Parallel implementation of multi-population differential evolution

Daniela Zaharie and Dana Petcu
West University of Timişoara, B-dul V.Pârvan 4, 1900 Timişoara, Romania,
e-mail: {dzaharie,petcu}@info.uvt.ro

Abstract. A coarse-grained parallelization of an adaptive differential evolution algorithm is described. The parallelization is based on the multi-population model, a random connection topology being used. The results obtained by the implementation of the proposed algorithm on a PC cluster show not only a speedup in execution time, but also a higher probability of convergence.

Keywords: parallel computing, evolutionary algorithms, parameter adaptation, multi-population models, PC cluster.

1 Introduction

Evolutionary algorithms are stochastic optimization methods which are particularly well suited for hard problems where little is known about the search space. They maintain a population of candidate solutions which are iteratively transformed by some nature-inspired operators: *mutation*, *recombination* and *selection*. The evolutionary process tends to find globally satisfactory, even if not optimal, solutions to the optimization problem. When they are applied to large real problems they may become too slow [14]. To overcome this difficulty either new evolutionary operators or parallelization methods have been proposed.

A result of the efforts in the first direction is the ”differential evolution” algorithm (DE), developed by Storn and Price [13], which is an efficient optimization technique on continuous domains.

On the other hand there are two main reasons for parallelizing an evolutionary algorithm: one is to achieve time savings by distributing the computational effort and the second is to benefit from the algorithmic point of view [14].

Efficient parallelization of evolutionary algorithms is not a trivial task despite the fact that these algorithms are inherently parallel. This is mainly due to the fact that the population elements interact during the evolution and the effect of operators is sensitive to their control parameters choice. This means that all the empirical knowledge concerning parameters choice gathered from serial implementations is not necessarily useful for parallel implementations (mainly if parallelization implies modification in the population structure). On the other hand there exists different classes of evolutionary algorithms: genetic algorithms, genetic programming, evolutionary strategies and evolutionary programming. Even if they are based on the same natural evolution principle they are different, and so are their parallel implementations.

During the last years, several parallel models have been proposed [1], [3], [14]. The parallelization of an evolutionary algorithm can be made at one of the following levels [14]: objective function evaluation level (*master-slave model*), population level (*multi-population model*, called also *island model* or *migration model*), elements level (*cellular model*). The cellular model leads to fine-grained parallelization while the other two lead to coarse-grained parallelization.

The aim of this work is to analyze a parallel implementation of the differential evolution based on the multi-population model. The reasons of choosing the multi-population model were: *(i)* it is inspired from the spatial structure of natural populations; *(ii)* its ability of preserving the population diversity through the migration process.

Due to the absence of a real mutation, differential evolution is highly predisposed to a fast decrease of the population diversity which leads to the undesirable premature convergence (the algorithm is trapped into a sub-optimal state). The multi-population approach could help avoiding such a situation.

Unlike other evolutionary algorithms for which different parallel implementations have been proposed there exists few parallel implementation of differential evolution [9], [12]. These are based on parallelization of the objective function evaluation and on the cellular model, respectively. As is motivated in [9], the multi-population model has been avoided because of the difficulties related with the parameters choice. Since the algorithm which we propose is adaptive, the parameter choice is no more a difficulty and, as we shall see, the multi-population approach can improve the DE behavior.

The paper is organized as follows. Section 2 presents a short overview of the parallel models for evolutionary algorithms. The adaptive differential evolution and the multi-population approach are detailed in Section 3. Section 4 describes the parallel implementation and the numerical tests performed on a PC cluster. Concluding remarks are presented in Section 5.

