluations required to solve problem 1 grows dramatically as the number of variables of the problem increases. This feature makes the search of a global minimum of a given function a very expensive computational task. On the other hand the latest computers in the market, equipped with more than one processor, and clusters of computers can be exploited. In [17] a sequential algorithm, called *Glob* was presented; this belongs to the improvement local minima family and carries on local search procedures. Specifically, a local minimum finder algorithm is run iteratively and in order to avoid to find the same local minimizer, a local search execution rule was introduced; this was chosen such that the average number of function evaluations needed to move from a local minimum to a new one, is minimal. The parameters used to define the execution rule at a given iteration were computed taking into account the previous history of the minimization process.

In this paper we present a parallel algorithm that distributes the computations carried out by *Glob* across two or more processors. To reduce to a low level the data passing operations between processors, the sequential algorithm is run on each processor, but the parameters of the execution rule are updated either after a fixed number of iterations are completed or straight as soon as new local minimizer is found. The new algorithm has been tested for solving several test functions commonly used in the literature. The numerical experiments have been carried out on two computers equipped with four and six processors; fourteen configurations of the computing resources have been investigated. To evaluate the algorithm performances the *speedup* and the *efficiency* are reported for each configuration.

2. Preliminaries

In this section we recall some results established in [17]. For Problem 1 we consider the following assumption.

Assumption 1

1) $f(\cdot)$ has m local minimum points $l_i, i=1,\cdots,m$ and $f(l_i) > f(l_{i+1})$;

2) meas $(S) = 1$, with meas (S) denoting the measure of S.

Consider the following algorithm scheme.

Algorithm 1 (Algorithm Glob)

Choose x_0 uniformly on S;

$i \leftarrow 1$; $j \leftarrow 1$; $(x_1, fx_1) \leftarrow local\_search(x_0)$;

$l_i = x_1$; $fl_i = fx_1$;

repeat

$j \leftarrow j+1$;

choose x_0 uniformly on S;

if

$f(x_0) \le fl_i$ or $(f(x_0) > fl_i$ and $rand(1) < d_i)$

$(x_1, fx_1) \leftarrow local\_search(x_0)$;

if $fx_1 < fl_i$

$i \leftarrow i+1$;

$l_i \leftarrow x_1$; $fl_i \leftarrow fx_1$;

end if

end if

until a stop rule is met;

end

The function *rand* (1) denotes a generator of random numbers in the interval $[0,1]$. Further, we denote by *local_search* (x_o) any procedure that starting from a point x_0 returns both a local minimum l_i of problem 1 and its function value.

In algorithm *Glob* a sequence of local searches is carried out. Once a local search has been completed and a new local minimum l_j is found, a point x_0 at random uniformly on S is chosen. Whenever $f(x_0)$ is less than $f(l_j)$ a new search is performed from x_0; otherwise a local search is performed with probability d_i.

We assume that Problem 1 satisfies all the conditions required to make the procedure *local_search* (x_0) convergent. We have the following proposition.

Proposition 1. *Let assumption 1 hold and consider a run of algorithm Glob. Then the probability that* l_i *is a global minimum of problem* 1 *tends to one as* $j \to \infty$.

First, we settle the following notation.

Definition 1

- $A_{0,j} \equiv \{x \in S \mid$ starting from x, *local_search*$(\cdot)$ returns local minimum $l_j\}$;
- $A_{i,j} \equiv \{x \in S \mid f(x) \le f(l_i)$; starting from x, *local_search*$(\cdot)$ returns local minimum $l_j\}$;
- $p_{0,j} = meas(A_{0,j})$;
- $p_{i,j} = meas(A_{i,j})$.

We have

$$\sum_{i=1}^{m} p_{0,i} = meas(S) = 1.$$

We consider the following definitions for algorithm *Glob*.

Definition 2

- $t_i \equiv$ the probability that having found the local minimum l_i, in a subsequent iteration no new local minimum is detected;
- $Prob_{i,j}(d_i) \equiv$ the probability that the algorithm, having found the local minimum l_i, can find the local minimum l_j in a subsequent iteration.

