Parallel Differential Evolution

D.K. Tasoulis[1,3], N.G. Pavlidis[1,3], V.P. Plagianakos[2,3] and M.N. Vrahatis[1,3]
[1]Department of Mathematics, University of Patras, GR-26110 Patras, Greece
[2]Department of Information and Communication Systems Engineering,
University of the Aegean, GR-83200 Samos, Greece
[3]University of Patras Artificial Intelligence Research Center
E-mail: {dtas,npav,vpp,vrahatis}@math.upatras.gr

***Abstract*— Parallel processing has emerged as a key enabling technology in modern computing. Recent software advances have allowed collections of heterogeneous computers to be used as a concurrent computational resource. In this work we explore how Differential Evolution can be parallelized in a virtual parallel environment so as to improve both the speed and the performance of the method. Experimental results indicate that the extent of information exchange among subpopulations assigned to different processor nodes, bears a significant impact on the performance of the algorithm. Furthermore, not all the mutation strategies of the Differential Evolution algorithm are equally sensitive to the value of this parameter.**

I. Introduction

Parallel processing, that is the method of having many small tasks solve one large problem, has emerged as a key enabling technology in modern computing. As a result of the demand for higher performance, lower cost, and sustained productivity, the past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing and for more general–purpose applications. The acceptance has been facilitated by two major developments: massive parallel processors, and the widespread use of distributed computing. The most important factor in parallel computing is the high cost of the hardware. In contrast, users see very little cost in running their problems on a local set of existing computers.

Exploiting recent software advances [1], [3], collections of heterogeneous computers can be used as a coherent and flexible concurrent computational resource. These technologies have allowed the vast number of individual Personal Computers available in most scientific laboratories to be used as parallel machines at no, or at a very low, cost. Network interfaces, linking individual computers, are necessary to produce such pools of computational power. Since network infrastructure is currently immature to provide sufficiently high speed data transfer interfaces, it comprises a bottleneck to the entire system. Thus applications that exploit specific strengths of individual machines on a network, while minimizing the required data transfer rate are best suited for network–based environments.

Differential Evolution (DE) is a novel minimization method [8], capable of handling nondifferentiable, nonlinear and multimodal objective functions. DE has been designed as a stochastic parallel direct search method, that utilizes concepts borrowed from the broad class of evolutionary algorithms (EAs). The method typically requires few, easily chosen, control parameters. Experimental results have shown that DE has good convergence properties and outperforms other well known evolutionary algorithms [7], [8].

In this work we explore how Differential Evolution can be parallelized in this kind of virtual parallel environment so as to obtain both speed and performance improvements. The paper is organized as follows. In Section II the Differential Evolution algorithm and the parallelization model are briefly described. Section IV-A reports the test functions used. Section IV-B is devoted to the presentation and the discussion of the experimental results. The paper ends with conclusions.

II. Differential Evolution

As previously mentioned, DE is a population–based stochastic algorithm that exploits a population of potential solutions, *individuals*, to probe the search space. New individuals are generated by the combination of randomly chosen individuals from the population. This operation in our context can be referred to as *mutation*. Specifically, for each individual w_g^k, $k = 1, \ldots, NP$, where g denotes the current generation, a new individual v_{g+1}^i (mutant individual) is generated according to one of the following equations:

$$v_{g+1}^i = w_g^{\text{best}} + \mu(w_g^{r1} - w_g^{r2}), \tag{1}$$

$$v_{g+1}^i = w_g^{r1} + \mu(w_g^{r2} - w_g^{r3}), \tag{2}$$

$$v_{g+1}^i = w_g^i + \mu(w_g^{\text{best}} - w_g^i) + \mu(w_g^{r1} - w_g^{r2}), \tag{3}$$

$$v_{g+1}^i = w_g^{\text{best}} + \mu(w_g^{r1} - w_g^{r2}) + \mu(w_g^{r3} - w_g^{r4}), \tag{4}$$

$$v_{g+1}^i = w_g^{r1} + \mu(w_g^{r2} - w_g^{r3}) + \mu(w_g^{r4} - w_g^{r5}), \tag{5}$$

$$\begin{aligned} v_{g+1}^i = (w_g^{r1} + w_g^{r2} + w_g^{r3})/3 + (p_2 - p_1)(w_g^{r1} - w_g^{r2}) \\ +(p_3 - p_2)(w_g^{r2} - w_g^{r3}) + (p_1 - p_3)(w_g^{r3} - w_g^{r1}) \end{aligned} \tag{6}$$

