Multiprocessor Scheduling by Simulated Evolution

Imtiaz Ahmad
Department of Computer Engineering
Kuwait University, P. O. Box: 5969, Safat 13060 Kuwait
Email: imtiaz@eng.kuniv.edu.kw

Muhammad K. Dhodhi
Ross Video Ltd., 9 Slack Road, Ottawa, ON, K2G 0B7 Canada
Email: mdhodhi@rossvideo.com

Ishfaq Ahmad
Department of Computer Science and Engineering
Box 19015, CSE, University of Texas at Arlington, Arlington, TX 76019 USA
Email: iahmad@cse.uta.edu

***Abstract*— This paper presents a variant of simulated evolution technique for the static non-preemptive scheduling of parallel programs represented by directed acyclic graphs including inter-processor communication delays and contention onto a multiprocessor system with the dual objectives of reducing the total execution time and scaling with the number of processors. The premise of our algorithm is Simulated Evolution, an effective optimization method based on the analogy with the natural selection process of biological evolution. The proposed technique, named Scheduling with Simulated Evolution (SES), combines simulated evolution with list scheduling, wherein simulated evolution efficiently determines suitable priorities which lead to a good solution by applying list scheduling as a decoding heuristic. SES is an effective method that yields reduced length schedules while scaling well and incurring reasonably low complexity. The SES technique does not require problem-specific parameter tuning on test problems of different sizes and structures. Moreover, it strikes a balance between exploration of the search space and exploitation of good solutions found in an acceptable CPU time. We demonstrate the effectiveness of SES by comparing it against two existing static scheduling techniques for the test examples reported in literature and on a suite of randomly generated graphs. The proposed technique produced good quality solutions with a slight increase in the CPU time as compared with the competing techniques.**

***Index Terms*— Software, Scheduling, Allocating Parallel Programs, Simulated Evolution**

I. Introduction

Parallel programs are typically represented by directed acyclic graphs (DAGs). In a DAG, nodes denote tasks and an arc between any two nodes represents data dependency among them. The weights associated with the nodes and the arcs of a DAG represent the computation cost and the communication cost, respectively. The multiprocessor scheduling problem is well known to be NP-complete except in a few restricted cases [1-2]. Hence, satisfactory suboptimal solutions obtainable in a reasonable amount of computation time are generally sought [3-13] by devising effective heuristics. The objective is not to propose another heuristic but to improve the effectiveness of a given heuristic.

Simulated evolution is a general purpose optimization method based on an analogy with the natural selection process in the biological environments. In biological processes, species adapt themselves to the environment as they evolve from one generation to the next. In this evolution process some of the bad characteristics of the old generation are eliminated and a new generation which is more suited to the environment is created [14-20]. Mutation and selection are the two main driving forces behind evolution. The simulated evolution models the evolution process by asexual reproduction with mutation and selection, while other probabilistic techniques, such as genetic algorithms [14], focus on recombination of different solutions via crossover and an occasional mutation. In the simulated evolution scheme mutation is the dominant operator and it is used for introducing variations into solutions. In nature, mutation refers to spontaneous and random changes in genes. The advantage is that optimization proceeds rapidly because the random distribution of new trials concentrates the computational effort on solutions that have provided evidence of success in the past [20]. Simulated evolution has been successfully applied to combinatorial optimization problems such as high-level synthesis [15], routing [16], circuit partitioning [17] and standard cell placement [18, 19]. A number of other evolutionary algorithms such as genetic algorithms [21], genetic list scheduling [22], particle swarm optimization [23], ant colony optimization [24] and artificial immune system [25] have been applied to multiprocessor scheduling problem with varying degree of success.

This paper presents an evolution-based technique, SES, applied to the problem of multiprocessor scheduling of task graphs with non-negligible inter-processor communication delays. As elaborated below, the priority assignment to tasks in list scheduling is critical in determining a good schedule. Accurate priority determination has led to a great deal of research to design efficient heuristics. In the proposed technique we apply simulated evolution to determine suitable priorities which lead to a good solution by applying the list scheduling as a decoding heuristic. SES presumes that the base heuristic has been designed to give reasonably good solutions to the problem at hand (such as the list scheduling in our case). If this heuristic is applied to a new set of problem data differing only slightly from the original problem, the resulting solution should also be a reasonably good solution to the original problem. Thus, by applying the base heuristic to problem data in the neighborhood of the original, the likelihood of good solutions enhances.

The remainder of the paper is organized as follows: the details of proposed SES technique are discussed in Section II, the performance results and comparisons are reported in Section III. Section IV concludes this paper.

II. EVOLUTION-BASED TECHNIQUE

In this section, first we formulate the scheduling problem, and then give a summary of the proposed technique followed by its detailed implementation.

A. Problem Formulation

Let $S = \{i: i = 1, \ldots, m\}$ be a set of m fully connected homogeneous processors and let the application program be modeled by a directed acyclic graph DAG = $\{j: j = 1, \ldots, n\}$ of n tasks. For any two tasks $i, j \in$ DAG, $i < j$ means that task j cannot be scheduled until i has been completed, i is a predecessor of j and j is a successor of i. Weights associated with the nodes represent the computation cost and the weights associated with arcs represent the communication cost. An example of a directed acyclic graph (DAG) consisting of 28 tasks adopted from [11] is shown in Fig. 1. The multiprocessor scheduling is to assign the set of tasks of DAG onto the set of processors S in such a way that precedence constraints are maintained, and to determine the start and finish times of each task with the objective to minimize the schedule length. We assume that the communication system permits the overlap of communication with computation and its communication channels are half-duplex. We do take care of network contention into consideration in determining the schedule length. Task execution can be started only after all the data have been received from its predecessor's nodes. Duplication of same task is not allowed. Communication is zero when two tasks are assigned to the same processor; otherwise they incur a communication cost given by the edge weight.