We calculate the average number of function evaluations so that algorithm *Glob* having found a local minimum, finds any new one. We assume that algorithm *Glob* can run an infinite number of iterations. Further it is assumed that the values $p_{0,j}$ and $p_{i,j}$, $i = 1, \cdots, m-1$ and $j = 1, \cdots, m$, are known and that the number of function evaluations required by *local_search* is $k =$ constant.

The following holds.

Theorem 1. *The average number of function evaluations so that algorithm Glob, having found a local minimum* l_i, *finds any new one is given by*

$$evals_1(d_i) = f_i \frac{1}{Prob_{i,*}}, \quad i = 1,\cdots,m-1,$$

with

$$Prob_{i,*} = \sum_{j=i+1}^{m} p_{i,j} + d_i\left(\sum_{j=i+1}^{m} p_{0,j} - \sum_{j=i+1}^{m} p_{i,j}\right),$$

$$f_i = k\sum_{j=i+1}^{m} p_{i,j} + kd_i\left(1 - \sum_{j=i+1}^{m} p_{i,j}\right) + (1-d_i)\left(1 - \sum_{j=i+1}^{m} p_{i,j}\right)$$

Problem 2. *Let us consider problem* 1 *and let the values* k, $p_{0,j}$ *and* $p_{i,j}$ *be given. Find value* d_i^* *such that*

$$evals_1(d_i^*) = \min_{d_i} evals_1(d_i).$$

We calculate which value of d_i gives the minimum of such a function. We have as $i = 1,\cdots,m-1$,

$$evals_1(d_i) = \frac{(k-1)\sum_{j=i+1}^{m} p_{i,j} + d_i\left(1 - \sum_{j=i+1}^{m} p_{i,j}\right)(k-1) + 1}{\sum_{j=i+1}^{m} p_{i,j} + d_i\left(\sum_{j=i+1}^{m} p_{0,j} - \sum_{j=i+1}^{m} p_{i,j}\right)}.$$

The derivative sign of $evals_1(d_i)$ is greater than or equal to zero for

$$k \ge \frac{\left(\sum_{j=i+1}^{m} p_{0,j}\right)\left(1 - \sum_{j=i+1}^{m} p_{i,j}\right)}{\left(\sum_{j=i+1}^{m} p_{i,j}\right)\left(1 - \sum_{j=i+1}^{m} p_{0,j}\right)}. \quad (1)$$

The condition (1) links the probability $p_{i,j}$ with the number k of function evaluations performed at each local search in order to choose the most convenient value of d_i: if the condition is met, we must take $d_i = 0$ otherwise $d_i = 1$.

In real problems usually we don't know the values $p_{0,j}$ and $p_{i,j}$; hence the choice of probabilities $d_1, d_2, \cdots, d_m$ in the optimization of the function in problem 2 cannot be calculated exactly. By making the following approximation of the values

$$p_2 = \sum_{j=i+1}^{m} p_{0,j},\ p_3 = \sum_{j=i+1}^{m} p_{i,j},$$

which appear in the definition of $evals_1$ in problem 2, we can device a rule for choosing the values d_i, $I = 1, \cdots, m$ in algorithm *Glob*. Specifically, from (1) we get

$$d_i = \begin{cases} 0 & \text{if } k > \left(p_2 \cdot (1-p_3)\right) \big/ \left(p_3 \cdot (1-p_2)\right), \\ 1 & \text{otherwise,} \end{cases}$$

where p_2, p_3 and k are approximated as follows

$$\begin{aligned} p_2 &= 1 \big/ (\textit{number of searches carried out}), \\ p_3 &= 1 \big/ (\textit{no. of iterations already carried out}), \\ k &= \textit{no. of function evaluations in the local searches}. \end{aligned} \tag{2}$$

Hence the line

if $f(x_0) \le fl_i$ or $\left(f(x_0) > fl_i \text{ and } rand(1) < d_i\right)$

of algorithm *Glob* is replaced by the line

$$\text{if } f(x_0) \le fl_i \text{ or } yes\_box(\ ) = 1 \tag{3}$$

where *yes_box* ( ) is a procedure that returns zero or 1 and is defined by

Procedure 1 (yes-box())

$p_2 = 1/(\textit{no. of searches already carried out})$
$p_3 = 1/(\textit{no. of iterations already carried out})$
$k = \textit{no. of function evals in the local searches}$;
if $p_2 = 1$
  $p_2 = 2 \times p_3$;
end if
if $p_2 < 1$
  ratio = $\left(p_2 \times (1-p_3)\right) \big/ \left(p_3 \times (1-p_2)\right)$;
else
  ratio = inf;
end if
if k > ratio
  yes = 0;
else
  yes = 1;
end if
end.