where w_g^{best} is the best member of the previous generation; $\mu > 0$ is a real parameter, called mutation constant, which controls the amplification of the difference between two individuals so as to avoid the stagnation of the search process; and $r_1, r_2, r_3, r_4, r_5 \in \{1, 2, \ldots, k-1, k+1, \ldots, NP\}$, are random integers mutually different and different from the running index k. The mutation strategy of Eq. (6) is known as the trigonometric mutation operator, and has been recently

proposed in [2]. This mutation strategy performs a mutation according to Eq. (6) with probability τ_μ and a mutation according to Eq. (2) with probability $(1-\tau_\mu)$. The values of p_i, $i=\{1,2,3\}$ and p' are obtained through the following equations.

$$\begin{aligned} p_1 &= \left|f(w_g^{r1})\right|/p', \\ p_2 &= \left|f(w_g^{r2})\right|/p', \\ p_3 &= \left|f(w_g^{r3})\right|/p', \text{ and} \\ p' &= \left|f(w_g^{r1})\right| + \left|f(w_g^{r2})\right| + \left|f(w_g^{r3})\right|. \end{aligned}$$

To increase further the diversity of the mutant individuals, the resulting individuals are combined with other predetermined individuals – the *target* individuals – this operation is called *recombination*. Specifically, for each component l ($l=1,2,\ldots,D$) of the mutant individual v_{g+1}^k, we randomly choose a real number r in the interval $[0,1]$. Then, we compare this number with the recombination constant, ρ. If $r \leqslant \rho$, then we select, as the l–th component of the trial individual u_{g+1}^k, the l–th component of the mutant individual v_{g+1}^k. Otherwise, the l-th component of the target vector w_{g+1}^k becomes the l–th component of the trial vector. This operation yields the *trial* individual. Finally, the trial individual is accepted for the next generation if and only if it yields a reduction in the value of the error function. This is the *selection* operation.

III. Proposed Algorithm

In this work we used the parallel virtual machine (PVM). PVM is a de facto standard message passing interface. It is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architectures. PVM is designed to link computing resources and provide users with a parallel platform for running their computer applications, irrespective of the number of different computer architectures they use and where those computers are located. It is capable of harnessing the combined resources of typically heterogeneous networked computing platforms to deliver high levels of performance and functionality. Its key concept is that it makes a collection of computers to appear as one large virtual machine, hence its name [1].

DE, like other EAs, is easily parallelized due to the fact that each member of the population is evaluated individually [6]. The only phase of the algorithm that requires communication with other individuals is reproduction. This phase can also be parallelized for pairs of individuals [4], [6]. Generally, there are two typical models for EA parallelization. The first employs fine grained parallelism, in which case, each individual is assigned to a different processor. This approach is problematic when the number of available processors is limited, or when the computation of the fitness function requires information from the entire population.

The second model, which is the one used in this paper, maps an entire subpopulation to a processor. Thus each subpopulation evolves independently toward a solution. This allows each subpopulation to develop its own solution independently. To promote information sharing, the best individual of each subpopulation is moved to other subpopulations, according to a predefined topology. This operation is called "migration". The topology of the proposed scheme is a ring, i.e. the best individuals from each subpopulation are allowed to migrate to the next subpopulation of the ring. This concept reduces the migration between the subpopulations and consequently the messages exchanged among the processors [5]. The migration of the best individuals is controlled by the migration constant, $\phi \in [0,1]$. At each iteration, a uniformly distributed random number in the interval $[0,1]$ is chosen and compared with the migration constant. If the migration constant is larger, then the best individuals of each subpopulation migrate and take the place of a randomly selected individual (different from the best) in the next subpopulation; otherwise no migration is permitted. A high level description of the parallel algorithmic scheme follows:

At the the master node
1. Spawn N subpopulations
each one on a different processor
2. For each generation
3. Receive a particle from each subpopulation
4. Decide for each particle if it is going
to migrate based on ϕ
5. Send the particles that will migrate to the
next subpopulation based on the ring topology
6. If the stop criterion for the objective function
is met send a termination signal
to all the subpopulations

At each subpopulation
1. For each generation
2. Perform a DE step
3. Send the best individual to the master node
4. Receive a migrated individual if such one
exists and assign it to a random individual
5. if a termination signal is received
terminate execution

The above algorithm could be modified by also including the concept of aging of individuals, which prevents an individual from surviving indefinitely [5], [7].

