Function Optimization using a Pipelined Genetic Algorithm

Malay K Pakhira [1], Rajat K De [2]
[1] *Department of Computer Science and Engineering, Kalyani Government Engineering College*
Kalyani, West Bengal - 741235, INDIA, malay_pakhira@yahoo.com
[2] *Machine Intelligence Unit, Indian Statistical Institute*
Kolkata - 700108, INDIA, rajat@isical.ac.in

Abstract

Genetic Algorithms (GAs) are very commonly used as function optimizers, basically due to their search capability. A number of different serial and parallel versions of GA exist. In this article, a pipelined version of the commonly used Genetic Algorithm is described. The main idea of achieving pipelined execution of different operations of GA is to use a stochastic selection function which works with the fitness value of the candidate chromosome only. GA with this selection operator is termed PLGA. When executed in a CGA (Classical Genetic Algorithm) framework, the stochastic selection scheme gives comparable performances with the roulette-wheel selection. In the pipelined hardware environment, PLGA will be much faster than the CGA. When executed on similar hardware platforms, PLGA may attain a maximum speedup of four over CGA. However, if CGA is executed in a uniprocessor system the speedup is much more. A comparison of PLGA against PGA (Parallel Genetic Algorithms) is also done.

Keywords : Function optimization, Genetic algorithm, Parallel GA, Pipelined GA, Stochastic selection.

1. Introduction

Genetic algorithm (GA), developed by Holland in 1975 [1], is known to be an efficient search and optimization mechanism which incorporates the principles of evolution and natural selection. Many different forms of the basic GA exist [2]. Several attempts have been made to exploit the inherent parallelism of GA [3], [4], [5], [6], [7]. Most of these methods maintain the basic serial nature of the operations. They simply divide the population into a number of sub-populations and execute genetic operations for each of the sub-populations separately. After parallel executions in all the processing units, at certain intervals, the newly developed information regarding the best solution strings of these units are exchanged through migration phase. This kind of parallel execution is supported by distributed processing environments.

In this article, we describe the design of a pipelined genetic algorithm (PLGA) which uses a stochastic selection scheme that eliminates the dependency for a complete pool of candidate solutions required in conventional methods at the selection stage and hence allows us to develop a pipeline using the basic operations of GA. Simulation experiments with various functional optimization problems of varying complexity demonstrates the effectiveness of this new selection scheme. The performance of PLGA, in terms of rate of convergence and speedup, is compared with classical genetic algorithm (CGA) and parallel genetic algorithms (PGA).

2. Basics of GA

In genetic algorithms [1], [2], the parameters of an optimization problem corresponding to a possible solution are encoded to form a chromosome. A collection of such chromosomes is called a population or pool. The initial population is generated randomly or using some domain specific knowledge. Three genetic operators selection, crossover and mutation are normally used in GAs. The selection operator finds the fittest candidate solutions (chromosomes) from the pool of current generation, which are then represented in the next generation, based on their figure of merit. Classical GAs (CGA) generally use the roulette-wheel selection scheme. By crossover, features of two selected strings (mates) from the parent population are intermixed to generate newer chromosome patterns called children. Mutation is used for fine tuning the solution. Crossover and mutation are done with certain probabilities. Mutation can take place in several different ways. In the binary representation for chromosomes, it generally toggles one or more bit position(s) depending upon mutation probability. Using these three operators, we can get a new population from the old one. Crossover and mutation strategies used in this work are similar to those mentioned in [8].

Parallel versions of GAs (PGA) are executed on a network of processors simultaneously. Difference among various parallel implementations are basically on the architecture of processor network, population distribution policy, rate of information exchange etc. We, however, have used a completely connected network in our experiments.

3. Stochastic Selection and Pipelining

In this section, we shall describe a stochastic selection method which allows us to develop a pipeline of GA operators. This selection method is a modified version of the existing SA-selection [9]. The functional form of the selection method is described along with an example. Then we shall provide the corresponding pipeline design of GA.

In *stochastic* selection method, a chromosome $\mathbf{x}$ is considered from a pool $P(g)$ of the current generation g, and is selected based on Boltzmann probability distribution function.

Let, f_{max} be the fitness value of the currently available best string. If the next string is having fitness value $f_{\mathbf{x}}$ such that $f_{\mathbf{x}} > f_{max}$, then it is selected. Otherwise, it is selected with Boltzmann probability

$$P = exp[-(f_{max} - f_{\mathbf{x}})/T]$$

where $T = T_0(1-\alpha)^k$ and $k = 100 \times \frac{g}{G}$; G is the maximum value of g. Ranges of α and T_0 are $[0,1]$ and $[5,100]$ respectively. From the above expression, it is clear that the value of T will decrease exponentially or at logarithmic rate with increase in g, and hence the value of the probability P. This is significant in terms of convergence. As $T \rightarrow 0$, the final state is approached.

