A New Asynchronous Parallel Evolutionary Algorithm for Function Optimization[1]

Pu Liu[1], Francis Lau[2], Michael J. Lewis[1] ,Cho-li Wang[2]

[1] Department of Computer Science, Binghamton University – SUNY,
Binghamton, N.Y., 13902, USA
{pliu1,mlewis}@binghamton.edu
[2] Department of Computer Science and Information Systems,
The University of Hong Kong, Pokfulam Road, Hong Kong
{fcmlau,clwang}@csis.hku.hk

Abstract. This paper introduces a new asynchronous parallel evolutionary algorithm (APEA) based on the island model for solving function optimization problems. Our fully distributed APEA overlaps the communication and computation efficiently and is inherently fault-tolerant in a large-scale distributed computing environment. For the scalable BUMP problem, our APEA algorithm achieves the best solution for the 50-dimension problem, and is the first algorithm of which we are aware that can solve the 1,000,000-dimension problem. For other benchmark problems, our APEA finds the best solution to G7 in fewer time steps than [16,17], and finds a better solution to G10 than [17].

1 Introduction

Evolutionary Algorithms (EAs) imitate nature's evolutionary process to solve complex problems. Their population-based searching mechanism makes them eminently suitable to be run in parallel on a massive scale. Traditionally, parallel EAs follow one of three models: the *master-slave* (or farming) model, the *island model*, or the *cellular model*. The island model is most popular [4] since it is easy to implement on a local area network with standard workstations. In the island model, the population is divided into subpopulations called *demes*, and each evolves separately on different processors with local memory. With some migration frequency, the demes exchange individuals to prevent premature convergence of subpopulations. This exchange requires communication in implementations of the model.

When implementing the island model, most algorithms adopt a synchronous approach [4]. That is, demes synchronize with one another when migration occurs and individuals are exchanged. The intervals of time between migrations are called *epochs* [4], and all EA processes stop at the end of each epoch to wait for all the others. This

[1] This work was partially supported by National Science Foundation (NFS) Instrumentation Grant EIA9911099.

barrier synchronization after each epoch results in considerable overhead, and allows the slowest processor to determine the speed of the algorithm. When loads are not balanced on all processors, especially when the number of processors is large, synchronous algorithms often perform poorly.

Evolution in nature is an asynchronous process. Individuals migrate independently from one population to another without central control or full coordination between populations. This is also the most efficient way to overlap the computation and communication in a parallel algorithm. But most parallel EAs are synchronous [1-7], presumably because of the seemingly complicated exchange of individuals. The few asynchronous algorithms require a central server to manage exchange of individuals [8], which also limits scalability and introduces additional software overhead.

In this paper, we introduce a new *asynchronous parallel evolutionary algorithm* (*APEA*), based on the island model, in which all communication operations are non-blocking. Each processor executes without waiting for network communication between epochs. In contrast to the synchronous algorithms, which scale poorly, our APEA achieves improved performance with additional processors. In addition, APEA is inherently fault-tolerant to failures in underlying computing environments. This is especially important for time-consuming computational problems.

We have implemented our APEA in both PVM [18] and MPI [19]. Using the PVM version of APEA, we solved a scalable optimization BUMP problem [10-16]. The extreme high dimensional BUMP problem allows us to test both the scalability and efficiency of APEA. In the literature, the largest BUMP problem solved is the 50 dimension problem [10]. Our APEA achieves a better solution for the 50 dimension BUMP, and finds a solution for the 1,000,000 dimension BUMP using a 256-node YH-4 MPP supercomputer. With the MPI version of APEA, we tested some widely used benchmark problems, namely G7 [16,17], and G10 [16,17]. Our APEA achieves the best solution for each, and for G10, our APEA finds a solution that is better than the best solution claimed in the literature [17].

The rest of this paper is organized as follows. Section 2 introduces the function optimization problem briefly, then describes our asynchronous parallel evolutionary algorithm and discusses its properties and our approaches to improve convergence characteristics. In the following sections, results of our solution for several benchmark problems are discussed. Concluding remarks are provided in the final section.

