High Level Synthesis using Learning Automata Genetic Algorithm

Huijing Yang
School of Software, Harbin University of Science and Technology, Harbin, China
Email: yhj833@gmail.com

Chunying Wang
School of Software, Harbin University of Science and Technology, Harbin, China
Email: spring_hrbust@yahoo.com.cn

Ning Du
Department of Software,Harbin University of Scienc&Technology,Harbin,China.
Email:dn.duning@gmail.com

***Abstract*—High-level synthesis consists of many interdependent tasks such as scheduling, allocation and binding. All tasks in high-level synthesis are NP-complete and the design objectives are in conflict for nature, most of the already proposed approaches are not efficient in the exploration of the design space and not effective in the identification of different trade-offs. For these reasons, genetic algorithms can be considered as good candidates to tackle such difficult explorations. A new algorithm that named Learning Automata Genetic Algorithm (LAGA) is used in this paper to perform scheduling and allocation concurrently. This algorithm is based on the Genetic Algorithm, the difference is that the Learning Process is added to the Genetic Algorithm. This strategy can complete the scheduling and the allocation effectively in the high-level synthesis under certain time and resource constraints. This algorithm is implemented in C language and is tested finally on a number of DSP benchmarks, and the test results then are compared with those obtained from four other different techniques which are commonly used in high-level synthesis. The experimental results show that the high-level synthesis using the LAGA algorithm is very effective, especially under the area constraint.**

***Index Terms*—Genetic algorithms, Learning Automata, High-level synthesis (HLS), Scheduling, design space exploration**

I. INTRODUCTION

There is a growing consensus among VLSI designers that one of the most effective methods to handle the complexity of today's system-on-chip (SoC) designs is to use computer-aided design (CAD) techniques. CAD techniques start with an abstract behavioral or algorithmic description of a circuit and automatically synthesize a structural description of a digital circuit that realizes the behavior. The behavioral description consists of computational operations (additions, multiplications, comparisons, logical operations) and control operations (conditional statements, loops, and procedure calls) [1]. The structural description maps the operations and data transfers onto functional units in a data path and a control unit that coordinates the flow of data between various functional units of the data path. The data path include hardware units (ALUs, multipliers, logical gates), storage units (registers, registers files, RAM, ROM), and interconnect units (multiplexers, buses) that are connected together to realize the specified behavior. This structural description is called a register transfer (RT)-level description. Once an RT-level design of a circuit is obtained, it can be transformed into a logic gate level netlist through logic synthesis, then into a layout via layout synthesis, and finally fabricated into an integrated circuit. Fig. 1 illustrates a typical high-level synthesis flow used for creating a chip design, starting from an abstract algorithmic specification [2].

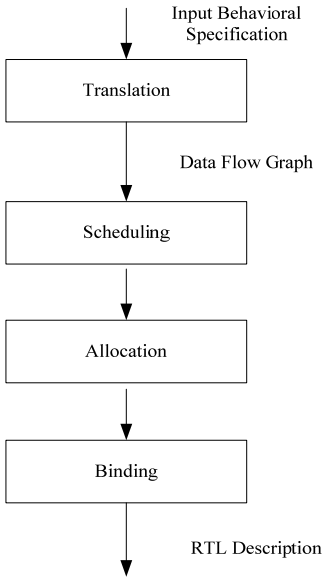


Figure 1. Interdependence of subtasks in high level synthesis

doi:10.4304/jcp.7.10.2534-2541

High-level synthesis (HLS) is the process of translating a behavioral description into a hardware implementation at register transfer level [3]. The design specification is usually written as a behavioral description, in a language such as C. The behavioral description is first compiled into an internal representation (such as data flow graphs - DFGs), which are then mapped to the functional units that are selected from the resource library to meet design goals (such as power, area, and performance). This process of transforming a behavioral description into a synthesizable structural description affords a methodology of automatically synthesizing a realizable digital circuit from an abstract algorithmic specification of the design, thus considerably reducing the design cycle time. VLSI designs are multiobjective by nature, since they have to trade-off several conflicting design objectives such as chip area, circuit delays, and power dissipation. The shorter design time using behavioral synthesis allows one to examine many alternative circuit realizations during the design process [4]. Often, the structural specification is divided into a data path comprised of the functional and storage units and a control unit that coordinates the flow of data between the data path elements [5]. Due to the division, high-level synthesis is traditionally divided into data path synthesis and controller synthesis[6]. The primarily focus will be on data path synthesis.