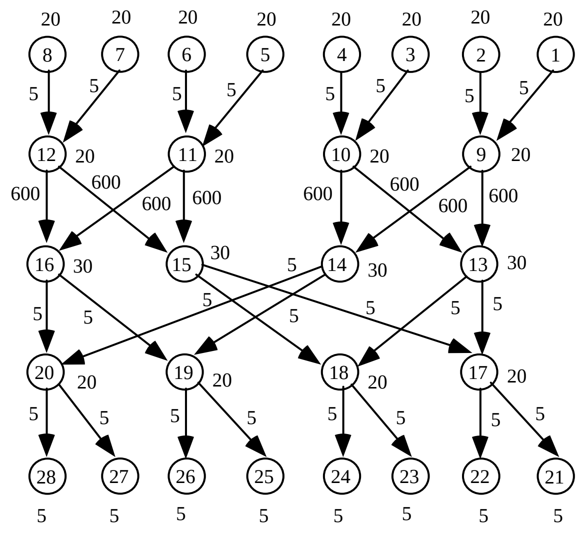
Figure 1. A directed acyclic graph for FFT-4 adopted from [11].

B. The Principle of List Scheduling

A major portion of task scheduling heuristics is based on the so called list scheduling approach [3-10]. In list scheduling each task is assigned a priority based on its estimated importance to determine the entire schedule. Initially a ready list holds all the tasks which have no predecessors. Whenever a processor becomes available, a task with the highest priority is selected from the ready list and assigned to the available processor. Two priority levels are associated with each task (node): the *t-level* and the *b-level*. The *t-level* of a node *i* is the length of the longest path between this node and an entry node in the DAG excluding the computation cost of node *i*. This level essentially determines the earliest start time of a node. The *b-level* of a node is the length of the longest path (computation + communication cost) from node *i* to an exit node. The *b-level* of a node is bounded by the critical path of the DAG. Different scheduling heuristic use the *t-level* and the *b-level* in different combinations such as smaller *t-level*, larger *b-level* or larger (*b-level* + *t-level*) etc. Detailed analysis of the design philosophies, principles behind these algorithms and performance comparisons for different classes of scheduling techniques are given in references [3, 11-12].

The priorities assignments to tasks play a key role in list scheduling. It was shown in [4] that, if priorities are assigned improperly, the resulting schedules may be worse, even if the precedence relationships are relaxed, task execution time decreased and the number of processors increased. It has been reported in [5, 6, 8] that the critical path (b-level) list scheduling heuristic is within 5% of the optimal solution 90% of the time when the communication cost is ignored, while in the worst case any list scheduling is within 50% of the optimal solution. The critical path list scheduling no longer provides the 50% performance guarantee in the presence of non-negligible communications cost [7-10]. In this paper we introduce a new technique based on simulated evolution [15-19] for the static, non-preemptive scheduling problem in homogeneous fully connected multiprocessor systems to reduce the schedule length and to increase the throughput of the system.

C. Algorithm Summary

The simulated evolution is a combination of both iterative and constructive methods [19]. To design a new evolution based technique involves various design decisions which include choosing a problem representation, deciding the decoding scheme, applying an appropriate cost function for the problem at hand, developing search operators and deciding selection mechanisms to be employed and the termination criteria. We will give details of all these questions for the proposed technique through an illustrative example. Outline of the proposed simulated evolution-based scheduling (SES) technique is shown in Fig. 2, where N_g denotes number of generations, P_m mutation rate, m number of processors, N_{gm} number of genes to be mutated and δ the maximum number of consecutive generations without improvement in the objective function.

In the proposed technique first we read the DAG of a given application program and build a database which includes the adjacency list, the number of predecessors and the number of successors for each node in the directed acyclic graph. Then we get the user defined parameters such as the N_g, P_m, N_{gm} and the parameter δ. The *t-level* and *b-level* of each node in the DAG are calculated and an initial chromosome (initial_chrom) is built based on the *b-level*. Then we copy the initial chromosome to current chromosome (current_chrom) and apply the decoding heuristic (list scheduling) to generate a solution (schedule) for the current chromosome, and its cost is evaluated by applying the function Evaluate(schedule). The schedule is stored as the best schedule, current chromosome is stored as the best chromosome and the cost is stored as the best objective. A counter called *count* which keeps track of the number of consecutive generations without improvement in the objective value is initialized to zero. Then we repeat the following until termination criteria are met. A mutation operator (function Mutate) is applied to generate offspring to form a new chromosome. Then the chromosome is decoded using the list scheduling heuristic and the solution is evaluated. If the new objective value is less than the previous one, we update the best objective, the best schedule, the best chromosome and count is set to zero. Otherwise, if *count* is less than δ, the count is incremented; else we copy the best chromosome to the current chromosome and reset count to zero. The detailed implementation of each step of the proposed algorithm is described next using the directed acyclic graph shown in Fig. 1 as an illustrative example.