2 Asynchronous Parallel Evolutionary Algorithm

2.1 Background

Many real world problems involve complex, non-linear, multi-constraint, mixed format (e.g. integers and floating point numbers) aspects that conventional algorithms cannot solve accurately or in reasonable time. Our goal is to solve numerical optimization problems, which are characterized by what are called the “3 S’s”, namely (*s*uper non-linear, *s*uper multi-peaked and *s*uper dimensional).

We solve the following general function optimization problem:

$$\max f(X) \qquad X = (x_1, x_2, \ldots, x_n)^T \in R^n$$

subject to

$$l_i \le x_i \le u_i, \quad l_i \in R, \quad u_i \in R, \quad i = 1,2,\ldots,n$$

$$g_i(X) \le 0, \quad i = 1,2,\ldots,q$$

2.2 The New Algorithm

Assuming a subpopulation with N individuals is assigned to each of NPROC processors, each processor executes the same process G, as follows, to steer the asynchronous parallel computation:

```
PROCESS G
1. t=0
2. Initialize P(t)={$X_1$, $X_2$, …, $X_n$}
3. Evaluate P(t)
4. While (not terminated) do
5.    If  (any  message arrived) then
6.        $X_{new}$ = Recv_Individual()
7.    else  $X_{new}$ = Local_Generate()
8.    P'(t) = P(t)$\cup${$X_{new}$}
9.    P(t) = select N best individuals from P'(t);
10.   Locate $X_{best}$ in P(t);
11.   If (t mod T = 0) and ($X_{best}$ changed)  then
12.       Send $X_{best}$ to Q other process(es)
13.   t = t+1
14. END WHILE
```

Line 5 checks the receiving buffer to see if a new message has arrived. This operation can be implemented by the “probe” communication primitive, which is provided by both PVM and MPI. Due to the non-blocking communication of Line 5, along with Line 12, the processes never wait for one another. That is, they run completely asynchronously, and the slowest process does not slow down the others. Thus, there is no need for a load-balancing mechanism. Furthermore, APEA is inherently fault-tolerant because it will continue to execute even when some nodes fail. This significant characteristic is especially important for large and time-consuming computational problems.

We also note that if the incoming individual in Line 6 is eliminated through selection in Line 9, then the immigration is useless. Because communication between processors is expensive, we should avoid this situation if possible. Line 12 ensures that only the individual with the best fitness migrates to the other demes. Likewise, Line 9 ensures that the worst individual is eliminated in each iteration. This allows the immigrated individual to have a high probability of surviving in its new deme.

The algorithm is SPMD; all processors run the same process G, and neither central control nor synchronous communication is required. This makes APEA scalable and suitable for solving complex problems via MPP supercomputers.

To be appropriate for parallel systems that are both loosely and tightly coupled, APEA contains two parameters to control the parallel granularity of APEA. Parameter Q indicates the number of processors with which one processor communicates when a new best individual is found within its deme. Parameter T determinates the frequency of individual migration. Together, Q and T determinate the total communication cost during each iteration of APEA. Thus, the parallel granularity and communication costs can be adjusted dynamically by altering the values of these parameters.

Unfortunately, the effects of these parameters on the quality of solutions are not well understood, especially in the asynchronous mode. Cantu-Paz has presented models that predict the effects of the parameters on the population size of demes and on migration rate [4]. However, this investigation considered that migration occurs only after each population converges, which assumes a synchronous algorithm. We discuss the setting of parameters based on the experimental results in Section 3.

Evolutionary programming and genetic algorithms are also applicable to produce a new offspring X'. For efficiency, our Local_Generate () method uses a multi-parent crossover [9]. The offspring X' is generated by parents X_i', i = 1, 2, … , m as follows.

$$X' = \sum_{i=1}^{m} a_i X_i'$$

where the coefficients a_i are chosen randomly and subject to

$$\sum_{i=1}^{m} a_i = 1, \quad -0.5 \le a_i \le 1.5, \quad i = 1,2,...,m$$

To solve constrained optimization problems, Michalewicz presented a comparison study of existing evolutionary algorithms [10]. According to this study, the existing algorithms are grouped as follows: (1) methods based on preserving feasibility of solutions, (2) methods based on penalty functions, (3) methods based on the superiority of feasible solutions, and (4) other hybrid methods. We choose the third method for its simplicity of implementation.

