Hindawi Publishing Corporation
VLSI Design
Volume 2008, Article ID 930610, 7 pages
doi:10.1155/2008/930610

Research Article

Particle Swarm Optimization for Constrained Instruction Scheduling

Rehab F. Abdel-Kader

Electrical Engineering Department, Faculty of Engineering - Port-Said, Suez Canal University, Port Fouad 42523, Port-Said, Egypt

Correspondence should be addressed to Rehab F. Abdel-Kader, rehabf98@gmail.com

Received 4 September 2008; Accepted 19 December 2008

Recommended by Xianlong Hong

Instruction scheduling is an optimization phase aimed at balancing the performance-cost tradeoffs of the design of digital systems. In this paper, a formal framework is tailored in particular to find an optimal solution to the resource-constrained instruction scheduling problem in high-level synthesis. The scheduling problem is formulated as a discrete optimization problem and an efficient population-based search technique; particle swarm optimization (PSO) is incorporated for efficient pruning of the solution space. As PSO has proven to be successful in many applications in continuous optimization problems, the main contribution of this paper is to propose a new hybrid algorithm that combines PSO with the traditional list scheduling algorithm to solve the discrete problem of instruction scheduling. The performance of the proposed algorithms is evaluated on a set of HLS benchmarks, and the experimental results demonstrate that the proposed algorithm outperforms other scheduling metaheuristics and is a promising alternative for obtaining near optimal solutions to NP-complete scheduling problem instances.

1. INTRODUCTION

High-level synthesis (HLS) is the process of generating the register transfer level (RTL) design from the behavioral description of the digital system [1–3]. In HLS, the description of the system is represented in an abstract form usually a data flow graph (DFG), and this representation is transformed and mapped onto architectural elements from a library of resources. The synthesis process involves three major tasks: instruction scheduling, allocation, and binding. Instruction scheduling (IS) is concerned with mapping the operations in the behavioral description to control steps without violating the precedence relations between them.

Instruction scheduling is an important optimization phase aimed at balancing the cost and performance tradeoffs in the design of digital circuits. It can be considered the most important step during the architecture synthesis [3–6]. Most of the practical formulations of the scheduling problem are NP-complete. A variety of heuristics exist in the literature in order to find optimal or near optimal solutions for the scheduling problems. Those algorithms that do not find the optimal solution often trade good solutions for better performance and time considerations.

Optimal scheduling algorithms have been typically based on integer linear programming (ILP) [1, 7]. The major limitation with ILP and other exact scheduling algorithms is that the applicability of these algorithms is limited to small problem sizes as they tend to be lengthy and intractable for large and difficult problems. In order to handle bigger problem instances, heuristic scheduling algorithms with polynomial runtime complexity have been proposed. These algorithms rely on heuristics to remove the examination of parts of the search space that appear fruitless. In the early days of scheduling in HLS, simple techniques like as soon as possible (ASAP) or as late as possible (ALAP) schedules were used pervasively [6]. Both of ASAP and ALAP schedules suffer from the inability to exploit the available resources as they employ local selection criteria for the next step. Several scheduling heuristics utilize more global criteria and thus try to overcome the shortcomings of ASAP and ALAP. These algorithms include list scheduling [3, 5], forced-directed scheduling (FDS) [5, 6], path-based scheduling [8], genetic

algorithm [4, 9, 10], tabu search, simulated annealing, and bipartite graph matching [7].

Among these heuristics, list scheduling is the most popular because of its simplicity, low run time, and capability to produce good scheduling results for small-size problems. In list scheduling, operations are sorted in a list in a topological order (top to bottom) using the precedence information dictated by data or control dependencies [3, 5, 6]. The sorted operations are then iteratively scheduled into control steps. When a resource conflict occurs due to insufficient hardware, one or more operations are deferred by one control step. The selection of the deferred operation(s) is determined by a local priority function that is applied to all operations that could be scheduled in the current control step. A number of alternatives are possible for the priority function utilized in list scheduling. Popular choices include instruction depth, successor number, and instruction mobility. The mobility of an instruction is simply the difference between its ASAP and ALAP schedules. Although list schedulers are fast, they are in many cases not sufficient because their performance depends heavily on the node order in the priority list and very often may lead to disappointing results. Therefore, several research efforts were carried out to incorporate list schedulers together with other heuristics to find good node orders for the list scheduler.