Datapath synthesis can be modeled as the process of searching a complex multidimensional space represented by the set of possible schedules, allocations, and bindings that can realize a given behavioral specification.

As modern VLSI and SoC designs become more complex, a major problem is the extremely large number of possible schedule and allocation combinations that must be examined in order to select a design that meets constraints and is optimal [7]. This process, called design space exploration, is further compounded by the need for shortening design times due to time-to-market pressures. Since an exhaustive search could be prohibitive and an ad hoc design exploration could be inefficient, designers often select a conservative architecture after some experimentation, which often results in a suboptimal design. Given this scenario, there is an acute need for techniques that automate the efficient exploration the large space in a reasonable time, during high-level synthesis of datapaths [8].

Searching a complex space of problem solutions often involves a tradeoff between two apparently conflicting objectives: exploiting the best solutions currently available and robustly exploring the design space. Genetic algorithms (GAS) manage this tradeoff in an intelligent way. GAS have recently been applied successfully to optimization problems in diverse fields, such as standard cell placement [9], searching and machine learning [10] and data path synthesis [11].

The Genetic Algorithm begins with a randomly selected population, and through recurrence of the production of the generation, looks for the best chromosome [12]. The aim of the Genetic Algorithm is to find the best chromosome. The position of genes in each chromosome, in the Genetic Algorithm is random. If we should select the appropriate position of genes, it would be possible to appropriate the nearly optimal answer in fewer generations [13]. The Genetic Algorithm, in fact chooses the best chromosome from among the existing ones, and the positions of the genes of chromosomes are totally random [14]. If it were possible to find the optimal place of the genes of chromosomes, we would be able to find the ideal answer in fewer generations. Through utilizing the advantage of both methods, the proposed algorithm tries to achieve the optimal answer in fewer generations.

In the LAGA algorithm each chromosome is equal to an automaton and each gene equal to an action of an automaton.

In this paper, we use LAGA performing subtasks of scheduling in high-level synthesis, and to trade-off conflicting design objectives the process of scheduling based on DFG with weights. And we set weights for DFG according to the constraints of resource.

The paper is organized as follows. Section II provides a brief review of the related work, with particular attention to the evolutionary approaches. Then, Section III describe in details methodology of LAGA, while Section IV presents the results of the experimental. Section V we summarize the paper and draw conclusions based on our experimental results.

II. Related Work

A. High Level Synthesis Methods

A large number of scheduling and allocation techniques have been developed for HLS over the past two decades. It is well-known that there is a strong interdependence between the HLS subtasks, and there is no clear consensus on their order of execution [18]. Such decision often has a large impact on the quality of solutions found and most of the early HLS systems performed those two subtasks separately, obtaining poor results. In literature, the high level synthesis techniques can be classified into four categories: constructive approaches, iterative transformational approaches, exact approaches and non-deterministic approaches. The constructive approaches operate on one operation or resource at a time until all elements are considered. Important algorithms following this approach, for example for scheduling, include common as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) scheduling, list based scheduling [15], force-directed scheduling [16] and path-based scheduling [17]. The iterative transformational approaches perform continuous refinements to the set of solutions while exact approaches [18] exploit a mathematical formulation of the problem to find the optimal solution, but the execution time of these algorithms grows exponentially with the number of variables and the number of inequalities. Therefore, these methods are impractical for large designs. Several high-level synthesis systems use nondeterministic approaches, and in particular GAs, to perform some or all of the synthesis subtasks. Most of them consider two phases and

problems separately like in [19] where GAs are used to schedule the operation while in [20] they are used to allocate and bind a scheduled graph. In the last years, the design of algorithms for DSE is becoming crucial to consider the effects of all the HLS subtasks.

B. Genetic Algorithm

Among the optimization methods inspired by the living nature, genetic algorithm, which is based on the principles of natural evolution, is considered one the best and most sophisticated [21]. Genetic Algorithm is a non-classic and random search optimization method that deals with the function itself, not its derivations, and is based on the theory of the survival of the fittest, inspire by Darwin's evolution theory, and natural genetics [22]. In this method, search begins from several points in solution space simultaneously and through point to point search. The variables of target function are evaluated and, finally, the point which the most or the least absolute is introduced as the optimal point [23]. Optimization is the most important and function of the Genetic Algorithm. In common optimization methods target function must necessarily be coherent and consistent [24]. In Genetic Algorithm, however, a consistent and devisable function is not needed. In accordance with Genetic Algorithm, a sample from among all decision variables, that affects the function, is regarded as a member, and a certain number of these samples, makes up a set of members[25]. In this method, a set of the population of variables is used in the process of search. As a result, as the chance of creating better variables is boosted, the possibility of finding the absolute or general optimal point is heighted. This quality is specifically suitable for functions sudden changes and possessing several situational optimal points. Complete information concerning Genetic Algorithm is brought in [26].