Frequently used termination criteria for EAs include running the algorithm until: (1) a satisfactory solution is found; (2) a fixed time limit is reached; or (3) all the individuals in the population are the same and no further improvement is possible. Our APEA uses the latter criterion. When individuals in a deme are all the same, its process broadcasts a 'termination' message and APEA terminates.

2.3 Related Work

There are many algorithms that explore alternative migration schemes and communication models to try to make parallel genetic algorithms (Ga's) more efficient. Muhlenbein [20] proposed a totally asynchronous cellular PGA, which runs on MIMD parallel computers. In this algorithm, individuals are distributed in a 2-D world. Each individual selects a partner for mating in its neighborhood. The active and intelligent individuals decide when to migrate and control the inter-processor communication. Hart [21] and Alba [22] applied this idea to a coarse-grain PGA whose subpopulations are distributed across *p* processors in a two-dimensional grid. Both report that the asynchronous algorithms outperformed their equivalent synchronous counterparts in real time. In this kind of "active" asynchronous PGA, when migration occurs, at least two messages, a request and a response, are needed. In most cases, communication-fault-handling is also needed to accomplish the migration of one individual. Our APEA uses a larger population of passive individuals. The migration of one single individual only uses one message to send the individual. The extra overhead is the probe, which doesn't incur much overhead.

Grefenstette [23] proposed three variations of the master-slave PGA with the one population maintained by the master, and use synchronous communication. Based on master-slave method, Martin [24] introduced a centralized coarse-grain PGA in which the migration is fully asynchronous. One drawback of this algorithm is its scalability, which is limited by its centralization, which we have eliminated in our APEA.

3. Numeric Experiments

The APEA is implemented by PVM and MPI under different hardware environments. The BUMP problem was solved via a MPP supercomputer YH-4 by PVM, and the other problems were solved on a 16-node PC cluster connected by a 100Mb/s Fast Ethernet switch. Each cluster node consists of a 300MHz Pentium II processor with 128MB of memory, running Linux 2.2.14 with MPICH 1.2.1.

When implemented by MPI, we used functions MPI_Iprobe(), MPI_Recv() and MPI_Isend(). The blocking MPI_Recv() is used only after MPI_Iprobe() returns "true". Both MPI_Iprobe() and MPI_Isend() are non-blocking. For the PVM version, we used similar functions pvm_probe(), pvm_recv() and pvm_mcast().

3.1 The BUMP Problem

The BUMP problem is described in [11] as follows:

$$f_n(X) \equiv \frac{\left|\sum_{i=1}^{n}\cos^4(x_i) - 2\prod_{i=1}^{n}\cos^2(x_i)\right|}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \qquad (1)$$

where $0 < x_i < 10$, i = 1, 2, 3, …, n (2)

subject to $\prod_{i=1}^{n} x_i > 0.75$ (3)

and $\sum_{i=1}^{n} x_i < 7.5n$ (4)

In 1994, Keane proposed the BUMP problem [11] in optimum structural design, and subsequently published several papers [12, 13, 14] discussing the problem. Because of its scalability, the BUMP problem has been used as a general benchmark problem to test optimization algorithms [16].

There are several potential difficulties in solving this problem. First, the objective function $f_n(X)$ is a nonlinear multimodal function of high dimension. Second, the global optimum is defined by the nonlinear constraint (3) above. When the dimension of the BUMP problem increases, the product in (3) is difficult to calculate (because of overflow). In fact, when *n* is greater than 20, a majority of constraint-handling methods have difficulty returning a high quality solution [9, 15, 16].

EI-Beltagy used metamodeling techniques for the BUMP problem and got some numerical results when n=2 and n=5 [15]. By incorporating some problem-specific knowledge, Michalewicz introduced an evolutionary operation called geometrical crossover and reported in [16] that when n=20 the best solution achieved was 0.803553 and when n=50, the best solution was 0.8331937. For n=20, the result Tao Guo reported in [9] is 0.803619. But when n is greater than 50, there is no published result of which we are aware. The expensive computation stunts sequential algorithms. Because evolutionary algorithms are inherently parallel, if we want to solve high dimensional BUMP problem, an obvious promising way is to parallelize the sequential algorithm.