Utilizing evolutionary searching techniques to find good node order for list scheduler has been investigated in the literature. Genetic algorithms were used in [4, 10], and the problem was modeled as a multiobjective optimization problem that searches the solution space for the best order of the operations. Another attempt was done in [11], where the MAX_MIN ant system (MMAS) optimization was used for the same purpose. Experimental results show that evolutionary-based searching techniques effectively search the solution space and produce excellent quality solutions. Encouraging results from previous work motivated this research effort that tends to explore the effectiveness of using the PSO algorithm in the same domain.

In this paper, an instruction scheduling algorithm using the PSO approach is presented. The algorithm utilizes a novel approach that employs PSO in conjunction with list scheduling to solve the resource constrained instruction scheduling (CIS) problem. The local and global search heuristics in PSO are iteratively adjusted making it an effective technique for exploring the discrete solution space for an optimal solution. Compared with a number of different scheduling heuristics, the proposed algorithm was found to generate better scheduling results in most of the tested benchmarks.

The main contributions of this work are

(i) formulation of the resource constrained instruction scheduling problem as a discrete optimization problem,

(ii) proposing a new hybrid scheduling algorithm that combines a discrete implementation of PSO and list scheduling algorithm,

(iii) evaluating the proposed algorithm using a set of HLS benchmarks and comparing its performance with existing scheduling heuristics.

The rest of this paper is organized as follows. An overview of particle swarm optimization is presented in Section 2. In Section 3, the constrained instruction scheduling problem is formally formulated as a discrete optimization problem. The proposed PSO-based instruction scheduling algorithm is explained in Section 4. Finally, Sections 5 and 6 attain the experimental results and conclusions.

2. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) proposed by Dr. Eberhart and Dr. Kennedy in 1995 is a computational paradigm based on the phenomenon of collective intelligence inspired by the social behavior of bird flocking or fish schooling [12–17]. Computation in PSO is based on a population (swarm) of processing elements called particles. Each particle represents a candidate solution and is identified with specific coordinates in the D-dimensional search space. The position of the ith particle is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$. The velocity of the particle (rate of the position change between the current position and the next) is denoted as $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. The fitness function is evaluated for each particle in the swarm and is compared to the fitness of the best previous result for that particle and to the fitness of the best particle among all particles in the swarm. After finding the two best values, the particles evolve by updating their velocities and positions according to the following equations:

$$V_i(t+1) = w * V_i(t) + c_1 * r_1 * (p_{i\_\text{best}} - X_i(t)) + c_2 * r_2 * (g_{\text{best}} - X_i(t)), \quad (1)$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \quad (2)$$

where $i = 1, 2, \ldots, M$, and M is the size of the swarm; p_{i_best} is the particle best reached solution, and g_{best} is the global best solution in the swarm. c_1 and c_2 are cognitive and social parameters that are bounded between 0 and 2. r_1 and r_2 are two random numbers, with uniform distribution $U(0,1)$. $V_{\max}$ is the maximum velocity that bounds the velocity vector, where $-V_{\max} \le V_i(t+1) \le V_{\max}$. In (1), the first component represents the inertia of pervious velocity. The inertia weight w is a factor used to control the balance of the search algorithm between exploration and exploitation; the second component is the "cognitive" component representing the private experience of the particle itself; the third component is the "social" component, representing the cooperation among the particles. The recursive steps will go on until we reach the termination condition (maximum number of iterations).

However, the original intent of PSO was to graphically simulate the choreography of a bird block or a fish school. It was found that particle swarm model can be used effectively as an optimizer in many domains such as training artificial neural networks, linear constrained function optimization, wireless network optimization, data clustering, and many other areas where GA can be applied [12–14, 17, 18].