D. Initial Chromosome

The chromosome representation of SES is given in Table I. Each position of the chromosome is called a gene. A gene *i* in the chromosome represents the priority of the node *i* in the directed acyclic task graph. The priority of node *i* for the initial chromosome is the length of the longest path (computation + communication cost) from node *i* to an exit node (*b-level*). In SES, only one chromosome is utilized in each generation, although each generation may consist of more than one chromosome as mentioned in [19], but it incurs more memory overhead. In our case the chromosome is represented in a problem domain instead of solution as compared with the previous approaches [15-19]. The chromosome is decoded using a fast list scheduling heuristic. This chromosome provides the priorities when we want to find a solution for the given problem using list scheduling.

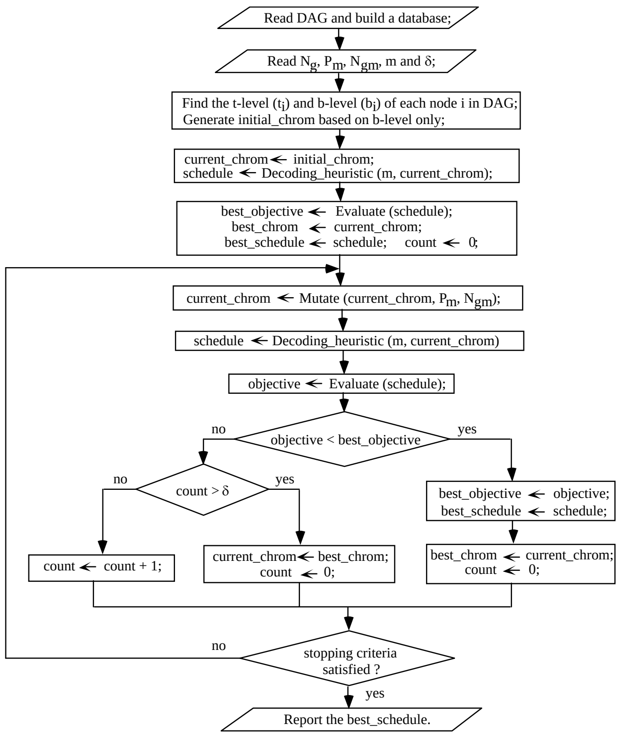
Read DAG and build a database;
Read N_g, P_m, N_{gm}, m and δ;
Find the t-level (t_i) and b-level (b_i) of each node i in DAG;
Generate initial_chrom based on b-level only;
current_chrom ← initial_chrom;
schedule ← Decoding_heuristic (m, current_chrom);
best_objective ← Evaluate (schedule);
best_chrom ← current_chrom;
best_schedule ← schedule; count ← 0;
current_chrom ← Mutate (current_chrom, P_m, N_{gm});
schedule ← Decoding_heuristic (m, current_chrom)
objective ← Evaluate (schedule);
no
objective < best_objective
yes
no
count > δ
yes
best_objective ← objective;
best_schedule ← schedule;
count ← count + 1;
current_chrom ← best_chrom;
count ← 0;
best_chrom ← current_chrom;
count ← 0;
no
stopping criteria satisfied ?
yes
Report the best_schedule.

Figure 2. Outline of the proposed scheme.

TABLE I.
Initial chromosome with node priorities.

Node number[node priority based on b-level]
1 [710], 2 [710], 3 [710], 4 [710], 5 [710], 6 [710], 7 [710], 8 [710], 9 [685], 10 [685], 11 [685], 12 [685], 13 [65], 14 [65], 15 [65], 16 [65], 17 [30], 18 [30], 19[30], 20 [30], 21 [5], 22 [5], 23 [5], 24 [5], 25 [5], 26 [5], 27 [5], 28 [5]

E. Decoding Heuristic

A scheduling algorithm consists of two steps: assigning the tasks to processors and determining the task execution ordering within a processor. Our decoding heuristic is an extended version of list scheduling [4-8] but assigns priorities and determines the execution order in the same step. The pseudo-code for the decoding heuristic is given in Fig. 3. In this heuristic we first build a task list from the given chromosome and initialize a ready list with only those tasks which do not have any predecessor. Then a task *i* from the ready list with the highest priority is selected. Then we check for each processor *j* its ready time by the procedure *find_ready_time*( ) and the data available time for task *i* on processor *j* from all the predecessor nodes of node i the procedure *find_data_available_time*(*i*, *j*) by scheduling the messages on the links and taking into consideration the contention. The early start time for processor j is the maximum of its data available time and its ready time. We check this for all the processors and find a candidate processor on which task *i* can be started the earliest. Then the task *i* is scheduled onto candidate processor. Then the task *i* is deleted from the ready list. This process is repeated on the ready list till there is no task in the ready list. A task cannot be scheduled unless its predecessors have been scheduled and data has been communicated. The *next_event* gets the earliest time when a task finishes its execution. Then, at the *next_event*, tasks whose predecessors have been scheduled are inserted into the ready list at their appropriate position by a function called *update_ready_list*( ). If there is no ready task at the *next_event*, it is then assigned the earliest time at which at least one more running task completes its execution. The algorithm repeats these simple steps until the task list becomes empty.

```
Decoding_heuristic (chromosome, m):
Build task_list from the chromosome;
ready_list ← Initialize_ready_list(task_list);
while (task_list < > null) do begin
for each task on the ready_list do begin
    Pick task i with the highest priority value;
    early_start_time = INFINITY;
    for j=1 to m begin
      pr_ready_time ← find_ready_time(j);
      data_available_time ← find_data_available_time(i, j);
      pr_start_time=Max(data_available_time, pr_ready_time);
      If pr_start_time < early_start_time then
      begin
        candidate_processor = j;
        early_start_time = pr_start_time;
      end;
    end for;
    Schedule task i onto candidate_processor;
    Delete task i from the ready_list;
end for;
next_event ← find_time_for_ready_list_update( );
ready_list ← update_ready_list(next_event);
end while;
end Decoding_heuristic.
```

Figure 3. Pseudo-code of decoding heuristic.

