PARALLEL DISTRIBUTED EVOLUTION STRATEGIES

Robertson S. Pereira, Osvaldo R. Saavedra, Omar A. Carmona[a]

Grupo de Sistemas de Energia Elétrica
Departamento de Engenharia de Eletricidade
Universidade Federal do Maranhão
São Luís - MA - 65085-580

Abstract— Evolutionary techniques have been emerged as powerful tools to handle complex optimization problems. However the computational time required may be unacceptable for some kind of applications. The availability of low-cost parallel computer resources and the natural parallelism of Evolutionary algorithm open a important alternative to overcome these difficulties. This paper is devoted to the study of the parallel implementation of evolution strategies and the effect of the parameters on the global efficiency. Results obtained show the beneficial effects in time processing and solution quality. Numerical results with three widely used functions are reported.

Key Words— Parallel processing, Evolutionary Computation, optimization

1 Introduction

Evolutionary Computation encloses different approaches for solving complex problems based on the emulation of mechanisms of natural evolution. Results obtained from application of simulated evolution for solving complex engineering problems have evidenced that search processes based on natural evolution are robust, and can be used in a vast domain variety (Fogel, D.B, 1995).

There are three main approaches where the majority of current implementations are classified: Genetic Algorithms (GA's); Evolution Strategies (ES's); Evolutionary Programming (EP).

Each of these main stream algorithms have clearly demonstrated their capability to yield good approximate solutions even in case of complicated multimodal, discontinuous, non-differentiable, noisy or moving response surfaces of optimization problems (Bäck, T., Hammel, U., Schwefel, H. P., 1997). In these approaches, a population of individuals is initialized and then evolves into a search space, through a stochastic process of selection, mutation, and in some cases, recombination.

The algorithms based on the principles of the natural evolution have been applied successfully to a set of problems of numerical optimization. They are adequate to solve complicated problems of optimization, such as those found in several engineering areas. On the other hand, it is recognized that the main obstacle of the practical applications of evolutionary techniques is processing time. However, evolutionary algorithms are good candidates for parallelization, reducing considerably the processing time by using multiprocessing and in some cases, competitive time are already achieved.

This paper is devoted to the study of the parallel distributed implementation of evolution strategies on a low-cost platform and the effect of the parameters on the global efficiency. Numerical results with three benchmark functions are also reported.

2 Classical Evolution Strategies

Evolution strategies (ES's) were developed in 1960 by Rechemberg and Schwefel in Germany and extended by other authors, such as Rudolph and Herdy (Schwefel, H.P. and Männer, R., 1991). In ES's, an individual is represented as a pair of float-valued vectors, i.e., $a = (x, \sigma)$, where the first vector x represents a point in the search space The second vector σ is a vector of standard deviations. This perturbation vector provides instructions on how to mutate x and is itself subject to mutation. In other words, both components, x and σ, are submitted to evolution process by application of the operators of mutation and also recombination. Thus, a suitable adjustment and diversity of parameter mutations should be provided under arbitrary circumstances.

More recently, two approaches have been explored, denoted by $(\mu + \lambda)$-ES and (μ, λ)-ES (Bäck, T., Hammel, U., Schwefel, H. P., 1997). In the former, μ parents generate λ offspring and all solutions compete for survival with the best μ individuals being selected as parents of the next generation. In the latter, only λ offspring compete for survival and the μ parents are completely replaced in each generation. Then, the life of an individual is limited to a single generation. this method is not elitist, thus facilitating the strategy to accept temporary deterioration that might help to leave the region of attraction of a local minimum and reach a better optimum (Bäck T., Rudolph, G. and Schwefel, H.P., 1997).

[a]Grupo de Sistemas Distribuídos, ICMC - Universidade de São Paulo, São Carlos - SP.

