Generational PipeLined Genetic Algorithm (PLGA) using Stochastic Selection

Malay K. Pakhira and Rajat K. De

Abstract**—** In this paper, a pipelined version of genetic algorithm, called PLGA, and a corresponding hardware platform are described. The basic operations of conventional GA (CGA) are made pipelined using an appropriate selection scheme. The selection operator, used here, is stochastic in nature and is called SA-selection. This helps maintaining the basic generational nature of the proposed pipelined GA (PLGA). A number of benchmark problems are used to compare the performances of conventional roulette-wheel selection and the SA-selection. These include unimodal and multimodal functions with dimensionality varying from very small to very large. It is seen that the SA-selection scheme is giving comparable performances with respect to the classical roulette-wheel selection scheme, for all the instances, when quality of solutions and rate of convergence are considered. The speedups obtained by PLGA for different benchmarks are found to be significant. It is shown that a complete hardware pipeline can be developed using the proposed scheme, if parallel evaluation of the fitness expression is possible. In this connection a low-cost but very fast hardware evaluation unit is described. Results of simulation experiments show that in a pipelined hardware environment, PLGA will be much faster than CGA. In terms of efficiency, PLGA is found to outperform parallel GA (PGA) also.

Keywords**—** Hardware evaluation, Hardware pipeline, Optimization, Pipelined genetic algorithm, SA-selection.

I. Introduction

Genetic algorithm (GA) [1], [2] is known to be an efficient search and optimization technique which incorporates the principles of evolution and natural selection. GA forms a basic tool of a new field of research called Evolutionary Computation [3], [4]. There have been several attempts [2], [3], [5], [6] for reformulation and customization of GAs. Because of their parallel search capability, they form a class of the most widely accepted techniques for solving complex problems. For enhancing the capabilities of GAs, their inherent parallelism can be exploited to develop parallel GAs. Several parallel implementations of GA (PGA) exist in literature [6], [7], [8], [9], [10], [11], [12].

Most of these parallel methods maintain the basic serial nature of GA 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 intervals of several generations, the newly developed information regarding the best solution chromosomes of these units are exchanged through migration phase. Such kind of parallel execution is supported by distributed processing environments or in a multiprocessor system. It is however possible to attain the speedup of a multiprocessor in a uniprocessor system if a proper pipeline can be developed.

In this paper, the design of a pipelined genetic algorithm (PLGA) is described that uses simulated annealing (SA)[13] based SA-selection. This SA-selection scheme (explained in Section III-A) 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. Use of SA-like selection functions in GAs is not new [14], [15], [16], [17]. Goldberg [16] proposed a Boltzmann Tournament selection. In [17], another form of the SA-selection scheme has been used. Here, a simple SA-selection scheme is used which is exactly similar to the one used in conventional simulated annealing. A temperature schedule, where the temperature varies as a function of the generation number of GA, is defined. This temperature schedule is described in Section III-A. The motivation behind using the SA-selection scheme is to maintain the basic generational nature of the GA in its pipelined execution too. Some FPGA based GA-pipeline can be found in literature [18], [19], [20], where binary tournament selection is used in such a way that one achieves only a non-generational GA.

In order to demonstrate the effectiveness of the SA-selection scheme, in comparison to the well known roulette-wheel selection scheme, simulation experiments are performed with various functional optimization problems of varying complexities. For these functional optimization problems, both unimodal and multimodal functions with dimensionality varying from 1 to 125 are considered. The performance of PLGA, in terms of rate of convergence, speedup and efficiency, is compared with conventional GA (CGA) and parallel GA (PGA).

It is realized that a pipelined algorithm can not be properly used without a corresponding pipelined hardware. Recently, researchers are trying to develop different hardware implementations of GAs [18], [19], [20], [21]. In these schemes, attempts are made to implement GA in FPGA platforms which support reconfigurable hardware design for selection, crossover, mutation and evaluation parts. A major problem, in this regard, is the need to develop a new evaluation circuit for each different problem. Sometimes this task may be very time consuming and in some cases it may not be possible implement the evaluation circuit due to some FPGA limitations. Hence, one needs a general purpose hardware evaluation unit that can be used for any optimization problem. One such evaluation unit and its generalized versions are presented in [22], [23],

Manuscript received February 3, 2007.

Department of Computer Science and Engineering, Kalyani Government Engineering College, Kalyani - 741235, INDIA, Email: malay_pakhira@yahoo.com

Machine Intelligence Unit, Indian Statistical Institute, Kolkata - 700108, INDIA, Email: rajat@isical.ac.in

[24]. Use of this circuit can fulfil the purpose of the proposed pipelining scheme.

The organization of this paper is as follows. In Section II, some conventional selection schemes and the serial genetic algorithm are described. Section III describes the SA-selection method and the proposed pipelined genetic algorithm. In Section IV, design of the pipeline is presented. A possible hardware evaluation unit for GAs is described in Section V. Section VI, deals with the selected benchmark functions and experimental results. A conclusion is drawn and direction to further research is cited finally in Section VII.

II. Conventional GA and Selection Methods

In conventional GA (CGA), parameters of an optimization problem, corresponding to a possible solution, are encoded to form a chromosomes. A collection of N such chromosomes is called a population or pool. The initial population is generated randomly or using some domain specific knowledge. The basic operations of a serial GA are selection, crossover, mutation and evaluation. The selection operation selects better chromosomes from current generation pool for generating the next generation pool. By crossover, features of two selected chromosomes (mates) from the parent population are intermixed to generate child chromosomes. Mutation is used for fine tuning the solutions. Crossover and mutation are done with certain probabilities. The processes of selection through evaluation are repeated for several generations until a stable pool of solutions is found.

In parallel versions of GA (PGA), generally the population is divided into subpopulations. Each subpopulation is then assigned to one processor in a network of processors. Individual processors execute CGA using their respective subpopulation, and exchange genetic information (chromosomes) obtained by them at specific intervals. Generally Master-Slave, Mesh Connected or Cyclic processor networks are used for PGAs [12]. The objective of PGAs is to benefit from both speedup and quality of solutions points of views.

The selection operator is responsible for selection of the fittest candidates (chromosomes) from the pool $P(g)$ of the current generation (g) in the population to be represented in the pool $P(g+1)$ of the next generation ($g+1$). Obviously, selection of a chromosome depends on its figure of merit toward optimization of the objective function. Here, two well known selection schemes, commonly used in conventional GAs, are described briefly.

Roulette-wheel selection

This is the most common selection scheme. Here, a chromosome $\mathbf{x}_i$ with fitness value $f(\mathbf{x}_i)$ from the current population is selected based on the probability values $Pr(.)$ such that

$$Pr(\mathbf{x}_i) = f(\mathbf{x}_i) / \sum_{j=1}^{N} f(\mathbf{x}_j)$$

Tournament selection

In tournament selection, we select the best chromosome out of a set of some randomly chosen chromosomes from the pool. This process is repeated N times, N being the population size, to create a completely new next generation pool.

From the description of the above two schemes, it is clear that both of them require a completely evaluated pool of chromosomes before the selection process starts. When these schemes are used, the operations of the genetic algorithm, within a generation, becomes serial in nature and follows the pseudo code given in Algorithm 1 below.

Algorithm 1 : *Outline of Serial genetic algorithm.*

begin
  $g = 0$;
  initialize population $P(g)$;
  evaluate chromosomes in $P(g)$;
    repeat
      select N chromosomes to generate $P(g+1)$;
      crossover selected chromosomes pairwise;
      mutate each crossed chromosome;
      evaluate each mutated chromosome;
      $g = g + 1$;
    until convergence;
end

From the above selection schemes and the serial genetic algorithm, it is seen that the selection operation needs a fully evaluated pool of chromosomes. Of course, this is true for generational GAs only. In case of steady state and other non-generational GAs the above restriction does not hold. In the following section, a stochastic selection scheme is described that allows us to pipeline a generational GA without affecting any of its basic features.

III. SA-Selection Scheme and Design of PLGA

The SA-selection method which is generally used in simulated annealing (SA)[13] is described below. In SA, at each iteration, an alternative solution $\mathbf{y}$ is selected from a set of alternatives. The solution $\mathbf{y}$ is accepted if $f(\mathbf{y}) \leq f(\mathbf{x})$, where $\mathbf{x}$ is the candidate solution selected in the previous step, otherwise it will be accepted with probability $\exp[-(f(\mathbf{y}) - f(\mathbf{x}))/T]$. At each iteration of the process, the parameter (temperature) T, is reduced by a small amount. The above function is used in the selection stage of PLGA. The selection method and the pipelined genetic algorithm are described below.