C. Learning Automata

A learning automaton is an abstract model that randomly selects an action from a set of the finite actions and applies it to the environment. The environment evaluates the selected action and informs the result of its evaluation, by a boosted signal, to the learning automata. By using the selected action and boosted signal, the learning automata results its internal situation and then selects its next action.

We can present the environment by E={α ,β , c} in which α={α1,α2,...,αr} is the set of inputs, β={β1,β2,...,βr }is the set of outputs and c={c1,c2,...,cr} is the set of penalty possibilities. When β is a two-member set, the environment is P type. In such environment, β1=1 is considered penalty and β2=0 reward. In a type Q environment, β set of processes an infinite number of members. ci is the possibility of a action's being penalized. Learning automata are directed into two groups: those with fixed structures, those with variable structures [27].

III. SCHEDULE USING LAGA

LAGA algorithm is constructed of two phase. In first phase with use of Genetic Algorithm, the result is optimized and in second phases the obtained results from Genetic algorithm improved using learning automata. In first phase the genetic Algorithm is endeavor to optimize the chromosomes and then the obtained chromosomes are putting into learning automata. Then Learning Automata is focus on Chromosome Genes and finding the most suitable place of Genes in Chromosomes. Figure 2 shows the flowchart of proposed algorithm.

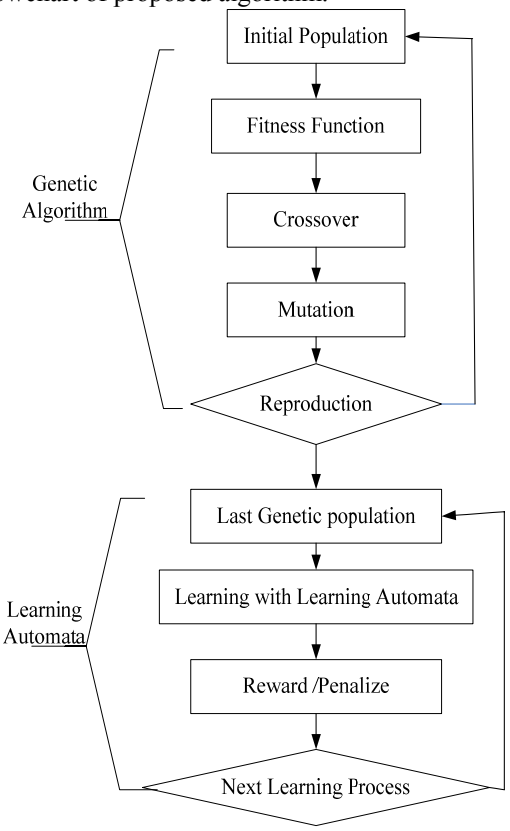


Figure 2. The Steps of proposed algorithm is described in below

A. Initial Population

At first, P (number of population) random generated chromosome and then all tasks are allocated to genes of chromosomes. Then a number is allocated randomly to all genes. The random allocated number to genes includes 2 concepts:

1. Task priority.
2. Processor's number, which executes the task.

After allocating random numbers to chromosome genes, the values of the genes are interchanged with the number of 2/N load.

We are going to perform, the shown graph in Figure 3, referring to Figure 4, you can observe how tasks are allocated to interior status in order. Since there are two processors in the system, so odd numbers indicate P1 processor and even numbers indicate P2 processor. If the system includes more than two processors, processor's number will consist of the result of the allocated number to the number of all processors.

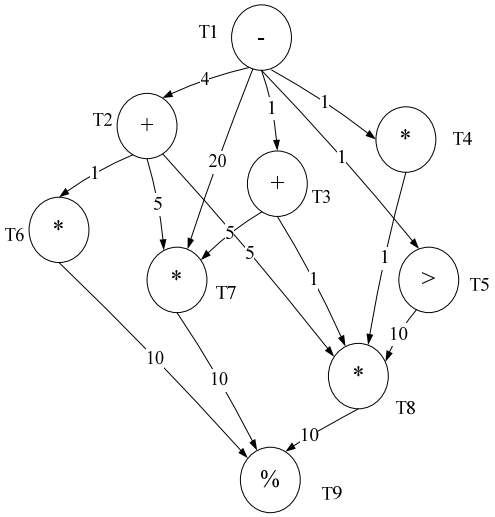


Figure 3. Operator task graph

B. Task Execution on Processors

When implementing a program on parallel processors, the data dependence between tasks should also be taken into consideration. In fact, a task cannot be implemented unless all of its parent tasks are implemented. In this section, how tasks are implemented on processors has been described [28]. Each task will be implemented in its relevant processor with regard to the automata of figure 4. From among all ready tasks, one, priority is higher than other tasks is implemented. In case, priorities of two tasks are the same as such other, one is randomly chosen for implementation. Ready task is one whose all parent tasks have been implemented.