In the sequel we shall denote by $Glob_{\text{new}}$ algorithm *Glob* completed with procedure 1.

3. The Parallel Algorithm

The message passing model will be used in the design of the algorithm we are going to introduce. This model is suitable for running computations on MIMD computers for which according to the classification of parallel systems due to Michael J. Flynn each processor operates under the control of an instruction stream issued by its control unit. The main point in this model is the communication where messages are sent from a sender to one or more recipients. To each send operation there must correspond a receive operation. Further the sender either will not continue until the receiver has received the message or will continue without waiting for the receiver to be ready.

In order to design our parallel algorithm in an environment of N processors we separate functions in two parts: server and client. The server task will executed by just one processor, while the remaining ones will execute the same code. The server will accomplish the following task.

- reads all the initial data and sends them to each client;
- receives the intermediate data from a sender client;
- combines them with all the data already received;
- sends back the updated data to the client sender;
- gathers the final data from each client.

Each client accomplishes the following tasks

- receives initial data from server;
- runs algorithm $Glob_{\text{new}}$;
- sends intermediate data to server;
- receives updated values from server;
- stops running $Glob_{\text{new}}$ whenever its stop rule is met in any client execution;
- sends final data to server.

The communication that takes place between the server and each client concerns mainly the parameters in (2), that is, p_2, p_3 and k. Each client, as soon as he either finds a new local minimizer or a fixed number of iterations have been executed, sends a message to the server containing data gathered after the last sent message; that is.

- last minimum found;
- the number of function evaluations since last message sending;
- the number of iterations since last message sending;
- the number of local searches carried out since last message sending;
- status variable of value 0 or 1 denoting that the stop rule has been met.

The server combines each set of intermediate data received with the ones stored in its memory and sends to the client the new data. If the server receives as status variable 1 in the subsequent messages sent to clients the status variable will keep the same value, meaning that the client has to stop running $Glob_{\text{new}}$ and has to send the final data to the server. The initial data and intermediate

data are embodied in the following data structures.

- Data_start = struct ("x", [ ], "fx", [ ], "sum_ev", [ ], "sum_tr", [ ], "sum_ls", [ ], "fun", [ ], "call_interval", [ ]).
- Data_mid = struct ("stop_flag", [ ], "client",[ ], "x", [ ], "fx", [ ], "sum_ev", [ ], "sum_tr" [ ], "sum_ls", [ ])

where the strings within single quotes denote the names of the members of the structure and [ ] its values.

In Data_start the members "x" and "fx" refer to the algorithm starting point as defined by the user; "sum_ ev", "sum_tr", "sum_ls" initialize the number of function evaluations, iterations and local searches. "fun" and "call_interval" denote the problem to solve and the number of iterations to be completed before an intermediate data passing has to take place.

In Data_mid "stop_ flag" and "client" refer to the status variable and to the client sender while the remaining members denote values as in Data_start but if the sender is a client the values refer to values gathered since the last message sending, while if the sender is the server the values concern the overall minimization process.

In the Appendix we report in pseudocode the basic instructions of the procedures to be executed in the server and client processors respectively.

4. Numerical Results

In this section we report the numerical results we got in the implementation of the algorithm outlined in the previous sections. First we describe the paradigm of our experiments.

Four test problems have been solved; to test the performance of our algorithm each problem has been chosen with specific features.