2 Overview of parallel evolutionary algorithms

The parallelization at the objective function level has as main motivation the fact that the most costly computation is the objective function evaluation. In this model, all computations excepting the evaluation operation is performed by a master processor. The evaluation step consists in computing the objective function values for all the elements of the population and this step is done in parallel in the following way. The master processor transfers elements of the population to slave processors which have only the role of evaluating the objective function for the received element. After the master receives the data from the slaves, it continues with the next step of the algorithm (e.g. selection). This synchronous updating of the population is not very efficient due to the need of waiting until the last slave return the function value. In order to prevent these idle times, in [2] is used an asynchronous update of the population: each new element obtained by mutation and recombination is send to a slave processor for evaluation without waiting to construct all the new candidates; moreover when a slave returns a result, if the corresponding element is better than the worst element in the population then will replace it. This algorithm was tested in [2] on a concrete problem on a LAN of Sun Sparc workstations using PVM. Parallelization at objective function level is efficient if the population size is huge and the objective function is complex [5]. PGA pack (http://www.mcs.anl.gov/pgapack) is a free software build on this model.

In the multi-population model, the population is divided into sub-populations called *is-*

lands or *demes*. In each island a standard sequential evolutionary algorithm is executed. The communication between sub-populations is assured by a migration process: after some generations several elements leave their island and migrate to another. This process has in important role in preserving the population diversity, thus in avoiding premature convergence cases. Its effectiveness depends on the communication topology (a graph structure in which the sub-populations are nodes and the connections indicate the communicating sub-populations). The frequency of migration (number of generations between two migrations), the migration topology, and the selection of migrants have to be established for each evolutionary algorithm because a general rigorous way to choose them is not known at present. From an implementation point of view, the coarse-grained models are suited to distributed memory multicomputers and clusters only if the ratio of computation to communication is high [14].

There exist several implementations of parallel genetic algorithms based on the multi-population model. For example, VEGA [18] implements a distributed genetic algorithm based on the island model with exchange of individuals over a defined connection topology (ring, grid, x-net, hypercube, fully connected); it works on the MIMD computer Intel Paragon and workstation clusters.

In the cellular model (also called neighborhood or diffusion models) the elements are placed on the nodes of a toroidal one- or two-dimensional grid. The evolutionary transformations take place within a small neighborhood. They are suited for massive parallel machines.

Different hybrid strategies were also considered. The idea proposed in [10] is to consider an architecture based on the island model at a top level where each island acts based on a cellular model. A process is dedicated to control the islander's evolution, to manage the migration, to collect the local statistics, and finally to generate the global ones.

Choosing the adequate model depends on the problem particularities, on the specific of the evolutionary algorithm, and on the available hardware support.

3 The multi-population adaptive differential evolution

The differential evolution algorithm has been successfully applied in solving continuous optimization problems encountered in engineering design [8]. To describe the algorithm structure we consider the problem of finding the minimum, $x^* \in D$, of an objective function $f : D \subset \mathbb{R}^n \to \mathbb{R}$ on which we do not impose any restriction.

The classical DE algorithm evolves a fixed size population, $P = \{x_1, \dots, x_m\}$, which is randomly initialized with elements from D. After population initialization, an iterative process is started, and, at each iteration (generation), a new population is produced until a stopping condition is satisfied. At each generation, each population element (x_l) could be replaced (with probability p) with a new generated element. The new element is a linear combination between a randomly selected element (x_{α^l}) and a difference between other two randomly selected elements (x_{β^l} and x_{γ^l}). For each l, the indices α^l, β^l and γ^l are selected without replacement from $\{1, \dots, m\}$. Besides the population size (m), the parameters of the algorithm are: $p \in [0, 1]$ (the probability of replacing an element with the new generated one) and $F > 0$ (the factor which amplify the "differential" term). Experimental studies show that DE convergence properties are highly dependent on the algorithm parameters [6], [13], [15]. Thus, as for other evolutionary algorithms, adaptation methods are highly desirable. In [16] is proposed a parameter adaptation based on the idea of controlling the population diversity.

