A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms*

Muhammad K. Dhodhi+, Imtiaz Ahmad+ and Ishfaq Ahmad++

+: Department of Electrical & Computer Engineering, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait.
++: Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong.

ABSTRACT

Efficient assignment and scheduling of tasks of a parallel program is of prime importance in the effective utilization of multiprocessor systems. In this paper, we describe an efficient scheme for static scheduling of precedence constrained task graphs with non-negligible intertask communication onto fully connected multiprocessor systems with the objective of minimizing the completion time. Our technique is based on problem-space genetic algorithms (PSGA). It combines the search power of genetic algorithms with list scheduling heuristic in order to reduce the completion time and to increase the resource utilization. We demonstrate the effectiveness of our technique by comparing against several of the existing static scheduling techniques for the test examples reported in literature.

I. INTRODUCTION

Task assignment and scheduling can be defined as assigning the tasks of a precedence constrained task graph onto a set of processors and determining the sequence of execution of the tasks at each processor. Once an application program has been partitioned, it can be represented by a directed acyclic graph (DAG). In a DAG, nodes denote tasks and an arc between any two nodes represents data dependency among them. Weights associated with the nodes and the arcs represent the computation cost and the communication cost, respectively. A major factor in the efficient utilization of multiprocessor systems is the proper assignment and scheduling of computational tasks of a DAG among processors. The multiprocessor scheduling problem is known to be NP-complete except in a few special cases [1]. Hence, satisfactory suboptimal solutions obtainable in a reasonable amount of computation time are generally sought [2-13].

One of the major set of heuristics for task scheduling onto multiprocessors is based on list scheduling [2-8]. It has been reported in [2, 5] that the critical path 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 intertask communications delays [2-5].

In this paper we introduce a technique based on a Problem-Space Genetic Algorithm (PSGA) for the static, non-preemptive scheduling problem in homogeneous fully connected multiprocessor systems with the objective of minimizing the job completion time. In the proposed technique we use a classical GA to determine suitable priorities which lead to a good solution by applying the list scheduling as a decoding heuristic. A classical genetic algorithm consists of reproduction, crossover and mutation operators [14, 15]. It starts with an initial population of potential solutions (each solution is called a chromosome in analogy to natural genetics). It then examines the solution space for a good solution to the problem being solved by the repetitive application of genetic operators such as crossover and mutation [14, 15]. GAs require problem-specific genetic operators (crossover, mutation) to get good solutions. A new search method based on problem-space, which integrates a fast, problem specific heuristic with the local search was proposed by Storer *et al.* [16]. The key concept in this method is to base the definition of search neighborhood on a heuristic/problem pair (h, p) where h is a known fast heuristic and p represents the problem data. Since a heuristic h is mapping from a problem to a solution, the pair (h, p) is an encoding of a specific solution. By perturbing the problem p, neighborhood of solutions is generated. The problem space is generated by perturbing the problem data. The perturbation range depends on the specific problem. In order to keep the generated "dummy" problem values in proximity of the original problem values, upper and lower limits on the perturbation can be introduced. The solution set s corresponding to the problem set p can be created by the application of an heuristic, h. In the PSGA a chromosome is based on the problem data and all the genetic operators are applied in problem-space. The solution is obtained by applying a simple and fast known heuristic to map from problem-space to solution-space. PSGA is different from the hybrid GA reported in [15] which requires a uniform order-based crossover operator and a special mutation operator, while PSGA can use any crossover and mutation operator. PSGA has been applied to solve many optimization problems such as multiprocessor synthesis [17], high-level synthesis of digital systems [18, 19], and task assignment [20]. In this paper we have applied PSGA to solve the multiprocessor scheduling problem.

The rest of the paper is organized as follows: the proposed scheme for multiprocessor scheduling is

*This research is supported by Kuwiat University research grant EE-073.

discussed in Section II, experimental results are reported in Section III and finally, Section IV concludes this paper.

II. TASK SCHEDULING SCHEME

In this section, first we formulate the scheduling problem, then give an outline of the proposed scheduling scheme followed by its detailed implementation.