IV. Experimental Results

A. Test Functions

The seven test functions selected, appear in [8].

1) Sphere:

$$f_1(x) = \sum_{j=1}^{5} x_j^2, \qquad x_j \in [-5.12, 5.12]. \tag{7}$$

The sphere test function is a considered to be a simple minimization problem. The minimum is $f_1(0,\ldots,0)=0$.

2) Rosenbrock's Saddle:

$$\begin{aligned} f_2(x) &= 100\cdot(x_1^2 - x_2)^2 + (1-x_1)^2, \\ & x_j \in [-2.048, 2.048]. \end{aligned} \tag{8}$$

This is a two–dimensional test function, which is known to be relatively difficult to minimize. The minimum is $f_2(1,1)=0$.

3) Step Function:

$$f_3(x) = 30 + \sum_{j=1}^{5} \lfloor x_j \rfloor, \quad x_j \in [-5.12, 5.12]. \tag{9}$$

The minimum of this function is $f_3(-5-\xi, \ldots, -5-\xi) = 0$, where $\xi \in [0, 0.12]$. This function exhibits many flat regions that can cause search stagnation.

4) Quartic Function:

$$f_4(x) = \sum_{j=1}^{30} \left(j \cdot x_j^4 + \eta \right), \; x_j \in [-1.28, 1.28]. \tag{10}$$

This is test function is designed to evaluate the behavior of minimization algorithms in the presence of noise. To this end, η is a random variable following the uniform distribution in the range $[0, 1]$. The inclusion of η makes f_4 more difficult to optimize. The functional minimum of the function is $f_4(0, \ldots, 0) \leqslant 30 \cdot E[\eta] = 15$, where $E[\eta]$ is the expectation of η.

5) Shekel's Foxholes:

$$f_5(x) = \frac{1}{0.002 + \psi_1(x)}, \; x_j \in [-65.536, 65.536], \tag{11}$$

where,

$$\psi_1(x) = \sum_{i=0}^{24} \frac{1}{1 + i + \sum_{j=1}^{2} (x_j - a_{ij})^6}.$$

The parameters for this function are:

$$\begin{aligned} a_{i1} &= \{-32, -16, 0, 16, 32\}, \text{ where} \\ &\quad i = \{0, 1, 2, 3, 4\} \text{ and } a_{i1} = a_{i \bmod 5, 1} \\ a_{i2} &= \{-32, -16, 0, 16, 32\}, \text{where} \\ &\quad i = \{0, 5, 10, 15, 20\} \text{ and} \\ &\quad a_{i2} = a_{i+k,2}, \; k = \{1, 2, 3, 4\}. \end{aligned}$$

The global minimum of $f_5(-32, -32) = 0.998004$.

6) Corana Parabola:

$$f_6(x) = \sum_{j=1}^{4} \begin{cases} \psi_2(x_j), \text{ if } |x_j - z_j| < 0.05, \\ \psi_3(x_j), \text{ otherwise.} \end{cases} \tag{12}$$

where $\psi_2(x_j) = 0.15 \left(z_j - 0.05 \text{sign}(z_j)\right)^2 d_j$, $\psi_3(x_j) = d_j x_j^2$, $z_j = \lfloor 5|x_j| + 0.49999 \rfloor \text{sign}(x_j) 0.2$ and $d_j = \{1, 1000, 10, 100\}$.

The Corana test function defines a paraboloid with axes parallel to the coordinate axes. The function is characterized by a multitude of local minima, increasing in depth as one moves closer to the origin. The global minimum of the function is $f_6(x) = 0$, for $x_j \in (-0.05, 0.05)$.

7) Griewangk's Function:

$$\begin{aligned} f_7(x) &= \sum_{j=1}^{10} \frac{x_j^2}{4000} - \prod_{j=1}^{10} \cos\left(\frac{x_j}{\sqrt{j}}\right) + 1, \\ &\quad x_j \in [-400, 400]. \end{aligned} \tag{13}$$

This test function is riddled with local minima. The global minimum of the function is $f_7(0, \ldots, 0) = 0$.