For the illustrative example the tasks schedule on processors and links for the initial chromosome onto a multiprocessor system consisting of two homogeneous processors is shown in Table II. The schedule length is 2585 for the initial chromosome, which is more than serial time (480) because of the communication overhead. We store this solution and the corresponding chromosome in the database before introducing any variation in the chromosome.

F. Mutation

In the simulated evolution based approach mutation is the main operator which introduces variations in the chromosome to find new points in the search space. Usually mutation is implemented by selecting one gene at random, with a mutation rate P_m, and replacing its value. But our technique alters multiple numbers of genes given by a parameter N_{gm} (the number of genes to be mutated) with a mutation rate P_m. The motivation behind mutating multiple genes at a time is to introduce enough variations into the chromosomes so that a different solution is generated by applying the decoding heuristic. If we mutate only one gene at a time, we need more generations to arrive at a good solution, hence requiring more CPU time. Pseudo-code for the mutation operator is shown in Fig. 4.

TABLE II.
Initial schedule for Fig. 1.

	Tasks scheduled on processors and links
P0	1 [0-20], 3 [20-40], 5 [40-60], 7 [60-80], 12 [85-105], 10 [105-125], 15 [1305-1335], 14 [1905-1935], 13 [2505-2535], 18 [2535-2555], 28 [2555-2560], 27 [2560-2565], 26 [2565-2570], 25 [2570-2575], 23 [2575-2580], 21 [2580-2585]
P1	2 [0-20], 4 [20-40], 6 [40-60], 8 [60-80], 11 [90-110], 9 [110-130], 16 [705-735], 20 [2510-2530], 19 [2530-2550],17 [2550-2570], 24 [2570-2575], 22 [2575-2580]
P0 xP1	(8 →12) [80-85], (5 → 11) [85-90], (4 → 10) [90-95], (1 → 9) [95-100], (12 → 16) [105-705], (11 → 15) [705-1305], (9 → 14) [1305-1905], (9 → 13) [1905-2505], (14 → 20) [2505-2510], (14 → 19) [2510-2515], (15 → 17) [2515-2520], (13 → 17) [2535-2540], (20 → 28) [2540-2545], (20 → 27) [2545-2550], (19 → 26) [2550-2555], (19 → 25) [2555-2560], (18 → 24) [2560-2565], (17 → 21) [2570-2575]

Note: 3 [20-40] means that node number 3 is scheduled on the processor from time units 20 to 40. (8 →12) [80-85] means that a message from node 8 to node 12 is scheduled on the channel from time units 80 to 85 from the processor on which node 8 is assigned to the processor on which node 12 is assigned.

In the mutation operator, we randomly select N_{gm} number of genes with probability P_m and perturb their values in the range -*t-level*$_j$/2 to *t-level*$_j$/2, where *t-level*$_j$ is the *t-level* of the node *j* in DAG with probability $P_{\tilde{m}}$. The priority of each node *j* in the chromosome is bounded in the range of *b-level*$_j$ to *b-level*$_j$ + *t-level*$_j$. If the priority value becomes more than *b-level*$_j$ + *t-level*, the priority is assigned the value *b-level*$_j$ + *t-level*$_j$. If the priority value becomes less than b-level$_j$, it is assigned the value *b-level*$_j$. The concept behind these ranges is to explore a wider space of priorities, but within the proximity to the original problem. After applying the mutation operator the old chromosome is replaced with a new one. The new generation is evaluated by applying the list scheduling heuristic. If the value of the objective function is less than the current best objective, the solution in the database is updated (i.e. the best objective is replaced with the new objective, best schedule gets new schedule).

G. Termination Criteria

We always saved the chromosome which resulted in the most recent best schedule. A counter *count* keeps track of the number of consecutive iteration without any improvement in the objective function value. When count becomes equal to δ, which is a user defined upper limit on the number of consecutive iterations without any improvement in the objective function value, we replace the current chromosome with the previous best chromosome and start the search again. This time we may go to a new neighborhood by altering the mutation rate or also by increasing/decreasing the number of genes to be mutated. Since genes are picked up randomly, we will end up in a new neighborhood. This helps to escape from local optima, thereby enabling the search to continue. The best solution discovered by the proposed technique is stored separately and is updated when a new best solution is found during the search. We terminate the search when N_g generations are completed.

```
Mutation (chromosome, $P_m$, $N_{gm}$)
for i= 1 to $N_{gm}$ do begin
  j ← random (n, 1);
  r ← random ( );
  if (r < $P_m$) then
  begin
      chromosome[j] = chromosome[j] + random(- t-level$_j$/2, t-level$_j$/2);
      if (chromosome[j] > (t-level$_j$ + b-level$_j$)) then
            chromosome[j] = t-level$_j$ + b-level$_j$;
      end if;
      if (chromosome[j] < b-level$_j$) then
            chromosome[j] = b-level$_j$;
      end if;
   end if;
end for;
end Mutation.
```

Figure 4. Pseudo-code of mutation operator.