Let $P = \{p_i: \ i = 1, \ldots, m\}$ be a set of m homogeneous fully connected processors and let the application program be modeled by a directed acyclic graph $T = \{T_j: \ j = 1, \ldots, n\}$ of n tasks. For any two tasks $i, j \in T$, $i < j$ 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 18 tasks adopted from [12] is shown in Figure 1 (a) and a fully connected multiprocessor systems consisting of three processors (m=3) is shown in Figure 1 (b). The multiprocessor scheduling is to assign the set of tasks T onto the set of processors P 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 completion time. We assume that the communication system is contention free and it permits the overlap of communication with computation. Task execution is started only after all the data have been received from its predecessors nodes. The communication links are full duplex. Duplication of same task is not allowed. Communication is zero when two tasks are assigned to the same processor, otherwise they incur the communication cost equal to the edge weight.

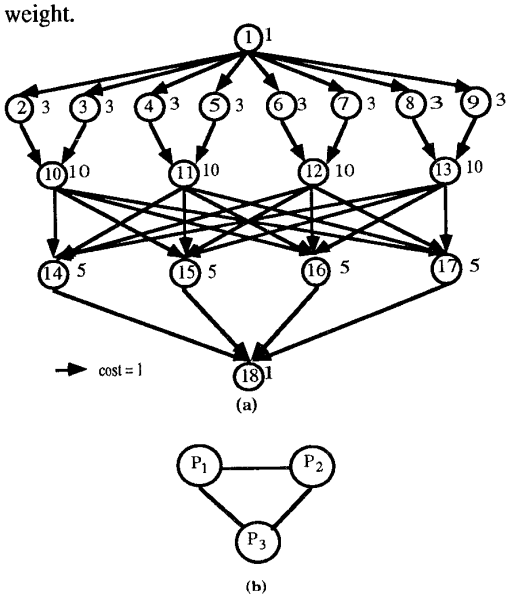

Figure 1 : (a) A directed acyclic graph [12], (b) a fully connected multiprocessor system.

Figure 2 gives the outline of the problem-space genetic algorithm-based task scheduling scheme.

Step 1. Read directed acyclic graph and build a design database. Also read population size (N_p), crossover rate (X_r), mutation rate (M_r), number of generations (N_g), and no. of processors (m).

Step 2. *Current_pop* := Gen_initial_pop (*Np*);

Step 4. **for** i := 1 to N_g **DO**

- Apply the decoding heuristic to generate solution for each chromosome in the *Current_pop*.
- Calculate the cost and the fitness for each chromosome solution in the *Current_pop*. Save the fittest current solution in the database.
- Select a pair of chromosomes from the current population probabilistically based on their fitness as parents to contribute to the next generation.
- Apply crossover and mutation to generate offspring to form a new population.
- Replace the entire current population (*Current_pop*) with the new population.

end for;

Step 5. Report the best final solution.

Figure 2: Outline of the scheduling scheme.

Initial Population

Chromosome representation and an initial population of size 4 for the proposed scheme is given in Figure 3. Each position of 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 $(W_{r0})^i$ of a node i for the first chromosome of the initial population is the total weight of the longest path from node i to exit node (only node weights are considered). The objective is to keep some knowledge about the problem data in determining the priority of the nodes. The priorities of rest of the chromosomes in the initial population are generated by a random perturbation in the priority of the first chromosome as given below:

$$(W_r)^i = (W_{r0})^i + Uniform\ (-\eta, \eta\ ). \qquad (1)$$

Where $(W_{r0})^i$ is the priority of node i in the first chromosome based on the original problem data, *Uniform* $(-\eta, \eta)$ is a random number generated uniformly between $-\eta$ and η. We are considering η=Max$\{(W_{r0})^i \ \forall \ i\}$/ but it can be chosen differently if desired. $(W_r)^i$ is the priority for node i of the chromosome calculated by perturbing the original problem data. The lower bound ($-\eta$) and upper bound (η) on the perturbation keep the dummy values in proximity to the original

problem. As one can see from Figure 3, each chromosome has a different priority values. So each chromosome guides the heuristic to generate a different solution. Each chromosome is decoded using a fast heuristic (e.g., list scheduling). This chromosome provides the priorities when we want to find a solution for the given problem using list scheduling.