Test problem 1

$$\min f(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i-1)^2\left(1+10\sin^2(\pi y_{i+1})\right)\right] + (y_n-1)^2\right\}$$

with

$$n = 100,\ y_i = 1+\frac{1}{4}(x_i-1),$$
$$S \equiv \left\{x \in R^n \mid -10 \le x_i \le 10,\ i = 1,\cdots,n\right\};$$

Test problem 2

$$\min\ f(x) = (x_1-1)^2 + \sum_{i=2}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$$

with

$$n = 25,\ S \equiv \left\{x \in R^n \mid -10 \le x_i \le 10,\ i = 1,\cdots,n\right\};$$

Test problem 3

$$\min\ f(x) = 10n + \sum_{i=1}^{n}\left(x_j^2 - 10\cos(2\pi x_j)\right)$$

with

$$n = 8,\ S \equiv \left\{x \in R^n \mid -2.56 \le x_i \le 2.56,\ i = 1,\cdots,n\right\};$$

Test problem 4

$$\min\ f(x) = \begin{cases} \left(\dfrac{2}{\rho_i^2}\dfrac{\langle x-m_i, x_t-m_i\rangle}{\|x-m_i\|} - \dfrac{2}{\rho_i^3}\left(\|x_t-m_i\|^2 + 4 - f_i\right)\right) \\ \times\|x-m_i\|^3 \\ +\left(1-\dfrac{4}{\rho_i}\dfrac{\langle x-m_i, x_t-m_i\rangle}{\|x-m_i\|} + \dfrac{3}{\rho_i^2}\left(\|x_t-m_i\|^2 + 4 - f_i\right)\right) \\ \times\|x-m_i\|^2 + f_i, & \text{for } x \in B_i, \\ \|x-x_t\|^2 + 4, & \text{for } x \notin B_i, \forall i, \end{cases}$$

with $n = 20$, $B_i \equiv \left\{x \in R^n \mid \|x-m_i\| \le \rho_i\right\}$, for $i = 1,\cdots,9$, $S \equiv \left\{x \in R^n \mid -1 \le x_j \le 1,\ j = 1,\cdots,n\right\}$, $m_i, (i = 1,\cdots,9)$, and x_t denoting ten points uniformly chosen in S such that the B_i balls do not overlap each other, J_i real values to be taken as the values of $f(\cdot)$ at m_i.

Test problems 1, 2, and 3 appeared in [18-20] respectively. Test problems 4 belongs to a set of problems introduced in [21] and implemented in the software GKLS (cfr. [22]).

We have been working in a Linux operating system environment according to the Ubuntu 10.04 LT implementation. All codes have been written in the C language in conjunction with the *OpenMPI* message passing library (version 1.4.2). The local minimization has been carried out by a code, called *cgtrust*, written by C. T. Kelley [23]. This code implements a *trust region* type algorithm that uses a polynomial procedure to compute the step size along a search direction. Since the *cgtrust* code was written according to the MatLab programming language, this has been converted in the C language. All software used is Open Source.

Two desktop computers have been used; the first equipped with an Intel Quad CPU Q9400 based on four processors, the second with an AMD PHENOM II X6 1090T based on six processors. Experiments have been carried out both on each single computer and on the two connected to a local network. In **Table 1** we report the fourteen configurations of the computing resources used in each of our experiments.

Whenever we have being exploiting just one processor of a computer, the running code was written leaving out

Table 1. Configurations of the computing resources.

No.	CPU 1	CPU 2	Procs in 1	Procs in 2
1	Quad		1	
2	Quad		2	
3	Quad		3	
4	Quad		4	
5	Phenom		1	
6	Phenom		2	
7	Phenom		4	
8	Phenom		6	
9	Quad	Phenom	2	2
10	Quad	Phenom	4	4
11	Quad	Phenom	4	6
12	Phenom	Quad	2	2
13	Phenom	Quad	4	4
14	Phenom	Quad	6	4

any reference to the *OpenMPI* library. Hence the code is largely simpler than the one used for using more than one processor. Since our algorithm makes use of random procedures, to get significant results in solving the test problems, 100 runs of the algorithm have been done on each problem. The data reported in the tables are all mean values. The parameter k that evaluates the computational cost of local searches has been computed as the sum of function and gradient evaluations of the current objective function. The algorithm stops whenever the global minimum has been found within a fixed accuracy. That is the stop rule is

$$|f^* - \bar{f}| < \varepsilon_1 |f^*| + \varepsilon_2, \quad \varepsilon_1 = 10^{-3}, \ \varepsilon_2 = 10^{-5}$$

with f^* and $\bar{f}$ the function values at the global minimum point and at the last local minimum found.

