Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 721–739
PARALLEL COMPUTING
www.elsevier.com/locate/parco

PSFGA: Parallel processing and evolutionary computation for multiobjective optimisation

F. de Toro Negro [a,*], J. Ortega [b], E. Ros [b], S. Mota [b], B. Paechter [c], J.M. Martín [a]

[a] *Department of Electronic Engineering and Computer Science, University of Huelva, E.P.S. La Rabida, Crtra Huelva – La Rabida s/n, 21047 Huelva, Spain*
[b] *Department of Computer Technology and Computer Architecture, ETSI informática, c/Daniel Saucedo Aranda, University of Granada, 18071 Granada, Spain*
[c] *School of Computing, Napier University, 10 Colinton Road, Edinburgh, EH10 5DT, Scotland, UK*

Received 10 November 2003; accepted 15 December 2003
Available online 20 May 2004

Abstract

This paper deals with the study of the cooperation between parallel processing and evolutionary computation to obtain efficient procedures for solving multiobjective optimisation problems. We propose a new algorithm called PSFGA (parallel single front genetic algorithm), an elitist evolutionary algorithm for multiobjective problems with a clearing procedure that uses a grid in the objective space for diversity maintaining purposes. Thus, PSFGA is a parallel genetic algorithm with a structured population in the form of a set of islands. The performance analysis of PSFGA has been carried out in a cluster system and experimental results show that our parallel algorithm provides adequate results in both, the quality of the solutions found and the time to obtain them. It has been shown that its sequential version also outperforms other previously proposed sequential procedures for multiobjective optimisation in the cases studied.

Keywords: Parallel evolutionary algorithms; Multiobjective optimisation; Cluster of computers

* Corresponding author.
E-mail address: ftoro@diesia.uhu.es (F. de Toro Negro).

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2003.12.012

1. Introduction

Most real-world optimisation problems are multiobjective in nature, since they normally have several (usually conflicting) objectives that must be satisfied at the same time. These problems are known as MOP (multiobjective optimisation problems) [1]. The notion of optimum has to be re-defined in this context, as we no longer aim to find a single solution; a procedure for solving MOP should determine a set of good compromises or trade-off solutions, generally known as Pareto optimal solutions from which the decision maker will select one. These solutions are optimal in the wider sense that no other solution in the search space is superior when all objectives are considered.

Evolutionary algorithms (EAs) have the potential to find multiple Pareto optimal solutions in a single run and have been widely used in this area [1–3]. After the first studies on multiobjective optimisation evolutionary algorithms (MOEAs) in the mid-1980s, a number of Pareto-based techniques were proposed [4–6]. These approaches did not explicitly incorporate elitism [2]: recently, however, the importance of this concept in multiobjective searching has been recognized and supported experimentally [2,7,8].

EAs are naturally prone to parallelism since most variation operators can be easily undertaken in parallel [9]. Much work has been reported about the performance of parallel EAs applied to single objective optimisation problems [10], but little insight has been given about the behaviour of these algorithms in multiobjective optimisation problems.

The development of parallel evolutionary algorithms for multiobjective problems involves the analysis of different paradigms for parallel processing and their corresponding parameters. Thus, this is not a simple question and not many publications have appeared dealing with the topic. In [11] a generic formulation for parallel multiobjective evolutionary algorithms (pMOEA) is provided and questions related with migration, replacement and niching schemes in the context of pMOEA are discussed. In this way [11] is a very good reference for a complete insight of the pMOEA field.

It is possible to speed up a multiobjective evolutionary algorithm by distributing the work involved in the evaluation of the fitness functions for the individuals in the population. In the case of multiobjective evolutionary algorithms we have a high amount of parallelism, not only considering that we have a set of solutions to evaluate in the population, but also taking into account the set of objectives to optimise. Nevertheless, in this paper we are not interested in this kind of parallelism that, of course, allows a very good way to accelerate the optimisation algorithm. Here, we consider another parallel approach based on a spatial decomposition of the population across the set of processors in the computer, not only to get a faster optimiser but also to obtain solutions with better quality.

In this sense, some of the authors proposed the SFGA [12], an elitist multiobjective evolutionary algorithm that has been successfully used in real-world optimisation applications [13–15] and the PSFGA [12], a pMOEA based on SFGA. Among the different four major paradigms for pMOEA [11] (*master-slave*, *island*, *diffusion*, or *hybrid*), PSFGA uses the island paradigm.

In [11] four basic pMOEA based on the island paradigm are described: (1) islands execute the same MOEA; (2) islands execute different MOEA; (3) each island evaluates a different subset of objective functions; and (4) each island considers a different region of the search domain. Taking into account this classification, PSFGA can be included in the fourth group. In order to provide an adequate cover of the search space and to provide a balanced workload to reach good levels of efficiency, migration of solutions among processors should be implemented. In PSFGA, this is done through communication between the processors that execute each island and a central or master processor that executes a number of serial generations and redistribute the population according to the known Pareto front.

This paper reviews SFGA and PSFGA and extend previous work [12] by adding new experimental results that explore the benefits of the parallel scheme and its performance in a cluster system.

In this paper, Section 2 introduces the MOPs and reviews the single front genetic algorithms. Section 3 is devoted to the experimental results. This section covers the comparison between SFGA (the sequential version of PSFGA), SPEA [7], one of the state-of-the art elitist MOEAs (multiobjective optimisation evolutionary algorithm) and PSFGA. The concluding remarks are summarized in Section 4.