3 Standard ES'S Algorithm

The process of ES's is described in (Bäck T., Rudolph, G. and Schwefel, H.P., 1997). The following pseudocode algorithm summarizes the components of the (μ, λ)-ES evolutionary algorithm, where each individual is characterized by a pair $a = (x, \sigma_i)$:

$t := 0$

1. initialize $P(t) := \{ a_1(0), ..., a_\mu(0) \} \in I^\mu$
   where $I = R^{n+n}$
   and $a_k = (x_i, \sigma_i) \forall i \in \{1, .., n\} \forall k \in \{1, ..., \mu\}$;
2. evaluate $P(t) : \{\Phi(a_1(t)), ..., \Phi(a_\mu(t))\}$
   where $\Phi(a_k(t)) = f(x_k(t)) \forall k \in \{1, ..., \mu\}$;

while termination criterion not fulfilled **do**

3. recombine: $a'_k(t) := r(P(t)) \forall k \in \{1, ..., \lambda\}$;
4. mutate: $a''_k(t) := m\{a'_k(t) \forall k \in \{1, ..., \lambda\}$;
5. evaluate $P'(t) := \{a''_1(t), ..., a''_\lambda(t)\}$;
   $\{\Phi(a''_1(t)), .., \Phi(a''_\lambda(t))\}$
6. select: $P(t+1) := S_d(P'(t))$;

   $t := t + 1$

end do

Where operators $r(.)$, $m\{.\}$ and $S_d(.)$ define the application of recombination, mutation and deterministic selection to the respective arguments. Search points in ES's are n-dimensional vectors $x \in R^n$, and the fitness value of an individual is identical to its objective function value, i.e $\Phi(a) = f(x)$ where x is the object variable component of a and each individual include up to n different variances σ_i $(i \in \{1, ..., n\})$.

3.1 Recombination

In *ES*, new individuals are generated using the mutation and recombination operators. In contrast with genetic algorithms, the recombination operator creates only one offspring. Different recombination mechanisms are used to produce one new individual from a set of randomly selected parent individuals. Basically, recombination works choosing ϱ $(1 \leq \varrho \leq \mu)$ parent vectors from $P(t) \in I^\mu$ with uniform probability. Next, characteristics of ϱ parents are mixed to create a new individual. When $\varrho = 2$, recombination is called bisexual, and if $\varrho > 2$, it is called multisexual. In particular, if $\varrho = \mu$, recombination is called global (Bäck, T., Hammel, U. , Schwefel, H. P, 1996). On this class, there are two variants:

- Global discrete recombination: This variant is similar to uniform crossover in genetic algorithms. Each component of an offspring is created by selecting at random an individual from parent population.
- Global intermediary recombination: Each component of an offspring is generated by the arithmetic average of the corresponding parent components, as follows:

  $b'_i = (1/\varrho) \sum_{k=1}^{\varrho} b_{k,i}$

  where b'_i is i-component of the offspring b'.

Notice that recombination is performed on strategy parameters as well as on the object variables, and the recombination operator may be different for object variables and standard deviations.

3.2 Mutation

The mutation operator $m : I \rightarrow I$ (where $I = R^{n+n}$) yields a mutated individual $m(\vec{a}) = (\vec{x}', \vec{\sigma}')$, by first mutating the standard deviations and then mutating the object variables as follow:

$$\sigma_i'^j = \sigma_i^j exp(\tau' N(0,1) + \tau N_j(0,1)) \qquad (1)$$

$$x_i'^j = x_i^j + \sigma_i'^j N(0,1) \qquad (2)$$

where $i = 1, ..., \lambda$ and $j = 1, ..., n$. $N(0,1)$ represents a Gaussian number with zero mean and variance 1, that should be the same for all vector positions. $N_j(0,1)$ also represents a Gaussian number; nevertheless, this value must be different for each j value.

The global factor $\tau' N(0,1)$ allows for any overall change of the mutability, whereas $\tau N_i(0,1)$ allows for individual changes of σ_i. The factors τ and τ' are defined as "learning rates" and are suggested by Back (Bäck T., Rudolph, G. and Schwefel, H.P., 1997) as $\tau = (\sqrt[4]{4n})^{-1}$ and $\tau' = (\sqrt{2n})^{-1}$, respectively, where n represents the problem size.