3. PROBLEM FORMULATION

The input to the resource-constrained instruction scheduling algorithm is a directed acyclic graph called the data flow graph (DFG) defining the instructions and the dependencies among them [3, 11]. A DFG is denoted by the ordered-pair $G(V, E)$, where each node $v_i \in V$ $(i = 1, \ldots, n)$ represents an operation that must be performed, and the edge $e_{ij} \in E$ denotes a dependency between nodes v_i and v_j. Edges provide a partial order on the nodes such that an edge between nodes specifies when nodes can execute relative to each other. In addition to the DFG, we have a resource library of R different resource types, where $r_j > 0$ gives the number of available units of resource type j $(1 \le j \le R)$.

A resource-constrained scheduling algorithm tends to schedule the instructions into the control steps such that the execution time of these instructions are minimized without violating constraints imposed by the available resources [11]. The resource-constrained scheduling problem can be formally modeled as a discrete optimization problem as follows.

Minimize "*schedule length (SL)*" subject to the following constraints:

(i) data dependencies imposed by the DFG must be obeyed. Let op_i and op_j be two operations in the DFG and op_i is the parent of op_j. Then, the control step in which op_j starts must be later than the finish time of op_i.; that is, start_time$_j \ge$ finish_time$_i$;

(ii) at any control step, the number of active operations of any type must be less than or equal to the number of available resources of that type; that is, at any given control cycle, the number of resources used is constrained by r_j, for every $1 \le j \le R$.

As mentioned in Section 1, the effectiveness of a list scheduler depends mainly on the method used to compute the input priority list [5, 6]. There exist many different heuristics on how to order the list; however, the best list depends on the structure of the input application. A priority list based on a single heuristic limits the exploration of the search space for the list scheduler. In this work, we address this problem in an evolutionary manner. The proposed two-phase algorithm incorporates the PSO approach customized for the discrete search space and the traditional list scheduling algorithm. The algorithm searches for the optimal solution by an iterative evolutionary searching process. In the first phase, PSO is used to generate particles that traverse the DFG to construct individual instruction lists using global and local heuristics; these lists will be used as priority lists for the second phase. In the second phase, instruction lists are scheduled using a traditional list scheduler, and the quality of schedules is evaluated based on the required fitness function. The fitness function is directly linked to the generated schedule length. The fitness of particle P is the schedule length when its position vector X is used as the node schedule list; that is, fitness of $P = f(X) = SL$. Based on this evaluation, the PSO heuristics are adjusted to favor better solution components.

In order to use PSO for instruction scheduling problems, a direct correlation must be found between the particle vector and the solution of scheduling problem. Suppose we would like to use a swarm of M particles to solve the CIS problem whose position vectors represent feasible solutions. Each position vector is a node list satisfying the DFG precedence order. This vector will be used as a priority list for subsequent list scheduling algorithm to generate schedules for all particles in the swarm. The quality of the schedule generated by each particle will be ranked according to the fitness function that ranks the favor of a particle in the population. In CIS, this can be modeled as a minimization problem where the result of the fitness function must reflect the length of the final schedule that a member of the population generates.

Since 1995, PSO has proven to be successful in solving many continuous optimization problems. The first use of PSO in discrete optimization was for solving the traveling salesman problem TSP [19]. The main issue was to modify the position and velocity vector (1) and (2) in the original PSO algorithm to span the discrete search domain. The swap operator was defined in [19] and subsequently used in [18, 20]. Consider a normal particle P for the CIS problem, where P is a vector of n nodes. If a new solution P' is obtained by exchanging node n_i and node n_j in solution P, we define the operation as a swap operator, denoted as $\mathrm{SO}(n_i, n_j)$ and the swap process as $P' = P + \mathrm{SO}(n_i, n_j)$; for example,

$$[n_1, n_2, n_3, n_4, n_5, n_6] + \mathrm{SO}(n_2, n_4) = [n_1, n_4, n_3, n_2, n_5, n_6]. \tag{3}$$

A swap sequence SS is made up of one or more swap operators; that is,

$$\mathrm{SS} = (\mathrm{SO}_1, \mathrm{SO}_2, \ldots, \mathrm{SO}_n). \tag{4}$$

The order of the swap operator in SS is important as the swap operators in the swap sequence act on the solution in order. This can be described by the following formula:

$$\begin{aligned} P' &= P + \mathrm{SS} = P + (\mathrm{SO}_1, \mathrm{SO}_2, \mathrm{SO}_n) \\ &= (((P, \mathrm{SO}_1), \mathrm{SO}_2), \ldots, \mathrm{SO}_n). \end{aligned} \tag{5}$$