2. Evolutionary multiobjective optimisation and the single front genetic algorithms

A multiobjective optimisation problem (MOP) can be defined [1] as one of finding a vector of decision variables $x \in \Phi \subseteq \Re^n$ that satisfies a set of constraints and optimises a vector function:

$$f(x) = [f_1(x), f_2(x), \ldots, f_k(x)]^{\mathrm{T}} \tag{1}$$

whose elements represent the objectives. These functions form a mathematical description of performance criteria, and are usually in conflict with each other. The meaning of optimum is not well defined in this context, so it is difficult to have a vector of decision variables that optimises all the objectives at the same time in these problems. Therefore, the concept of Pareto-optimality is used. Thus, a point $x^* \in \Phi \subseteq \Re^n$ is defined as Pareto optimal (for minimization problems) if the following condition is satisfied:

$$\forall \mathbf{x} \in \Phi, \quad \exists i \in \{1, \ldots, k\} / f_i(x^*) < f_i(x) \wedge \forall j \neq i \in \{1, \ldots, k\} f_j(x^*) \leqslant f_j(x) \tag{2}$$

This means that x^* is Pareto optimal if no feasible vector x exists that would decrease one criterion without causing a simultaneous increase in at least one of the others. The notion of Pareto optimum almost always gives not a single solution, but rather a set of solutions called non-inferior or solutions. This set of solutions is known as the Pareto front. As, in general, it is not easy to find an analytical expression for the Pareto front, the usual procedure is to determine a set of Pareto optimal points that provide a good approximate description of the Pareto front.

Evolutionary algorithms are stochastic optimisation procedures which apply a transformation process, inspired in the species natural selection, to a set (population)

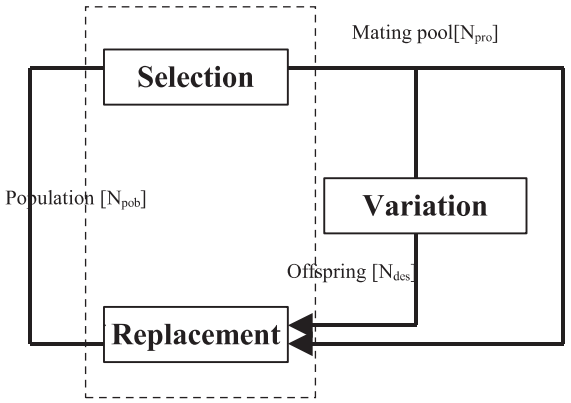

Fig. 1. General functioning of an evolutionary algorithm: A selection procedure select a subset of individuals from the population to the mating pool by using a (set of) fitness function(s). Individuals from the mating pool are transformed by applying variation operators (crossover and mutation) and then the offspring set is obtained. Finally next population is obtained using best solutions from the offspring set and the Mating pool. The procedure continues till an stop condition is satisfied (e.g. some iterations are completed).

of coded candidate solutions (individuals) of the problem (Fig. 1). These procedures are especially suited for multiobjective optimisation because they are able to capture multiple Pareto-optimal solutions in a single run, and may exploit similarities of solutions by recombination. Indeed, some research suggests that multiobjective optimisation might be an area where EAs perform better than other search strategies. The considerable amount of research related to MOEAs currently reported in the literature is evidence of present interest in this subject. A completed repository of research publications in the field of multiobjective optimisation evolutionary algorithms (MOEAs) can be found in [14].

In order to maximize the knowledge of the search space the Pareto-optimal solutions have to be uniformly distributed along the Pareto front, so MOEAs have to incorporate diversity preservation methods [1]. In addition, the use of elitism in a controlled way in this algorithms is becoming the cornerstone of the current research in the field [1].

On the other hand, it has been reported [9] that the use of parallel structured-population EAs may lead not only to a faster algorithm—with regard to the sequential version—but also to a superior numerical performance. However, little work has been reported about the performance of this kind of algorithms when applied to multiobjective optimisation problems. In what follows, both SFGA and PSFGA are described. For further details about these algorithms [12] should be consulted.

2.1. SFGA (single front genetic algorithm)

The SFGA [12], is an elitist Pareto-based algorithm for multiobjective optimisation (Fig. 2). In SFGA, all non-dominated solutions from the population are copied

In:

N_{pob} (population size)
T (generations)
p_m (mutation rate)

Out:

A (Pareto Optimal Solutions)

01 Create randomly population $P_t=P_0$
(e. g. using normal distribution)
02 Evaluate P_t for each objective function
03 Compute non-dominated individuals in P_t as P_{nd}=*NonDominated*(P_t)
04 Compute elite set as $P_t elite=P_{nd}$
05 Compute diversified elite set as $P_t elite_{diversified}$=*Clearing*($P_t$elite)
06 Copy $P_t elite_{diversified}$ to offspring population, $D_t= P_t elite_{diversified}$
07 **while** (size(Dt)<N_{pob})
08 Choose randomly without replacement x_i, $x_j \in P_t$elite.
09 Recombine x_i and x_j and produce x_k.
10 Mutate x_k with probability p_m to produce x_k'.
11 $D_t=D_t U \{x_k'\}$.
end while
12 $t=t+1$, $P_t= D_t$.
13 **If** $t<T$ go to step 02
else A=*NonDominated*(P_t) and stop

Fig. 2. Pseudo-code for the single front genetic algorithm.

to both the mating pool and the offspring set. The rest of individuals necessary for the offspring set to meet the size of the population are obtained from the application of variation operators to the mating pool (individuals). Finally the offspring set replace the previous population and the process goes on. To avoid the stagnation of the evolution (the number of solutions become equal to the size of the population so no new solutions are created) a clearing procedure is introduced (see pseudo-code in Fig. 3). This procedure eliminates some individuals, so only one representative solution remains in each area of the objective space (Fig. 4). Furthermore, the clearing procedure function as a diversity preservation method [1] encouraging the dissimilarity between the solutions in terms of distance in the objective space.

2.2. PSFGA (parallel single front genetic algorithm)

In PSFGA [12], the population is sorted with respect to one of the components of the vector function (Eq. (1)) and divided into subpopulations of equal size. In each

In:

Sigma	(clearing radius)
Kappa	(niche capacity)
U_{cl}	(Clearing threshold)
P	(Population set)
N_{pob}	(Population size)

Out:

P_{diver}	(Diversified PopulationSet)

```
01    if (size(P))>U_cl * N_pob / 100
02        for i:=0 to N_pob-1
03            Mark P[i] as "representative individual"
          end for
04        for i:=0 to N_pob-1
05            if P[i] is "representative individual"
06                for j:=i+1 to N_pob-1
07                    if P[j] is "representative individual" and
                      Distance (P[i],P[j])<Sigma
08                        if nbwinners<Kappa then nbwinners:=nbwinners+1
09                        else mark P[j] as "non-representative individual"
                  endfor
          endfor
10        P_diver=0
11        for i:=0 to N_pob-1
12            if P[i] is "representative individual" then P_diver=P_diverU{P[i]}
          endfor
13    else P_diver=P
```

Fig. 3. Pseudo-code for the clearing procedure integrated in SFGA.

subpopulation, the SFGA (Fig. 2) is executed. After some generations, all individuals are gathered and SFGA is applied to the whole population for some further generations. Then, individuals are sorted by a different component of the vector function and redistributed again. Thus, the PSFGA has alternate sequential iterations with the whole population and parallel iterations within the subpopulations.