In conventional selection schemes, before starting the selection process, all the chromosomes in the earlier generations must be evaluated. But evaluation is the most time consuming process and is a bottleneck in attaining a pipeline of the genetic operators. The new selection scheme eliminates this bottleneck. We can express the new selection operator as a function of the input chromosome $(\mathbf{x})$. Let, $\mathbf{x}_{max}$ be the chromosome corresponding to the currently available maximum fitness. Then the selection operator, expressed functionally, is

$$Sel(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } (f_{\mathbf{x}} > f_{max}) \\ \mathbf{x} & \text{if } (f_{\mathbf{x}} \leq f_{max}) \ \wedge \ (P > P_1) \\ \mathbf{x}_{max} & \text{if } (f_{\mathbf{x}} \leq f_{max}) \ \wedge \ (P \leq P_1) \end{cases}$$

where, $P_1 = random[0,1)$.

Let us consider an example, for describing the operation of the selection scheme, with a population of size 6. Let, in the $(g-1)$th generation, the maximum fitness value is 45.0. This value is stored in the variable f_{max} and is used in generation g also. In any generation, the value of f_{max} is altered whenever a chromosome with a greater fitness is encountered (and selected). Note that, using elitist strategy, we are storing the best chromosome, in a generation, along with its fitness value in the very first location of the pool of chromosomes. Table 1 shows how chromosomes are selected for generation g.

TABLE 1: SAMPLE EXECUTION OF STOCHASTIC SELECTION

Input chromosome number	Fitness of input chromosome	f_{max}	Chromosome number with maximum fitness	P	P1	Selected chromosome number
0	45.0	45.0	0	0.931	0.882	0
1	48.0	45.0	0	-	-	1
2	35.0	48.0	1	0.810	0.853	1
3	43.0	48.0	1	0.616	0.447	3
4	55.0	48.0	1	-	-	4
5	12.0	55.0	4	0.317	0.591	4

A pair of selected chromosomes may be used for crossover, mutation, and evaluation and then put into the population pool for the next generation. When a chromosome is evaluated, it is put into population pool along with its fitness value for the next generation. Thus the processes corresponding to selection, crossover, mutation and evaluation in a particular generation can work simultaneously, in an overlapped fashion.

Using the above mentioned selection method, the concept of pipelining has been incorporated within the genetic algorithm framework. The algorithmic difference of PLGA with respect to CGA is only due to the use of the stochastic selection scheme. However, since each chromosome can be selected (or rejected) as soon as it is evaluated, we can continue looping through the steps of genetic operations if only a pair chromosomes are evaluated and selected. The algorithm is formulated below.

begin
  initialize population
  evaluate population for fitness
  while (NOT $termination\_condition$) **do**
    begin
      select a pair of chromosomes
      cross the pair of chromosomes
      mutate the pair of chromosomes
      evaluate children for fitness
      put children into population
        with fitness values
    end
end

It is possible to implement the algorithm using *appropriate hardware*, where, selection, crossover, mutation and evaluation operations will be done in specially designed hardware circuits forming a pipeline among them. Multiplicity of processing units to be used at each of the stages may, in general, be different and depends on the complexity of the problem concerned.

4. DESIGN OF THE PIPELINE

In this section, we explore the structural parallelism of GA that are hidden in its strictly serial use of genetic operators discussed in Section 3. For this purpose, we have to streamline these operations so that their functioning becomes overlapped in nature. The motivations behind PLGA are twofold. First, there is an advantage in terms of higher speedup as a result of overlapped execution in a pipeline. The second, possibly more important one, is the chance of incorporating more population and thus increasing the diversity among them.