According to (2), a new position vector is obtained by imposing the velocity vector to the old position vector. The velocity vector can be defined as a swap sequence SS acting on the position vector representing the node scheduling list. The velocity vector evolves according to the formula given in (1). The $p_{i\_\mathrm{best}} - X_i(t))$ component in (1) means the basic swap sequence SS that should act on $X_i(t)$ to get to p_{i_best}; $\mathrm{SS} = p_{i\_\mathrm{best}} - X_i(t)$. We can swap the nodes in $X_i(t)$ according to p_{i_best} from left to right to get SS. This is also true for $(g_{\mathrm{best}} - X_i(t))$. For example,consider $X_i(t) = [2, 3, 5, 1, 4]$, and $p_{i\_\mathrm{best}} = [5, 2, 3, 4, 1]$. The third element in $X_i(t) =$ first element in $p_{i\_\mathrm{best}} = 5$, then the first swap operator to operate on $X_i(t)$ is SO(1,3); that is, $X_i'(t) = X_i(t) + \mathrm{SO}(1, 3) = [5, 3, 2, 1, 4]$. Similarly, the second swap operator is SO (2,3), and $X_i''(t) = [5, 2, 3, 1, 4]$. The third swap operator to operate on X_i'' is SO (4,5) producing $X_i'''(t)$, where

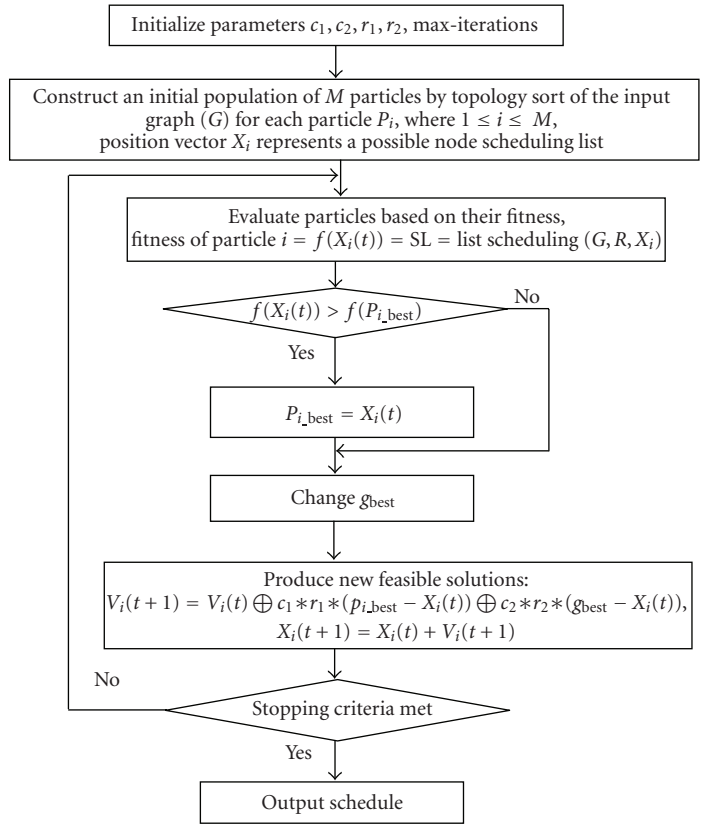

Figure 1: PSO-CIS algorithm.

$X_i'''(t) = p_{i\_\mathrm{best}} = [5, 2, 3, 4, 1]$. Finally, we get the basic swap sequence $\mathrm{SS} = p_{i\_\mathrm{best}} - X_i(t) = (\mathrm{SO}(1,3), \mathrm{SO}(2,3), \mathrm{SO}(4,5))$.