The classical DE is modified as follows. The parameters F and p are replaced with two sets of parameters: $\{F_i\}_{i=\overline{1,n}}$ and $\{p_i\}_{i=\overline{1,n}}$ (a pair of parameters, (F_i, p_i) for each component). At each generation the variances for all n components are computed as follows:

$$\text{Var}(x^i(g)) = \frac{1}{m}\sum_{l=1}^{m}\left(x^i_l(g) - \frac{1}{m}\sum_{k=1}^{m} x^i_k(g)\right)^2, \quad i = \overline{1,n}.$$

These values are used to compute the new parameters. The parameter modification is based on the value $c_i(g+1) = \gamma\text{Var}(x^i(g))/\text{Var}(x^i(g+1))$, with $\gamma > 0$ a *new* control parameter. The values $c_i(g+1)$, $i = \overline{1,n}$ are used to adjust the parameters involved in the computation of the new population, $x(g+2)$. At each generation only one set of parameters is modified. For instance, at even generations the values F_i are modified as follows:

$$F_i = \begin{cases} \sqrt{\dfrac{m(c_i-1)+p_i(2-p_i)}{2mp_i}} & \text{if } m(c_i-1)+p_i(2-p_i) \geq 0 \\ F_{\text{inf}} & \text{if } m(c_i-1)+p_i(2-p_i) < 0 \end{cases} \tag{1}$$

with F_{inf} the minimal value for F. A sufficient condition for increasing the population variance by recombination is that $F \geq 1/\sqrt{m}$, thus we shall use $F_{\text{inf}} = 1/\sqrt{m}$. An upper bound for F_i can also be imposed (empirical results suggest $F_{\text{sup}} = 2$).

At odd generations, the parameters p_i are adapted as follows:

$$p_i = \begin{cases} -(mF_i^2-1)+\sqrt{(mF_i^2-1)^2-m(1-c_i)} & \text{if } c_i \geq 1 \\ p_{\text{inf}} & \text{if } c_i < 1 \end{cases} \tag{2}$$

with p_{inf} the minimal value for p_i. Since $p_i \in [0,1]$ a minimal value for p_i should be near 0, e.g. $p_{\text{inf}} = 0.01$ while the maximal value should be $p_{\text{sup}} = 1$. When the computed values of the parameters are outside their domain, they are corrected as follows: a value less than the lower bound is replaced with the lower bound, while a value greater than the upper bound is replaced with the upper bound. The general structure of the adaptive DE algorithm is illustrated in Fig. 1.

Unlike other adaptation rules [4] (e.g. self-adaptation) where each population element has specific values of the parameters, the proposed method uses different values for each component. In this way the particularities of the fitness landscape could be ”captured” by the parameters adaptation process. Moreover, this will simplify the communication between sub-populations in the multi-population approach.

We consider now a multi-population approach for the adaptive DE. Our model consists in dividing the population in s sub-populations of the same size, μ. On each sub-population an adaptive DE is executed for a fixed number, τ, of generations. Each DE corresponding to a sub-population works with its own sets of randomly initialized adaptive parameters.

After each τ generations a migration process, based on a random connection topology, is started. More specifically, the migration strategy consists in: each element from each sub-population can be swaped (with a given *migration probability*, p_m) with a randomly selected element from a randomly selected sub-population (including the sub-population which contains the initial element).

Due to the migration process, a sub-population with a low diversity can be ”revived” after the migration takes place. Hence the multi-population approach allows avoiding premature convergence situations in DE. We shall illustrate this by numerical results obtained for some benchmark test functions (see Table 1).

```
Random initialization of $P(0)=\{x_1(0),\dots,x_m(0)\}$, $\{F_i\}_{i=\overline{1,n}}$ , $\{p_i\}_{i=\overline{1,n}}$
$g=0$
Compute Var$(x^i(0))$, $i=\overline{1,n}$
Repeat
  Perturbation step:
    $z_l^i = \begin{cases} x^i_{\alpha^l}(g)+F_i\cdot(x^i_{\beta^l}(g)-x^i_{\gamma^l}(g)) & \text{with probability } p_i \\ x^i_l(g) & \text{with probability } 1-p_i \end{cases}$   $l=\overline{1,m}, i=\overline{1,n}$
  Evaluation step:
    Compute $f(x_1),\dots,f(x_m)$
  Selection step:
    If $f(z_l)<f(x_l(g))$ then $x_l(g+1)=z_l$ else $x_l(g+1)=x_l(g)$   $l=\overline{1,m}$
  Variance computation:
    Compute Var$(x^i(g+1))$, $i=\overline{1,n}$
  Parameters adaptation:
    If $g$ is even then adapt $F_i$, $i=\overline{1,n}$ according with (1)
     else adapt $p_i$, $i=\overline{1,n}$ according with (2)
  $g=g+1$
Until  a stopping criterion is satisfied.
```