In addition, for a solution vector $X(x_1, x_2,..., x_n)$, the value of the numerator does not change no matter the order of $x_1, x_2,..., x_n$. However, the value of the denominator may change only when the order of $x_1, x_2,..., x_n$ changes. In fact, the value of $f_n(X)$ achieved by a solution vector $X(x_1, x_2,..., x_n)$ is no better than the one achieved by the vector $X^*(x^*_1, x^*_2,..., x^*_n)$ where $x^*_1, x^*_2,..., x^*_n$ is the non-increasing sequence of $x_1, x_2,..., x_n$. So we add a mutation operation sort() to function Local_Generate().

Recalling the nonlinear constraint (3), the sorting process is necessary when *n* is fairly great, i.e. 400, because a product of 400 random numbers ranging from 0 to 10 is liable to overflow or underflow during calculation, even though the final product does not overflow or underflow. For example, if the first 100 random numbers are greater than 2, then the temporary product will be greater than 2^{100} or less than 2^{-100} both of which are far beyond the capability of most architectures. Fortunately, for the descending sequence vector $X^*(x^*_1, x^*_2,..., x^*_n)$, we can use an algorithm to calculate the product and avoid the above situation in many cases. The algorithm multiplies a running product by a small value then the product is in danger of overflowing, and multiplies in a large value when it is in danger of underflowing.

According to the research of Michalewicz [16], the constraint (4) has no effect on the optimal results, so we also ignore constraint (4) in our experiments.

To verify the efficiency and effectiveness of the APEA, we have completed numerical experiments for *n* = 50, 100, 200, 300, 400, 500, 1500.

Table 1. Results for n = 50 to 1500

Dimension	50	100	200	300
Best Result	0.8352620	0.8448539	0.8468442	0.8486441
Dimension	400	500	1500	
Best Result	0.8511074	0.8504975	0.8449622	

From Table 1, when *n* equals 50, we achieved the best results of which we are aware. In order to compare with other algorithms, we display the whole solution here. F(X) is 0.83526201238794 with $\prod_{i=1}^{50} x_i$ = 0.75000291468818, where

X = {6.28324314967593, 3.16959135278874, 3.15587932289134, 3.14221834166447, 3.12875141132280, 3.11533720178037, 3.10135016319809, 3.08867068084463, 3.07467526362337, 3.06187302198439, 3.04886636373126, 3.03519777233540,

3.02235650101037, 3.00791087929595, 2.99431484723465, 2.98078207510642,
2.96617535809182, 2.95213454834831, 2.93826263614623, 2.92347051874431,
0.48770508444941, 0.48608797212231, 0.48388281752034, 0.48168154669018,
0.47901882085524, 0.47712972009898, 0.47501098097910, 0.47259800785213,
0.47128211104035, 0.46871206605219, 0.46726830592743, 0.46606878481357,
0.46376534354877, 0.46204633230610, 0.45965521445116, 0.45884822857585,
0.45717243025277, 0.45481532957423, 0.45342381481910, 0.45166163748202,
0.45021480667760, 0.44845064437796, 0.44673980402160, 0.44526519733653,
0.44338085480658, 0.44213030342910, 0.44043946842174, 0.43904935652294,
0.43874135178293, 0.43634175968446}

We set out to determine, via a state-of-the-art MPP supercomputer, the highest dimension BUMP problem we could solve by APEA. We had to speed up the convergence rate without compromising the quality of solution. Parameters T and Q affect both aspects, but unfortunately, it is difficult to predict their exact effects. We performed two groups of experiments to find effective values. The first group observes Q with fixed *n* and T, while the second group observes T with fixed *n* and Q. Table 2 contains parts of our experimental results.

Table 2. Results of APEA with NPROC=128, n=1000

T = 200			Q = 2		
Q	Iterations	Results	T	Iterations	Results
2	14653	0.84421	100	11830	0.8410597
2	14115	0.84421	100	8867	0.8410594
4	5187	0.83924	200	14653	0.8442193
4	4998	0.83889	200	14115	0.8442193
6	3130	0.83376	300	12095	0.8448454
6	3996	0.84218	300	12389	0.8448539
8	30169	0.84193	400	11329	0.8410598
8	28877	0.84259	400	11720	0.8421879

We observe that as communication increases, the algorithm converges and the solution degrades. Based on Table 2, we choose Q=10 and T=300 and get the following numerical results in Table 3:

Table 3. Results of APEA with NPROC=256, Q=10, T=300

	N	Terminated t	Time (s)	Solutions
1	10000	20072	313	0.8455810
2	10000	16385	260	0.8456407
3	20000	24236	382	0.8452293
4	20000	22169	344	0.8455882

When n = 100,000, NPROC=128, APEA converged with t = 46252 and solution 0.8448940. The total execution time was 7106 seconds. When n=1,000,000, NPROC=128, we terminated APEA after 16 hours calculation with t=6027 and result 0.8445861. For n=10,000,000, system resources were exhausted.