A. Use of SA-selection for pipelining GA

In SA-selection method, a chromosome $\mathbf{x}_i$, with value $f(\mathbf{x}_i)$ is considered from a pool $P(g)$ of generation g, and is selected based on Boltzmann probability distribution function. Let, f_{mrp} be the fitness value of the chromosome selected in the most recent past. If the next chromosome is having fitness value $f(\mathbf{x}_i)$ such that $f(\mathbf{x}_i) > f_{mrp}$, then it is selected. Otherwise, it is selected with Boltzmann probability

$$Pr = exp[-(f_{mrp} - f(\mathbf{x}_i))/T] \quad (1)$$

where $T = T_0(1-\alpha)^k$ and $k = \left(100\frac{g}{G}\right)$. g is the current generation number, and its maximum value is represented by G. The value of α can be chosen from the interval $[0, 1]$, and

TABLE I
SAMPLE EXECUTION OF STOCHASTIC SELECTION

Input chrom. number	Fitness of input chrom.	f_{mrp}	Chrom. no. with max. fitness	Pr	Pr_1	Selected chrom. 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

T_0 may be selected from the interval $[5, 100]$. These choices are in the line with those in SA. 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 Pr. This is significant in terms of convergence. As computation proceeds toward T = 0, the final state is reached, i.e., a near-global solution is achieved at this point.

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. One can express the new selection operator as a function of the input chromosome $(\mathbf{x})$. Let, $\mathbf{x}_{mrp}$ be the fitness of the chromosome selected in most recent past. Then the selection operator, expressed functionally, is

$$Select(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } (f_{\mathbf{x}} > f_{mrp}) \\ \mathbf{x} & \text{if } (f_{\mathbf{x}} \leq f_{mrp}) \ \wedge \ (Pr > Pr_1) \\ \mathbf{x}_{mrp} & \text{if } (f_{\mathbf{x}} \leq f_{mrp}) \ \wedge \ (Pr \leq Pr_1) \end{cases}$$

where, Pr_1 = $random[0, 1)$. Let us consider an example, for describing the operation of the selection scheme, with a population of size 6. Let, after the $(g-1)$th generation, the fitness value of the chromosome selected in most recent past is 45.0. This value is stored in the variable f_{mrp} and is used in generation g also. In any generation, the value of f_{mrp} is altered whenever a chromosome with a greater fitness is encountered (and selected). Note that, using elitist strategy, one reserves the best chromosome, in a generation, along with its fitness value in the very first location of the pool of chromosomes. Table I shows how chromosomes are selected for generation g.

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. It is interesting to note that the generations are also overlapped. This leads to reduction of appreciable amount of execution time as compared to conventional GA. The pseudo code showing the streamlined operations of selection, crossover, mutation and evaluation, within a generation of PLGA, is given in Algorithm 2 below. Note that in this algorithm, within each generation, an inner loop is inserted which considers 2 chromosomes at a time and performs operations of GA over them before the other chromosomes are taken into account.

Algorithm 2 : *Outline of Pipelined genetic algorithm.*

begin
  $g = 0$;
  create initial pool $P(g)$ and initialize temperature T;
  evaluate initial population;
  repeat
    for $i = 1$ to N in steps of 2 **do**
    begin
      select a pair of chromosomes from pool $P(g)$;
      cross the selected pair;
      mutate the crossed pair;
      evaluate the mutated pair and put in $P(g+1)$;
    end
    $g = g + 1$;
    lower temperature T;
  until convergence;
end

Note that, unlike the serial CGA, in PLGA two chromosomes are selected at a time, instead of a complete set of N chromosomes. Therefore, only two evaluated chromosomes are needed for the selection operation to start. The restriction of 2 chrosomes is implied by the fact that a crossover operation requires two selected chromosomes.

B. Basic operators of the pipelined algorithm

Using the selection strategy mentioned above, the concept of pipelining can be incorporated within the genetic algorithm framework in order to make it faster. Let us call this algorithm as pipelined genetic algorithm (PLGA). Note that it is possible to implement this algorithm using appropriate hardware for selection, crossover, mutation and evaluation operations. The functional forms of selection, crossover and mutation operations of the PLGA are formulated below.

Selection operation:

As mentioned earlier, selection operation is responsible for reproduction of better chromosomes from pool $P(g)$ into pool $P(g+1)$. In case of PLGA, the pseudo code of Algorithm 3, for selection of pairs of chromosomes for the crossover operation, is suggested .

Algorithm 3 : *Selection procedure.*

procedure selection(i, T)
begin
  for $j = i$ to $i + 1$ **do**
  begin
  if (f_{mrp} - pool[j].fitness) ≤ 0
    then select chromosome j from $P(g)$
      and update f_{mrp} to pool[j].fitness
    else
    if exp [-(f_{mrp}-pool[j].fitness)/T]> $random[0, 1)$
      then select chromosome j from $P(g)$
        and update f_{mrp} to pool[j].fitness
      else select chromosome corresponding to f_{mrp}
  end
end

Crossover operation:

The crossover operation takes two selected chromosomes (called parents) p_1 and p_2 from the current generation pool $P(g)$. and exchanges genetic informations between them to produce two offspring (called child) c_1 and c_2 for the next generation pool $P(g+1)$. Crossover takes place with a probability P_c. In the present implementation, single point crossover is used as mentioned in [25]. The crossover procedure is provided in Algorithm 4 below.

Algorithm 4 : *Crossover procedure.*

procedure crossover(p_1, p_2, c_1, c_2)
begin
  if $random(0,1) \leq P_c$
  begin
    $crossover\_point = random(chromosome\_length)$
    for $i = 1$ to $chromosome\_length$ **do**
    begin
      if $i \leq crossover\_point$
      begin
        new_pool[c_1].bit[i] = pool[p_1].bit[i]
        new_pool[c_2].bit[i] = pool[p_2].bit[i]
      end
      else
      begin
        new_pool[c_1].bit[i] = pool[p_2].bit[i]
        new_pool[c_2].bit[i] = pool[p_1].bit[i]
      end
    end
  end
end

Mutation operation:

Mutation operation introduces new genetic structures within the crossed child chromosomes. Mutation for binary chromosomes is done by bit complementation. Whether a bit will be mutated or not is determined by the mutation probability P_m. This type of mutation is called uniform mutation [25], [26]. After the mutation operation a child chromosome is evaluated and enters the pool for generation $P(g+1)$, if selected. A pseudocode for the mutation operation is presented in Algorithm 5.

Algorithm 5 : *Mutation procedure.*

procedure mutation (c_1, c_2)
begin
  for each of the chromosomes $c \in \{c_1, c_2\}$ **do**
  begin
  for $i = 1$ to $chromosome\_length$ **do**
    if $random(0,1) \leq P_m$
      pool[c].bit[i]= complement(new_pool[c].bit[i])
    else pool[c].bit[i]= new_pool[c].bit[i]
  end
end

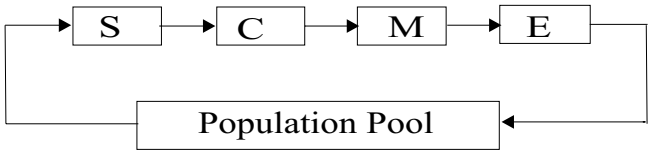

Fig. 1. Pipeline stages for the GA. Here S, C, M and E stand for selection, crossover, mutation and evaluation respectively

Note that, using the above procedures, the structural parallelism of GA can be converted into a pipelined framework, where the four basic operations of GA can be performed in a streamlined and overlapped fashion through a pipe of four stages.

IV. Design of The Pipeline Architecture

In this section, the structural parallelism of GA that are hidden in its strictly serial use of genetic operators can be explored which streamline these operators so that they can function in an overlapped fashion. 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 is the chance of incorporating more population and thus increasing the diversity among them, when executed in a hardware platform.

A. Architecture

There are four major functions that are identified as: (i) selection, (ii) crossover, (iii) mutation and (iv) evaluation. Thus a four stage pipeline can be constructed, where each stage corresponds to one of these functions, as shown in Figure 1.