We experimented with various numbers of generations N_g, mutation rates P_m, the number of genes to be mutated (N_{gm}) which give good results at a reasonable computation cost. The number of generations between 50-150, P_m=0.6, and δ =10 are sufficient to arrive at reasonably good solutions. The effects of these parameters on schedule length are shown in the experimental results section. The final schedule and the chromosome which resulted in the best solution by applying the proposed technique to the DAG of Fig. 1 using two processors, is shown in Table III and Table IV, respectively. Note that the communication system allows overlap of communication with computation. SES generates a schedule length of 240, where both the processors are 100 % utilized. MH [9] and DLS [10] generate a schedule length of 2585 and 2635, respectively. One of the main drawbacks of MH is that it uses static priorities, while DLS does use dynamic priorities but the exploration space is very limited. The proposed technique outperforms all other techniques and provides the shortest schedule length. This example shows that priorities determination in list scheduling is the key element to determine a good schedule and the proposed scheme is providing a technique to exploit this feature.

TABLE III.
Chromosome which resulted in the best schedule for Fig. 1.

Node number[node priority]
1 [710], 2 [710], 3 [710], 4 [710], 5 [710], 6 [710], 7 [710], 8 [710], 9 [692], 10 [687], 11 [705], 12 [688], 13 [95], 14 [642], 15 [65], 16 [509], 17 [403], 18 [87], 19[541], 20 [476], 21 [418], 22 [16], 23 [19], 24 [58], 25 [220], 26 [147], 27 [706], 28 [384]

TABLE IV.
Schedule generated by SES for Fig. 1.

	Tasks scheduled on processors and links
P0	1 [0-20], 3 [20-40], 5 [40-60], 7 [60-80], 11 [80-100], 12 [100-120], 16 [120-150], 15 [150-180], 19 [180-200], 17 [200-220], 27 [220-225], 21 [225-230], 26 [230-235], 23 [235-240]
P1	2 [0-20], 4 [20-40], 6 [40-60], 8 [60-80], 9 [80-100], 10 [100-120], 14 [120-150], 13 [150-180], 20 [180-200], 18 [200-220], 28 [220-225], 25 [225-230, 24 [230-235], 22 [235-240]
P0 xP1	(6 →11) [60-65], (1 → 9) [65-70], (8 → 12) [80-85], (3 → 10) [85-90], (14 → 19) [150-155], (16 → 20) [155-160], (13 → 17) [180-185], (15 → 18) [185-190], (20 → 27) [200-205], (19 → 25) [205-210], (18 → 23) [220-225], (17 → 22) [225-230]

III. EXPERIMENTAL RESULTS

The proposed simulated evolution-based technique, SES, for multiprocessor scheduling has been tested on a number of examples reported in literature and on a suite of randomly generated graphs. The results are very promising. The proposed evolution-based technique offers considerable improvement in the schedule length over previous work. We compared our results with the two competing techniques DLS [10] and MH [9]. For all the test examples the following values of different parameters were used: the number of generations (N_g) = 80; the mutation rate (Pm) =0.6; the number of genes to be mutated (Ngm) = 10; and the control parameter δ =10.

A. A Suite of Test Graphs [11]

As a first example, we have selected a suite of test graphs such as FFTs, trees (SUM1, SUM2) and irregular graph (IRR) used by McCreary *et al.* [11] to compare the performance of different scheduling algorithms. The FFT graph is shown in Fig. 1. The node weights for FFT-1 through FFT-3 are given in Table V. The communication cost is 25 units per edge for FFT-1 and FFT-2, while communication cost is 500 units per edge for FFT-3. The comparison of schedule lengths for all the test graphs is given in Table VI. The proposed technique gives the shortest schedule length as compared with the DLS and MH techniques for all the test cases.

TABLE V.
Nodes weight for FFTs graphs.

Node #	FFT-1	FFT-2	FFT-3
1-8	1	60	20
9-12	20	50	20
13-16	30	5	30
17-20	20	5	20
21-28	1	5	5

B. Example 2

The second example consists of three different types of directed acyclic graphs: Out-Tree, Fork-Join and Laplace Equation Solver. The comparison of schedule length with

DLS [10] and MH [9] techniques are given in Table VII. The proposed technique outperforms other techniques by providing a considerable improvement for all the test cases. This demonstrates the strength of the proposed technique to explore good solutions for different types of graph structures.

TABLE VI.
Comparison of schedule lengths for test graphs of Example 1.

DAGs	SES	DLS [10]	MH [9]
SUM-1	65	75	84
SUM-2	50	51	51
IRR	650	710	755
FFT-1	173	175	175
FFT-2	255	275	280
FFT-3	1630	2100	2570

The effects of different control parameters on the schedule length for the Out-Tree graph are shown in Fig. 5. This Figure shows that if we mutate only one gene per generation like the traditional mutation operator, the solution quality does not improve with the number of generations (N_g) because the alterations in the chromosome are not enough to generate a different solution by applying the decoding heuristic. If we increase the value of *Ngm* the solution quality improves, because we are introducing enough variations into the chromosomes, so that a different solution is generated by applying the decoding heuristic. The solution quality depends mainly on the number of generations and on the value of *Ngm*. The mutation rate (P_m) does affect the solution quality, but impact appears only with an increase in the number of generations. With the increase in the number of generations, the solution quality certainly improves, but then it requires more CPU time. To achieve a reasonable balance between the quality of solution and the computation cost, one has to select suitable values of these parameters. The values of these parameters selected for our experimentation is a reasonable choice.