B. Presentation of Results

Numerical experiments were performed using PVM version 3.4.4 and a Parallel Differential Evolution (PARDE) C++ Interface developed under the Fedora Linux 1.0 operating system using the GNU compiler collection (gcc) version 3.3.2.

In Table I the parameter setup used in the numerical experiments conducted is summarized. Specifically, D denotes the dimensionality of the problem, *NP* stands for the size of the subpopulation assigned to each of the processors employed, g is the maximum number of generations allowed, finally, μ and ρ are the values of the mutation and recombination constants, respectively. Little effort has been devoted to the selection of the values of *NP*, μ and ρ since the scope of this work is to study extensively, the implications of information sharing in a parallel environment, which is controlled by the migration constant, ϕ. It is worth noting that further performance improvements can be achieved by further fine–tuning *NP*, μ, and ρ. The parameter τ_μ used by the trigonometric mutation strategy, Eq. (6), was set to 0.1.

Test function	D	*NP*	g	μ	ρ
Sphere function	5	30	1000	0.9	0.3
Rosenbrock's saddle	2	30	1000	0.9	0.5
Step function	5	20	1000	0.8	0.3
Quartic function	30	100	2000	0.8	0.5
Shekel's foxholes	2	30	1000	0.9	0.3
Corana's parabola	4	15	2000	0.4	0.2
Griewangk's function	10	50	10000	1.0	0.3

TABLE I
PARAMETER VALUES

Figure 1 illustrates the speedup achieved by assigning each subpopulation to a different processor, relative to assigning all subpopulations to a single processor. To obtain the plotted values, the algorithm performed 1000 generations with a migration constant equal to 0.5. The setup used to obtain this Figure was 32 Pentium III Celeron 900MhZ connected through a 100Mbit Fast Ethernet network interface.

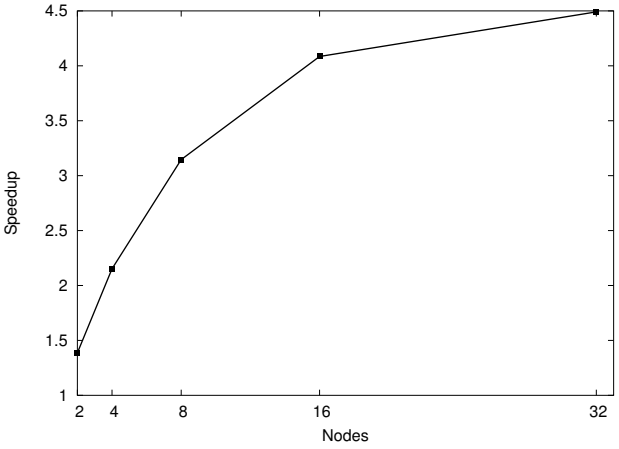

Fig. 1.

Table II reports the mean number of generations required

to locate the global minimum of each test function, averaged over all the considered mutation constants. It is clear from Table II that the best performing mutation strategy, for all test problems, was the first. Furthermore, Griewangk's function (test function 7) appears to be the hardest one to minimize.

Test function	Mutation Strategy					
	1	2	3	4	5	6
Sphere	**231.09**	611.58	247.02	341.08	409.79	642.02
Rosenbrock	**82.21**	386.01	130.14	220.77	271.39	371.59
Step	**21.91**	76.85	25.48	47.59	25.40	31.59
Quartic	**244.92**	249.90	260.21	406.13	454.18	244.11
Shekel	**63.39**	136.96	96.21	186.91	158.30	101.55
Corana	**282.35**	364.09	513.36	329.48	491.01	398.85
Griewangk	**1872.17**	1885.19	1975.61	2644.47	4448.66	2434.45

TABLE II

MEAN NUMBER OF GENERATIONS FOR THE 16–NODE MODEL

Figures 2–7 illustrate the performance of the 16–node model for all the considered mutation strategies, on all the test functions, for a particular migration constant. In all the $3D$ plots, the mutation strategies are given by the x–axis, the test functions by the y–axis, and finally, the mean number of generations required is reported in the z–axis. Concerning the overall performance of the alternative mutation strategies, the worst performance is exhibited by strategies 5 and 6. Strategies 1 and 3 appear to be the most efficient and robust.