The new velocity vector consists of three SSs, the old velocity vector $V_i(t)$, $(p_{i\_\mathrm{best}} - X_i(t))$, and $(g_{\mathrm{best}} - X_i(t))$. The three swap sequences can be merged into a new equivalent swap sequence. Suppose SO_1, SO_2, SO_3 act on one solution P in this particular order, SO_1 first, SO_2 second, and SO_3 third, to get a new solution P'. This is equivalent to a new swap sequence SS' described as follows:

$$\mathrm{SS}' = \mathrm{SO}_1 \oplus \mathrm{SO}_2 \oplus \mathrm{SO}_3. \tag{6}$$

Assuming that the inertia weight factor $w = 1$, the new position and velocity evolution equations in the discrete domain can be rewritten as follows:

$$\begin{aligned} V_i(t+1) &= V_i(t) \oplus c_1 * r_1 * (p_{i\_\mathrm{best}} - X_i(t)) \oplus c_2 * r_2 * (g_{\mathrm{best}} - X_i(t)), \\ X_i(t+1) &= X_i(t) + V_i(t+1). \end{aligned} \tag{7}$$

4. PROPOSED PSO-CIS ALGORITHM

The specific steps of the PSO-CIS algorithm are presented in Figure 1. The inputs to the algorithm are the input DFG (G) and the resource library (R). The output of the algorithm is the instruction schedule. The process begins by generating an initial group of M candidate solutions as particles. Each particle is a node list generated by topology sort on the input DFG. The performance of the candidate solutions is evaluated using a list scheduler. The best previous position of the ith particle should be put in P_{i_best}, and the best position among all the particles should be put in g_{best}. The next generation of particles is produced by updating the position and the velocity vectors of the particles. The evolutionary cycle will repeat until the maximum number of iterations is reached.

5. EXPERIMENTAL RESULTS

In this section, we present the experimental results for the PSO-CIS algorithm described in Section 4. The proposed algorithm was implemented in C++, and experiments were carried out on an Intel Xeon-based Linux machine. The performance of the algorithm was compared to the results obtained from three heuristic scheduling algorithms reported in [11]. In addition, the performance of the PSO-CIS algorithm was compared to an optimal ILP-based scheduler solved by CPLEX [7, 11]. For the consistency of the comparison, the same test conditions were used as in [11]. Five different types of functional units are included in the resource library, which are ALUs (a), fast multipliers (fm), multipliers (m), input units (i), and output units (o). The functionality of these resources overlaps; that is, each

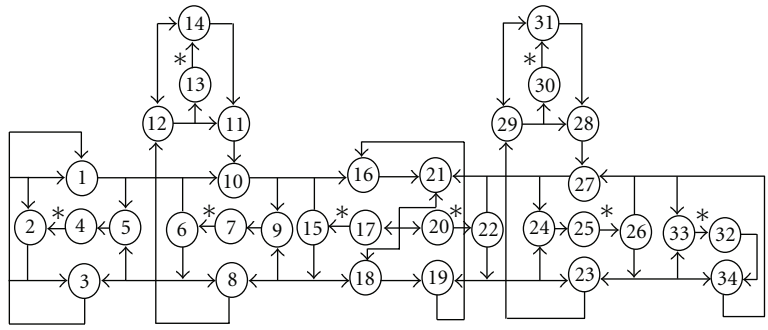

Figure 2: DFG of elliptic wave filter benchmark.

Table 1: Schedule length in cycles generated by different scheduling algorithms.

Benchmark (node/edges)	Resource-constraints	ILP-based CPLEX	FDS	MB-list scheduling	MB-MMAS (average of 5 runs)	PSO-CIS (average of 5 runs)
HAL(21/25)	1a, 1fm, 1m, 3i, 3o	8	8	8	8	8
ARF(28/30)	2a, 1fm, 2m	11	11	11	11	11
EWF(34/47)	1a, 1fm, 1m	27	28	28	27.2	27.4
FIR(40/39)	2a, 2m, 3i, 3o	13	19	19	17.2	17
COSINE1(66/76)	2a, 2m, 1fm, 3i, 3o	X	18	19	17.4	17.2
COSINE2(82/91)	2a, 2m, 1fm, 3i, 3o	X	23	23	21.2	21.2

instruction can be performed by at least one resource type. It is assumed that additions and subtractions can be mapped to ALUs and last 1 cycle, whereas multiplications are mapped to multipliers and take two cycles.