Thus, PSFGA has a two-level mechanism to maintain the diversity in the population: at high-level PSFGA presents a structured-population island model that minimizes the recombination between individuals from different subpopulations; at low-level, the clearing procedure prevents the crowding of the individuals in each subpopulation. Both mechanisms are applied in the objective space.

PSFGA uses a Pareto-local selection scheme [1]during the parallel iterations. Because the condition of Pareto-optimality is applied locally, some individuals may be locally but globally dominated (see Fig. 8). However, the Pareto-local selection scheme is only applied in parallel iterations, and the impact on the overall perfor-

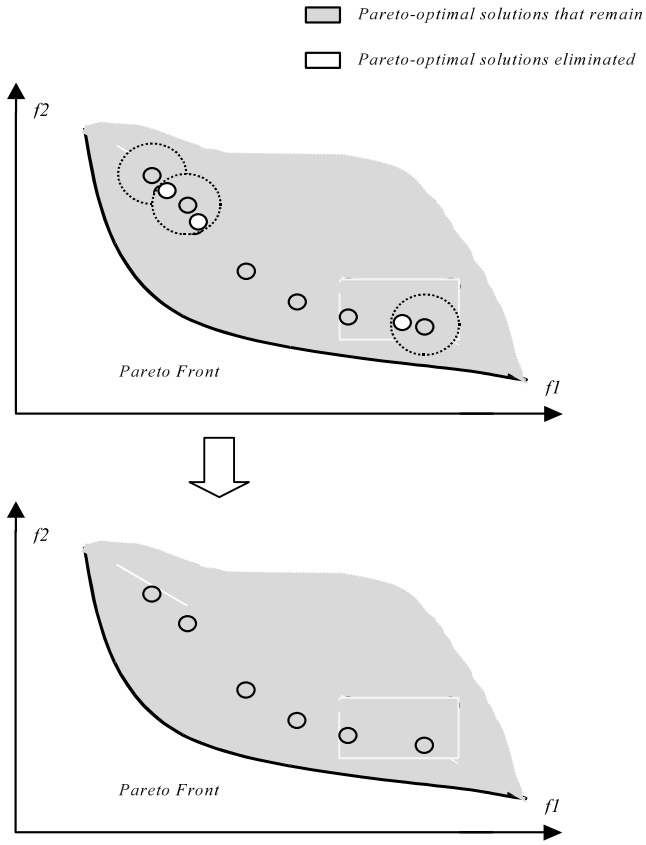

Fig. 4. Clearing procedure depiction (objective space) in a biobjective problem.

mance may be inferior to the benefits in time execution when a parallel hardware to execute the algorithm is used. In addition to this and as mentioned before, the Pareto-local selection scheme helps to preserve diversity in the population, so an empirical performance analysis is needed in order to find out the influence of these different effects.

3. Experimental results

Zitzler benchmark functions [7] (see Appendix A) have been used to evaluate the algorithms. These functions were selected by taking into account a wide range of features that may cause difficulties for an MOEA such as convexity (function ZDT1), non-convexity (ZDT2), discreteness (ZDT3), multimodality (ZDT4), and non-uniformity (ZDT6).

For performance comparison, the hyperarea metric [1,7] has been used. The hyperarea for minimization problems is the volume of the objective domain that is

no dominated by a given set of individuals. In SFGA, the clearing radius is set to 0.01 and the clearing threshold to 0.8 (the clearing procedure is only activated when more than 80% of individuals are ones).

3.1. SPEA–SFGA comparison

The mutation probability per gene used was 0.01. The algorithms were executed 30 runs for each experiment. The size of the external archive of SPEA was set to 80 and the crossover probability used was 0.6. In Tables 1 and 2 the results of the performance comparison (hyperarea) between SFGA and SPEA are shown for different number of evaluations (10 000, 20 000 and 40 000). As can be seen, SFGA obtains better results (lower hyperareas) than SPEA in most of the cases. Furthermore, SPEA has showed a severe variability in the results for ZDT4 and ZDT6 functions. Fig. 5 shows the Pareto-optimal solutions obtained by both algorithms using the same initial population.

3.2. SFGA–PSFGA comparison

The pseudo-code of PSFGA is shown in Figs. 6 and 7. The tasks of the algorithms have been organized in two different algorithms. The *master* algorithm (Fig. 6) runs SFGA on the whole population and distributes the individuals. The *worker* algorithm (Fig. 7) runs SFGA on the subpopulation.

The total number of function set evaluations performed by PSFGA in each epoch of the algorithm (a parallel computing on the subpopulations followed by a serial computing on the whole population, an a population redistribution) is given by

Table 1
Hyperarea of the Pareto-optimal solutions obtained by SFGA

Test function	Hyperarea (10 000)	Hyperarea (20 000)	Hyperarea (40 000)
ZDT1	0.554 ± 0.082	0.442 ± 0.053	0.398 ± 0.042
ZDT2	1.042 ± 0.263	0.854 ± 0.043	0.812 ± 0.046
ZDT3	0.173 ± 0.162	0.113 ± 0.045	0.082 ± 0.035
ZDT4	5.242 ± 3.124	2.483 ± 1.643	1.967 ± 1.123
ZDT6	3.858 ± 0.623	3.523 ± 0.532	3.245 ± 0.520