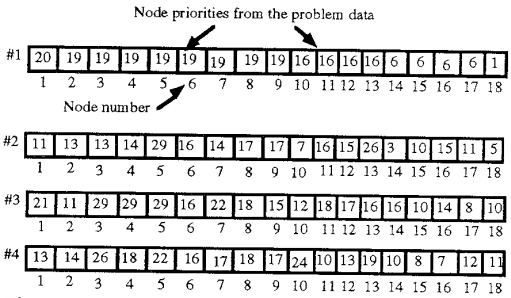

Figure 3: Initial population of four chromosomes.

Decoding Heuristic

The scheduling problem can be thought of as consisting of two parts: the assignment of tasks to processors and task execution ordering within a processor. A list scheduling heuristic solves both problem at once. Our decoding heuristic is an extended version of list scheduling [2-5]. We applied this heuristic to generate a schedule from a given chromosome with the objective of minimizing the completion time. Pseudocode for the decoding heuristic is given in Figure 4. In this heuristic we first build a task list from a chromosome and initialize a ready list with only those tasks which do not have any predecessor. Then a task from the ready list with the highest priority is selected and scheduled to the available processor on which the start time of the task is the earliest taking into account the communication of data from the predecessor tasks with the assigned tasks. Then the task is deleted from the ready list. This process is repeated on the ready list till either no idle processor is available or there is no task in the ready list. A task cannot be scheduled unless its predecessors have been scheduled and data have been communicated. The clock is set to the earliest time when a processor becomes available. Then, at the next clock, the processors are recovered, if possible, and the ready list is updated to have more tasks whose predecessors have been scheduled. If there is no ready task at the clock, the clock 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.

For the illustrative example the tasks schedules for the first three chromosomes of the initial population onto a multiprocessor system consisting of three homogeneous processors is shown in Figure 5. The schedule for the fourth chromosome is omitted for the sake of space. The minimum completion time for the initial population is 39.0 time units.

```
Decoding_heuristic (chromosome, m):
  Build task_list from the chromosome;
  ready_list := Initialize_ready_list(task_list);
  clock := 0;
  while (task_list < > null) do begin
     for each task on the ready_list do begin
         Pick task i with the highest priority value;
         j :=find_processor_early_time(i ) ;
         Schedule task i onto processor j ;
         Delete task i from the ready_list;
     end for;
     clock := find_next_clock( );
     recover _processors (clock);
     ready_list :=update_ready_list( );
  end while;
end Decoding_heuristic.
```

Figure 4: Pseudocode for decoding-heuristic.

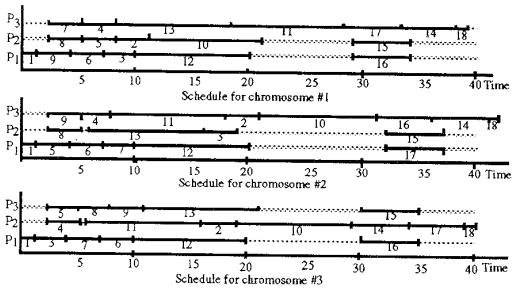

Figure 5: Schedules for the first three chromosomes of the initial population onto 3 processors.