To evaluate the performance of our algorithm we consider two indices, the speedup and the efficiency; the first estimates the decrease of the time of a parallel execution with respect to a sequential run. The second index estimates how much the parallel execution exploit the computer resources. In **Tables 2-5** we report the results gathered for each configuration given in **Table 1**; in each table for each function we report the computational expired time, the speedup and the efficiency. The function evaluation is done in two ways: 1) We evaluate the function as it is defined, 2) We introduce an extra computation that consists of a loop of 5000 iterations where at each iteration the square root of 10.99 is calculated. Hence we carried out our experiments by assigning different weights to the function evaluations. Note that the columns in **Tables 4** and **5** referring to one processor has been calculated as the means of the values in the corresponding columns in 2, and 3. From the data in the tables we can state the following remarks.

1) Whenever the function evaluation cost is evaluated

Table 2. Results working with Intel Quad.

No extra computation		***Processors***			
		1	**2**	**3**	**4**
	Secs	0.08	0.11	0.08	0.08
Fun_1	*Speed*		0.73	1.00	1.00
	Eff		0.36	0.33	0.25
	Secs	17.77	20.42	12.35	7.94
Fun_2	*Speed*		0.87	1.44	2.24
	Eff		0.44	0.48	0.56
	Secs	22.39	33.25	21.69	11.76
Fun_3	*Speed*		0.67	1.03	1.90
	Eff		0.34	0.34	0.48
	Secs	19.52	11.27	6.19	4.49
Fun_4	*Speed*		1.73	3.15	4.35
	Eff		0.87	1.05	1.09
Extra computation		***Processors***			
		1	**2**	**3**	**4**
	Secs	0.18	0.14	0.09	0.09
Fun_1	*Speed*		1.29	2.00	2.00
	Eff		0.64	0.67	0.50
	Secs	167.92	208.62	104.15	85.01
Fun_2	*Speed*		0.80	1.61	1.98
	Eff		0.40	0.54	0.49
	Secs	305.63	294.49	185.17	90.14
Fun_3	*Speed*		1.04	1.65	3.39
	Eff		0.52	0.55	0.85
	Secs	103.59	55.04	23.33	15.88
Fun_4	*Speed*		1.88	4.44	6.52
	Eff		0.94	1.48	1.63

Table 3. Results working with AMD Phenom 6.

No extra computation		***Processors***			
		1	**2**	**4**	**64**
	Secs	0.1	0.1	0.07	0.07
Fun_1	*Speed*		1.00	1.43	1.43
	Eff		0.50	0.36	0.24
	Secs	12.15	28.3	11.48	7.38
Fun_2	*Speed*		0.43	1.06	1.65
	Eff		0.21	0.26	0.27
	Secs	21.84	57.1	19.73	15.24
Fun_3	*Speed*		0.38	1.11	1.43
	Eff		0.19	0.28	0.24
	Secs	19.96	17.27	5.25	3.3
Fun_4	*Speed*		1.16	3.80	6.05
	Eff		0.58	0.95	1.01
Extra computation		***Processors***			
		1	**2**	**4**	**6**
	Secs	0.11	0.16	0.08	0.08
Fun_1	*Speed*		0.69	1.38	1.38
	Eff		0.34	0.34	0.23
	Secs	133.32	176.87	64.36	40.41
Fun_2	*Speed*		0.75	2.07	3.30
	Eff		0.38	0.52	0.55
	Secs	256.75	288.45	104.45	63.89
Fun_3	*Speed*		0.89	2.46	4.02
	Eff		0.45	0.61	0.67
	Secs	94.42	41.01	15.18	10.49
Fun_4	*Speed*		2.30	6.22	9.00
	Eff		1.15	1.56	1.50

Table 4. Results working with Intel Quad and AMD Phenom 6.