Table 2
Hyperarea of the Pareto-optimal solutions obtained by SPEA

Test function	Hyperarea (10 000)	Hyperarea (20 000)	Hyperarea (40 000)
ZDT1	0.602 ± 0.086	0.445 ± 0.057	0.395 ± 0.024
ZDT2	1.153 ± 0.356	0.853 ± 0.825	0.823 ± 0.723
ZDT3	0.256 ± 0.133	0.123 ± 0.052	0.092 ± 0.042
ZDT4	6.823 ± 6.234	2.924 ± 1.654	1.945 ± 1.245
ZDT6	4.123 ± 4.023	3.334 ± 3.102	3.128 ± 0.923

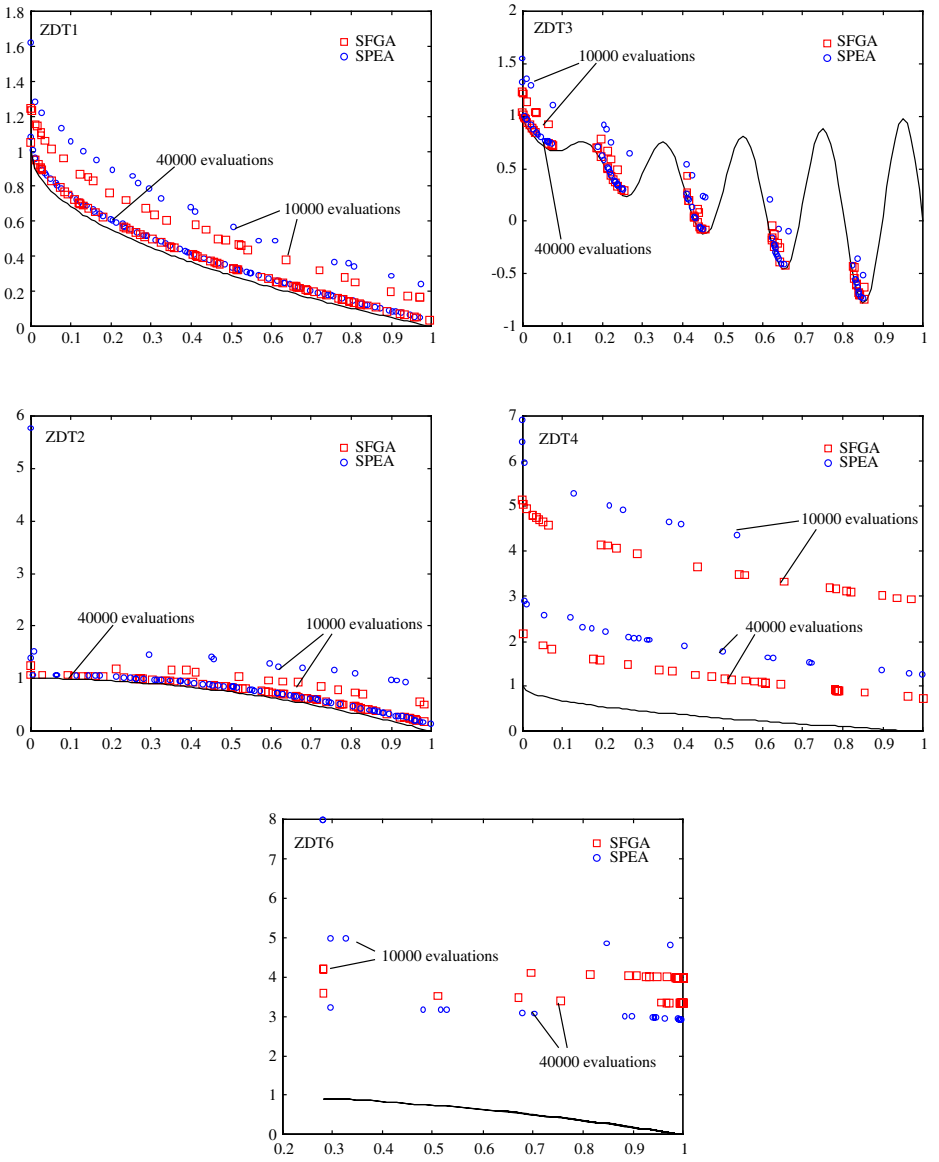

Fig. 5. Pareto-optimal solutions to the test problems (executions with the same initial population) found by SFGA and SPEA. The known Pareto front is also represented as a reference of the quality of the solutions.

Eq. (3), where genser is the number of iterations performed with the whole population, genpar is the number of iterations performed in subpopulations, and *n_workers* is the number of subpopulations.

```
Master Process
01  Initiate algorithm parameters
02  Initiate random population P of size pop_size
03  Create n_workers slave processes
04  obj ← 0
05  Rank individuals in P according to objective function f_obj
06  for i:=1 to n_workers
07      Create subpopulation SP [i] of size subpop_size
08      for j:=1 to subpop_size
09          SP[i][j] ← P[subpop_size*(i-1)+j]
        end for
10      Send parameters to i-slave process
11      Send subpopulation SP [i] to i-slave process
    end for
12  for k:=1to n_epoch
13      for i:=1 to n_workers
14          Receive subpopulation SP [i] from i-slave process
        endfor
15      for j:=1 to subpop_size
            P[subpop_size*(i-1)+j] ← SP[i][j]
        endfor
16      Execute SFGA on P for genser generations
18      obj ← (obj++)% n_obj;
17      Rank individuals in P according to objective function f_obj
18      for i:=1 to n_workers
19          for j:=1 to subpop_size
                SP[i][j]← P[sub pop_size*(i-1)+j]
            endfor
20          Send subpopulation SP[i] to i-slave process
        endfor
    endfor
```

Fig. 6. Pseudo-code of the master process of PSFGA.

$$n\_evaluations := (pop\_size \cdot genser + subpop\_size \cdot genpar \cdot n\_workers) \tag{3}$$

If the size of the population (*pop_size*) remains constant, then the size of the subpopulations depends on the number of subpopulation in the following way:

$$subpop\_size = \frac{pop\_size}{n\_workers} \tag{4}$$

In this context, the work load of the algorithm, when identified with the number of evaluations, is constant under next condition:

```
i-Slave process
21  Receive parameters from master process
22  Receive subpopulation SP[i] from master process
23  Execute SFGA on SP [i] for genpar generations
24  Send SP[i] to master process
25  for k:=1 to n_epoch
26      Execute SFGA on SP [i] for genpar generations
27      Send SP [i] to master process
    endfor
```

Fig. 7. Pseudo-code of the slave process of PSFGA.