Cauchy Mutation.- The evolution strategy with Cauchy-based mutation follows the same general algorithm above, except that the eq. (2)) is replaced by:

$$x_i'^j = x_i^j + \sigma_i^j \delta_j \qquad (3)$$

where δ_j is random number with Cauchy distribution with scalar parameter setting as $t = 1$. This number must be obtained for each value of j.

Due to Cauchy distribution be more expanded than Gaussian distribution, it allows, probabilistically speaking, larger mutations and in this way, generating more different individuals (Nogueira, M. L., Saavedra, O. R., 1999).

3.3 *Selection*

In contrast with evolutionary programming, selection in classical *ES* (S_d) is completely deterministic (Fogel, D.B., 1995). In case of $(\mu + \lambda)$-*ES*, the μ best individuals are selected from the union of parents and offspring. Thus, this selection is elitist and therefore guarantees a monotonic improving performance. In (μ, λ) strategies, the μ best individuals are selected only from offspring population and replace the parents in the next generation (not elitist selection).

4 Parallel Distributed ES'S

Evolutionary algorithms are good candidates for parallelization. There are several approaches in the technique literature, that can be separated into two categories:

- Diffusion model;
- Migration model.

In the diffusion model, each individual is placed on a single processor and selects another individual to share information in order to generate a new trial solution point. This model is well suited for SIMD machines. Numerical results seem to indicate that algorithms using the diffusion model perform better in average. In (Rudolph, G, 1992) is implemented a parallel ES using diffusion model and applied it to global optimization problems. The efficiency improves as soon as the population size increases or function evaluations become more expensive, so that more time must be spent upon one iteration.

In the Migration model, one sequential version is allocated on each processor and exchange information between the processors during the search. Suppose that there are λ individuals forming on each processor, the exchange of information can be regarded as migration of individuals, where typically the best individuals are exchanged. Several variations of algorithms can be derived, depending of the number of individuals exchanged, when this exchange is executed, etc. Due to the parallel version be other algorithm as the sequential one, it is possible to obtain efficiencies above 100%.

This paper is concentrated on the study and implementation of the parallel evolution strategies using the migration model. An important reason is the exploitation of the significant asynchronism level obtained from this paradigm.

4.1 *The Motivation for Parallel Distributed Processing*

The local network utilized was a cluster formed by 5 PC's, connected by a switch Fast-Ethernet and running Linux Slackware. The platform PVM version 3.4.3 allows a network of parallel and serial machines to be viewed as a unique concurrent computational resources. In this way, this platform provides, with very low cost, a parallel distributed environment that offers the best features of sequential and parallel processing, due to the availability of high-speed processors and to natural scalability.

The PVM provides the functions to automatically start up tasks on the parallel virtual machine and allows the information interchange among theses tasks. Also, the available PVM version has host failure detection. If a host fails, the PVM system will run continuously and will automatically delete this host from the virtual machine. It is still the responsibility of the application developer to make his application tolerant to host failure (Geist, A.. et al, 1994). The PVM provides non-blocking asynchronous routines for sending and receiving messages; these routines are especially suited for the asynchronous approach implemented here.

4.2 *Implementation of the Migration model*

There are several ways how to implement the migration process. We suggest three ways in the following.
After k generations:

a) each processor send the γ best individuals to the others processors. In the sequence, receives individuals sent for the other processors, selects the γ best individuals and substitutes the γ poor individuals of the local population.

b) Each processor send the γ best individuals to the master processor. It controls the overall process by receiving the best individuals at each *epoch*, selecting the γ best individuals of this epoch and sending it to all processors.

   When no individual is arrived, the master processor executes a copy of evolutionary code, evolving it population similarly to others processors.

c) Similarly to (b), but considering only single individual for interchange, i.e. $\gamma = 1$.