The selection operation should be performed in two parallel units so that it can provide two selected chromosomes to the crossover unit in due time. Mutation and fitness evaluation should be done in multiple units that operate in parallel as shown in Figure 2. The number of units for mutation is determined by the length of a chromosome. In most of the cases, evaluation is a time consuming process compared to the other operations, and the number of units is determined by the complexity of fitness function.

Let us assume that a chromosome consists of l number of genes each of which represents a variable x_i ($1 \leq i \leq l$) in binary form having equal number of bits. 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. Let us 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, for efficient utilization of resources, one needs to use s, m and e pairs of units for selection, mutation and evaluation respectively. Here chromosomes are counted as pairs because one crossover needs two selected chromosomes. But s, m and e may not be integers. Thus for one crossover unit, the number of pairs of units to

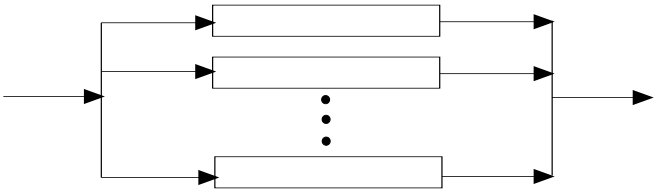

Fig. 2. Multiplicity of any particular unit in the pipeline

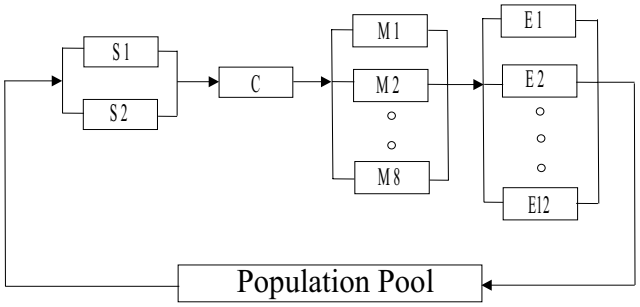

Fig. 3. A possible design of the pipeline for an example one-dimensional problem

be allocated at the respective stages will be the next nearest integral values, i.e., $\lceil s \rceil$, $\lceil m \rceil$ and $\lceil e \rceil$. For sake of simplicity, let us consider, from now on, $s = \lceil s \rceil$, $m = \lceil m \rceil$ and $e = \lceil e \rceil$. From the above ratio, it is clear that, if the crossover unit takes 1 T-cycle to perform one crossover, the selection, mutation and evaluation units take s, m and e T-cycles to perform one selection, mutation, distribution and evaluation operations respectively. Thus for proper and efficient utilization of the resources, s, m and e pairs of respective units should be used for one crossover unit. An example of such configuration is shown in Figure 3, where it is assumed s = 1, m = 4 and e = 6 for a single crossover unit. This means that, the time required to compute one mutation is four times to that required for a selection or a crossover and one should use four pairs of mutation units for one crossover unit. Similar is the case for the evaluation units. In practice, for complex problems, the values of m and e may become very large and variable. So in such situations, we need to incorporate large and variable number of units at the corresponding stages.

Note that, in Figure 3, a buffer (population pool) is used to store children chromosomes along with their fitness values. By using multiplicity of processing elements, we actually get a pipeline where each chromosome will occupy a particular stage for one T-cycle on the average. The pipeline shown in Figure 3 is a simple one, and is designed for an example one dimensional problem only. The reservation table for the example pipeline (Figure 3) can be constructed as shown in Table II. In this table, T-cycles are numbered from 01-17, and chromosomes are numbered from 01-12 respectively.

A pool of 12 chromosomes is considered in this example in order to keep the table smaller. The entries in mth column (i.e., at mth T-cycle) indicate the chromosomes being processed by a stage corresponding to the row. For example, at the 1st T-cycle, chromosomes 1 and 2 in the population are being processed at the stage S, while all other stages remain

TABLE II
RESERVATION TABLE FOR THE EXAMPLE PIPELINE

Stage	T-cycles and associated pairs of chromosomes 01-02-03-04-05-06-07-08-09-10-11-12-13-14-15-16-17
S	01-03-05-07-09-11 02-04-06-08-10-12
C	01-03-05-07-09-11 02-04-06-08-10-12
M	01-01-01-01 02-02-02-02 03-03-03-03 04-04-04-04 05-05-05-05 06-06-06-06 07-07-07-07 08-08-08-08 09-09-09-09 10-10-10-10 11-11-11-11 12-12-12-12
E	01-01-01-01-01-01 02-02-02-02-02-02 03-03-03-03-03-03 04-04-04-04-04-04 05-05-05-05-05-05 06-06-06-06-06-06 07-07-07-07-07-07 08-08-08-08-08-08 09-09-09-09-09-09 10-10-10-10-10-10 11-11-11-11-11-11 12-12-12-12-12-12

idle. Once the pipe is filled up, all the stages will be active simultaneously.

It should be noted that, as problem dimension increases, more mutation and evaluation units must be added in parallel. For example, for a two variable fitness function of similar form, 8 pairs of mutation units and 12 pairs of evaluation units are needed. However, the other two stages may remain unaltered.

We can use lesser number of units at the mutation and evaluation stages. In such a situation, the frequency of pipeline initiation will be reduced and hence, lesser speedup results. If the number of units in the M and E stages be reduced by a factor $r = 2$, the pipeline needs to be initialized at intervals of 2 T-cycles. The modified reservation table, for this situation, is shown in Table III. As seen from this table, now 24 T-cycles are needed to complete a generation (with 12 chromosomes only) instead of 17 T-cycles (as shown in Table II).

TABLE III
MODIFIED RESERVATION TABLE FOR THE EXAMPLE PIPELINE WITH REDUCED STAGE MULTIPLICITY

Stage	T-cycles and associated pairs of chromosomes 01-02-03-04-05-06-07-08-09-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24
S	01-01-03-03-05-05-07-07-09-09-11-11 02-02-04-04-06-06-08-08-10-10-12-12
C	01-01-03-03-05-05-07-07-09-09-11-11 02-02-04-04-06-06-08-08-10-10-12-12
M	01-01-01-01 02-02-02-02 03-03-03-03 04-04-04-04 05-05-05-05 06-06-06-06 07-07-07-07 08-08-08-08 09-09-09-09 10-10-10-10 11-11-11-11 12-12-12-12
E	01-01-01-01-01-01 02-02-02-02-02-02 03-03-03-03-03-03 04-04-04-04-04-04 05-05-05-05-05-05 06-06-06-06-06-06 07-07-07-07-07-07 08-08-08-08-08-08 09-09-09-09-09-09 10-10-10-10-10-10 11-11-11-11-11-11 12-12-12-12-12-12

B. Speedup

Assuming identical stage time at each stage, the speedup of a general pipeline is defined as

$$S = \frac{T_{NP}}{T_P} \tag{2}$$

where T_{NP} ($= nk$, $n =$ number of executions and $k =$ number of stages) and T_P ($= n + k - 1$) are computation times (in terms of number of T-cycles) required for non-pipelined and pipelined systems respectively. Equation (2) is valid for the suggested pipeline as well, because the difference in stage times are tackled by using multiple units in parallel. In our case, it is assumed that the crossover time is equal to one T-cycle. As selection time is s times that of the crossover time, s pairs of selection units are considered in parallel, in order to make the average selection time per chromosome as one T-cycle. Similarly, for mutation and evaluation stages, m and e pairs of units respectively are required. This is the ideal hardware configuration. In our case, n = population size × number of generations.

For the pipeline, $(s + 1 + m + e)$ T-cycles are required to get the first pair of children chromosome. 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 $\left(\frac{n}{2} - 1\right)$. Since one pair of chromosomes is always needed for a crossover operation, the value of s is set to 1. Thus, the total number of T-cycles required for the pipeline is

$$T_p = 1 + 1 + m + e + \left(\frac{n}{2} - 1\right) = 1 + m + e + \frac{n}{2}. \tag{3}$$

For a non-pipelined system configured with the same multiplicity of stages (as that of the pipelined one), the number of T-cycles is

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

Consider the pipeline of Figure 3, where s = 1, m = 4 and e = 6. Here,

$$T_P = 11 + \frac{n}{2}.$$

So, the speedup attained is

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

Since n is large compared to 11, $S \approx 4$. This is the *ideal speedup*.