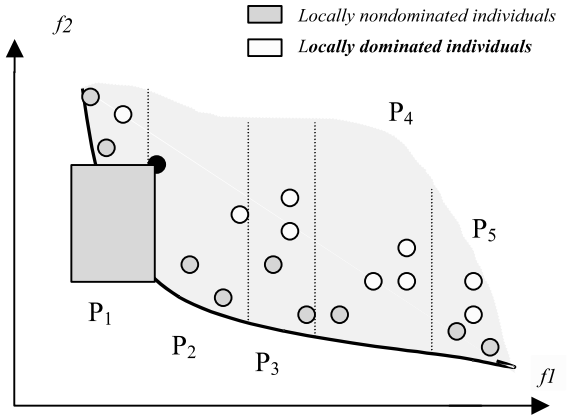

Fig. 8. Five subpopulations of PSFGA converging in a biobjective problem depiction: All subpopulations have the same size. The reference solution (black) is a local non-dominated solution in subpopulation P2, but a dominated solution in the overall population. This effect introduces an error in the non-dominated solution set computing.

$$gen\_epoch = genser + genpar = Cte \tag{5}$$

PSFGA has been implemented using MPI [16] and run in a Pentium II based PC Linux cluster of 9 nodes, so one copy of the master algorithm run in one of the nodes and every one of the *n_workers* copies of the worker algorithm run in a different node of the cluster.

In order to explore the influence of the parameters genser and genpar, two different configurations have been tried: (1) genser = 10 and genpar = 10 (labeled as (10,10)), and (2) genser = 0 and genpar = 20 (labeled as (0,10)). The first one (10,10) corresponds to an equal distribution of parallel and sequential iterations in the workers nodes and in the master one respectively. The second configuration presents a higher degree of parallelism as genser = 0. Moreover, the first configuration

represent a situation were the rate of migration of individual is higher than the in the second one, due to the fact that migration (redistribution) occur whenever master node takes control of the computing. It is important to remark that there exist redistribution of the individuals (migration) even when genser = 0 (see PSFGA master algorithm pseudocode in Fig. 6).

Tables 3 and 5 show the performance (hyperarea) of SFGA and PSFGA for each of the test problem. Stop algorithm condition occurred after the complexion of 80 000 evaluations of the function set. Tables 4 and 6 show the performance of SFGA and PSFGA when the time execution has been set to the 50% and 75% of the SFGA time convergence.

From Tables 3–6 it is possible to derive some conclusions:

- When the execution time is fixed (Tables 4 and 6), PSFGA provides better solutions than its sequential version SFGA for both (10,10) and (0,20) configurations considered with some number of subpopulations. The best performance for fixed running time is obtained for an average of 4.6 processors (i.e. 4 processors) in case of (10,10) configuration, and for an average of 4.2 processors (i.e. also 4 proces-

Table 3
Hyperarea of the Pareto-optimal solutions obtained by SFGA (*) and PSFGA with (10,10 configuration) after 80.000 evaluations

$Test_{n\_workers}$	$Hyperarea_{final}$	Speedup
ZDT1*	0.372 ± 0.005	1
ZDT1-$_2$	$\mathbf{0.351} \pm 0.002$	1.07
ZDT1-$_4$	0.356 ± 0.001	1.43
ZDT1-$_6$	0.359 ± 0.004	1.48
ZDT1-$_8$	0.356 ± 0.004	1.51
ZDT2*	0.723 ± 0.004	1
ZDT2-$_2$	$\mathbf{0.701} \pm 0.006$	1.05
ZDT2-$_4$	0.709 ± 0.005	1.40
ZDT2-$_6$	0.711 ± 0.006	1.44
ZDT2-$_8$	0.710 ± 0.005	1.48
ZDT3*	0.069 ± 0.003	1
ZDT3-$_2$	$\mathbf{0.065} \pm 0.010$	1.03
ZDT3-$_4$	0.070 ± 0.004	1.41
ZDT3-$_6$	0.054 ± 0.022	1.44
ZDT3-$_8$	0.069 ± 0.013	1.48
ZDT4*	$\mathbf{0.687} \pm 0.056$	1
ZDT4-$_2$	0.795 ± 0.028	1.08
ZDT4-$_4$	1.126 ± 0.101	1.40
ZDT4-$_6$	0.928 ± 0.077	1.52
ZDT4-$_8$	0.936 ± 0.076	1.56
ZDT6*	$\mathbf{2.440} \pm 0.038$	1
ZDT6-$_2$	2.565 ± 0.136	1.16
ZDT6-$_4$	2.678 ± 0.131	1.53
ZDT6-$_6$	2.766 ± 0.073	1.60
ZDT6-$_8$	2.740 ± 0.020	1.63

Table 4
Hyperarea of the Pareto-optimal solutions obtained by SFGA with (10,10 configuration) when the time execution is set