No extra computation		Processors			
		1	**2 + 2**	**4 + 4**	**4 + 6**
	Secs	0.09	0.07	0.07	0.07
Fun_1	*Speed*		0.29	1.29	1.29
	Eff		0.32	0.16	0.13
	Secs	14.96	9.15	4.75	4.54
Fun_2	*Speed*		1.63	3.15	3.30
	Eff		0.41	0.39	0.33
	Secs	22.115	27.26	11.06	11.49
Fun_3	*Speed*		0.81	2.00	1.92
	Eff		0.20	0.25	0.19
	Secs	19.74	6.3	3.36	2.52
Fun_4	*Speed*		3.13	5.88	7.83
	Eff		0.78	0.73	0.78
Extra computation		***Processors***			
		1	**2 + 2**	**4 + 4**	**4 + 6**
	Secs	0.145	0.08	0.08	0.09
Fun_1	*Speed*		1.81	1.81	1.61
	Eff		0.45	0.23	0.16
	Secs	150.62	57.82	26.27	29.23
Fun_2	*Speed*		2.60	5.73	5.15
	Eff		0.65	0.72	0.52
	Secs	281.19	70.62	52.5	45.71
Fun_3	*Speed*		3.98	5.36	6.15
	Eff		1.00	0.67	0.62
	Secs	99.01	17.75	8.94	6.63
Fun_4	*Speed*		5.58	11.07	14.93
	Eff		1.39	1.38	1.49

Table 5. Results working with AMD Phenom 6 and Intel Quad.

No extra computation		Processors			
		1	**2 + 2**	**4 + 4**	**6 + 4**
	Secs	0.09	0.07	0.07	0.07
Fun_1	*Speed*		1.29	1.29	1.29
	Eff		0.32	0.16	0.13
	Secs	14.96	12.57	5.12	6.4
Fun_2	*Speed*		1.19	2.92	2.34
	Eff		0.30	0.37	0.23
	Secs	22.115	33.85	15.62	19.57
Fun_3	*Speed*		0.65	1.42	1.13
	Eff		0.67	0.18	0.11
	Secs	19.74	8.53	3.42	3.37
Fun_4	*Speed*		2.31	5.77	5.86
	Eff		0.58	0.72	0.59
Extra computation		***Processors***			
		1	2 + 2	4 + 4	6 + 4
	Secs	0.145	0.09	0.09	0.09
Fun_1	*Speed*		1.61	1.61	1.61
	Eff		0.40	0.20	0.16
	Secs	150.62	77.74	33.58	26.67
Fun_2	*Speed*		1.94	4.49	5.65
	Eff		0.48	0.56	0.56
	Secs	281.19	137.94	53.14	46.18
Fun_3	*Speed*		2.04	5.29	6.09
	Eff		0.51	0.66	0.61
	Secs	99.01	21.57	8.64	6.45
Fun_4	*Speed*		4.59	11.46	15.35
	Eff		1.15	1.43	1.53

according to 1) The use of more than one processor does not give significant improvements only for the first test problem. Indeed this is a very easy problem to solve and does not require a large computational cost.

2) Working with two processors the speedup becomes less than one. Clearly this has to be related to the fact that the complexity of the multi processor code is not balanced by the use of additional processors.

3) As the function is evaluated according to 2) the advantage of the multiprocessor code becomes clear.

4) The output of the four test problems is quite different; while problem 1 even with the extra computational cost does not exhibits a good efficiency, the remaining problems show significant improvements. For problem 4, the parallel algorithm improves a great deal its performances with respect to the serial version.

5. Conclusion

In order to find the global minimum of a real function of n variables, a new parallel algorithm of the multi-start and local search type is proposed. The algorithm distributes the computations across two or more processors. The data passing between cores is minimal. Numerical experiments are carried out in a linux environment and all code has been written in the C language linked to the Open Mpi libraries. Two desktop computers have been used; the first equipped with an Intel Quad CPU Q9400 based on four processors, the second with a AMD Phenom II X6 1090T based on six processors. Numerical experiments for solving four well-known test problems, have been carried out both on each single computer and on the two connected to a local network. Several configurations with up to ten processors have considered; for each configuration, speedup and efficiency are evaluated. The results show that the new algorithm has a good performance especially in the case of problems that require a large amount of computations.