In this work, we have analyzed the option (b). In order to observe the relationship between γ and the convergence of algorithm, tests varying the size of γ have been performed and are reported in the next section. In figure (1) the conceptual parallel model is illustrated. In each processor is allocated a copy of evolutionary algorithm. Additionally, the master processor hosts tasks related with the global control of the parallel process and send/receive the best individuals to/from others

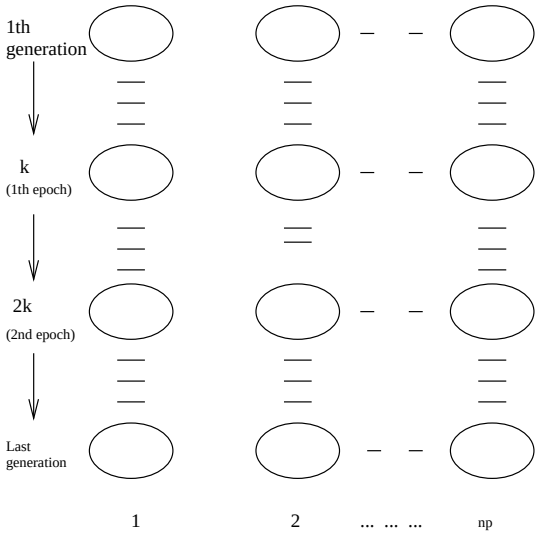

Figure 1. Migration model: conceptual parallel model

processors. At each epoch, slave processors send the best individual to the master processor. As soon it receives individual from all processors, the master chooses the best individual and send it immediately to all processors.

In the slave processor the individual received from the master processor substitutes the worst individual in the local population. This process is made asynchronously, i.e., slave processors do not remain waiting data in the buffer. By the opposite, they keep evolving, and alternately, check the input buffer searching for new data. Thus the epoch is then marked by sending the best individual, but the reception and inclusion of individual is done as soon as they are available.

5 Test Results

In the following, the test results of $(\mu + \lambda)$ parallel evolutionary strategy are illustrated using three well-known functions used in the references (Yao, Xin and Li, Yong, 1997), (Nogueira, M. L., Saavedra, O. R., 1999). The evolution strategy implemented is based on Cauchy mutation rather than the traditional Gaussian mutation. The platform used has been a PC-based Linux cluster running PVM version 3.4.3.

The test functions used are complex multidimensional multimodal functions, with several local minima, and are given by the following expressions:

$$f_1 = \sum_{i=1}^{n} -x_i sin(\sqrt{|x_i|}) \tag{4}$$

$$f_2 = -20exp\left(-0.2\sqrt{u(x)}\right) - exp\left(v(x)\right) + 20 + e \tag{5}$$

with:

$$u(x) = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

and:

$$v(x) = \frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)$$

$$f_3 = \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{6}$$

Functions (4 - 6) have been chosen due to similarity with formulation problems found in power systems (Gomes, J. R. and Saavedra O. R., 1999). It is interesting to make use of this work and to export some concepts and approaches to improve current evolutionary applications in this area.

5.1 Parameters

The population size used was $\mu = 30$, while offspring population size was $\lambda = 200$. The tests have been performed over 10 simulations for each function. For all cases, the problem dimension has been fixed in $n = 30$. The constraints for x variables are given by:

f_1: [-500;500];

f_2: [-32;32];

f_3: [-600;600];

In the parallel implementation, the number of generations of serial version was divided by the number of processors. Migration is executed each $k = 100$ generations. In all the case, the number of generations of serial version was 5000.

In figures (6) is presented, for each function, the effect of the number of individuals interchanged at each epoch, considering five processors. The stochastic behaviour allows only to analyse the trend of the phenomenon, that basically indicates that small γ tax are, in average, more appropriate for good performance, because the communication between processor is small. In this way, it is expected a most asynchronous behavior from algorithm.

Serial Version			
f	Mean Best	Std Dev	Time(s)
f_1	-12095.73327	118.43833	155.95
f_2	0.089115	0.3733721	195.81205
f_3	0.0036980	0.00370	170.07

Table 1. Average performance of serial algorithm.