$Test_{n\_workers}$	$Hyperarea_{(1)}$	$Hyperarea_{(2)}$
ZDT1*	0.398 ± 0.053	0.378 ± 0.012
ZDT1-$_2$	0.383 ± 0.006	0.366 ± 0.006
ZDT1-$_4$	**0.378** ± 0.008	0.365 ± 0.004
ZDT1-$_6$	0.385 ± 0.015	0.366 ± 0.003
ZDT1-$_8$	0.382 ± 0.002	**0.364** ± 0.009
ZDT2*	0.812 ± 0.046	0.741 ± 0.010
ZDT2-$_2$	0.764 ± 0.017	0.731 ± 0.012
ZDT2-$_4$	**0.746** ± 0.018	**0.720** ± 0.008
ZDT2-$_6$	0.754 ± 0.020	0.724 ± 0.007
ZDT2-$_8$	0.752 ± 0.005	0.720 ± 0.008
ZDT3*	0.082 ± 0.035	0.069 ± 0.012
ZDT3-$_2$	**0.075** ± 0.025	0.069 ± 0.007
ZDT3-$_4$	0.079 ± 0.024	**0.069** ± 0.005
ZDT3-$_6$	0.080 ± 0.020	0.076 ± 0.013
ZDT3-$_8$	0.084 ± 0.013	0.070 ± 0.013
ZDT4*	1.967 ± 1.123	1.821 ± 0.375
ZDT4-$_2$	1.894 ± 0.206	1.812 ± 0.331
ZDT4-$_4$	1.941 ± 0.225	1.366 ± 0.041
ZDT4-$_6$	**1.540** ± 0.165	**1.068** ± 0.037
ZDT4-$_8$	1.852 ± 0.358	1.098 ± 0.114
ZDT6*	3.245 ± 0.520	3.142 ± 0.192
ZDT6-$_2$	3.162 ± 0.078	2.958 ± 0.208
ZDT6-$_4$	**3.118** ± 0.097	2.963 ± 0.156
ZDT6-$_6$	3.169 ± 0.266	**2.861** ± 0.081
ZDT6-$_8$	3.491 ± 0.590	2.995 ± 0.049

sors) in the (0,20) configuration. The (0,20) configuration is the best in five cases, while the (10,10) configuration outperforms the (0,20) one in the other five cases.

- When the number of evaluations is set as a constant (Tables 3 and 5), PSFGA outperforms SFGA when the functions ZDT1, ZDT2 and ZDT3 are considered, but PSFGA obtains worse solutions than SFGA for the ZDT6, and ZDT4. In this last case, the difference between the performance of PSFGA and SFGA is higher, and the results of PSFGA get worse as the number of subpopulations increases (above all in the (0,20) configuration). These results seem to indicate that in problems with local Pareto fronts (Function ZDT4), the iterations that involve the whole population (sequential iterations) make improve the convergence of the algorithm.
- After 80 000 evaluations, the (10,10) configuration (Tables 3 and 4) provide better results in terms of quality of solutions than (0,20) configuration (Tables 5 and 6). This means that the iterations involving the whole population have a beneficial effect as they provide a way, at least along a 50% of generations, to increment the genetic diversity and to obtain global Pareto solutions instead of local Pareto solutions.

Table 5
Hyperarea of the Pareto-optimal solutions obtained by SFGA (*) and PSFGA with (0,20 configuration) after 80.000 evaluations

$Test_{n\_workers}$	$Hyperarea_{final}$	Speedup
ZDT1*	0.372 ± 0.005	1
ZDT1-$_2$	$\mathbf{0.368} \pm 0.005$	1.51
ZDT1-$_4$	0.378 ± 0.006	5.36
ZDT1-$_6$	0.407 ± 0.005	9.83
ZDT1-$_8$	0.437 ± 0.010	11.80
ZDT2*	0.723 ± 0.004	1
ZDT2-$_2$	$\mathbf{0.713} \pm 0.010$	1.47
ZDT2-$_4$	0.753 ± 0.019	5.90
ZDT2-$_6$	0.794 ± 0.020	9.83
ZDT2-$_8$	0.829 ± 0.012	11.80
ZDT3*	0.069 ± 0.003	1
ZDT3-$_2$	0.066 ± 0.010	1.48
ZDT3-$_4$	0.062 ± 0.010	6.20
ZDT3-$_6$	$\mathbf{0.059} \pm 0.016$	10.33
ZDT3-$_8$	0.076 ± 0.008	12.40
ZDT4*	$\mathbf{0.687} \pm 0.056$	1
ZDT4-$_2$	1.114 ± 0.171	1.52
ZDT4-$_4$	1.543 ± 0.244	6.09
ZDT4-$_6$	2.401 ± 0.751	11.17
ZDT4-$_8$	3.178 ± 0.819	13.40
ZDT6*	$\mathbf{2.440} \pm 0.038$	1
ZDT6-$_2$	2.983 ± 0.013	1.67
ZDT6-$_4$	2.945 ± 0.023	6.54
ZDT6-$_6$	2.935 ± 0.324	12.10
ZDT6-$_8$	3.701 ± 0.425	14.40

- The best speed-up with respect to SFGA is obtained for (0,20) configuration. This result is not surprising as the parallelism implemented is bigger in the (0,20) configuration that in the (10,10) one.

Moreover, taking into account Fig. 9, it is apparent the PSFGA provides sets of solutions that are well distributed along the Pareto front. Both configurations, (10,10) and (0,20), behave sufficiently well in this respect. As in the comparison with SFGA, it can be also seen that the benchmarks ZDT4 and ZDT6 are the harder to optimise. Thus, in these cases, even after 80 000 evaluations the solutions provided are far from the known Pareto front.

4. Concluding remarks

A parallel algorithm for multiobjective optimisation based on an elitist evolutionary algorithm, called SFGA, has been described and analysed in this paper. The sequential implementation of PSFGA provides very good results, as it has been also shown in the paper from a performance comparison between single front genetic

Table 6
Hyperarea of the Pareto-optimal solutions obtained by SFGA with (0,20 configuration) when the time execution is set