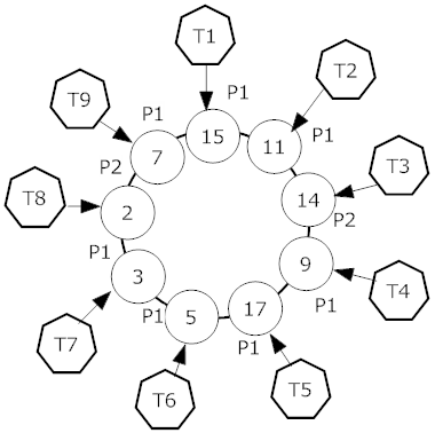


Figure 4. Example of automata for Figure 3 operator task graph

For example, by considering the automata of figure 4 and the task graph of figure 3 in the first phase, it becomes evident that only T1 task is ready. In the first phase, then, T1 will be executing. After the execution of T1 task, T2, T3, T4, T5 tasks will be on the ready. In this phase, priority of T5 task is higher than other ready tasks, so it is executed. In the same fashion, all tasks are implemented on their own specific processors [29].

With regard to Figure 4 automata, tasks T1, T5, T2, T4, T6, T7, T9 will be run on P1 processor and T3, T8 on P2 processor as well. Tasks execution order will be as TABLE 1.

TABLE I.
TASK EXECUTION ORDER

J	S'	v(j)	J*	TS
0	{1}	v(1)-15	1	S-{1}
2	{2,3,4,5}	v(2)-11,v(3)-14,v(9)-9,v(5)-17	2	S-{1,5}
1	{2,3,4}	v(2)-11,v(3)-14,v(9)-9	3	S-{1,5,3}
3	{2,4}	v(2)-11, v(9)-9	4	S-{1,5,3,2}
5	{4,6,7}	v(4)-9,v(6)-5,v(7)-3	5	S-{1,5,3,2,4}
4	{6,7,8}	v(6)-5,v(7)-3,v(8)-2	6	S-{1,5,3,2,4,6}
7	{7,8}	v(7)-3,v(8)-2	7	S-{1,5,3,2,4,6,7}
9	{8}	v(8)-2	8	S-{1,5,3,2,4,6,7,8}
6	{9}	v(9)-7	9	S-{1,5,3,2,4,6,7,8,9}

All tasks, now, will be executed according to table2. It shows task execution order on processors in details.

TABLE II.
DISPLAY OF TASK EXECUTION ORDER IN DETAILS

J*	S	pi	tj=ej+pj
1	P1-{1},P2-{}	1	T1 – 0+2-2
2	P1-{1,5},P2-{}	1	T5 – 2+5-7
3	P1-{1,5},P2-{3}	2	T3 – 3+3-6
4	P1-{1,5,2},P2-{3}	1	T2 – 7+3-10
5	P1-{1,5,2,4},P2-{3}	1	T4 – 10+4-14
6	P1-{1,5,2,4,6},P2-{3}	1	T6 – 14+4-18
7	P1-{1,5,2,4,6,7},P2-{3}	1	T7 – 18+4-22
8	P1-{1,5,2,4,6,7},P2-{3,8}	2	T8 – 17+4-21
9	P1-{1,5,2,4,6,7,9},P2-{3,8}	1	T9 – 31+1-32

C. Fitness Function

In Genetic Algorithm, fitness function determines whether chromosomes are going to stay alive. In the problem of task scheduling, the object is to find a short makes pan. Analysis function for scheduling problem is:

$eval(v_k) = 1/f^k$, k=12,..., pop size

f^k : the makes pan resulting from k^{th} chromosome.