Six high-level syntheses DFG benchmarks were used to evaluate the proposed approach, namely, a second order differential equation solver (HAL), an autoregressive lattice filter (ARF), an elliptic wave filter (EWF), a finite impulse response filter (FIR), and two implementations for computing the discrete cosine transform (COSINE1) and (COSINE2). The number of nodes in these DFGs ranges from 21 to 82 nodes, and they present a set of challenging testing samples for the instruction scheduling problem that is widely used in the HLS literature [5, 6, 21]. A sample DFG for the elliptic wave filter is presented in Figure 2 [21]. The DFG consists of a set of 34 nodes that perform two types of operations, addition or multiplication (marked by "$*$" in Figure 2), submitted to over 47 precedence constraints.

For each benchmark, we run the proposed algorithm under the constraints imposed by the predefined set of resources. A population of 10 particles is generated. The cognitive and social components in the heuristic are balanced by setting parameters c_1, c_2, r_1, and r_2 as follows: $c_1 = c_2 = 1$, and r_1, r_2 are randomly selected with uniform distribution from the interval [0, 1]. The use of randomly generated coefficients for r_1 and r_2 balances between the exploration and exploitation of the search space and prevents premature convergence to local minima which presents a critical issue in all evolutionary algorithms. The maximum number of iterations is set to 100. For each benchmark, 5 runs are conducted, and the minimum schedule length is reported at the end of each run. The average value of the 5 runs is reported as the performance of the algorithm for this benchmark under these particular test conditions.

Experimental results are summarized in Table 1. The second column in the table indicates the available resources for each test case. Results from the proposed algorithm are compared to the results reported in [11] which are obtained from the mobility-based list scheduling algorithm (MB-list scheduling), the force-directed scheduling algorithm (FDS), and the mobility-based MAX-MIN ant system (MB-MMAS). As shown in Table 1, the proposed algorithm generates near optimal schedules consistently over most testing cases. In all tested benchmarks, the PSO-CIS and the MMAS algorithms are found to have comparable performance that outperforms the traditional list scheduler and the force-directed scheduling algorithms. The proposed algorithm achieves up to 10.52% performance enhancement compared to the FDS and list scheduling algorithms. The algorithm also finds a solution for the *COSINE*1 and *COSINE*2 benchmarks, where the CPLEX fails to find an optimal solution before running out of memory.

The reason behind the results is due to the fact that PSO-CIS and MB-MMAS algorithms are both based on list scheduling, and utilize an evolutionary-based searching approach for constructing a better scheduling list. The combination of different methods for producing the scheduling list and for generating the actual schedule is the key factor in improving the overall schedule quality produced by these hybrid algorithms compared to the stand alone list scheduler and the force-directed scheduling algorithms.

Detraction from these encouraging results is the time complexity of the PSO-CIS algorithm. For all the benchmarks, the runtime of the PSO-CIS algorithm ranges from 0.12 second to 1.66 second. This is close to the run time obtained by the MB-MMAS which ranges between 0.1 and 1.76 second. List scheduling is always the fastest due to its one-pass nature with complexity $O(n^2)$, where n is the number of operations to be scheduled. It typically finishes within a small fraction of a second. The force-directed scheduler runs much slower than the list scheduler, because its complexity is $O(n^3)$. The proposed algorithm utilizes list scheduling, and repeatedly performing this operation to evaluate all particles in the population increases the time complexity. If the number of iterations is linear with respect to the number of nodes in the DFG, the complexity becomes $O(n^3)$. Increasing the population size also results in similar complexity increase. An interesting extension to this work will be the exploration of other forms of scheduling algorithms that fit within this framework and possibly reduce the time complexity of the algorithm.