$Test_{n\_workers}$	$Hyperarea_{(1)}$	$Hyperarea_{(2)}$
ZDT1*	0.398 ± 0.053	0.378 ± 0.012
ZDT1-$_2$	0.387 ± 0.009	0.373 ± 0.005
ZDT1-$_4$	**0.371** ± 0.010	**0.368** ± 0.010
ZDT1-$_6$	0.382 ± 0.002	0.383 ± 0.002
ZDT1-$_8$	0.404 ± 0.007	0.404 ± 0.007
ZDT2*	0.812 ± 0.046	0.741 ± 0.010
ZDT2-$_2$	0.754 ± 0.027	0.732 ± 0.011
ZDT2-$_4$	**0.730** ± 0.021	**0.731** ± 0.020
ZDT2-$_6$	0.740 ± 0.007	0.740 ± 0.007
ZDT2-$_8$	0.755 ± 0.005	0.755 ± 0.005
ZDT3*	0.082 ± 0.035	0.069 ± 0.012
ZDT3-$_2$	0.077 ± 0.013	0.059 ± 0.008
ZDT3-$_4$	**0.060** ± 0.012	**0.056** ± 0.009
ZDT3-$_6$	0.063 ± 0.012	0.060 ± 0.009
ZDT3-$_8$	0.077 ± 0.015	0.077 ± 0.018
ZDT4*	1.967 ± 1.123	1.821 ± 0.375
ZDT4-$_2$	2.295 ± 0.415	1.465 ± 0.306
ZDT4-$_4$	2.095 ± 0.524	**1.245** ± 0.242
ZDT4-$_6$	**1.763** ± 0.823	1.606 ± 0.658
ZDT4-$_8$	2.844 ± 0.913	2.844 ± 0.913
ZDT6*	3.245 ± 0.520	3.142 ± 0.192
ZDT6-$_2$	3.222 ± 0.148	3.062 ± 0.102
ZDT6-$_4$	**3.102** ± 0.156	**3.054** ± 0.145
ZDT6-$_6$	3.306 ± 0.216	3.306 ± 0.216
ZDT6-$_8$	3.495 ± 0.345	3.495 ± 0.345

algorithm (SFGA) and strength Pareto evolutionary algorithm (SPEA). Both algorithms use an elitist approach to assure the maintenance of best solutions through the iterations. In the case of SFGA, a clearing procedure helps in both controlling the elitism and preserve diversity in the population. Experiments have been carried out using a well-known multiobjective benchmark function set that covers a wide range of difficulties in finding the Pareto front. Although SFGA has a very simple design, it has shown a faster convergence rate than SPEA through the experiments. Moreover, SPEA has showed a severe variability in the results for ZDT4 and ZDT6 functions.

On the other hand, the experimental results retrieved on the cluster of computers show that PSFGA can obtain better results than SFGA when a fixed execution time is considered. For the cases studied, the benefits of the execution of PSFGA on a parallel hardware seem to be greater than the drawback of using a Pareto-local selection scheme in the parallel iterations, so better solutions can be obtained by PSFGA than in SFGA when the same time execution time is considered in both algorithms. Serial iterations with the whole population seem to work well in some problems (e.g. ZDT4) dealing with local Pareto fronts (multimodality). PSFGA configurations with high degree of parallelisation (the number of parallel iterations are superior to the

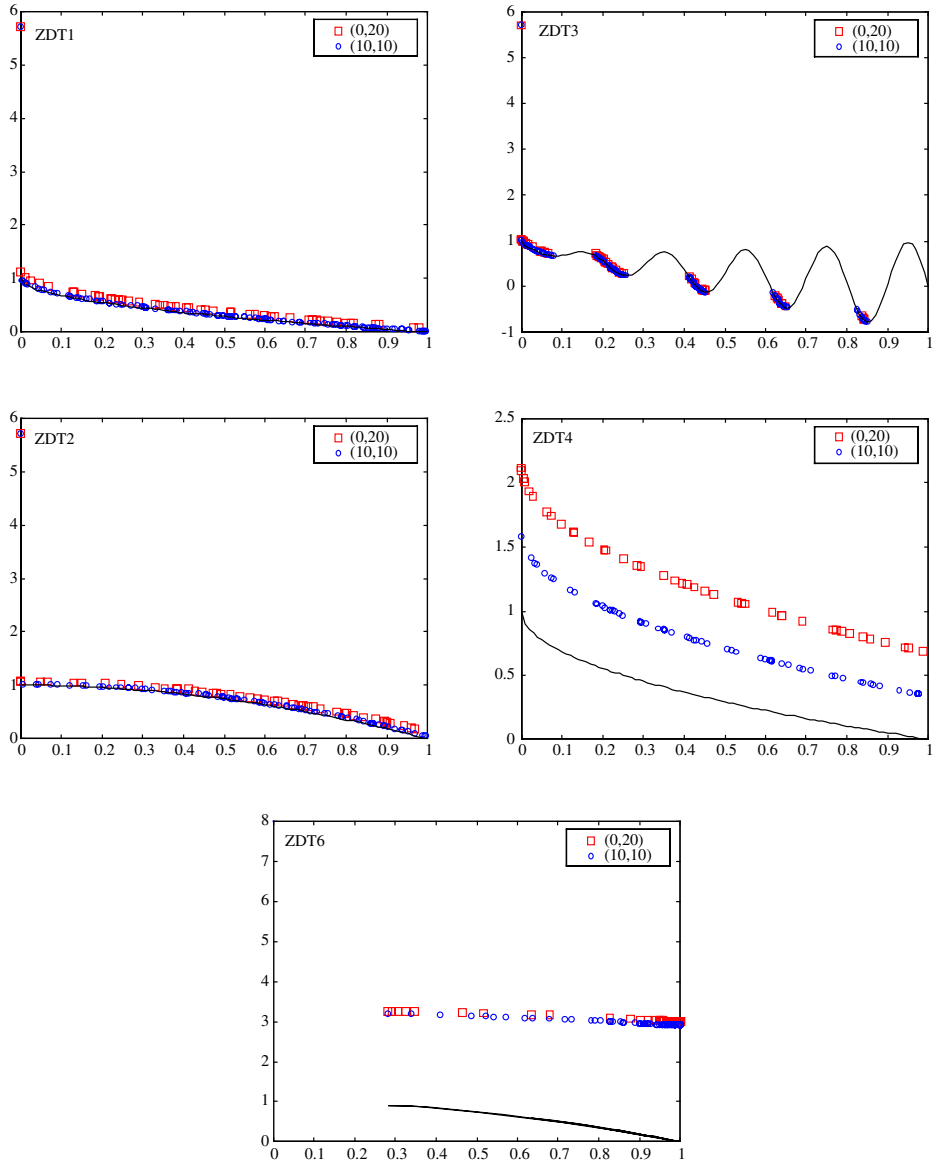

Fig. 9. Pareto-optimal solutions to the test problems (executions with the same initial population) found by PSFGA with two different configurations.

number of serial interactions) cab be useful when the optimisation problem requires an speed-up more than an increase in the quality of the solutions. In general terms, the results shown in this paper lead to think that PSFGA may be very competitive in solving real multiobjective optimisation problems where the computational cost of

the evaluation of vector function is very high, obtaining better solutions than SFGA for a given execution time.