Since, in case of non-pipelined execution, stage times cannot be directly measured in units of T-cycles, Equation 4 should be replaced by Equation 5, given below, for calculation of equivalent non-pipelined execution time in units of T-cycles.

$$T_{NP} = \frac{n}{2} \times (S_t + C_t + M_t + E_t) \times \frac{1}{One\ T - cycle\ time}. \tag{5}$$

As mentioned earlier, we can use less number of units at mutation and evaluation stages. Let, for any arbitrary configuration, $m\prime$ and $e\prime$ 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, $m\prime < m$ and $e\prime < e$, i.e., the number of units at the mutation and evaluation stages are less than that needed for full multiplicity of these stages. Let $r_m = \lceil m \rceil$ and $r_e = \lceil e \rceil$, 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, distribution, 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 others are less than or equal to r. So, two different situations can arise:

- **case 1:** $r_m = r_e = r$. In this case, we can initialize the pipeline at an interval of r T-cycles, with a pair of chromosomes. And the pipeline will produce outputs (child chromosome pairs) at intervals of r T-cycles.
- **case 2:** $r_m = r$ and $r_e < r$. Here, for streamlined operation of the stages, we have to initialize the pipeline at intervals of r T-cycles and outputs are also generated at intervals of r T-cycles. However, some of the units in the E-stage will remain idle for some T-cycles. This results in a lower speedup than the full pipeline.
- **case 3:** $r_e = r$ and $r_m < r$. As in case 2, here, some of the units in the M-stage will remain idle for some T-cycles. and hence, a lower speedup is obtained.

For both $case2$ and $case3$, we obtain lesser speedups than that of a uniformly reduced pipeline. It is, however, notable that, due to initialization latency of r T-cycles, speedup is

reduced in both the cases. Again, if we employ more number of units than are necessary at a particular stage, lower speedup results.

Let r_m and r_e be the reduction factors of a reduced pipeline at the mutation, distribution and evaluation stages. For such a reduced system we get

$$T_P = 1+1+m+e+\left(\frac{n}{2}-1\right)\times r = 2+m+e+\left(\frac{n}{2}-1\right)\times r \quad (6)$$

and

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times r_m+\frac{n}{2}\times r_e \quad (7)$$

or,

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times \lceil\frac{m}{m\prime}\rceil+\frac{n}{2}\times \lceil\frac{e}{e\prime}\rceil \quad (8)$$

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

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times r+\frac{n}{2}\times r \quad (9)$$

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

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

and

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times 2+\frac{n}{2}\times 2 = 3n \quad (11)$$

Therefore, speedup $S \approx 3$ for $n >> 10$. On the other hand, if $r_m = 2$ and $r_e = 1$, then also $r = 2$, but now we have,

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times 2+\frac{n}{2} = 2.5n \quad (12)$$

Similar will be the situation with $r_m = 1$ and $r_e = 2$.

Further simplification for a fixed hardware setup is also possible. Let us consider that we have a fixed hardware setup having $m\prime = 10$ and $e\prime = 10$. Also let for a particular problem, the requirements are $S_t = 10, C_t = 10, M_t = 200$ and $E_t = 400$. Assuming one T-cycle = 10 units of time, we get, $s = 1, c = 1, m = 20$ and $e = 40$. Here, $r_m = 2$ and $r_e = 4$. So, the pipeline should be initialized at intervals of 4 T-cycles. Here,

$$T_P = 62+\left(\frac{n}{2}-1\right)\times 4 = 2n+58 \quad (13)$$

and

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times\frac{20}{10}+\frac{n}{2}\times\frac{40}{10} = 4n \quad (14)$$

Therefore, speedup $S \approx 2$ for $n >> 58$.

However, if we have another problem where, $S_t = 10, C_t = 10, M_t = 200$ and $E_t = 200$ and hence $s = 1, c = 1, m = 20$ and $e = 20$, then the pipeline becomes a uniformly reduced one with reduction factor $r = 2$. In this case,

$$T_P = 62+\left(\frac{n}{2}-1\right)\times 2 = n+58 \quad (15)$$

and

$$T_{NP} = \frac{n}{2}+\frac{n}{2}+\frac{n}{2}\times\frac{20}{10}+\frac{n}{2}\times\frac{20}{10} = 3n \quad (16)$$

Therefore, speedup becomes $S \approx 3$ for $n >> 58$.

Note also that in the non-pipelined system, termination of the process is checked at the end of each generation. On the other hand, for pipelined system, once the pipeline is activated, there is no necessity of such termination checking. Moreover, the pipeline is designed in such a manner that one can add or remove a number of units from M and E stages according to the complexity of the problem concerned. This will ensure maximum use of all the units keeping the speedup at the same level, and hence ensures scalability of the proposed design.

V. A Possible Hardware Implementation

Recently hardware platforms for GAs are being developed. Without a proper hardware platform it is not possible to realize the true benefits of PLGA. In the proposed pipeline, different number of units are needed at different stages of the pipeline. Here, two selection units and one crossover unit are used, because a crossover needs two selected chromosomes. Number of mutation and evaluation units depend on the corresponding computational complexities at the respective stages. Use of general purpose computers for each of the stage units may not be cost-effective. Hence, use of simple and dedicated hardware units is necessary. In the following subsections, some of the hardware design schemes for different units is presented. The evaluation unit, presented here, is absolutely suitable for the proposed pipelined design.

A. Selection, crossover and mutation units

Hardware selection units for binary tournament selection have been proposed in literature. However, a hardware unit for the SA-selection is needed. Although it seems to be a bit complicated, it can be developed with suitable hardware components or using reconfigurable memory devices (FPGAs). Bitwise crossover and mutation units are very easy to implement, and a number of such devices are found in literature. But, regarding evaluation units, many efforts to develop reconfigurable devices are found, one for each different application. Thus, it is necessary to develop a general hardware evaluation unit. One such unit has been presented in [23],[24]. For convenience, the said scheme for hardware evaluation and the corresponding hardware unit are illustrated here briefly in the following subsection. For detailed operational features, the cited reference may be consulted.

B. A simple stack based hardware evaluation unit

A simple stack based hardware evaluation unit can be used to evaluate a fitness expression, if it is assumed that the expression is converted to its postfix form a priori. Postfix evaluation technique is generally employed in simple calculators where numerical values of the operands are present in the expression itself.

In the circuit used here, the symbolic postfix expression is maintained in a buffer, and actual numeric parameter values replace the symbolic parameters for the purpose of evaluation. A diagram of the proposed scheme is shown in Figure 4. In this figure, x_i represents the ith symbolic variable and v_i represents its numeric value.

The hardware is designed keeping it in mind that, in a GA process, the same function is evaluated for a number of times with different operand values at each instance. Here, the

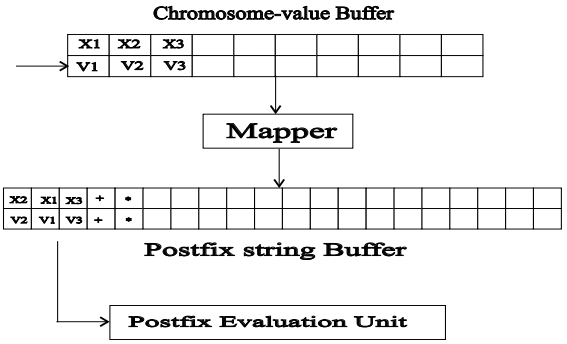

Fig. 4. Postfix evaluation scheme for GA fitness function

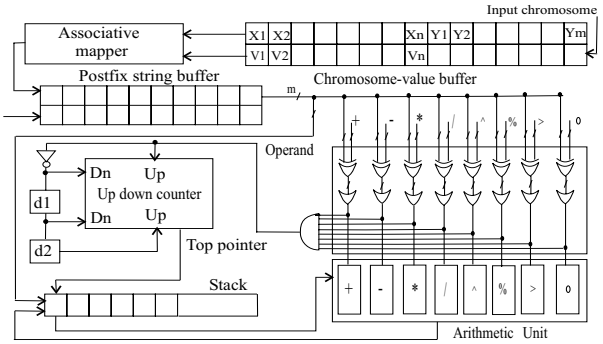

Fig. 5. Evaluation unit for general optimization

postfix expression is generated before the GA process starts and loaded into the postfix buffer. The symbolic chromosome is stored in the chromosome-value buffer beforehand. Each time a numeric chromosome enters the hardware unit (in the value buffer), a mapping unit replaces the symbolic operands in the postfix expression with the corresponding numeric value. The final numeric postfix expression can be easily evaluated in the simple hardware unit.