Figure 1: The general structure of the adaptive DE algorithm

Table 1: Test functions

Name	Expression	Domain
Sphere	$f_1(x)=\sum_{i=1}^{n} x_i^2$	$[-100,100]^n$
Rastrigin	$f_2(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	$[-5.12,5.12]^n$
Griewank	$f_3(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos(x_i/\sqrt{i})+1$	$[-600,600]^n$
Ackley	$f_4(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	$[-32,32]^n$
Rosenbrock	$f_5(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	$[-30,30]^n$

Table 2: Influence of migration on the adaptive DE. Test functions: f_1 and f_2, $\epsilon = 10^{-6}$, $\gamma = 0.5$.

			Sphere (f_1)			Rastrigin (f_2)		
s	μ	p_m	Success cases/gen.	PC cases/gen.	$\langle f_* \rangle$	Success cases/gen.	PC cases/gen.	$\langle f_* \rangle$
1	60	0	-	10/2002	9.2829	-	10/2011	8.1777
2	30	0.5	7/1223	3/1700	0.0001	-	10/2398	3.0844
3	20	0.5	10/1282	-	10^{-6}	2/2852	8/2988	0.9950
4	15	0.5	10/1337	-	10^{-6}	4/3447	6/3552	0.7959
5	12	0.5	10/1410	-	10^{-6}	7/3967	3/4385	0.2984
6	10	0.5	10/1478	-	10^{-6}	7/4631	3/4874	0.3979

Table 3: Influence of migration on the adaptive DE. Test functions: f_3 and f_4, $\epsilon = 10^{-6}$, $\gamma = 0.5$.

			Griewank (f_3)			Ackley (f_4)		
s	μ	p_m	Success cases/gen.	PC cases/gen.	$\langle f_* \rangle$	Success cases/gen.	PC cases/gen.	$\langle f_* \rangle$
1	60	0	-	10/2251	0.3926	-	10/1613	0.2730
2	30	0.5	6/1205	4/1946	0.0031	-	10/1562	0.0011
3	20	0.5	10/1285	-	10^{-6}	-	10/1582	$4 \cdot 10^{-6}$
4	15	0.5	10/1314	-	10^{-6}	-	10/1658	$4 \cdot 10^{-6}$
5	12	0.5	10/1391	-	10^{-6}	-	10/1740	$4 \cdot 10^{-6}$
6	10	0.5	10/1456	-	10^{-6}	-	10/1825	$4 \cdot 10^{-6}$

The minimal value for all test functions is 0. Functions f_1 and f_5 have a unique minimum, while f_2, f_3 and f_4 have many local minima beside the global minimum. For instance Rastrigin's function f_2 has 11^n local minima.

During the experiments, the following parameters were fixed: $m = 60$ (the population size), $n = 100$ (the problem dimension), $\tau = 100$ (the number of generations between migrations), $p_m = 0.5$ (the migration probability), $r = 10$ (the number of independent runs of the algorithm, used to compute the averaged values). The stopping condition which we used is: "$f_* < \epsilon$ or $\langle Var(x) \rangle < 10^{-12}$" ($f_*$ is the best value of the objective function found into the population). The algorithm is considered successful (*Success*) if the first part of the condition is satisfied, and it prematurely converges (*PC*) if only the second part is true. In Tables 2 and 3, for each situation (success or premature convergence) the number of cases (from 10 independent trials) and the averaged number of generations are presented.