Table (1) shows results obtained from serial version using Cauchy mutation rather than the classical Gaussian Mutation (Nogueira, M. L., Saavedra, O. R., 1999). Columns $3-5$ show the mean best obtained over 10 runs, the associated standard deviations and the CPU time in seconds,

respectively. The the minimum values f^{min} of each test functions are -12569.5, 0 and 0, respectively

Parallel Version						
f	np	Mean Best	Std Dev	Time (s)	S	E (%)
f_1	2	-12569.48	1.0e-4	78.76	1.98	99.00
	3	-12569.48	3.6e-4	50.79	3.07	102.34
	4	-12569.48	1.3e-4	37.78	4.12	103.19
	5	-12569.48	4.0e-5	31.65	4.92	98.4
f_2	2	0.08921	2.e-4	97.02	2.01	100.91
	3	0.08948	2.e-4	63.03	3.10	103.55
	4	0.10097	0.0038	50.44	3.88	97.05
	5	0.11798	0.0484	40.69	4.81	96.24
f_3	2	2.5e-8	1.25e-5	83.91	2.02	101.34
	3	6.95e-7	2.22e-05	55.63	3.05	101.90
	4	1.15e-5	1.95e-5	42.91	3.96	99.08
	5	0.00063	6.75e-4	32.24	5.27	105.50

Table 2. Average performance of parallel distributed algorithm

Table (2), shows results obtained using the parallel distributed algorithm. Column np indicates the number of processors used, and columns 6-7 show the speed-up (S) and efficiency (E) obtained, respectively. The γ tax used was 6, 6, and 1 for f_1, f_2 and f_3, respectively. Due to the stochastic characteristic of algorithm, in some cases a superlinear performance is observed. In general, good solutions, efficiencies and speed-ups are reached using parallel processing. The beneficial impact of the parallel processing is observed mainly in the processing time and the quality of the obtained solutions. These results are important, because the main obstacle of the application of evolutionary techniques to engineering problems is the CPU time required to reach competitive solutions. Although parallel computers be expensive, it is possible get a low-cost parallel distributed platform using a PC-based cluster. In this way, parallel processing stops being a myth and becomes an all users tool.

6 Conclusions

In this work the study of the parallel distributed implementation of evolution strategies has been presented. Results obtained using a migration model show the beneficial effects in time processing and solution quality. Numerical results using three widely used functions have been reported, showing the beneficial effect of parallel processing in the solution quality. The parallel processing is achieved by using a low-cost platform formed by PC-based Linux cluster running PVM version 3.4.3.

Acknowledgments

This project was supported by CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico and BNB, Banco do Nordeste do Brasil.

References

Fogel, D.B. (1995), "A Comparison of Evolutionary Programming and Genetic Algorithms on Selected Constrained Optimization Problems", Simulation , pp. 397-404.

Bäck, T., Hammel, U. , Schwefel, H. P. (1996) "Evolutionary Computation: an Overview", in Proc. 3rd IEEE Conf. on Evolutionary Computation, Piscataway, NJ, IEEE Press, pp. 20 - 29.

Schwefel, H.P and Rudolph, G. (1995), "Contemporary Evolution Strategies", Proceedings of European Conference on Alife, Granada, Spain.

Bäck, T., Hammel, U., Schwefel, H. P. (1997), "Evolutionary Computation: Comments on the History and Current State", IEEE Transaction on Evolutionary Computation, Vol 1, No 1, pp. 3 -17.

Yao, Xin and Li, Yong (1997), "Fast Evolution Strategies", Control and Cybernetics, 26(3):467-496.

Nogueira, M. L., Saavedra, O. R. (1999), "Multimodal Optimization using Standard Evolution Strategies with Cauchy Mutation", Proceedings of the 1999 North American Power Symposium, pp. 283-288, EEUU.

Geist, A., Beguellin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V. (1994), "PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing", The MIT Press.

Rudolph, G (1992), "Parallel Approaches to Stochastics Global Optimization", in: W. Joosen and E. Milgrom (eds.): Parallel Computing: From Theory to Sound Practice, Proc. of the European Workshop on Parallel Computing, pp. 256-267, Amsterdam.

Schwefel, H.P. and Männer, R., Eds. (1991), "Parallel Problem Solving from Nature", Proc. 1st Workshop PPSNI, Berlin, Germany, Vol. 496 of Lecture Notes in Computer Science, Springer.