Figures 8–14 exhibit the mean number of generations required for each migration constant, $\phi \in [0, 1]$ with stepsize 0.1, for all the mutation strategies on each of the test problems. It is evident that selecting the appropriate migration constant has a significant impact on the performance of the algorithm. Moreover, it appears that setting ϕ close to one or to zero can lead to a substantial increase in the number of generations required. A superior performance is typically obtained for intermediate values of ϕ. It has already been noted from the results of Table II and Figures 2–7 that the first mutation strategy, Eq. (1), is the most efficient. From Figures 8–14 we can infer that this strategy also exhibits the most robust behavior with respect to the migration constant. The third mutation strategy, Eq. (3), also exhibits a relatively robust behavior with respect to ϕ on test functions 1–6. It is worth noting, however, that the other considered mutation strategies can achieve a comparable, or even better, performance after fine–tuning the value of ϕ. Mutation strategies 2, 4 and 6, Eq. (2), (4) and (6) appear to be the most heavily influenced from the choice of ϕ.

V. CONCLUSION

In this work we investigate how the Differential Evolution optimization algorithm can be parallelized in a virtual parallel environment so as to reduce computational time and improve performance. The parallelization model we employed

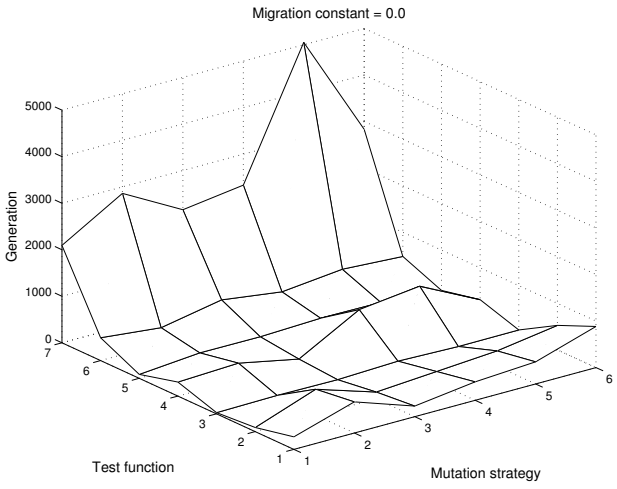

Fig. 2.

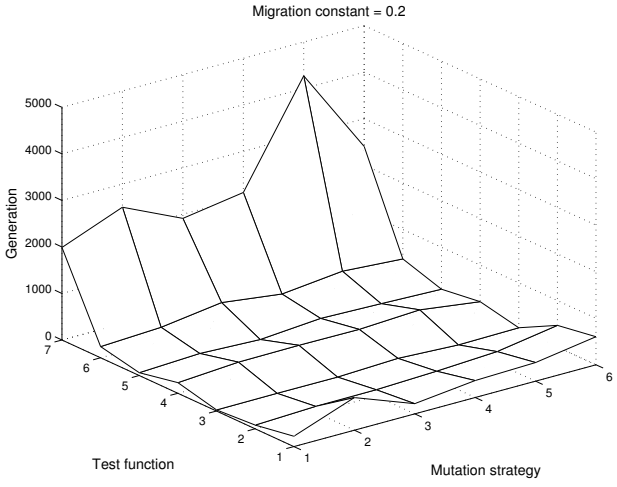

Fig. 3.

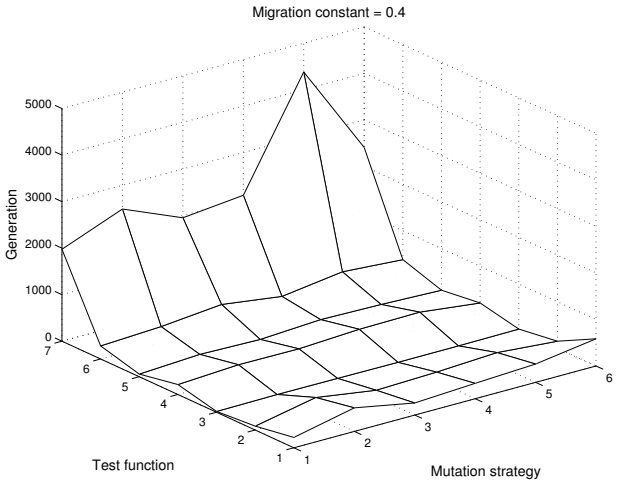

Fig. 4.