A. Pipeline Architecture

In order to solve functional optimization problems using GA, we need to maintain a population of probable solutions (chromosomes). The chromosomes are evaluated for the objective function. The fittest candidates are selected for the next generation which then undergo the crossover and mutation operations to generate offspring. The whole process is repeated for a number of generations. Thus, we can identify four major functions : (i) selection (S), (ii) crossover (C), (iii) mutation (M) and (iv) evaluation (E), and can construct a four stage pipeline as shown in Figure 1. We need to maintain a buffer

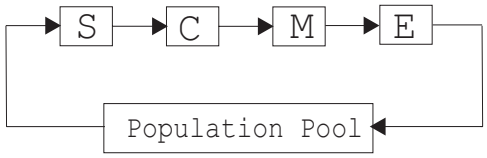

Fig. 1: Pipeline stages for the GA. Here S, C, M and E stand for selection, crossover, mutation and evaluation respectively

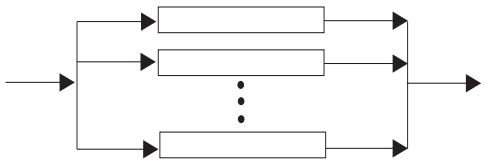

Fig. 2: Multiplicity of any particular unit in the pipeline

memory to reserve the chromosomes after evaluation. This buffer is organized in a FIFO manner.

The selection operation requires two parallel units so that it can provide two strings to the crossover unit in due time. Mutation and fitness evaluation should be done in multiple units that operate in parallel. The number of units for mutation is determined by the length of a chromosome. It is found that evaluation is the most time consuming process compared to the other operations, and the number of units for this stage is determined by the complexity of the function considered.

Let, S_t, C_t, M_t and E_t be the stage times for selection, crossover, mutation and evaluation operations respectively. Among them, C_t is normally found to be the minimum. We call this minimum time as one T-cycle. Let, $S_t = sC_t$, $M_t = mC_t$ and $E_t = eC_t$. Therefore, the ratio of S_t, C_t, M_t and E_t becomes $s:1:m:e$. That is, s, m and e number of T-cycles are required for selection, mutation and evaluation operations respectively. Thus for one crossover unit we need, for efficient utilization of resources, $\lceil s \rceil$, $\lceil m \rceil$ and $\lceil e \rceil$ pairs of units for selection, mutation and evaluation respectively. For sake of simplicity, let us consider, from now on, $s = \lceil s \rceil$, $m = \lceil m \rceil$ and $e = \lceil e \rceil$. Here the units are counted in pairs because one crossover needs two selected strings. From the above ratio, it is clear that, if the crossover unit takes 1 unit of time (T-cycle) to perform one crossover, the selection, mutation and evaluation units take s, m and e units of time to perform one selection, mutation and evaluation operations respectively. Thus for proper and efficient utilization of the resources, we should use s, m and e pairs of respective units for one crossover unit.

B. Speedup

Speedup of a general pipeline is defined as $S = \frac{T_{NP}}{T_P}$ where T_{NP} ($= nk$, n = number of executions and k = number of stages) and T_P ($= n + k - 1$) are the computation times (in terms of number of T-cycles) required for non-pipelined and pipelined systems respectively. In the proposed clustering, if appropriate number of units are considered, the average time per chromosome in each stage becomes equal to one T-cycle. This is the ideal hardware configuration. However, we can use less number of units at the stages. Let, for any arbitrary configuration, m_n and e_n be the number of pairs of units used at mutation and evaluation stages corresponding to one crossover unit and one pair of selection units. In our case, n = population size $\times$ number of generations.

Consider a pipeline where s = 1, m = 4 and e = 6. Here we require $s+1+m+e$ = 12 T-cycles to get the first pair of children chromosomes. Since n = population size $\times$ number of generations, we may assume, without loss of generality, n to be equal to the population size executed for one generation only. After obtaining the first pair of children, the remaining children will come out in pairs at each successive T-cycles. Therefore, the number of T-cycles required for the remaining pairs is $\left(\frac{n}{2} - 1\right)$. Thus, the total number of T-cycles required for the pipeline is $T_P = 11 + \frac{n}{2}$ For a non-pipelined system configured with the same multiplicity of stages (as that of the pipelined one), the number of T-cycles considering all the five stages sequentially is

$T_{NP} = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} + \frac{n}{2} = \frac{4n}{2} = 2n$

So, the speedup attained is

$S = \frac{T_{NP}}{T_P} = \frac{2n}{\frac{n}{2}+11}$

for situations when $n >> 11$, $S \approx 4$. This is the ideal speedup.