REFERENCES

[1] R. G. Strongin and Y. D. Sergeyev, "Global Optimization with Non-Convex Constraints. Sequential and Parallel Algorithms," Kluver Academic Press, Dordrecht, 2000.

[2] R. Horst and P. M. Pardalos, "Handbook of Global Optimization," Kluver Academic Press, Dordrecht, 1995.

[3] R. Horst, P. M. Pardalos and N. V. Thoay, "Introduction to Global Optimization," Kluver Academic Press, Dordrecht, 1995.

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

[5] J. D. Pintér, "Global Optimization in Action," Kluwer Academic Publishers, Dordrecht, 1996.

[6] H. Tuy, "Convex Analysis and Global Optimization," Kluwer Academic Publishers, Dordrecht, 1998.

[7] P. M. Pardalos and H. E. Romeijn, "Handbook of Global Optimization Volume 2," Kluwer Academic Publishers, Dordrecht, 2002.

[8] P. M. Pardalos and T. F. Coleman, "Lectures on Global Optimization," Fields Communications Series, Vol. 55, American Mathematical Society, 2009, pp. 1-16.

[9] V. P. Gergel and Y. D. Sergeyev, "Sequential and Parallel Global Optimization Algorithms Using Derivatives," *Computer & Mathematics with Applications*, Vol. 37, No. 4-5, 1999, pp. 163-180. doi:10.1016/S0898-1221(99)00067-X

[10] Y. D. Sergeyev, "Parallel Information Algorithm with Local Tuning for Solving Multidimensional GO Problems," *Journal of Global Optimization*, Vol. 15, No. 2, 1999, pp. 157-167. doi:10.1023/A:1008372702319

[11] E. Eskow and R. B. Schnabel, "Mathematical Modelling of a Parallel Global Optimization Algorithm," *Parallel Computing*, Vol. 12, No. 3, 1989, pp. 315-325. doi:10.1016/0167-8191(89)90089-6

[12] R. Čiegis, M. Baravykaité and R. Belevičius, "Parallel Global Optimization of Foundation Schemes in Civil Engineering," *Applied Parallel Computing. State of the Art in Scientific Computing*, *Lecture Notes in Computer Science*, Vol. 3732, 2006, pp. 305-312.

[13] G. Rudolph, "Parallel Approaches to Stochastic Global Optimization," In: W. Joosen and E. Milgrom, Eds., *Parallel Computing*: *From Theory to Sound Practice*, *IOS*, IOS Press, Amsterdam, 1992, pp. 256-267.

[14] C. G. E. Boender and A. H. G. Rinooy Kan, "Bayesian Stopping Rules for a Class of Stochastic Global Optimization Methods," Erasmus University Rotterdam, Report 8319/0, 1985.

[15] A. S. Nemirovsky and D. B. Yudin, "Problem Complexity and Method Efficiency in Optimization," John Wiley and Sons, Chichester, 1983.

[16] S. A. Vavasis, " Complexity issues in global optimization: a survey," In: R. Horst and P. M. Pardalos, Eds., *Handbook of Global Optimization*, Kluwer Academic Publishers, Dordrecht, 1995, pp. 27-41.

[17] M. Gaviano, D. Lera and A. M. Steri, "A Local Search Method for Continuous Global Optimization," *Journal of Global Optimization*, Vol. 48, 2010, pp. 73-85.

[18] A. Levy and A. Montalvo, "The Tunneling Method for Global Optimization," *SIAM Journal on Scientific Computing*, Vol. 6, No. 1, 1985, pp. 15-29.

[19] C. A. Floudas and P. M. Pardalos, "A Collection of Test Problems for Constraint Global Optimization Algorithms," Springer-Verlag, Berlin Heidelberg, 1990. doi:10.1007/3-540-53032-0

[20] L. A. Rastrigin, "Systems of Extreme Control," Nauka, Moscow, 1974.