TABLE VII.
Comparison of schedule lengths for test graphs of Example 2.

DAGs	SES	DLS [10]	MH [9]
Out-Tree	723	761	1070
Fork-Join	1924	3533	3406
Laplace	6390	7340	8370

C. A Suite of Random Graphs

To demonstrate the effectiveness of our SES technique, we consider a large suite of 175 of randomly generated graphs. The size of the graphs varied from 50 to 350 nodes with increments of 50. The cost of each node was randomly selected from a normal distribution with the mean equal to the specified average computation cost. The cost of each edge was also randomly generated using a normal distribution with the mean equal to the product of the average computation cost and the communication-to-computation ratio (CCR). Five different values of CCR were selected: 0.1, 0.5, 1.0, 2.0, and 10.0. For generating the random task graphs, we used another parameter called parallelism (P). This parameter determines the average number of immediate descendents for each node. Five different values of parallelism were chosen: 1, 2, 3, 4, 5. Thus, the suite consists of 25 graphs for each size. We compared our results with the two competing techniques MH [9] and DLS [10] for 8 fully connected homogeneous processors. Two types of comparisons were carried out based on the results obtained by running each algorithm on this suite of 175 graphs. First, we compared the speedup for graphs with different values of CCR and P, and then we compared the average speedup for all the random graphs generated by the technique. Finally, we compared the average run times (CPU +I/O) of these algorithms. The discussion of these comparisons is as follows:

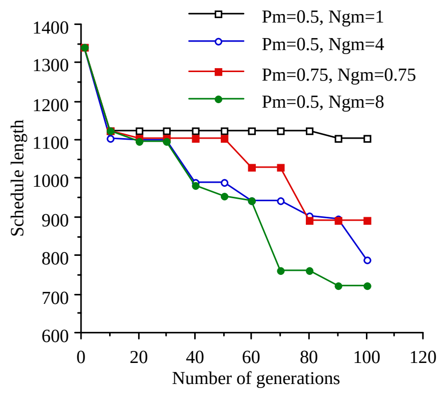

Figure 5. The effects of different parameters on the schedule length.

The typical pattern of speedup with different values of CCR is shown in Fig. 6-10. As the value of CCR is increased, the speedup decreases. The average speedup curves of SES, MH and DLS algorithms for 8 processors are given in Fig. 11. Each point in this figure is the average of 25 tests cases with various values of CCR and parallelism. We did not encounter a single instance during all the tests cases the schedule length generated by either DLS or MH are better than SES. The proposed technique outperformed both the MH and DLS algorithms. The average running times for various numbers of nodes in the task graph for 8 processors are given in Fig. 12. Each point in the figure is also the average of 25 tests cases. The running times of SES are large as compared with MH, but are comparable with DLS. The running times of SES becomes less as compared with DLS as the size of the graph is increased, since the overhead of the number of generations remains the same for all graph sizes. The proposed technique produces much better results in terms of the quality of the solution as compared with MH and DLS and with comparable running times with DLS.

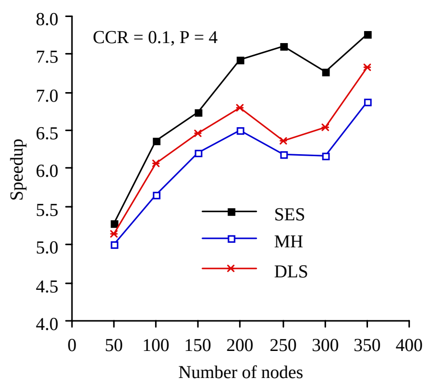

Figure 6. Speedup versus number of nodes.

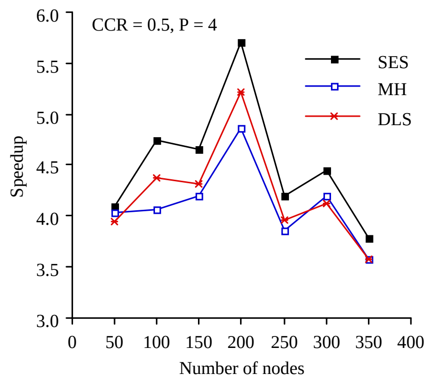

Figure 7. Speedup versus number of nodes.

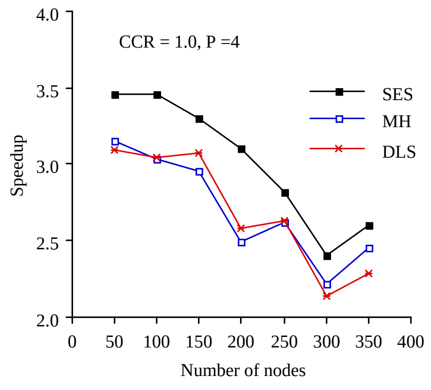

Figure 8. Speedup versus number of nodes.