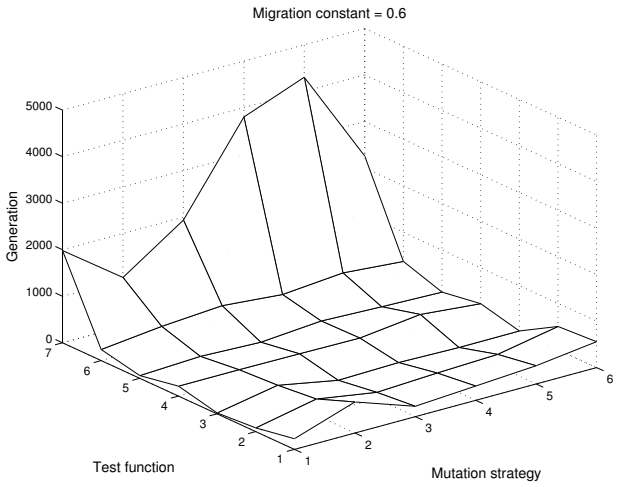

Fig. 5.

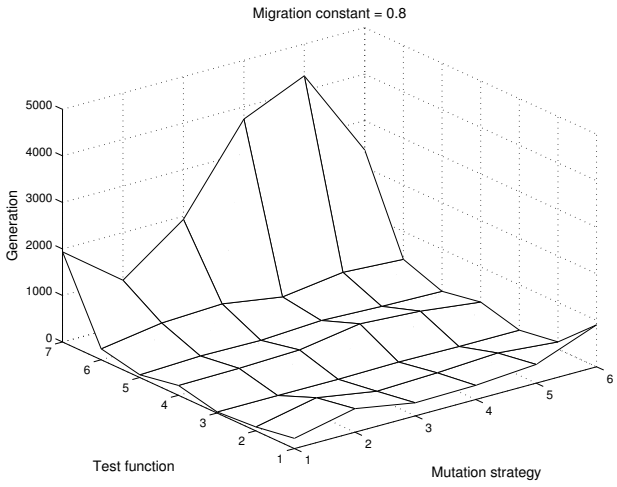

Fig. 6.

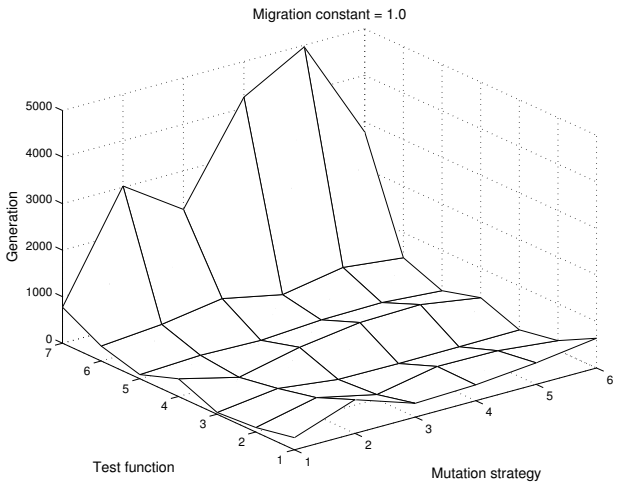

Fig. 7.

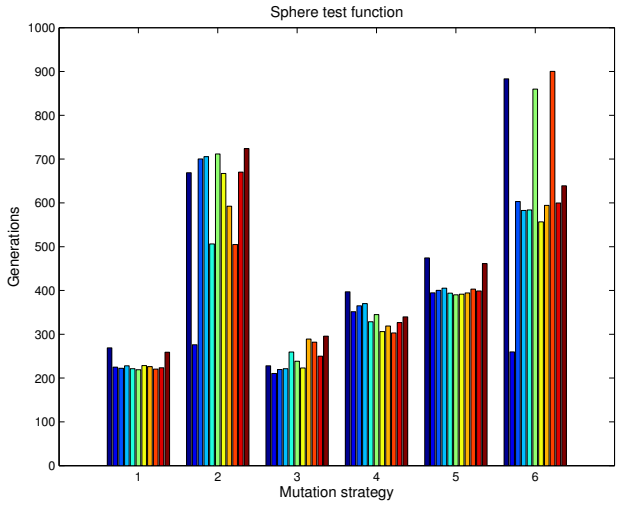

Fig. 8.