3.2 Benchmark Problems Solved by APEA Using MPI

1. G7 [16][17]: Minimize

$$f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2$$
$$+ 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1) + 5x_7^2$$
$$+ 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

subject to:

$$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0$$
$$-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$
$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$
$$-x_1^2 - 2(x_2 - 2) + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$
$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$
$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$$
$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$$
$$-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$
$$-10 \leq x_i \leq 10, i = 1,2,...,10$$

The global minimum is known to be f(X*) = 24.3062091 [16][17], where

X* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548,
1.430574, 1.321644, 9.828726, 8.280092, 8.375927)

The APEA always achieves the optimal solution with fewer iterations. Table 3 provides the best results under different parameters. The Algorithm CEALM in [17] and other algorithms listed in [17] do not reach the global optimal every time. APEA achieves the optimal value for each of the ten separate runs.

Table 3. The results of G7 with NPROC = 16

NPROC	T	O	Median time(s)	Result
1	0	0	39	24.3062090690560
2	1	2	18	24.3062090685401
4	30	4	21	24.3062090683975
8	30	8	28	24.3062090683429
16	1	16	43	24.3062090683032

Table 3 shows that increasing communication improves the quality of the solution on average but requires more time to converge.

2. G10 [16][17]: Minimize $f(x) = x_1 + x_2 + x_3$

subject to

$$
\begin{aligned}
&1-0.0025(x_4+x_6)\geq 0\\
&1-0.0025(x_5+x_7-x_4)\geq 0\\
&1-0.01(x_8-x_5)\geq 0\\
&x_1x_6-833.33252x_4-100x_1+83333.333\geq 0\\
&x_2x_7-1250x_5-x_2x_4+1250x_4\geq 0\\
&x_3x_8-1250000-x_3x_5+2500x_5\geq 0\\
&100\leq x_1\leq 10000\\
&1000\leq x_2\leq 10000\\
&1000\leq x_3\leq 10000\\
&10\leq x_i\leq 1000, i=4,5,...,8
\end{aligned}
$$

The global minimum is said to be f(X*)=7049.330923 [17], where

X* = (579.3167, 1359.943, 5110.071, 182.0174,295.5985, 217.9799, 286.4162, 395.5979)

The algorithm CEALM in [17] and other algorithms listed in [16, 17] do not find this value. Our APEA finds a better solution to be f(X) = 7049.24802055468081 where

X = (579.30828537221191, 1359.96826293047116, 5109.97147225199751,
182.01783328980514, 295.60114110998057, 217.98216670925277,
286.41669217921429, 395.60114110996898)

The parameters of APEA are set as follows: NPROC= 4, T=10, Q=2. Each time, APEA converges at the solution above in no more than 26 seconds. In contrast, the sequential simulation of APEA converges at a much worse solution quickly.

4. Conclusion

In this paper, we describe an effective asynchronous parallel evolutionary algorithm and get a series of best solutions for the BUMP problem and other benchmark problems. Still, there are many other issues that should need further research, such as the effect of different Q and T on numerical result, and the topology of neighborhoods.

References

1. H. T. Yang, “A Parallel Genetic Algorithm Approach to Solving the Unit Commitment Problem: Implementation on the Transputer Networks,” *IEEE Transactions on Power Systems*, Vol. 12, No.2, pp. 661-668, 1997.
2. A. Wu, K. Y. Wu, R. M. M. Chen, Y. Shen, “Parallel Optimal Statistical Design Method with Response surface modeling using genetic algorithms,” *Circuits, Devices and Systems, IEE Proceedings-*, Vol. 145, No.1, 1998.
3. J. D. Lohn, “A Circuit Representation Technique for Automated Circuit Design,” *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 3, pp. 205-219, 1999.
4. E. Cantu-Paz, “Markov Chain Models of Parallel Genetic Algorithms,” *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 216-226, 2000.