IV. CONCLUSIONS

SES blends a simulated evolution and a heuristic and uses a neighborhood structure to efficiently search a large solution space in order to find the best possible solution within an acceptable CPU time. In SES, the chromosomal representation is based on problem data, and the solution is generated by applying a fast decoding heuristic (list scheduling) in order to map from problem domain to solution domain. Experimental results on test examples demonstrated that SES reduces the schedule length in a scalable fashion as compared to the existing approaches for different types of graph structures. SES can be easily extended for heterogeneous processors and can also be integrated with other heuristics.

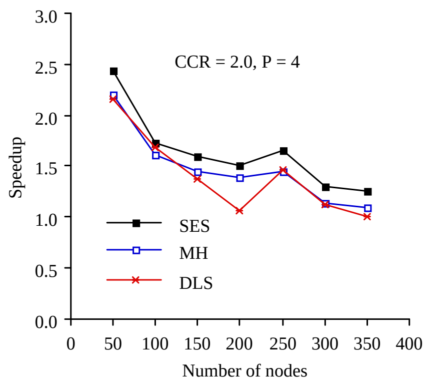

Figure 9. Speedup versus number of nodes.

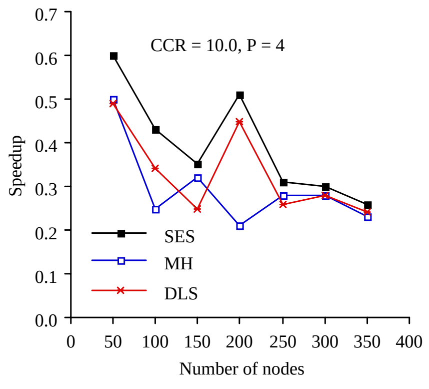

Figure 10. Speedup versus number of nodes.

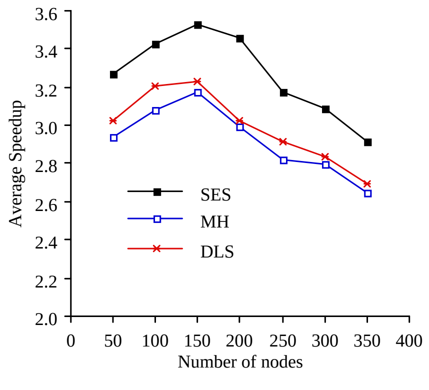

Figure 11. Average speedup for 8 fully connected processors.

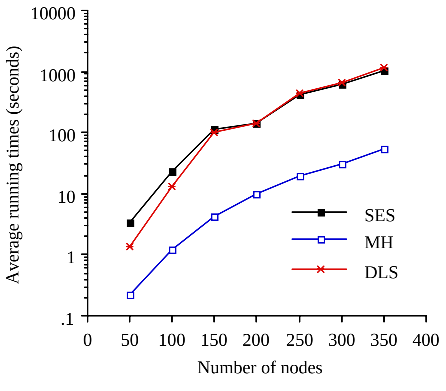

Figure 12. Average CPU running times (logscale) for 8 processors.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*, San Francisco, CA, W. H. Freeman, 1979.

[2] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, Vol. 31, No. 4, pp. 406-471, December 1999.

[3] Y. Kwok and I. Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, pp. 381-422, December 1999.

[4] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal of Applied Math.*, 17, pp. 416-429, 1969.

[5] T. L. Adam, K. M. Chandy, and J. R. Dicson, "A Comparison of List Schedules for Parallel Processing Systems," *Communications of the ACM*, Vol. 17, No. 12, pp. 685-690, December 1974.

[6] W. H. Kohler, "A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems," *IEEE Trans. on Computers*, Vol. 24, No. 12, pp. 1235-1238, December 1975.

[7] C. Y. Lee, J. J. Hwang, Y. C. Chow, and F. D. Anger, "Multiprocessor Scheduling With Interprocessor Communication Delays," *Operations Research Letters*, Vol. 7, No. 3, pp. 141-147, June 1988.

[8] T. Yang and A. Gerasoulis, "List Scheduling with and without Communication Delays," *Parallel Computing*, 19, pp. 1321-1344, 1993.

[9] H. El-Rewini and T. G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, Vol. 9, No. 2, pp. 138-153, June 1990.

[10] G. C. Sih and E. A. Lee, "Scheduling to Account for Interprocessor Communication Within Interconnection-Constrained Processor Network," *1990 International Conference on Parallel Processing*, Vol. 1, pp. 9-17, August 1990.

[11] C. L. McCreary, A. A. Khan, J. J. Thompson, and M. E. McArdle, "A Comparison of Heuristics for Scheduling DAGs on Multiprocessors," *8th International Parallel Processing Symposium*, pp. 446-451, April 1994.

[12] G. Liao, E. R. Altman, V. K. Agarwal and G. R. Gao, "A Comparative Study of Multiprocessor List Scheduling Heuristics," *Twenty-Seventh Annual Hawaii International Conference on System Sciences*, pp. 68-77, January 1994.

[13] Y Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs onto Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506-521, May 1996.

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

[15] T. A. Ly and J. T. Mowchenko, "Applying Simulated Evolution to High Level Synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 12, No. 3, pp. 389-409, March 1993.

[16] Y. L. Lin, Y. C. Hsu, and F. S. Tsai, "SILK: A Simulated Evolution Router," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 8, No. 10, pp. 1108-1114, October 1989.

[17] Y. Saab and V. Rao, "An Evolution-Based Approach to Partitioning ASIC Systems," *26th ACM/IEEE Design Automation Conference*, pp. 767-770, 1989.