We can use less number of units at mutation and evaluation stages. Let, for any arbitrary configuration, $m^{'}$ and $e^{'}$ be the number of pairs of units used at mutation and evaluation stages corresponding to one crossover unit and one pair of selection units. Here, $m^{'} < m$ and $e^{'} < e$, i.e., the number of units at the mutation and evaluation stages are less than that needed for full multiplicity of these stages. Let $r_m = \lceil \frac{m}{m'} \rceil$ and $r_e = \lceil \frac{e}{e'} \rceil$, i.e., r_m and r_e are the factors by which multiplicity is reduced at the corresponding stages. We define the $reduction\ factor$ for a pipeline as the maximum of r_m and r_e, i.e., $reduction\ factor,\ r = max(r_m, r_e)$. When $r = 1$, the pipeline has full multiplicity and is referred to as a $full\ pipeline$. For $r > 1$, it is called a $reduced\ pipeline$.

Now, let us a consider a reduced pipeline where r_m, r_e and r represent reduction factors for mutation, evaluation stages and that for the whole pipeline. By definition, at least one of r_m and r_e is equal to r and the other is less than or equal to r. For such a reduced system we get$T_P = 1 + 1 + m + e + \left(\frac{n}{2} - 1\right) \times r = 2 + m + e + \left(\frac{n}{2} - 1\right) \times r$ and

$T_{NP} = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} \times \lceil \frac{m}{m'} \rceil + \frac{n}{2} \times \lceil \frac{e}{e'} \rceil$.

If the pipeline is a uniformly reduced one with $r_m = r_e = r$, we have,

$T_{NP} = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} \times r + \frac{n}{2} \times r$.

Now, in our example system, if the reduction factor be $r =$ 2, then we get,

$T_P = 12 + \left(\frac{n}{2} - 1\right) \times 2 = 10 + n$ and

$T_{NP} = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} \times 2 + \frac{n}{2} \times 2 = 3n$.

Therefore, speedup $S \approx 3$ for $n >> 10$. From the above discussion it is clear that a fixed hardware setup is also possible. However, for the present article, we have not performed experiments with a reduced pipeline.

5. Experimental Results

We have demonstrated effectiveness of PLGA on various benchmark functions. The benchmarks include both unimodal and multi-modal functions. Experiments are performed to compare the performance of the pipelined algorithm (PLGA) with its serial (CGA) and parallel (PGA) counterparts. The following subsections describe the benchmark functions and results of comparison respectively.

A. Benchmark Functions

The selected benchmark functions, which are unimodal or multi-modal in nature, are available in literature. These functions are mentioned below.

1. *Sphere Model* function:

$$f_1(x) = \sum_{i=1}^{l} {x_i}^2.$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

2. $Step$ function:

$$f_2(x) = \sum_{i=1}^{l} integer(x_i).$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

3. $Rastrigin's$ function:

$$f_3(x) = \sum_{i=1}^{l} [x_i^2 - 10\cos(2\pi x_i) + 10]$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This multi-modal function has its minimum value of 0 at $x_i = 0, \forall i$.

4. $Rosenbrock's$ function:

$$f_4(x) = \sum_{i=1}^{l} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 1, \forall i$.

5. $Ackley's$ function:

$$\begin{aligned} f_5(x) = \quad & -20\exp\left(-0.2\sqrt{\tfrac{1}{l}\textstyle\sum_{i=1}^{l} x_i^2}\right) \\ & - \exp\left(\tfrac{1}{l}\textstyle\sum_{i=1}^{l}\cos 2\pi x_i\right) + 20 + e \end{aligned}$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

6. $Schwefel's$ function 1:

$$f_6(x) = \sum_{i=1}^{l} |x_i| + \Pi_{i=1}^{l} |x_i|$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

7. $Schwefel's$ function 2:

$$f_7(x) = \sum_{i=1}^{l} \left(\sum_{j=1}^{i} x_j\right)^2$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

8. $Schwefel's$ function 3:

$$f_8(x) = \max_i \{\ |x_i|, \quad 1 \leq i \leq l\}$$

The range of x_i is $-5.12 \leq x_i \leq 5.12$. This function has its minimum value of 0 at $x_i = 0, \forall i$.

For all these functions we have coded the variables using 25 bit binary code. However, larger binary codes may be used to increase the accuracy of the numbers represented.

B. Results

Results of experiments are described in two phases. In the first phase, we compared stochastic and roulette-wheel selection schemes in CGA framework, in terms of quality of solutions and rate of convergence and speedup. Also, the speedup obtained in the pipelined system over uniprocessor based CGA is presented. In the second phase, we demonstrate comparative performances of PLGA and PGA in terms of rate of convergence.

1) PLGA vs. CGA : For this part of investigations, with the stochastic and roulette-wheel selection schemes in the CGA framework, we have used the same values for the control parameters, viz., population size = 50, P_c = 0.6 and P_m = 0.05. The value of α is taken to be 0.05. The initial temperature, T_0, is set to 50.