Acknowledgements

This paper has been supported by the Spanish Ministerio de Ciencia y Tecnología under grant TIC2000-1348. The authors would also like to acknowledge the support of the European Commission through grant number HPRI-CT-1999-00026 (the TRACS Program at EPCC).

Appendix A

The benchmark functions set used in this work addresses the following minimization problem:

Minimize $\mathbf{t}(\mathbf{x}) = (f_1(x_1), f_2(\mathbf{x}))$
with $f_2(\mathbf{x}) = g(x_2, \ldots, x_n) \cdot h(f_1(x_1), g(x_2, \ldots, x_n))$
and $\mathbf{x} = (x_1, \ldots, x_n)$.

ZDT1:

$f_1(x_1) = x_1$

$$g(x_2, \ldots, x_n) = 1 + 9 \cdot \left(\sum_{i=2}^{n} x_i \right) \Big/ (n-1)$$

$h(f_1, g) = 1 - \sqrt{f_1/g}$
with $n = 30$ and $x_i \in [0, 1]$.

ZDT2:

$f_1(x_1) = x_1$

$$g(x_2, \ldots, x_n) = 1 + 9 \cdot \left(\sum_{i=2}^{n} x_i \right) \Big/ (n-1)$$

$h(f_1, g) = 1 - (f_1/g)^2$
with $n = 30$ and $x_i \in [0, 1]$.

ZDT3:

$f_1(x_1) = x_1$

$$g(x_2, \ldots, x_n) = 1 + 9 \cdot \left(\sum_{i=2}^{n} x_i \right) \Big/ (n-1)$$

$h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)$
with $n = 30$ and $x_i \in [0, 1]$.

ZDT4:

$$f_1(x_1) = x_1$$

$$g(x_2, \ldots, x_n) = 1 + 10(n-1) + \sum_{i=2}^{n} (x_i^2 - 10\cos(4\pi x_i))$$

$$h(f_1, g) = 1 - \sqrt{f_1/g}$$

with $n = 10, \quad x_1 \in [0,1]$ and $x_2, \ldots, x_n \in [-5,5]$.

ZDT6:

$$f_1(x_1) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$$

$$g(x_2, \ldots, x_n) = 1 + 9 \cdot \left(\left(\sum_{i=2}^{n} x_i \right) \Big/ (n-1) \right)^{0.25}$$

$$h(f_1, g) = 1 - (f_1/g)^2$$

with $n = 10, \quad x_i \in [0,1]$.

References

[1] C.A. Coello Coello, D.A. Van Veldhuizen, G.B. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers, 2002.

[2] G.T. Parks, I. Miller, Selective breeding in a multiobjective genetic algorithm, in: A.E. Eiben, T. Bäck, M. Schoenauer, H.P. Schwefel (Eds.), Proc. PPSN-V, Springer, Berlin, 1999, pp. 250–259.

[3] C.M. Fonseca, P.J. Fleming, Multiobjective optimisation and multiple constraint handling with evolutionary algorithms—part i: A unified formulation, IEEE Transactions on Systems, Man, and Cybernetics 28 (1998) 38–47.

[4] C.M. Fonseca, P.J. Fleming, Genetic algorithms for multiobjective optimisation: Formulation, discussion and generalization, in: S. Forrest (Ed.), Proc. of the 5th International Conference on Genetic Algorithms, Morgan Kauffman Publishers, California, 1993, pp. 416–423.

[5] N. Srinivas, K. Deb, Multiobjective optimisation using nondominated sorting in genetic algorithms, Evolutionary Computation 2 (1994) 221–248.

[6] J. Horn, N. Nafpliotis, Multiobjetive optimisation using the Niched Pareto genetic algorithm, in: Proc. 1st IEEE Conference on Evolutionary Computation, vol. 1, 1994.

[7] E. Zitzler, L. Thiele, Multiobjective optimisation using evolutionary algorithms: A comparative case study, in: A.E. Eiben, T. Bäck, M. Schoenauer, H.P. Schwefel (Eds.), Proc. PPSN-V, Springer-Verlag, Berlin, 1999, pp. 292–301.

[8] K. Deb, T. Goel, Controlled elitist sorting genetic algorithms for better convergence, in: Proc. First International Conference on Evolutionary Multi-Criterion Optimisation, Lecture Notes in Computer Science, vol. 1993, Springer-Verlag, Berlin, 2001, pp. 67–81.

[9] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002).

[10] E. Cantú Paz, D. Goldberg, A survey of parallel genetic algorithms, Calculateurs Parallèles, Réseaux et Systèmes Répartis 10 (2) (1998) 141–171.

[11] D.A. Veldhuizen, J.B. Zydallys, G.B. Lamont, Considerations in engineering parallel multiobjective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 144–173.

[12] F. de Toro, J. Ortega, B. Paechter, Parallel single front genetic algorithm: Performance analysis in a cluster system, in: Proc. International Parallel and Distributed Processing Symposium, Nice, France, 2003.

[13] F. de Toro, E. Ros, S. Mota, J. Ortega, Multi-objective optimisation evolutionary algorithms applied to paroxysmal atrial fibrillation diagnosis based on the k-nearest neighbors classifier, in: Lecture Notes in Artificial Intelligence, vol. 2527, Springer-Verlag, Berlin, 2002, pp. 313–318.
[14] F. de Toro, E. Ros, S. Mota, J. Ortega, Non-invasive Atrial disease diagnosis using decision rules: A multi-objective optimisation approach, in: Proc. 2nd International Conference on Evolutionary Multi-criterion Optimisation, Lecture Notes in Computer Science, vol. 2632, Springer-Verlag, Berlin, 2003, pp. 638–647.
[15] http://www.lania.mx/~ccoello/EMOO.
[16] Message passing interface forum, MPI: A message-passing interface standard, International Journal of Supercomputer Applications 8(3–4) (1994) 165–414.