[18] Y. H. Hu and C. Y. Mao, "Solving Gate-Matrix Layout Problems by Simulated Evolution," *IEEE International Symposium on Circuits and Systems*, pp. 1873-1875, 1993.

[19] R. M. King and P. Banerjee, "ESP: Placement by Simulated Evolution," *IEEE Trans. on CAD of Integrated Circuits and Systems*, Vol. 8, No. 3, pp. 245-256, March 1989.

[20] V. Nissen, "Solving the Quadratic Assignment Problem with Clues from Nature," *IEEE Trans. on Neural Networks*, Vol. 5, No. 1, pp. 66-72, January 1994.

[21] A. S. Wu, H. Yu, S. Jin, K. C. Lin and G. Schiavone, "An Incremental Genetic Algorithm to Multiprocessor Scheduling," *IEEE Trans. on Parallel and Distributed Computing*, Vol. 15, No. 9, pp. 824-834, September 2004.

[22] M. Grajcar, "Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System," *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, pp. 280-285, 1999.

[23] T. Chen, B. Zhang, X. Hao and Y. Dai, "Task Scheduling in Grid based on Particle Swarm Optimization," *The Fifth international Symposium on Parallel and Distributed Computing*, pp. 238-245, July 2006.

[24] C. Chiang, Y. Lee, C. Lee and T. Chou, "Ant Colony Optimization for Task Matching and Scheduling," *IEE-Proceedings Computers and Digital Techniques*, Vol. 153, No. 6, pp. 373-380, November 2006.

[25] H. Yu, "Optimizing Task Schedules Using an Artificial Immune System Approach," *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 151-158, 2008.

Imtiaz Ahmad received his B.Sc. in Electrical Engineering from University of Engineering and Technology, Lahore, Pakistan, a M.Sc. in Electrical Engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, and a Ph.D. in Computer Engineering from Syracuse University, Syracuse, New York, in 1984, 1988 and 1992, respectively. Since September 1992, he has been with the Department of Computer Engineering at Kuwait University, Kuwait, where he is currently a professor. His research interests include design automation of digital systems, high-level synthesis, and parallel and distributed computing.

Muhammad K. Dhodhi received his B.Sc. (with honors) in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan in 1982. He received two Master degrees, one in Computer and Systems Engineering and another in Electric Power Engineering, from Rensselaer Polytechnic Institute, Troy, New York, in 1984 and 1986, respectively. He received a Ph.D. in Electrical Engineering from Lehigh University, Bethlehem, PA, in 1992. Dr. Dhodhi has research and development experience both in the industry as well as in academia. Dr. Dhodhi is currently a Senior Member of hardware development team at Ross Video Ltd., Ottawa, Canada. In the past, he has worked as a member of technical staff for distinguished global organizations such as Nortel Networks, Lucent Technologies, IBM Corporation, and a number of start-ups such as Diablo Technologies, Silicon Optix, The VHDL Technology Group, Silc Technologies and as a Principal Consultant at Hayat ECAT, Inc. In industry, Dr. Dhodhi has been actively involved in all the phases of VLSI design, modeling and verification of System-on-Chip (SoC) devices, ASICs/FPGAs used in Networking (i.e., Multiservice Core Switching Products, Terabit Switch Routers), Video/Image Processors, and Advanced Memory Buffers for DDR2/DDR3. He has also played a key role in the development of state-of-the-art constrained random versification and assertion-based design and verification methodologies.

In academia, Dr. Dhodhi had worked as an assistant professor with the Department of Electrical and Computer Engineering, Kuwait University, from February 1993 to May 1997. He was an associate professor of electrical and computer engineering, Kuwait University, Kuwait from 1997 to 1998. Dr. Dhodhi research interests are in Wireless Sensor Networks, Hardware/Software Co-Design Verification, VLSI Design Automation and Parallel/Distributed Computing.

Ishfaq Ahmad received a B.Sc. degree in Electrical Engineering from the University of Engineering and Technology, Pakistan, in 1985, and an MS degree in Computer Engineering and a PhD degree in Computer Science from Syracuse University, New York, U.S.A., in 1987 and 1992, respectively. He is currently a professor of Computer Science and Engineering at the University of Texas at Arlington (UTA). Prior to joining UTA, he was on the faculty of Computer Science Department at the Hong Kong University of Science and Technology (HKUST). At UTA, he leads the Multimedia Laboratory and the Institute for Research in Security (IRIS). IRIS, an inter-disciplinary research center spanning several departments, is engaged in research on advanced technologies for homeland security and law enforcement. Professor Ahmad is known for his research contributions in parallel and distributed computing, multimedia computing, video compression, and security. His work in these areas is published in 200 plus technical papers in peer-reviewed journals and conferences.

Dr. Ahmad is a recipient of numerous research awards, which include three best paper awards at leading conferences and 2007 best paper award for IEEE Transactions on Circuits and Systems for Video Technology, IEEE Service Appreciation Award, and 2008 Outstanding Area Editor Award from the IEEE Transactions on Circuits and Systems for Video Technology.

His current research is funded by the Department of Justice, National Science Foundation, SRC, Department of Education, and several companies. He is an associate editor of the Journal of Parallel and Distributed Computing, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Transactions on Multimedia, IEEE Distributed Systems Online, and Hindawi Journal of Electrical and Computer Engineering. He is a Fellow of the IEEE and a member of the advisory board of Lifeboat Foundation.