Having the aim of verifying if the migration process can avoid premature convergence we used a low value for the control parameter, $\gamma = 0.5$. This value induces a premature convergence on the adaptive DE, but as numerical results in Tables 2 and 3 shows, the multi-population approach assures in most cases a repairing effect.

4 Numerical tests on the parallel implementation

In this section we present the results obtained running a multi-population adaptive DE implementation on a PC cluster: 8 PC IV 1500 MHz with 256 Mb RAM interconnected via a Myrinet switch and optical fiber cables ensuring a transmission of 2 Gb/s. Such a system is suited for a random communication topology between the processes of a parallel code. The

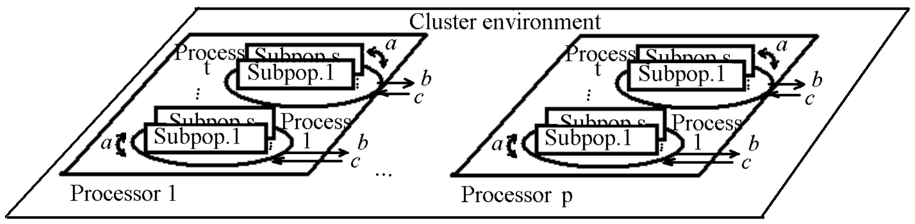

Figure 2: Implementation strategy – the migration process has three stages: (a) internal exchanges of elements from the same process; (b) send the data of the elements migrating from the current process; (c) receive the elements replacing the migrants

code is written in C and PVM (Parallel Virtual Machine, http://netlib.org/pvm). We used the test functions from Table 1.

Similar numerical tests were performed in [7] on a 16 PC cluster system (Intel PII 400 MHz, 128 Mb, FastEthernet, MPICH) for Rastrigin's function. A genetic algorithm based on gray-coding, one point crossover and standard mutation is used. The migration is based on a ring topology with connections established in a random manner each time a migration takes place. The tests used the following parameters: migration rate equal to 0.5, population size between 20 and 270, sub-populations number between 1 and 16, and 30 independent trials. The simulation of the island model is terminated when all islands satisfy the termination condition (synchronous operations are needed). The number of generations that ensure a reasonable speedup (12 for 16 processors) is around 40.

The model adopted in [11] distributes all the population elements on a toroidal landscape and the sub-populations are obtained by introducing logical boundaries. The migration strategy is defined by letting a random walk path to cross the boundaries between sub-populations on the bases of a given probability function (flip rate, the same for all boundaries). The master-slave model is used in the implementation on a SGI Power Challenge system with 10 R-1000 processors. The algorithm was applied to the Rastrigin's function with $n = 16$. The minimum value for the objective function is set to 10^{-6}, three step length for the random walk, populations sizes 64-900, 10 trials, toroidal landscape split in 4 equal regions, flip rate between 0 and 1. Tests show that a clear correlation between the optimal flip rate and the population size cannot be settled and extreme flip rates (0 and 1) are not recommended. For 10 generations the speedup obtained using 4 processors and 900 individuals is 3.8.

We expect to obtain better speedup results for smaller population sizes due to the faster and newest computing and communication architecture.

We have adopted the following strategy. The user can decide if the sub-populations will be treated in one or more processes. One processor of the cluster system can treat one or more processes. A random communication topology is used in the migration process. An element is moved with respect to a user-defined probability (p_m) in a random position of a randomly selected sub-population. The selected position being occupied by another element, the later one will migrate to the former position of the incoming individual. If the destination sub-population is treated by the same process, it suffices a simple exchange; otherwise the element will be gathered in a message buffer together with the others willing to migrate from the current process. This message buffer is send to all other processes which will extract data corresponding to the incoming elements and send back the data corresponding to the elements being replaced (Figure 2). The algorithm stops when one of sub-population satisfies the termination condition (distance to the minimal value less than 10^{-6}).