Cost and Fitness Functions

The objective of optimization is to minimize the job completion time for an application program represented by a directed acyclic graph (DAG). For a given schedule s, the completion time C_S is given by:

$$C_S = \max \{F_1, F_2, \ldots F_m\}. \quad (2)$$

Where F_j is the completion time of a processor P_j, j=1 ... m. The completion time includes the computation time, the communication time and the waiting time because of the precedence constraints. We are using equation 2 as cost function to be minimized. In our study, as in Storer *et al.* [16], the following cost-to-fitness mapping function was used:

$$f(i) = \frac{(C_{max} - C_i)^{\pi}}{\sum_{j=1}^{j=N_p} (C_{max} - C_j)^{\pi}} \quad (3)$$

Where $f(i)$ is the fitness of chromosome i, C_{max} is the maximum cost of a chromosome in the population, C_i is the cost of chromosome i computed from equation 2, N_p is the population size, and π is a parameter which determines the selectivity of the fitness function. For high values of π, only the fittest few chromosomes will

survive, diversity will be lost, and the algorithm will converge to a population of identical solutions. Thus π must be chosen to balance convergence and diversity. It was established in Storer *et al.* [16] that 1 to 5 is a reasonable range of π. Table 1 shows the cost calculation and fitness for each chromosome's solution assuming $\pi = 3$.

Table 1: Cost and fitness for Generation G_0.

Solution #	Cost	Fitness
1	39.0	0.435484
2	42.0	0.000000
3	40.0	0.129032
4	39.0	0.435484

Selection, Crossover and Mutation

By operating in problem space, we have utilized standard roulette wheel selection technique [14] where each chromosome has a slot sized in proportion to its fitness. Each time we require an offspring, a simple spin of the roulette wheel gives a parent chromosome. A simple one-point crossover [14] was applied to the chromosomes. The crossover was applied with a crossover rate X_r. Mutation was implemented by randomly selecting a gene with mutation rate M_r and perturbing its value in the same perturbation range that was established for initial population. For example, Figure 6 shows the crossover operation when chromosome #1 and chromosome #4 were selected as parent with a cross site 5. The offspring chromosomes are also shown. Similarly chromosome #1 and chromosome #3 were selected as parents to generated other two offspring chromosomes.

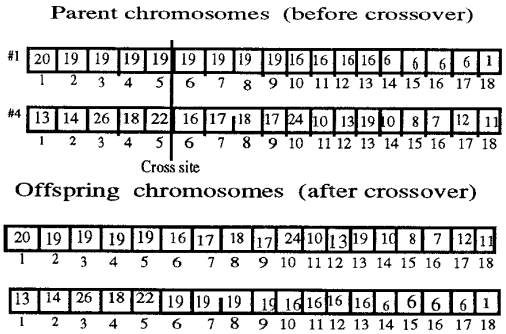

Figure 6: Chromosome before and after crossover.

The final best schedule obtained by applying the proposed scheme to the DAG of Figure 1 (a) onto the multiprocessor system given in Figure 1 (b), is shown in Figure 7. The completion time obtained by our technique is 36.0 time units. Note we did not consider network contention into account.

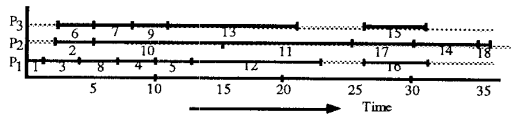

Figure 7: The final schedule for the illustrative example of Figure 1 (a).

Speed-up: Speed up (SpdUp) is defined as the completion time on a uniprocessor divided by completion time on a multiprocessor system.
The SpdUp achieved for the illustrative example is:

SpdUp = 86.0/36.0 =2.388.

Efficiency : Efficiency (E) = (Speed-up*100)/m.

Where m is the number of processors .
The percentage efficiency (E) for the illustrative example is given by :

E =79.629 percent.

III. EXPERIMENTAL RESULTS

The proposed problem-space genetic algorithm-based scheme, called PSGAs, for multiprocessor scheduling was implemented in C on a SUN SPARCstation 10 and has been tested on examples reported in literature. The results are very promising. The proposed scheme offers improvement in the completion time and efficiency over previous work. We compared our results with the CLANS [9], MCP [8] and LAST[6]. Note for all the test examples we have used population size Np=60, number of generations Ng=50, mutation rate M_r=0.009 and the crossover rate X_r=1.0.

Note: Running time used by our scheme is in the range of 5.0 sec. for the FFT1, FFT2, NEQ on Sun SPARCStation 10. The running time for IRR is 8.5 sec.