6. CONCLUSION

In this paper, a novel instruction scheduling algorithm utilizing particle swarm optimization was presented. PSO is used in conjunction with the traditional list scheduling algorithm to minimize the total schedule time in resource-constrained scheduling problems. The algorithm is based on the phenomenon of collective intelligence as a group of particles use local and global search heuristics to explore the discrete search space for an optimal solution. The solutions are iteratively constructed from efficient operation swaps guided by a fitness function that ranks the quality of the solution in the population. Experimental results over a set of HLS benchmarks demonstrate that PSO is an efficient method for solving the discrete problem of the instruction scheduling. The proposed algorithm produced better schedules in most test cases compared to other instruction scheduling heuristics.

REFERENCES

[1] P. Arató, Z. Á. Mann, and A. Orbán, "Time-constrained scheduling of large pipelined datapaths," *Journal of Systems Architecture*, vol. 51, no. 12, pp. 665–687, 2005.

[2] K. Ito, T. Iwata, and H. Kunieda, "An optimal scheduling method for parallel processing system of array architecture," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '97)*, pp. 447–454, Chiba, Japan, January 1997.

[3] S. O. Memik, R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "A scheduling algorithm for optimization and early planning in high-level synthesis," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 33–57, 2005.

[4] S. Beaty, "Genetic algorithms and instruction scheduling," in *Proceedings of the 24th Annual International Symposium on Microarchitecture*, pp. 206–211, Albuquerque, NM, USA, November 1992.

[5] P. G. Paulin and J. P. Knight, "Force-directed scheduling in automatic data path synthesis," in *Proceedings of the 24th ACM/IEEE Conference on Design Automation*, pp. 195–202, Miami Beach, Fla, USA, June 1987.

[6] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, 1989.

[7] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Boston, Mass, USA, 1994.

[8] R. Camposano, "Path-based scheduling for synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 85–93, 1991.

[9] E. Bonsma and S. Gerez, "A genetic approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays," in *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing*, Mierlo, The Netherlands, November 1997.

[10] M. J. M. Heijligers and J. A. G. Jess, "High-level synthesis scheduling and allocation using genetic algorithms based on constructive topological scheduling techniques," in *Proceedings of IEEE Conference on Evolutionary Computation*, vol. 1, pp. 56–61, Perth, Australia, November-December 1995.

[11] G. Wang, W. Gong, and R. Kastner, "Instruction scheduling using *MAX-MIN* ant system optimization," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI '05)*, pp. 44–49, Chicago, Ill, USA, April 2005.

[12] J.-F. Chang, S.-C. Chu, J. F. Roddick, and J.-S. Pan, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science and Engineering*, vol. 21, no. 4, pp. 809–818, 2005.

[13] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS '95)*, pp. 39–43, IEEE Service Center, Nagoya, Japan, October 1995.

[14] R. C. Eberhart and Y. H. Shi, "Comparison between genetic algorithm and particle swarm optimization," in *Proceedings of the 7th International Conference on Evolutionary Programming (EP '98)*, pp. 611–619, San Diego, Calif, USA, March 1998.

[15] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks (ICNN '95)*, vol. 4, pp. 1942–1948, IEEE Service Center, Perth, Australia, November-December 1995.

[16] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Proceedings of the 7th Annual Conference on Evolutionary Programming (EP '98)*, pp. 591–600, San Diego, Calif, USA, March 1998.

[17] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 69–73, Anchorage, Alaska, USA, May 1998.

[18] S.-C. Chu, Y.-T. Chen, and J.-H. Ho, "Timetable scheduling using particle swarm optimization," in *Proceedings of the 1st International Conference on Innovative Computing, Information and Control (ICICIC '06)*, pp. 324–327, Beijing, China, August-September 2006.

[19] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC '03)*, vol. 3, pp. 1583–1585, Shanghai, China, August 2003.

[20] X. Kong, J. Sun, and W. Xu, "Particle swarm algorithm for tasks scheduling in distributed heterogeneous system," in *Proceedings of the 6th International Conference on Intelligent Systems Design and Applications (ISDA '06)*, vol. 2, pp. 690–695, Jinan, China, October 2006.

[21] A. Shatnawi, M. O. Ahmad, and M. N. S. Swamy, "Optimal scheduling of digital signal processing data-flow graphs using shortest-path algorithms," *The Computer Journal*, vol. 45, no. 1, pp. 88–100, 2002.