We have tested the parallel algorithm, for the test functions from Table 1, with the problem dimensions $n =$30, 100, 250, 500, the population sizes $m =$50, 100, 300, the sub-populations number $s =$1, 2, 3, 4, 6, 8, the number of processes $t =$1, 2, 3, 4, 6, 8, the number of processors $p =$1, 2, 3, 4, 6, 8, and the probability of migration from one island to another set to 0.5. The migration process follows after 100 iterations.

In order to measure the speedup of the code due to the parallel implementation we look to the mean time spent in 100 iterations of the adaptive DE algorithm and the corresponding migration stage. Let $T(p, t, s)$ denote this time spent by p processors treating t processes dealing with s sub-populations.

Figures 3 (a) and (b) show that the mean time spent by the implementation of the algorithm with several sub-populations and migrations is approximately the same like that of the algorithm which do not use sub-populations, independent on the problem dimension n and the population size m: $T(1, 1, s) \approx T(1, 1, 1)$ (in several cases, the first one is smaller). $T(1, 1, s)$ with $s > 1$ differs from $T(1, 1, 1)$ in that it includes the migration (implying $T(1, 1, s) > T(1, 1, 1)$), and the access to data is more simple (implying $T(1, 1, s) < T(1, 1, 1)$).

Figures 3 (c) and (d) show that a significant time decrease can be obtained by treating the sub-populations on different processes, due to the fact that each process deals with smaller sets of elements: $T(1, t, s) \leq T(1, 1, s)$ where $1 \leq t \leq s$, at least for $m \geq 100$ or $n \geq 100$.

The computation of the speedup at one stage of the algorithm can be done in three ways:
- algorithmic speedup: $S_p^{(1)} = T(1, 1, 1)/T(p, p, p)$;
- concurrent implementation speedup: $S_p^{(2)} = T(1, 1, p)/T(p, p, p)$;
- parallel implementation speedup: $S_p^{(3)} = T(1, p, p)/T(p, p, p)$.

According the above remarks we expect that $S_p^{(3)} \leq \min(S_p^{(1)}, S_p^{(2)})$. Figure 3 (e) suggests also these inequalities in the case of $m = 300$ and $p = 4$ processors. The speedup values are comparable with those from [11] (in our case we have a smaller population size and a higher problem dimension). Figure 3 (f) treats the dependence of the $S_p^{(2)}$ values on the problem dimension and population size in the case of $p = 2$ processors (no major differences).

The code efficiency is measured using the above defined speed-ups and the number of processors: $E_p = S_p/p$. Figure 3 (g) refers to $E_p^{(2)}$ in the case of the population size $m = 300$ and $s = p$ sub-populations. Note that high efficiency values ($E_p^{(2)} \geq 85\%$) also in the case of $p = 8$ processors.

Figure 3 (h) shows that only small differences will be encountered by changing the problem: $E_p^{(3)}$ is computed in the case of $m = 300$ and $n = 250$.

5 Conclusions

We implemented a coarse-grained model of an adaptive differential evolution algorithm on a PC cluster. The parallelization is based on the multi-population model. In summary, we obtained the following results: parallel execution on the cluster is able to speedup the DE significantly, and the global optimum is found with a higher probability even if there exist many similar sub-optima. Improvements of the convergence have been obtained even by sequential implementation due to the ability of the migration process to preserve the population diversity, thus to avoid premature convergence.

In [17] an adaptive Pareto differential evolution algorithm for multi-objective optimization and its parallelization are further proposed.