Example 1: Fast Fourier Transform Graphs

Our first example is a set of Fast Fourier Transform (FFT) graphs called FFT1, and FFT4 selected from McCreary *et al.* [10]. Details of these graphs can be found in [10]. The task execution ordering and completion time (C_S) for FFT1 and FFT2 obtained by the proposed technique are given in Table 2. Comparison of results (i.e., number of processors (m) used, C_S, speed-up, and percentage efficiency) for the FFT graphs are given in Table 3. For FFT1, proposed scheme produces the reported best time using the same number of processors. For FFT4, the completion time produced by the proposed scheme is better than all except the LAST [6], whereas efficiency of the schedules obtained by the proposed scheme is the best among all techniques which is 100%.

Table 2: Tasks execution ordering for FFT1 and FFT2 by the proposed scheduling scheme.

DAG	n	m	Tasks execution ordering on processors	C_S
FFT1	28	4	{2, 1, 9, 17, 21, 22}, {3, 4, 10, 13, 19, 26, 25}, {5, 6, 11, 14, 18, 23, 24 }, {7, 8, 12, 16, 20, 27, 28}	124
FFT4	28	2	{6, 1, 2, 5, 11, 12, 15, 16, 19, 25, 26, 20, 27, 23}, {7, 4, 8, 3, 10, 9, 14, 13, 17, 22, 21, 18, 24, 28}	240

Table 3: Comparison of results for FFT1 and FFT4 selected from McCreary *et al.* [10].

System	DAG	m	C_S	SpdUp	% E
CLANS[9]	FFT1	4	124	2.387	59.6
	FFT4	2	205	1.185	59.2
MCP [8]	FFT1	8	148	2.0	25.0
	FFT4	8	710	0.676	8.4
LAST [6]	FFT1	4	146	2.027	50.6
	FFT4	8	170	2.823	35.2
Proposed	FFT1	4	124	2.387	59.6
	FFT4	2	240	2.0	100

Example 2: Irregular Asymmetric Graphs

There are two graphs in this example, NEQ and IRR selected from McCreary *et al.* [10]. Task execution ordering and C_S for NEQ, and IRR graphs obtained by our scheme are give in Table 4. Comparison of results is given in Table 5. We are reporting the new best time for NEQ using less number of processors. The proposed scheme outperforms all other techniques both in terms of completion time and efficiency. For IRR only MCP [8] has a better C_S, but efficiency of the schedule obtained by our scheme is higher than all other techniques.

Table 4: Tasks execution ordering for test graphs by the proposed scheduling scheme, PSGAs.

DAG	n	m	Tasks execution ordering on processors	C_S
NEQ	20	3	{1, 2, 4, 8, 10, 13, 15, 17, 19, 20}, {5, 7, 12, 9, 16, 18} {3, 6, 11, 14}	1586
IRR	41	4	{1, 4, 8, 11, 9, 14, 15, 22, 28, 23, 24, 31}, {2, 5, 29, 34, 18, 26, 32, 37, 40}, {3, 7, 13, 20, 21, 17, 19, 27, 35, 38}, {10, 6, 12, 16, 25, 33, 30, 36, 39, 41}	615

Table 5: Comparison of results for NEQ and IRR selected from McCreary *et al.* [10].

System	DAG	m	C_S	SpdUp	% E
CLANS[9]	NEQ	5	1652	1.915	38.3
	IRR	7	725	1.834	26.2
MCP [8]	NEQ	5	1597	1.981	39.6
	IRR	12	605	2.198	18.3
LAST [6]	NEQ	4	2082	1.519	31.9
	IRR	3	840	1.583	52.7
Proposed	NEQ	3	1586	1.995	66.6
	IRR	4	615	2.162	54.1

To demonstrate the strength of the proposed PSGAs scheme we have generated a variety of random graphs (including computation intensive as well as communication intensive) from 40 to 200 nodes and compared results against MH [12]. We have achieved considerable improvement is speed-up over [12]. We have omitted these results due to the space limitation.