The evaluation unit described above is able to handle only one simple algebraic expression. In [23], it is shown that by decomposing a complex expression into a number of simpler expressions and arranging them in a hierarchy, evaluation of any general expression can be performed in this circuit. Here, the evaluation scheme for any general expression is described briefly.

In the circuit (Figure 5), a chromosome-value buffer and a postfix expression buffer are used. The chromosome-value buffer is intended to hold the symbolic chromosome vector $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$ and the corresponding numeric value vector $\mathbf{v} = \{v_1, v_2, \cdots, v_n\}$. There are extra space in the chromosome-value buffer to hold extra values. These values are represented by components of a symbolic vector $\mathbf{y} = \{y_1, y_2, \cdots\}$, where, y_is hold computed values of different sub-expressions. Initially the value buffer locations for this **y** vector are undefined, but as soon as a sub-expression corresponding to a y_i is evaluated, its value is placed in the value buffer so that it can be used for evaluating the next higher levels of sub-expressions. A separate block of buffers is also required to hold a number of postfix sub-expressions. These buffers hold sub-expressions for different levels in the hierarchy in a suitable order. One of these expressions are taken, at a time, to the postfix buffer for execution. However, to simplify the hardware, it is assumed that the main processor is responsible to send the sub-expressions of the hierarchy in proper order, to extract the computed values of these sub-expressions and to load them in the chromosome-value buffer (y-locations). In the hardware, one can add other evaluation units like those for modulus computation(%), comparison($>$) and others(o), as shown in Figure 5.

VI. Experimental Results

Here, the effectiveness of PLGA, along with its comparison with conventional serial GA (CGA) and parallel GA (PGA), are demonstrated on various benchmark functions. This section has two parts. In the first part, the selected benchmark functions are described. Results of comparison of the selection schemes used for PLGA (SA selection) and CGA (Roulette-wheel selection) are provided in the second part. This part also includes comparative measures of speedup and efficiency of PLGA with respect to CGA and PGA.

A. Some Benchmark Optimization Problems

The selected benchmark functions are available in literature [2], [3], [27]. Since basic GA considers only binary representation of a chromosome, ten functions are selected whose arguments can be represented in binary in a straightforward manner. These functions may be found in [2], [3], [27]. Out of these ten functions, the function f_B is used by Michalewicz, f_A and f_C are De Jong's test functions. f_D is a 2-dimensional complicated sine function, f_E is the Goldstein-Price function, f_F is the Rastrigin's function. f_G and f_H are Rosenbrock's function and Ackley's function respectively and f_I and f_J are Schwefel's test functions. Note that f_B, f_F, f_G, f_H, f_I and f_J are multimodal in nature. Except f_A, f_B, f_D and f_E all other functions are of variable dimensions and can be tested by varying their dimension (or complexity).

1. A binary function:

$$f_A(x) = \sum_i 2^i b_i$$

where $b_i \in \{0, 1\}$ indicates the ith bit in the binary representation of x. This function was originally used by Goldberg. The function is considered over the interval $[0, 2^{25} - 1]$. The normalized range of x is $[0, 1]$. Normalization is done by dividing any x-value by 2^{25}. The function has only one parameter x. The minimum value of this function is 0 at $x = 0$.

2. A simple $sine$ function:

$$f_B(x) = x.sin(10\pi x) + 1.0$$

This multimodal function was originally used by Michalewicz [3]. Here also the function contains a single variable x which can be encoded as a chromosome of 25 bits. The problem is to find x in the interval $[-1, 2]$ which maximizes the function $f(x)$. It is known that the maximum value of $f_B(x) \approx 2.85$ at $x \approx 1.85$.

3. *Sphere Model* function:

$$f_C(\mathbf{x}) = \sum_{i=1}^{3} {x_i}^2$$

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

4. A complex $sine$ function:

$$f_D(x_1, x_2) = \sin^2(x_1) + \sin^2(x_2) - 0.1\exp(-(x_1)^2 - (x_2)^2)$$

The range of x_1, x_2 is $-10 \leq x_1, x_2 \leq 10$. This function attains its minimum value of 0.9 at (x_1, x_2) = $(0, 0)$.

5. $Goldstein - Price$ function:

$$\begin{aligned} f_E(x_1,x_2) \quad &= \left[1 + (x_1 + x_2 + 1)^2 \times \right. \\ &\quad \left. (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \\ &\quad \times \left[30 + (2x_1 - 3x_2)^2 \times \right. \\ &\quad \left. (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right] \end{aligned}$$

The range of x_1, x_2 is $-2 \leq x_1, x_2 \leq 2$. This function attains its minimum value of 3.0 at (x_1, x_2) = $(0, -1)$.

6. $Rastrigin's$ function:

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

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 $x_i = 0, \forall i$.

7. $Rosenbrock's$ function:

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

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

8. $Ackley's$ function:

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

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

9. $Schwefel's$ function 1:

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

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

10. $Schwefel's$ function 2:

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

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

Note that, like f_B and f_F, f_H is also multimodal. All the functions except f_B, f_G and f_H, have their minimum at $x_i = 0, \forall\, i$. The function f_B attains its maximum at $x = 1.85$. The function f_E have its minima at $(x_1, x_2) = (0, -1)$ and the function f_G has a minima at $x_i = 1, \forall\, i$. For all these functions, variables are coded using 25 bit binary code. However, larger binary codes may be used to increase the accuracy of the numbers represented.

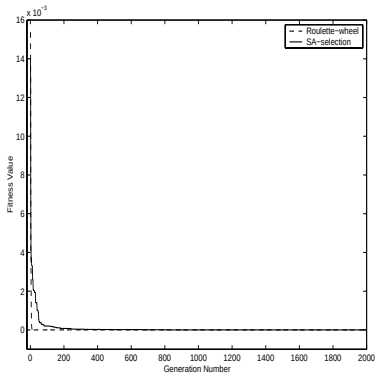

Fig. 6. Convergence of f_A using roulette-wheel and SA-selection

B. Results

For empirical investigations with both conventional GA (CGA) and PLGA, the same values for the control parameters are used, viz., population size, $N = 50$, P_c = 0.6, P_m = 0.05, α =0.05 and T_0 = 50. The individual values in the chromosomes are encoded in 25-bit binary for all the selected problems. The genetic search spaces for individual functions are already mentioned in Section VI-A. Experiments are done using the roulette wheel selection scheme for conventional GA and the SA-selection scheme for PLGA. Regarding mutation, the bit complementation strategy mentioned in [25] is used for both CGA and PLGA.

Comparison of SA-selection and Roulette-wheel selection schemes

For all the selected benchmark functions simulation experiments are performed to illustrate the rate of convergence of SA-selection and Roulette-wheel selection schemes. Figures 6 to 15 show the rate of convergence of CGA (using Roulette-wheel selection) and PLGA (using SA-selection), when they were executed for 2000 generation, for the selected benchmark problems. The variation of fitness values are shown against generations number. Functions f_C and f_F are executed for high dimensions (=125) also. Due to higher problem complexity, convergence rates are slower in these cases. Hence, curves are plotted for 10,000 number of generations. The rate of convergence for these high dimensional benchmark functions are shown in Figures 16 and 17 respectively. The observations show the usability of the SA-selection scheme for GAs.

Table IV provides the optimal values for the objective functions which are reached by the respective algorithms when executed for 2000 generations, and they are found to be almost equal.