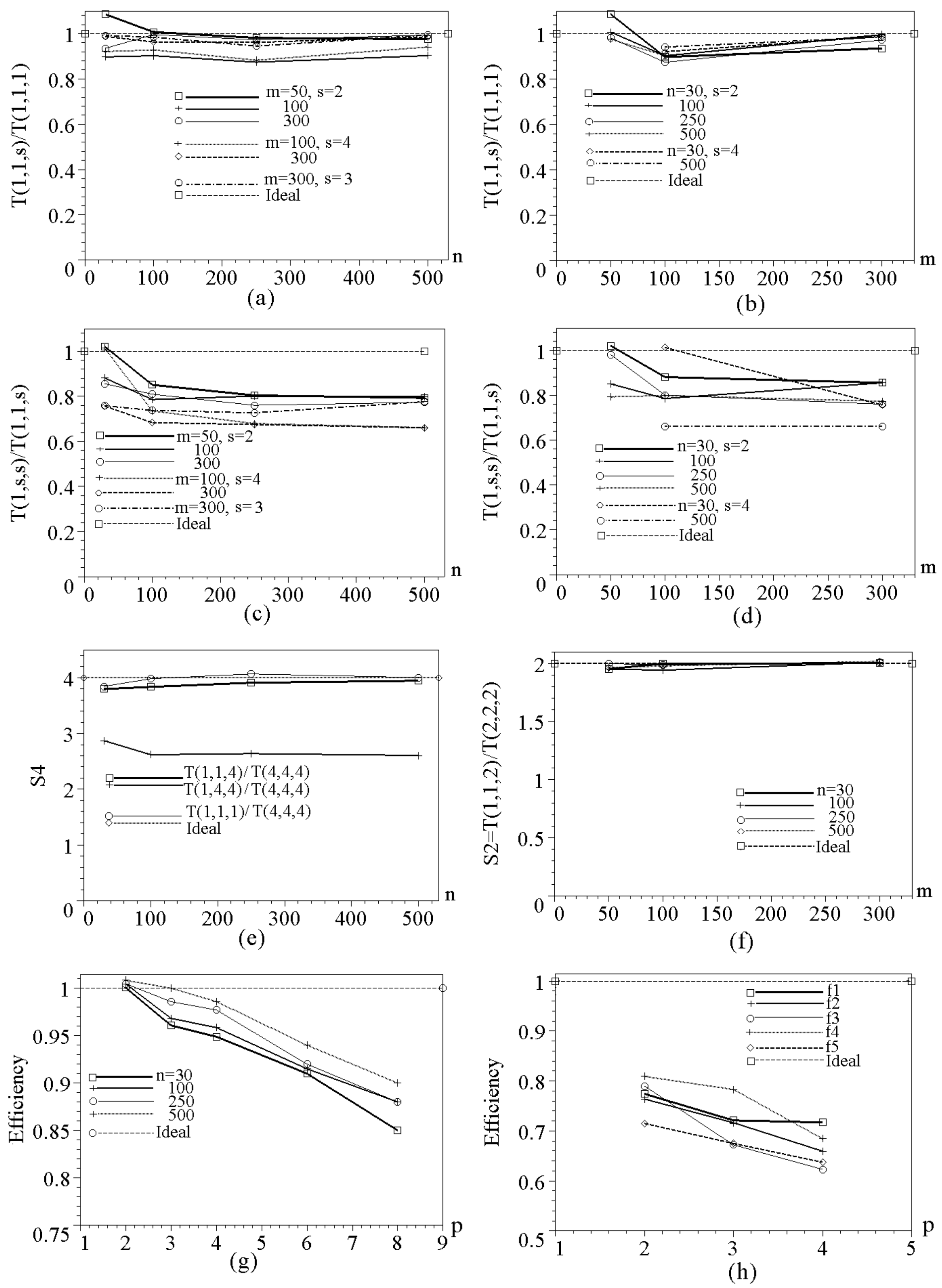

Figure 3: Time, speedup and efficiency of the parallel algorithm applied to Rastrigin's function f_2: (a) time differences when using or not sub-populations and migrations – n is variable; (b) the same – m is variable; (c) time differences when using or not different processes to treat the sub-populations – n is variable; (d) the same – m is variable; (e) different measurements for the speedup in the case of $p = 4$; (f) dependence of the speedup on m and n in the case $p = 2$; (g) parallel code efficiency in the case $m = 300$; (h) efficiency variation when the code is applied to several functions

Acknowledgements. This work has been supported by the project RO-InfoSoc-3 no. 61/2002.