VI. CONCLUSIONS

In this paper we have proposed a Problem-Space Genetic Algorithm-based technique for the static multiprocessor scheduling problem including the communication delays to reduce the completion time and to increase the throughput of the system. The proposed scheme is a blend of the standard genetic algorithm and an heuristic, which uses a different neighborhood structure to search a large solution space in an intelligent way in order to find the best possible solution within an acceptable CPU time. In the proposed scheme the chromosomal representation is based on problem data, and 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 showed that the completion time can be reduced and the resources can be utilized more efficiently by using the proposed technique as compared to the existing approaches. We are extending our work for other architecutures such as hypercube, mesh *etc.* and to include network contention.

ACKNOWLEDGMENT

This research is supported by Kuwait University's Research Grant EE-073.

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] T. L. Adam, K. M. Chandy, and J. R. Dicson, "A Comparison of List Schedules for Parallel Processing Systems,"

Communication of the ACM, Vol. 17, pp. 685-690, December 1974.

[3] C. Y. Lee, J. J. Hwang, Y. C. Chow, and F. D. Anger, "Multiprocessor Scheduling With Interprocessor Communication Delays," *Operations Research Letters*, Vol. 7, N0. 3, pp. 141-147, June 1988.

[4] S. Selvakumar and C. S. R. Murthy, "Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors, " *IEEE Trans. on Parallel and Distributed Computing*, Vol. 5, No. 3, pp. 328-336, March 1994.

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

[6] J. Baxter and J. H. Patel, "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm," *1989 International Conference on Parallel Processing*, Vol. 2, pp. 217-222, 1989.

[7] 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, 1990.

[8] M. Y. Wu and D. D. Gajski, "Hypertool: A Programming Aid for Message_Passing Systems," *IEEE Trans. on Parallel and Distributed Computing*, Vol. 1, No. 3, pp. 330-343, July 1990.

[9] C. McCreary and H. Gill, "Automatic Determination of Grain Size for Efficient Parallel Processing," *Communication of the ACM*, Vol. 32, No. 9, pp. 1073-1078, September 1989.

[10] 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.

[11] A. Gerasoulis and T. Yang, "Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors," *J. of Parallel and Distributed Computing*, 16, pp. 276-153, 1992.

[12] H. El-Rewini and T. G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *J. of Parallel and Distributed Computing*, 9, pp. 138-153, 1990.

[13] G. C. Sih and E. A. Lee, "Declustering: A New Multiprocessor Scheduling Technique," *IEEE Trans. on Parallel and Distributed Computing*, Vol. 4, No. 6, pp. 625-637, June 1993.

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

[15] Lawrence Davis, *Handbook of Genetic Algorithms* , Van Nostrand Reinhold, 1991.

[16] R.H. Storer, D. S. Wu and R. Vaccari, "New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, " *Management Science*, Vol. 38, No. 10, pp. 1495-1509, October, 1992.

[17] Muhammad K. Dhodhi, Imtiaz Ahmad and Robert Storer, "SHEMUS: Synthesis of Heterogeneous Multiprocessor Systems," *Microprocessors and Microsystems*, Vol. 19, No. 6, pp. 311-319, 1995.

[18] M. K. Dhodhi, F. Hielscher, R. Storer, and J. Bhasker, "Datapath Synthesis Using a Problem-Space Genetic Algorithm," *IEEE Transsactions on CAD*, Vol. 14, No. 8, August 1995.

[19] I. Ahmad, M. K. Dhodhi and C. Y. R. Chen, "Integrated Scheduling, Allocation and Module Selection for Design-Space Exploration in High-Level Synthesis," *IEE Proc.-Computers and Digital Techniques*, Vol. 142, No. 1, pp. 65-71, January 1995.

[20] I. Ahmad and Muhammad K. Dhodhi, "Task Assignment Using a Problem-Space Genetic Algorithm", *Concurrency: Theory and Practice*, Vol. 7, No. 5, August 1995.