PLGA vs. CGA

Speedups of PLGA compared to CGA are measured in two different situations, viz., when a similar hardware platform (like the pipeline itself) is used for CGA, and when a single uniprocessor is used only. For the former situation, results are

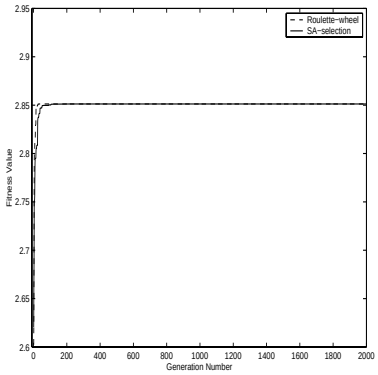

Fig. 7. Convergence of f_B using roulette-wheel and SA-selection

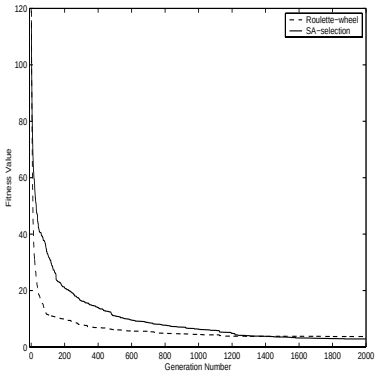

Fig. 11. Convergence of f_F using roulette-wheel and SA-selection

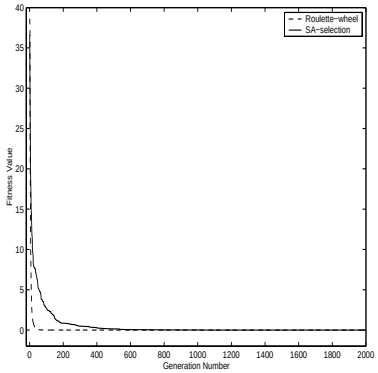

Fig. 8. Convergence of f_C using roulette-wheel and SA-selection

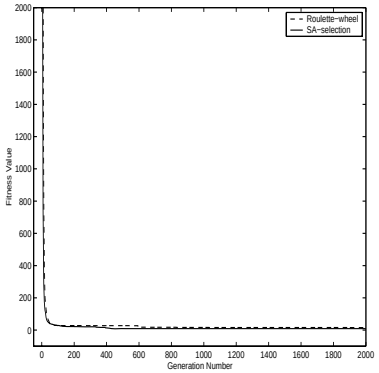

Fig. 12. Convergence of f_G using roulette-wheel and SA-selection

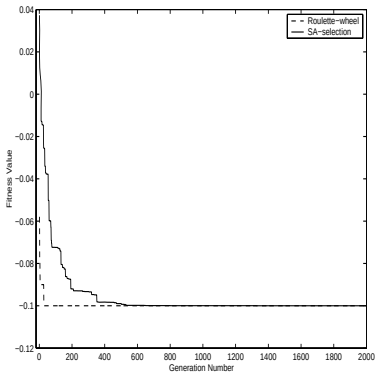

Fig. 9. Convergence of f_D using roulette-wheel and SA-selection

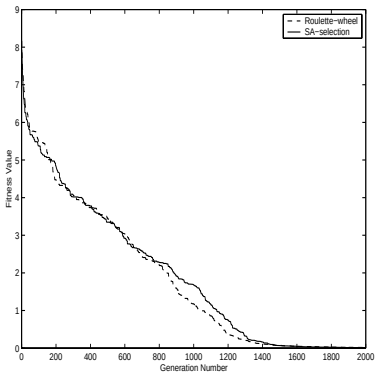

Fig. 13. Convergence of f_H using roulette-wheel and SA-selection

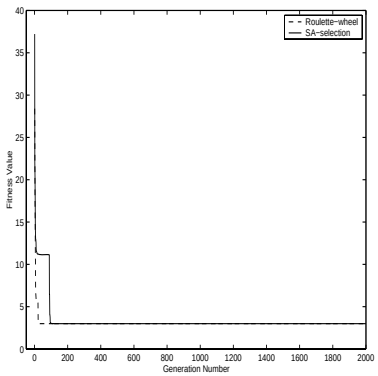

Fig. 10. Convergence of f_E using roulette-wheel and SA-selection

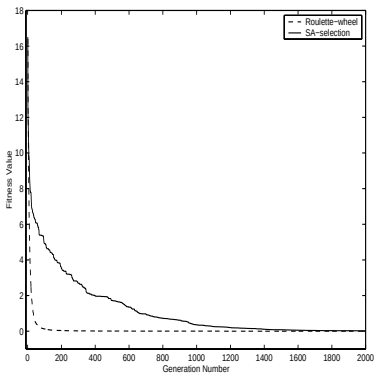

Fig. 14. Convergence of f_I using roulette-wheel and SA-selection

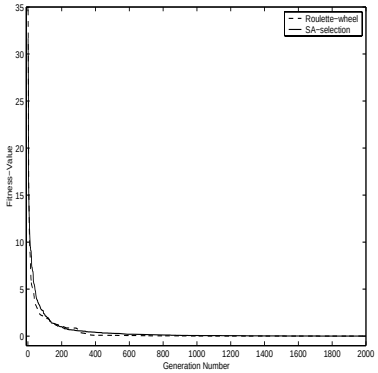

Fig. 15. Convergence of f_J using roulette-wheel and SA-selection

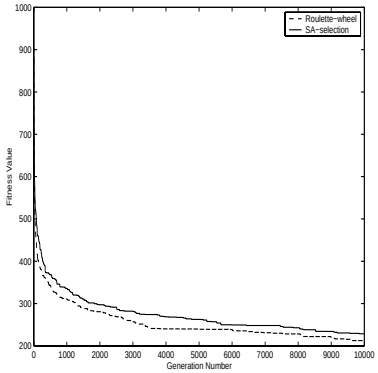

Fig. 16. Convergence of f_C using roulette-wheel and SA-selection when problem dimension = 125

shown in Table V and for the later results are presented in Table VI. The stage times given in Table V are proportional to the time required for two selections, one crossover, two mutations and two evaluations respectively. This is because, this time PLGA is executed for 1000 generations over a population size of 50, and sum of the corresponding stage times are measured. So it can be assumed, without any loss of generality, that entries in Table V are proportional to the time required for one selection, one crossover, one mutation and one evaluation respectively.

It is observed that the stage times of crossover, mutation and evaluation functions increases almost proportionately with

TABLE IV

PERFORMANCES OF STOCHASTIC AND ROULETTE-WHEEL SELECTION SCHEMES ON FUNCTIONS $f_A - f_J$ (AVERAGED OVER 50 RUNS EACH WITH 2000 GENERATIONS). "MEAN BEST" INDICATES THE MEAN BEST FUNCTION VALUES AND "STD DEV" MEANS STANDARD DEVIATION

Function	Pipelined GA		conventional GA	
	Mean Best	Std Dev	Mean Best	Std Dev
f_A	0.0	0.0	0.0	0.0
f_B	2.851212	0.0	2.851212	0.0
f_C	0.0	0.0	$3.0 \times E-10$	$6.0 \times E-10$
f_D	0.9	$2.5 \times E-9$	0.84	$4.8 \times E-2$
f_E	3.0	0	3.0	0
f_F	2.5626	1.3813	3.0030	1.2641
f_G	0.0722	0.005	0.0735	0.006
f_H	0.02	0.0003	0.02	0.0002
f_I	$2.90 \times E-3$	$6.50 \times E-4$	$3.15 \times E-3$	$9.42 \times E-4$
f_J	$2.60 \times E-3$	$4.09 \times E-4$	$5.01 \times E-2$	$7.85 \times E-2$

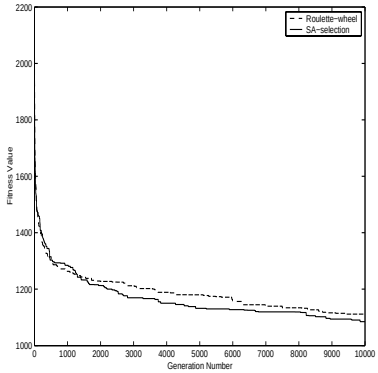

Fig. 17. Convergence of f_F using roulette-wheel and SA-selection when problem dimension = 125

increase in the number of variables (Table V), but selection time remains almost fixed. Thus it can be said that the number of crossover, mutation and evaluation units to be used depends on the number of variables. For example, for a single variable problem, the time needed for different stages may be as follows. Let 50 units of time is needed for each of selection and crossover stages, 150 units for mutation stage and 350 units for evaluation stage. Therefore, assuming one T-cycle = 50 time units, 2 S-units, 1 C-unit, 6 M-units and 14 E-units can be used. For problems with higher dimensions, more number of M and E units should be used. The above number of units are needed at different stages of the pipeline to attain full benefit from it. However, it is already mentioned that one can allocate lesser number of different units at the cost of minor losses in speedup. For function f_A, as seen from Table V, the number of T-cycles needed for different stages may be found to be in the ratio of 1:1:2:4 (this is obtained by allocating 50 units of time to each of S and C stages and allocating required time to the M and E units in multiple of 50s), where, one T-cycle = 50 units of time. Therefore, for using this function, 2 S-units, 1 C-unit, 4 M-units and 8 E-units can be used.