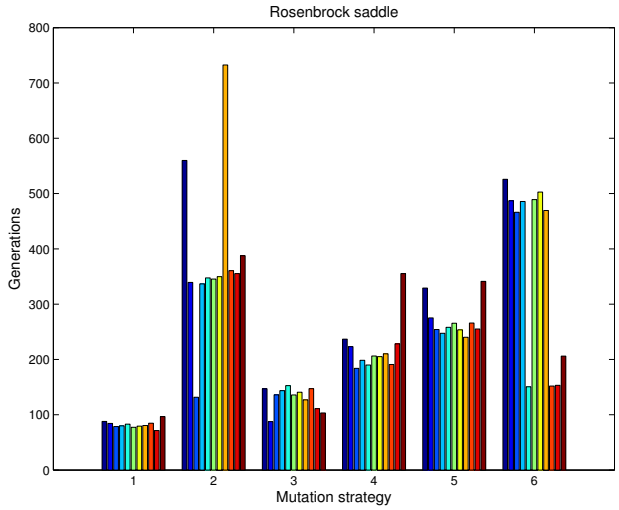

Fig. 9.

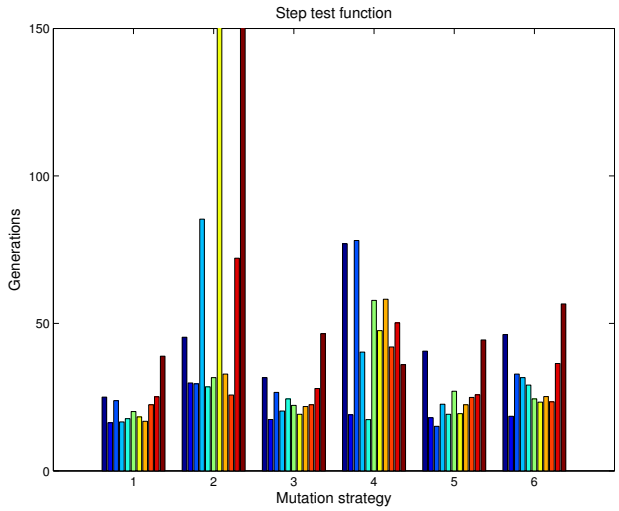

Fig. 10.

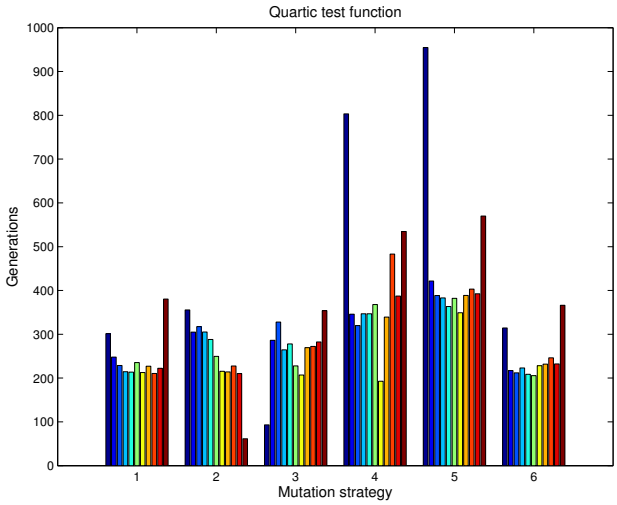

Fig. 11.

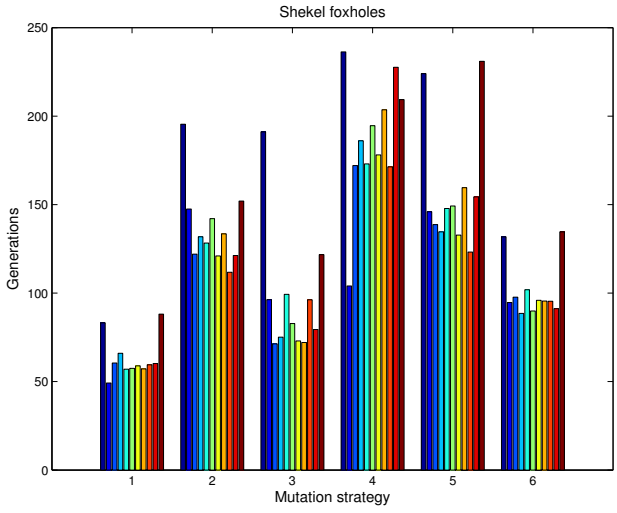

Fig. 12.