At first, we have compared the stochastic selection scheme with the roulette wheel one when both of them are used in a CGA framework. Then, we have executed the PLGA and CGA using the stochastic selection function in both the cases for computing speedup. Simulation experiments are performed on the selected problems mentioned in Section 5-A. We have used, here, a dimension of 10 for all the problems. Table 2 provides the optimal values for the objective functions which are reached by the CGA, using two selection schemes, when executed for 2000 generations.

Speedup of PLGA compared to CGA is measured in two different situations, viz., when a similar hardware platform (like the pipeline) is used for CGA, and when a single uniprocessor is used only. For the first situation, results are provided in Table 3, and for the second, it is provided in Table 4. The stage times given in Table 3 are proportional to the actual times. This is because, we have executed PLGA for a number of generations, and measured sum-total of the corresponding stage times. The speedup is found to be less than that obtained in Section 4-B. This is due to the fact that in computation of T_P in Section 4-B, we have allocated extra time to all the units for synchronous operation of the pipeline stages. However, selection and crossover could be done in less amount of time than that allocated (50 units).

TABLE 2: COMPARISON OF STOCHASTIC AND ROULETTE WHEEL SELECTIONS. RESULTS ARE AVERAGED OVER 50 RUNS. "MEAN BEST" AND "STD DEV" INDICATE MEAN BEST FUNCTION VALUES AND STANDARD DEVIATIONS

Function	Stochastic		Roulette wheel	
	Mean Best	Std Dev	Mean Best	Std Dev
f_1	$1.50 \times E-6$	$1.28 \times E-6$	$3.06 \times E-6$	$1.37 \times E-6$
f_2	0.00	0.00	0.00	0.00
f_3	2.5626	1.3813	4.0030	2.0641
f_4	8.6902	0.8009	34.7507	39.2635
f_5	$1.32 \times E-3$	$4.30 \times E-4$	$1.50 \times E-3$	$6.08 \times E-4$
f_6	$2.90 \times E-3$	$6.50 \times E-4$	$3.15 \times E-3$	$9.42 \times E-4$
f_7	$2.60 \times E-3$	$4.09 \times E-4$	$5.01 \times E-2$	$7.85 \times E-2$
f_8	$5.55 \times E-3$	$1.68 \times E-3$	$5.67 \times E-3$	$1.27 \times E-3$

TABLE 3: STAGE TIMES OF PLGA AND SPEEDUPS OBTAINED FOR $f_1 - f_8$. TIMES (IN UNITS OF 10^4 CLOCK TICKS) SHOWN ARE PROPORTIONAL TO ACTUAL STAGE TIMES. ONE T-CYCLE = 50.

Function	Dimension	Stage Times				No. of T-Cycles				Speedup
		S_t	C_t	M_t	E_t	s	c	m	e	
f_1	3	17	13	150	599	1	1	3	12	2.60
f_2	5	18	14	247	998	1	1	5	20	2.62
f_3	10	15	46	472	2129	1	1	10	43	2.90
f_4	3	14	19	134	539	1	1	3	11	2.53
f_5	3	16	18	134	527	1	1	3	11	2.53
f_6	3	16	10	137	481	1	1	3	10	2.40
f_7	5	16	23	216	842	1	1	5	16	2.70
f_8	3	14	15	138	482	1	1	3	10	2.46

TABLE 4: SPEEDUP OF PLGA OVER CGA EXECUTED IN A SERIAL UNIPROCESSOR SYSTEM WITH NO SPECIAL HARDWARE PROCESSING ELEMENTS. THE TIMES (IN UNITS OF 10^4 CLOCK TICKS) SHOWN ARE PROPORTIONAL TO ACTUAL STAGE TIMES. ONE T-CYCLE = 50. $k = n/2$

Function	Dimension	Total Execution Time		Speedup
		$Serial\ Uniprocessor$	$Pipelined\ System$	
f_1	3	$779k$	$(16+k)50$	15.58
f_2	5	$1277k$	$(26+k)50$	25.54
f_3	10	$2662k$	$(54+k)50$	53.24
f_4	3	$706k$	$(15+k)50$	14.12
f_5	3	$695k$	$(15+k)50$	13.90
f_6	3	$644k$	$(14+k)50$	12.88
f_7	5	$1097k$	$(22+k)50$	21.94
f_8	3	$649k$	$(14+k)50$	12.98

2) PLGA vs. PGA: We have executed both PLGA and PGA for a number of selected benchmark problems. In this case, again we have set the dimension of benchmark functions to 10. As a measure of comparison, we have selected the number of generations needed to converge to a near optimal solution. For all the benchmark functions, a particular limiting value is selected as the stopping criteria.