For calculation of speedup, for each function, let us consider that the total number of chromosomes to be processed through the pipeline be sufficiently large. It is also considered that the non-pipelined system has the same hardware components, i.e., the stages are having the same multiplicity as those for the pipelined one. Now two different situations may occur. Considering that sufficient amount of hardware components are available for each of the stages such that one pair of chromosomes will come out from each stage on the average, at each T-cycle, the maximum speedup can be attained. Otherwise, if a fixed hardware setup is only available, for example, with 1 pair of S unit, 1 C unit, 5 pairs of M units and 10 pairs of E units, speedup may have a lower value.

In Table V, crossover time is also seen to increase with the number of variables. This is because a chromosome is represented here in an array structure. However, with a linked list representation or in case of proper hardware implementation, crossover time would become constant. So while computing speedup from experimental data, a maximum crossover time of 50 units only is considered here.

The speedup is found to be less than that obtained in Section IV-B. This is due to the fact that in computation of T_P in Section IV-B, extra times have been allocated 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). For example, the crossover unit for function f_A (Table V) needs 7 units of time, although 50 units is allocated to it. Thus the actual crossover time is very less than that is allocated. Similarly, for mutation and evaluation stages, the execution times are less than those allocated. That is, the execution time for each of mutation and evaluation, for a population of size n, is less than n T-cycles. If it is considered that selection and crossover units require half of the allocated time (i.e., 25 units), then we can write,

$$T_{NP} \approx \frac{n}{2} + \frac{n}{2} + n + n = 3n. \quad (17)$$

Thus the actual speedup that will be obtained in such a case is approximately 3, which can be easily verified from the observed results (Table V). In the following, let us show how to compute the speedup for function f_A. As mentioned earlier, for this function, $s : c : m : e$ = 1 : 1 : 2 : 4. Therefore,

$$T_P = \left[1 + 1 + 2 + 4 + \left(\frac{n}{2} - 1\right)\right] \times 50 = \left[7 + \frac{n}{2}\right] \times 50$$

and

$$T_{NP} = \frac{n}{2} \times 18 + \frac{n}{2} \times 7 + \frac{n}{2} \times \frac{62}{2} + \frac{n}{2} \times \frac{200}{4} = \frac{n}{2} \times 106$$

Thus,

$$S \approx \frac{106}{50} = 2.12$$

As mentioned earlier, in a hardware implementation, the crossover time will be same for any chromosome length. In that case, if it is seen that selection and crossover can be performed each within 25 units of time, then one can set one T-cycle to be 25 units of time and in such a situation speedup will be substantially higher than the earlier. For all of our selected functions, except f_F, the crossover time is found below 25 time units. Thus considering one T-cycle = 25, recomputed speedup for these functions are shown in Table VI.

From Tables V and VI, the speedup may seem to be relatively small. However, here it is assumed that both PLGA and CGA are executed in the same simulated pipeline. The difference between executions of PLGA and CGA in this case is that, the former allows overlapping among operations of different stages, whereas, the second does not allow this. However, if we compare speedup of PLGA executed in the proposed pipeline with respect to CGA executed on a serial uniprocessor system, the speedup will be much more. The corresponding speedups obtained (using Equations 3 and 5 and Table V) are presented in Table VII. These speedups are computed assuming $n = 10,000$.

PLGA vs. PGA

As a part of the investigations, simulation experiments are done on both PLGA and PGA for some of the selected

TABLE V

STAGE TIMES OF PLGA AND CORRESPONDING SPEEDUPS OBTAINED FOR FUNCTIONS $f_A - f_J$. THE 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_A	1	18	7	62	200	1	1	2	4	2.12
f_B	1	20	9	56	214	1	1	2	5	2.00
f_C	3	17	13	150	599	1	1	3	12	2.60
f_D	2	20	8	108	430	1	1	3	9	2.24
f_E	2	21	9	109	419	1	1	3	9	2.26
f_F	10	15	46	472	2129	1	1	10	43	2.90
f_G	3	14	19	134	539	1	1	3	11	2.53
f_H	3	16	18	134	527	1	1	3	11	2.53
f_I	3	16	10	137	481	1	1	3	10	2.40
f_J	5	16	23	216	842	1	1	5	16	2.70

TABLE VI

STAGE TIMES OF PLGA AND CORRESPONDING SPEEDUPS OBTAINED FOR FUNCTIONS $f_A - f_J$. THE TIMES (IN UNITS OF 10^4 CLOCK TICKS) SHOWN ARE PROPORTIONAL TO ACTUAL STAGE TIMES. ONE T-CYCLE = 25.

Function	Dimension	Stage Times				No. of T-Cycles				Speedup
		S_t	C_t	M_t	E_t	s	c	m	e	
f_A	1	18	7	62	200	1	1	3	8	2.83
f_B	1	20	9	56	214	1	1	3	9	2.86
f_C	3	17	13	150	599	1	1	6	24	3.20
f_D	2	20	8	108	430	1	1	5	18	2.94
f_E	2	21	9	109	419	1	1	5	17	3.06
f_F	10	15	46	472	2129	-	-	-	-	-
f_G	3	14	19	134	539	1	1	6	22	3.19
f_H	3	16	18	134	527	1	1	6	22	3.21
f_I	3	16	10	137	481	1	1	6	20	2.92
f_J	5	16	23	216	842	1	1	9	34	3.51

TABLE VII

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.

Function	Dimension	Execution Time		Speedup
		$SerialUniprocessor$	$PipelinedSystem$	
f_A	1	$287\frac{1}{2}$	$(7 + \frac{1}{2})50$	5.73
f_B	1	$299\frac{1}{2}$	$(8 + \frac{1}{2})50$	5.97
f_C	3	$779\frac{1}{2}$	$(16 + \frac{1}{2})50$	15.53
f_D	2	$566\frac{1}{2}$	$(13 + \frac{1}{2})50$	11.29
f_E	2	$558\frac{1}{2}$	$(13 + \frac{1}{2})50$	11.13
f_F	10	$2662\frac{1}{2}$	$(54 + \frac{1}{2})50$	52.67
f_G	3	$706\frac{1}{2}$	$(15 + \frac{1}{2})50$	14.08
f_H	3	$695\frac{1}{2}$	$(15 + \frac{1}{2})50$	13.86
f_I	3	$644\frac{1}{2}$	$(14 + \frac{1}{2})50$	12.84
f_J	5	$1097\frac{1}{2}$	$(22 + \frac{1}{2})50$	21.84

benchmark functions using dimension 10 for each, in order to compute the relative efficiencies of the concerned algorithms. Functions f_A, f_B, f_D and f_E are selected out of this set of experiments as their dimensionalities are small and fixed. In this case, PLGA and PGA are executed for a number of generations needed to converge to a near optimal solution. For each of the benchmarks a particular limiting value is selected as the stopping criteria.

Here, the population size is considered to be 40. For the purpose of executing PGA a four processor network is considered. The population is distributed among the four processors, each getting a subpopulation of size 10. The processors are completely connected and they can communicate chromosomes after every five 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 VIII.