Gomes, J. R. and Saavedra O. R.(1999), "Optimal Reactive Power Flow Using an Extended Evolution Strategy", in: Computational Intelligence and Applications, N. Mastorakis (ed.), World Scientific and Engineering Society Press, pp. 230-238.

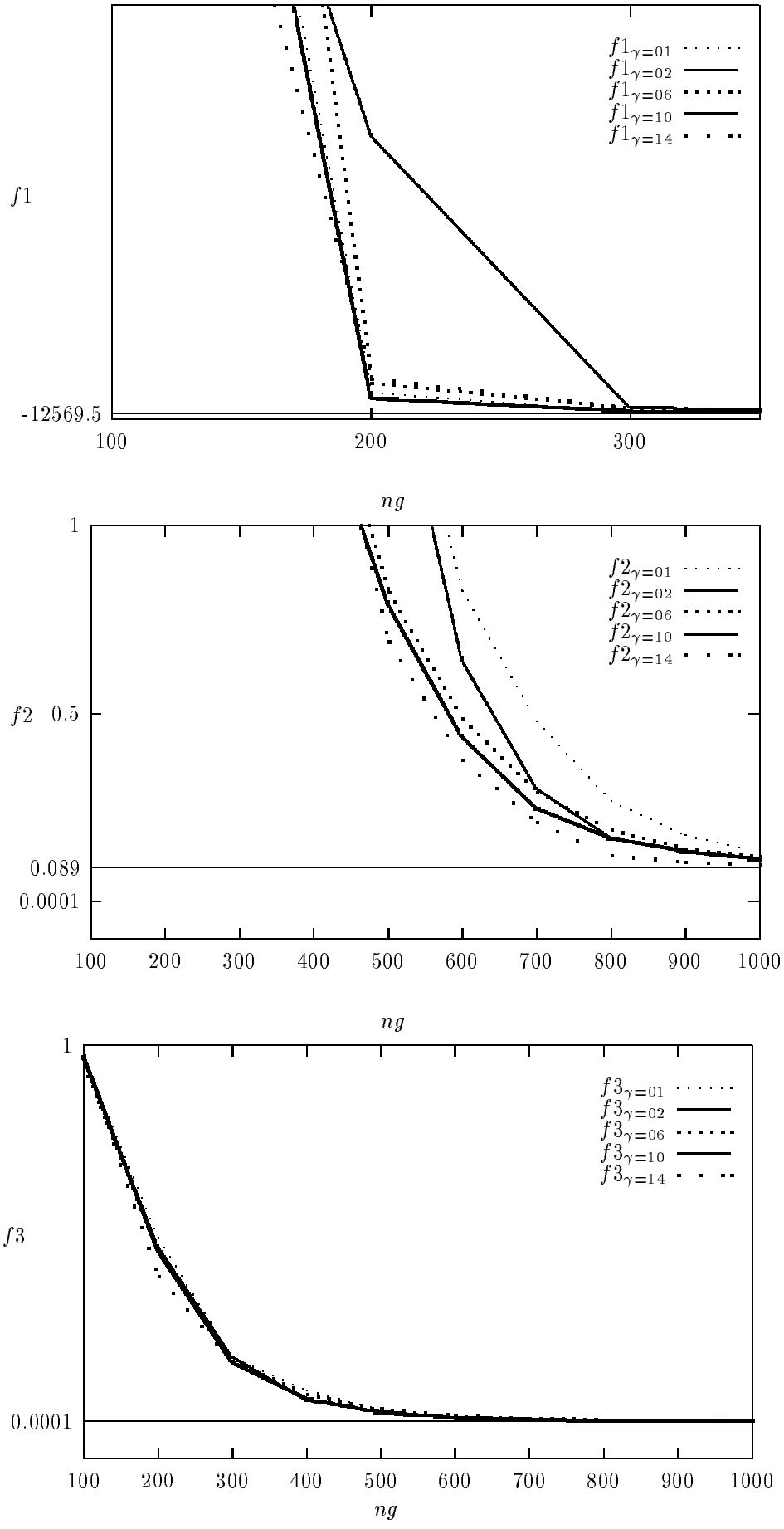

Figure 2. The effect of γ tax on the convergence, using 5 processors.