References

[1] P. Adamidis, Parallel Evolutionary Algorithms: A Review, 4th Hellenic-European Conference on Computer Mathematics and its Applications (1998).

[2] Thomas Bäck, Thomas Baielstein, Boris Naujoks, Jochen Heistermann, Evolutionary algorithms for the optimization of simulation models using PVM. In Proc. EuroPVM'95, eds. J. Dongarra, M. Gengler, B. Tourancheau, X. Vigouroux (1995) 277–282.

[3] E. Cantu-Paz. A survey of parallel genetic algorithms. IlliGal Report No. 97003 (1997).

[4] A.E. Eiben, R. Hinterding, Parameter Control in Evolutionary Algorithms; IEEE Trans. on Evolutionary Computation, vol. 3, no. 2 (1999) 124–141.

[5] C. T. Forgaty and R. Huang, Implementing the genetic algorithm on transputer based parallel processing systems. In *Proc. of Parallel Problem from Nature* (1991) 145–149.

[6] R. Gämperle, S.D. Müller, P. Koumoutsakos, A Parameter Study for Differential Evolution, in A. Grmela, N.E. Mastorakis (eds.) Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS Press (2002) 293–298.

[7] Tomoyuki Hiroyasu, Mitsunori Miki and Yusuke Tanimura, The difference of parallel efficiency between the two models of parallel genetic algorithms on PC cluster systems. In Proc. 4th Intern. Conf. High Performance Computing in Asia-Pacific region (1999) 945–948.

[8] J. Lampinen, A Bibliography of Differential Evolution Algorithm, Technical Report, Lappeenranta University of Technology, IT Department, Laboratory of Information Processing (1999).

[9] J. Lampinen J., Differential Evolution-New Naturally Parallel Approach for Engineering Design Optimization, in Barry H.V. Topping(ed.), Developments in Computational Mechanics with High Performance Computing, Civil-Comp Press, Edinburgh (1999) 217–228.

[10] Michael Rocha, Filipe Pereira, Sónia Afonso and Jóse Neves, A genetic and evolutionary programming environment with spatially structured populations and built-in paralallelism. In Proc. IEA/AEI 2001, eds. L. Monosori, J. Váncza and M. Ali, LNAI **2070** (2001) 383–392.

[11] D. Quagliarella and A. Vicini, Sub-population policies for a parallel multiojjective genetic algorithm with applications to wing design. In Proc. of Intern. Conf. on Systems, Man & Cybern. (1998) 3142–3147.

[12] M. Salomon, Parallélisation de l'évolution différentielle pour le recalage rigide d'images médicales volumiques, RenPar'2000, Besançon (2000).

[13] R. Storn, K. Price, Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Technical Report TR-95-012, ICSI (1995).

[14] M. Tomassini, Parallel and Distributed Evolutionary Algorithms: A Review In Evolutionary Algorithms in Engineering and Computer Science, J. Wiley and Sons, Chichester, 1999, K. Miettinen, M. Mkel, P. Neittaanmki and J. Periaux (eds.) (1999) 113–133.

[15] D. Zaharie, Critical Values for the Control Parameters of Differential Evolution Algorithms, in R. Matoušek, P. Ošmera (eds.), Proc. of Mendel 2002, 8th International Conference on Soft Computing (2002) 62–67.

[16] D. Zaharie, Parameter Adaptation in Differential Evolution by Controlling the Population Diversity, in D.Petcu et al. (eds), Proc. of 4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania (2002) 385–397.

[17] D. Zaharie and D. Petcu, Adaptive Pareto Differential Evolution and its Parallelization, submitted to PPAM 2003.

[18] A. Zell and R. Baumann, Distributed Genetic Algorithms on the Intel Paragon, a large MIMD Computer. In Proc. of the 5th Ann. Conf. on Evolutionary Programming, T. B. L. J. Fogel, P. J. Angeline (eds) (1996).