5. P. B. Grosso, “Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multi-locus Model,” *Ph.D. Dissertation*, University of Michigan, 1985.
6. L. A. Anbarasu et al., “Multiple Sequence Alignment by Parallely Evolvable Genetic Algorithms,” in A. S. Wh(ed.) *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program*, Orlando, Fla, July 13, 1999, pp. 154-156.
7. H. Lienig, “A Parallel Genetic Algorithm for Performance-Driven VLSI Routing,” *IEEE Transactions on Evolutionary Computation*, Vol. 1, No.1, pp. 29-39, 1997.
8. K. Kojima, W. Kawamata, et al, “Network Based Parallel Genetic Algorithm using Client-Server Model”, in *Proceedings of the Conference on Evolutionary Computation 2000*, pp. 244-250.
9. T. Guo, L. S. Kang, “A New Evolutionary Algorithm for Function Optimization,” *Wuhan University Journal of Natural Sciences*, Vol. 4, No. 4, pp. 409-414, 1999.
10. Z. Michalewicz and M. Schoenauer, “Evolutionary Algorithms for Constrained Parameter Optimization Problems,” *Evolutionary Computation*, Vol. 4, No. 1, pp. 1-32, 1996.
11. A. J. Keane, “Experiences with Optimizers in Structural Design,” *in Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 94*, ed. (I. C. Parmee, Plymouth, 1994), pp. 14-27.
12. A. J. Keane, “A Brief Comparison of Some Evolutionary Optimization Methods,” *Proceedings of Applied Decision Technologies (Modern Heuristic Methods)*, 1995.
13. A. J. Keane, “Genetic Algorithm Optimization of Multi-peak Problems: Studies in Convergence and Robustness.” *Artificial Intelligence in Engineering*, 1995.
14. A. J. Keane, “Passive Vibration Control Via Unusual Geometric: Application of Genetic Algorithm Optimization to Structural Design,” *Journal of Sound and Vibration*, 1995.
15. M. A. EI-Beltagy, et al. “Metamodeling Techniques For Evolutionary Optimization of Computationally Expensive Problems: Promises and Limitations.” *Genetic Algorithms and Classifier Systems*, 1999.
16. Z Michalewicz, S. Esguvel et al., “The Spirit of Evolutionary Algorithms.” *Journal of Computing and Information Technology*, Vol. 7, pp. 1-18, 1999.
17. M. J. Tahk, B. C. Sun, “Coevolutionary Augmented Lagrangian Methods for Constrained Optimization.” *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 2, 2000.
18. Sunderam, V. S., “PVM: A Framework for Parallel Distributed Computing.” *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315-339, December, 1990.
19. “MPI: A Message-Passing Interface Standard.” Message Passing Interface Forum, 1994.
20. Heinz Muhlenbein, “Evolution in Time and Space - The Parallel Genetic Algorithm”, In Gregory J.E. Rawlins, Editor, *Foudation of Genetic Algorithms 1*, Page 316--337, San Mateo, CA, USA, 1991, Morgan, Kaufmann.
21. William E. Hart, Scott Baden, Richard K. Belew, Scott Kohn **,** **“**Analysis of the Numerical Effects of Parallelism on a Parallel Genetic Algorithm**”,** *Proc 10th Intl. Parallel Processing Symp*. pp 606-612, 1996.
22. Enrique Alba, Jos M Troya Dpto. de Lenguajes y Ciencias de la Computation, “An Analysis of Synchronous and Asynchronous Parallel Distributed Genetic Algorithms with Structured and Panmictic Islands”, *Future Generation Computer Systems*, 17(4):451-465, January 2001.
23. Grefenstette J. J., “Parallel adaptive algorithms for function optimization”, Tech. Rep. No. CS-81-19, Vanderbilt University, Computer Science Department, Nashville, TN, 1981.
24. Martin F. J., Trelles-Salazar O., Snadoval F., “Genetic algorithms on LAN-message passing architectures using PVM: Application to the routing problem”, In Davidor Y. Schwefel H. –P., Manner R., Eds., *Parallel Problem Solving from Nature, PPSN III*, p.534-543, Springer-Verlag (Berlin), 1994.