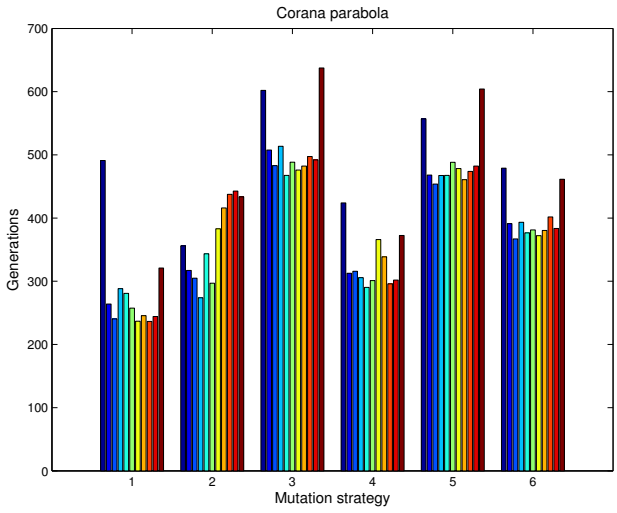

Fig. 13.

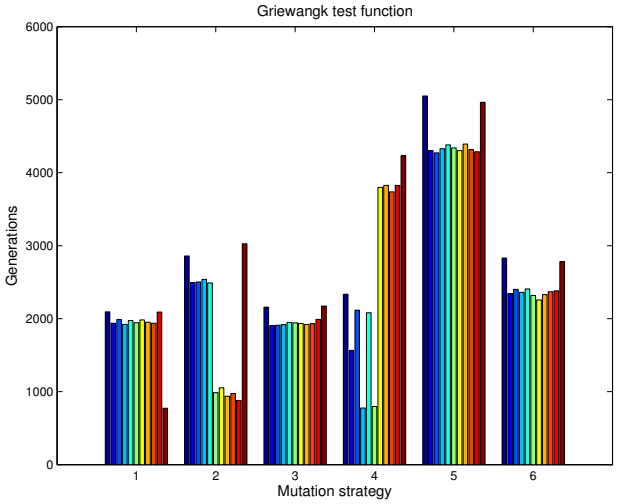

Fig. 14.

assigns each subpopulation, to a different processor node. The topology of the proposed parallel model was a ring. The experimental results obtained suggest that the impact of the migration constant (the factor that determines the extent of information exchange among subpopulations) is significant for the performance of the algorithm. Overall, selecting a migration factor around 0.5 appears to be a good first choice, while setting this factor close to 0 or to 1 is not advisable. The impact of this factor on performance varies across the different mutation strategies employed by the algorithm. The first and third mutation strategies appear to be more robust with respect to this factor, whereas the second, fourth, and sixth mutation strategies are highly sensitive. Fine–tuning the migration constant can produce significant performance improvements. In a future correspondence we intend to investigate alternative topologies, as well as the potential benefits from employing an evolutionary adapted migration scheme along with an implementation of an aging concept for the individuals of the populations [5], [7].

REFERENCES

[1] A.Geist, A.Beguelin, J.Dongarra, W.Jiang, R.Manchek, and V.Sunderam, *Pvm: Parallel virtual machine. a user's guide and tutorial for networked parallel computing*, MIT Press, Cambridge, 1994.

[2] H. Y. Fan and J. Lampinen, *A trigonometric mutation operation to differential evolution*, Journal of Global Optimization **27** (2003), 105–129.

[3] W. Gropp, E. Lusk, and A. Skjellum, *Using mpi*, 2nd ed., Scientific and Engineering Computation, MIT Press, 1999.

[4] Z. Michalewicz and D. B. Fogel, *How to solve it: Modern heuristics*, Springer, 2000.

[5] V.P. Plagianakos and M.N. Vrahatis, *Parallel evolutionary training algorithms for 'hardware–friendly' neural networks*, Natural Computing **1** (2002), 307–322.

[6] H.-P. Schwefel, *Evolution and optimum seeking*, Wiley, New York, 1995.

[7] R. Storn, *System design by constraint adaptation and differential evolution*, IEEE Transactions on Evolutionary Computation **3** (1999), 22–34.

[8] R. Storn and K. Price, *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces*, Journal of Global Optimization **11** (1997), 341–359.