Here, the population size considered is 40. For the purpose of executing PGA a four processors network is considered. The population of size 40 is distributed among the four processors, each getting a subpopulation of size 10. The processors are completely connected and they can communicate strings (chromosomes) after every 5 generations. During communication, each processor selects four chromosomes, including the current best, from self, and two from each of the other processors. The results of comparison are shown in Table 5.

6. CONCLUSION

A function optimizer, using a pipelined version of the conventional genetic algorithm, called PLGA, has been described in this article. A stochastic selection scheme is used for this purpose. The unique feature of the stochastic selection scheme is that, it does not depend on a complete pool of pre-evaluated chromosomes. Hence, as soon as a chromosome is evaluated, it can be passed to the selection unit for possible selection.

TABLE 5: COMPARISON OF PLGA AND PGA IN TERMS OF NUMBER OF GENERATIONS. "MEAN" AND "STDDEV" INDICATE THE AVERAGE NUMBER OF GENERATIONS AND THE STANDARD DEVIATIONS RESPECTIVELY.

Function	Dimension	PLGA		PGA		Stopping Value
		Mean	Stddev	Mean	Stddev	
f_1	10	180.52	42.92	406.50	61.15	0.005
f_2	10	33.72	13.71	54.52	15.49	0.005
f_3	10	8.06	3.08	17.42	6.58	50.0
f_4	10	11.18	2.94	20.76	5.93	500.0
f_5	10	65.38	23.86	128.98	33.54	0.005
f_6	10	134.18	32.72	284.26	36.71	0.5
f_7	10	132.02	138.19	202.40	109.82	0.5
f_8	10	236.00	58.62	383.56	64.01	0.5

By use of proper hardware, one can develop an extremely fast version of the GA based function optimizer. We have executed the algorithm in software form, on a serial uniprocessor system, and have shown that, in a hardware implementation, if proper multiplicity of different stage units are used, a maximum speedup of 4 is attainable compared to conventional GAs executed serially using similar multiplicity of stage units. However, when compared to CGA executed on a uniprocessor, speedup is found to be much more. We have also compared PLGA and PGA with a certain processor architecture. It is seen that speedup obtained in PLGA is better than PGA.

Although PLGA is presented here only as a functional optimizer, one may use it for any combinatorial optimization problem also. The authors are currently working in that direction.

ACKNOWLEDGMENTS

This research is partly supported by a project titled *Pipelined Genetic Algorithm and its Applications in Satellite and Medical Image Segmentation :* Number 8022/RID/NPROJ/RPS-97/2003-04 funded by *All India Council for Technical Education (AICTE)*, Government of India.

REFERENCES

[1] J. Holland, *Adaptation in Neural and artificial Systems*. Ann. Arbor, MI: University of Michigan, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Addison-Wesley, 1989.

[3] J. J. Grefenstette, "Parallel adaptive algorithms for function optimization," tech. rep., Vanderbilt University, Computer Science Department, Nashville. TN, 1981.

[4] H. Mühlenbein, M. Scomisch, and J. Born, "The parallel genetic algorithm as function optimizer," in *Proc. of Fourth Intl. Conf. on Genetic Algorithms*, pp. 271–278, San Mateo, Calif: Morgan Kaufmann, 1991.

[5] V. S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers," in *Proc. of the Fifth International Conference on Genetic Algorithms*, (Morgan Kaufmann, San Mateo, CA), pp. 177–183, 1993.

[6] S. Baluja, "Structure and performance of fine-grain parallelism in genetic search," in *Proc. of the Fifth International Conference on Genetic Algorithms*, (Morgan Kaufmann, San Mateo, CA), pp. 155–162, 1993.

[7] R. Shonkwiler, "Parallel genetic algorithms," in *Proc. of 5th Intl. Conf. on Genetic Algorithms*, pp. 199–205, San Mateo, CA: Morgan Kaufmann, 1993.

[8] J. L. R. Filho, P. C. Treleaven, and C. Alippi, "Genetic algorithm programming environments," *IEEE Computer*, pp. 28–43, June, 1994.

[9] B. T. Zhang and J. J. Kim, "Comparison of Selection Methods for Evolutionary Computation," in *Evolutionary Optimization*, pp. 55–70, vol. 2, no. 1, 2000.