[21] M. Gaviano and D. Lera, "Test Functions with Variable Attraction Regions for Global Optimization Problems," *Journal of Global Optimization*, Vol. 13, No. 2, 1998, pp. 207-223. doi:10.1023/A:1008225728209

[22] M. Gaviano, D. E. Kvasov, D. Lera and Y. D. Sergeyev, "Software for Generation of Classes of Test Functions with Known Local and Global Minima for Global Optimization," *ACM*, *Transaction on Mathematical Software*, Vol. 29, No. 4, 2003, pp. 469-480. doi:10.1145/962437.962444

[23] C. T. Kelley, "Iterative Methods for Optimization," SIAM, Philadelphia, 1999. doi:10.1137/1.9781611970920

Appendix

```
Procedure (Glob_server) fun=function to minimize;
np=number of processors;
call_interval=max interval between two server-client messages;
x1=starting point; fx1=fun(x1);
sum_ev=1; sum_tr=0; sum_ls=0;
Data_start=struct('x',x1,'fx',fx1,'sum_ev',sum_ev,
'sum_tr',sum_tr,'sum_ls',sum_ls,'fun',fun,'call_ interval',[ ]).
stop_flag=0;
no_stop=0;
Send Data_start to all clients.
while no_stop<np-1
   Receive Data_Mid from any client
   sum_ev=sum_ev+Data_mid.sum_ev;
   sum_ls=sum_ls+Data_mid.sum_ls;
   sum_tr=sum_tr+Data_mid.sum_tr;
   if Data_mid.fx>fx1;
        Data_mid.x=x1; data_mid.fx=fx1.
   else
        x1=Data_mid.x; fx1=Data_mid.fx;
   end
   if Data_mid.stop_flag==1;
        no_stops=no_stops+1;
        stop_flag=1;
        continue
   elseif stop_flag==1
        Data_mid.stop_flag=1;
        no_stops=no_stops+1;
   end
   send to client Data_mid
end.
Procedure (Glob_client)
client=client_name;
Receive Data_start from server;
x1=Data_start.x;   fx1=Data_start.fx;
sum_ls=sum_ls+Data_start.sum_ls;
sum_tr=sum_tr+Data_start.sum_tr;
sum_ev=sum_ev+Data_start.sum_ev;
call_interval=Data_start.call_interval;
buf=struct('sum_ls', 0, 'sum_tr', 0,'sum_ev', 0);
stop_flag=0;
iter_client=0;
yes=1;
while stop_flag==0
   flag_min=0;
   iter_client=iter_client+1;
   Choose  x0  uniformly on S;
   fx=fun(x0);
   buf.sum_ev=buf.sum_ev+1;
   buf.sum_tr=buf.sum_tr+1;
   if fx<fx1 | yes
        (x2,fx2,evals)=local_search(x)
        buf.sum_ls=buf.sum_ls+1;
        buf.sum_ev=buf.sum_ev+evals;
        if fx2<fx1
             x1=x2; fx1=fx2;
             flag_min=1;
        end
   end
   if stop condition satisfied
        stop_flag=1;
   end
   if   iter_client==call_interval or flag_min==1 or
        stop_flag==1;
        Data_mid.stop_flag= stop_flag;
        Data_mid.sum_ls=buf.sum_ls;
        Data_mid.sum_tr=buf.sum_tr;
        Data_mid.sum_ev=buf.sum_ev;
        Data_mid.x=x1; Data_mid.fx=fx1;
        Data_mid.client=client;
        Send Data_mid to server;
        Receive Data_mid from server
        sum_ls=Data_mid.sum_ls;
        sum_tr=Data_mid.sum_tr;
        sum_ev=Data_mid.sum_ev;
        x1=Data_mid.x;
        fx1=Data_mid.fx1;
        stop_flag=Data_mid.stop_flag;
        buf.sum_ev=0;
        buf.sum_tr=0;
        buf.sum_ls=0;
        iter_client=0;
   end
   [yes,p2,p3]=yes_box(1,sum_ls,sum_tr,sum_ev,
   prob_value,iter,itmax);
end.
Send final data to server.
```