A more direct way to compare the performance of a parallel

TABLE VIII
COMPARISON OF PLGA AND PGA IN TERMS OF NUMBER OF GENERATIONS NEEDED TO CONVERGE TO A CERTAIN STOPPING VALUE. "MEAN" INDICATES THE AVERAGE OF NUMBER OF GENERATIONS AND "STD DEV" MEANS STANDARD DEVIATION

Function	Dimension	PLGA		PGA		Stopping
		Mean	Std Dev	Mean	Std Dev	Fitness Value
f_C	10	180.52	42.92	406.50	61.15	0.005
f_F	10	8.06	3.08	17.42	6.58	50.0
f_G	10	11.18	2.94	20.76	5.93	500.0
f_H	10	65.38	23.86	128.98	33.54	0.005
f_I	10	134.18	32.72	284.26	36.71	0.50
f_J	10	132.02	138.19	202.40	109.82	0.50

TABLE IX
COMPARISON OF PLGA OVER PGA IN TERMS OF EFFICIENCY. DATA USED FOR PLGA AND PGA ARE TAKEN FROM TABLES V AND VIII

Function	PLGA Efficiency	PGA Efficiency
f_C	0.650	0.444
f_F	0.725	0.462
f_G	0.633	0.538
f_H	0.633	0.507
f_I	0.600	0.472
f_J	0.675	0.652

GA is to derive an efficiency measure. The expression for efficiency may be developed as follows. Let us denote the serial and parallel execution times by T_{serial} and $T_{parallel}$ respectively. It is assumed that the execution time is proportional to the number of generations (ignoring the communication overhead). Thus T_{serial} and $T_{parallel}$ may be replaced by the corresponding average number of generations given in Table VIII. Since, the coarse grain PGA is executed by equally subdividing the population among the processors, actual parallel execution time should measured as $\frac{T_{parallel}}{P}$. Here, P is the number of processors used. Now, cost of parallel execution is defined as

$$Cost = \frac{T_{parallel}}{P} \times P = T_{parallel}.$$

Efficiency of the parallel system can now be defined as

$$Efficiency = \frac{Cost\ of\ Serial\ Execution}{Cost\ of\ Parallel\ Execution}$$

$$= \frac{Average\ Number\ of\ Generations\ in\ Serial\ Execution}{Average\ Number\ of\ Generations\ in\ Parallel\ Execution}$$

For the pipelined system, the efficiency is

$$Efficiency = \frac{Speedup\ Attained}{Maximum\ Attainable\ Speedup}$$

The values of computed efficiencies of pipelined and parallel execution of GAs are listed in Table IX, which depicts the superiority of the former.

VII. CONCLUSION

A pipelined version of the well known GA, called PLGA, has been proposed in this paper. For designing PLGA, the SA-selection scheme is used which does not affect GA's basic features to reach the global optima. By use of proper hardware units, one can implement an extremely fast hardware platform for PLGA. In this regard, a possible hardware evaluation unit for general function optimization using GA is presented. In absence of a physical hardware, PLGA is executed in software, on a uniprocessor working serially and iteratively, and it is observed that with proper multiplicity of different stages a maximum speedup of 4 is attainable compared to conventional GAs executed serially using similar multiplicity of stage units. However, speedups of PLGA compared to a uniprocessor based CGA are found to be much more. The performance of PLGA is tested against a version of PGA also. It is seen that PLGA outperforms PGA in terms of efficiency measures. Although, experiments are performed using software simulation on a uniprocessor system, it is realized that synthesizing a real hardware is essentially needed and includes a part of further investigation. The authors are working in that direction.

ACKNOWLEDGMENT

This research is partly supported by a sponsored 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] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.
[4] T. Bach, F. Hoffmeister, and H. P. Schwefel, "A survey of evolution strategies," in *Proc of Fourth international conference on genetic algorithms*, pp. 2–9, San Mateo, CA: Morgan Kaufmann, 1991.
[5] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. on Syst., Man and Cybern.*, vol. 16, pp. 122–128, 1986.
[6] 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.
[7] 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.
[8] 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.
[9] R. Shonkwiler, "Parallel genetic algorithms," in *Proc. of 5th Intl. Conf. on Genetic Algorithms*, pp. 199–205, San Mateo, CA: Morgan Kaufmann, 1993.
[10] E. Cantú-Paz, "A survey of parallel genetic algorithms," tech. rep., University of Illinois, Illinois GA Laboratory, Urbana Champaign, Urbana, IL, 1997.
[11] E. Cantú-Paz, "On scalability of parallel genetic algorithms," *Evolutionary Computation*, vol. 7, no. 4, pp. 429–449, 1999.
[12] E. Cantú-Paz, *Effective and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.
[13] S. Kirkpatrik, C. Gellat, and M.P.Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
Upper Saddle River, NJ: Prentice Hall PTR, 1999.
[14] L. Yong, K. Lishan, and D. J. Evans, "The annealing evolution algorithm as function optimizer," *Parallel Computing*, vol. 21, pp. 389–400, 1995.
[15] A. Prügel-Bennett and J. L. Shapiro, "Analysis of genetic algorithms using statistical mechanics," *Physical Review Letters*, vol. 72, no. 9, pp. 1305–1309, 1994.
[16] D. E. Goldberg, "A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing," *Complex Systems*, vol. 4, pp. 445–460, 1990.

[17] B. T. Zhang and J. J. Kim, "Comparison of selection methods for evolutionary optimization," *Evolutionary Optimization*, vol. 2, no. 1, pp. 55–70, 2000.
[18] P. Martin, "A pipelined hardware implementation of Genetic Programming using FPGAs and Handle-C," tech. rep., University of Essex, Department of Computer Science, Colchester, UK, 2002.
[19] M. Tommiska and J. Vuori, "Implementation of genetic algorithms with programmable logic devices," in *Proc. of the 2NWGA*, pp. 71–78, 1996.
[20] S. D. Scott, A. Samal and S. Seth, "HGA: A Hardware-Based Genetic Algorithm", in *Intl. Symposium on Field-Programmable Gate Array*, pp. 53–59, 1995.
[21] I. M. Bland and G. M. Megson, "Efficient operator pipelining in a bit serial genetic algorithm engine," *Electronic Letters*, vol. 33, pp. 1026–1028, 1997.
[22] M. K. Pakhira and R. K. De, "A hardware pipeline for function optimization using genetic algorithms," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO - 05)*, (Washington DC, USA), pp. 949–956, 2005.
[23] M. K. Pakhira, "Postfix hardware evaluation unit for genetic algorithms: Application in fuzzy clustering," in *Proc. of Intl.conf. on Advanced Computing and Communications (ADCOM - 06)*, (Mangalore, INDIA), pp. 357–360, 2006.
[24] M. K. Pakhira, "Genetic evaluation in hardware: Application in fuzzy clustering," accepted in *Foundations of Computing and Decision Sciences*, 2007 (to appear).
[25] J. L. R. Filho, P. C. Treleaven, and C. Alippi, "Genetic algorithm programming environments," *IEEE Computer*, pp. 28–43, June, 1994.
[26] M. D. Vose, *The simple Genetic Algorithms: Foundations and Theory (Complex Adaptive Systems)*. New York: The MIT Press, 1999.
[27] D. D. Cox and S. John, "SDO: A statistical method for global optimization," in *Multidisciplinary Design Optimization (Hampton, VA), 1995*, pp. 315–329, Philadelphia, PA: SIMA, 1997.

Malay K. Pakhira received his Masters of Computer Science and Engineering degree from the Jadavpur University, Kolkata, India in 1993, and Ph. D. in Technology from the Kalyani University, Kalyani, West Bengal, India in 2005. He is currently a faculty member in the Computer Science and Engineering Department of Kalyani Government Engineering College, Kalyani, West Bengal, India at the Assistant Professor level. His research interests include *Image Processing, Pattern Recognition, Evolutionary Computation, Soft Computing and Data Mining.* He has a number of publications in various International and National journals. He is a reviewer of many International and National journals also. Dr. Pakhira is a member of the Institution of Engineers (India), Computer Society of India and the Indian Unit of the International Association of Pattern Recognition and Artificial Intelligence. Dr. Pakhira is featuring in the 2007 edition of the Marquis World Who's Who Directory for his achievements and contributions in the fields of Scientific and Technological research.

Rajat K. De is an Associate Professor of the Indian Statistical Institute, Kolkata, India. He did his Bachelor of Technology in Computer Science and Engineering, and Master of Computer Science and Engineering in the years 1991 and 1993, at Calcutta University and Jadavpur University, India respectively. He obtained his Ph. D. degree from the Indian Statistical Institute, India, in 2000. Dr. De was a Distinguished Postdoctoral Fellow at the Whitaker Biomedical Engineering Institute, the Johns Hopkins University, USA, during 2002-2003. He has about 30 research articles published in International Journals, Conference Proceedings and in edited books to his credit. He has been serving as a chair/member of Program and Organizing Committee of various national/international conferences. His research interests include bioinformatics and computational biology, soft computing, and pattern recognition.