D. Crossover Operator

Crossover is a technique which produces off-springs when two parents mate together. The parents are selected by binary tournament selection method [30]. In this paper, a novel method for combining chromosomes has been put forward. The combination method used in this paper is a two-point one. First two points are randomly chosen as subclasses, and then their contents and orders are analyzed. For instance, the substring chosen from 1 v , has a weight order of 1-2-3-4.This weight order is used for changing the subclass chosen by v2 . Thus, the 6-13-15-11 is changed to 15-13-11-6 and changes with the weight order of v1 subclass.

WMX algorithm is not one, which changes only the contents of two points selected from two chromosomes, but it also changes the contents of classes according to weight priorities. WMA is comprised of three steps.

Step1: random substring selection for two chromosomes. In figure 5, an example of the step1 of

combining chromosome by using WMA algorithm is displayed.

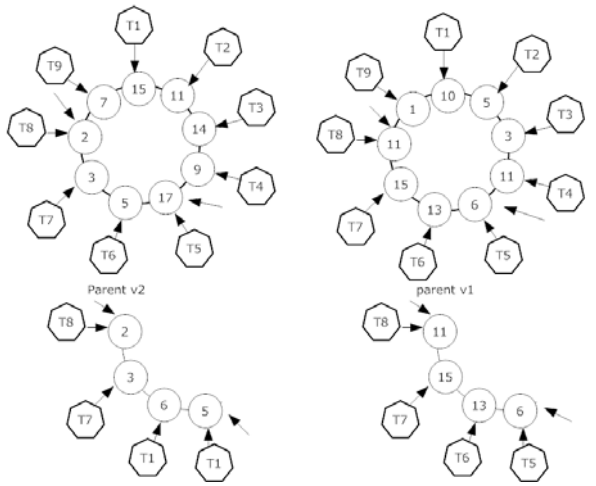


Figure 5. Step1 of WMA algorithm

Step2: defined genes mapping relation. Such as TABLE III.

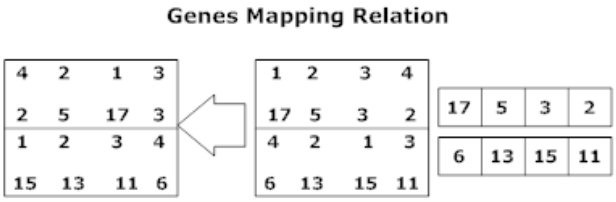


Figure 6. Step2 of WMA algorithm

Step3: creating two new offspring generation, the result of above example indicated in Figure6.

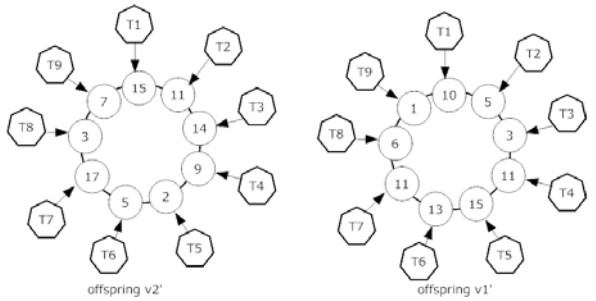


Figure 7. Step3 of WMA algorithm

E. Mutation Operator

For operating mutation, two Genes are randomly selected from a chromosome and their amounts are changed with each other. The manner of swapping actions values of a chromosome is departed into three step.

Step1: automata action status after allocating values randomly.

Step2: Selection of two random actions.

Step3: The output for automata after swapping actions randomly.

F. Selection Operator

Selection operator in this paper is as follows: In each step of new population production, (1-p) % of chromosome, which has least amount of FT, are selected and enter the new population directly. The rest of the population is, than produced through combining chromosomes.

G. Reward and Penalize Operators

Since, each chromosome is presented as a learning automaton, in each automaton, after considering the fitness of a gene (either processor or action), which is selected on a random basis, that gene, is duly penalized or rewarded. As a result of rewarding or penalizing a gene, its position in the boundary position of an action, its punishment leads to a change in its action and, in consequence, creation of a new makes pan. Departing on the type of learning automata, reward and penalize operator will be different.

Reward action occurs when the fitness of a task is smaller than threshold.

Fitness of ti is: x/y

x: is the sum of connection cost of all parent and offspring nodes of tj node so that. [Σ c(ti,tj) if(pti≠ptj)] pti: A processor that i t task is performed on it. ptj: A processor that j t task is performed on it.

y: is the sum of costs of all parent and offspring nodes of ti node. Σ c(ti,tj)

c(ti,tj) : Communication cost between tj and ti tasks. Threshold rate is equal T/Ntaks

T: Consist of a number of related tasks to ti task that is executed on a processor which ti task is run in it.

Ntasks: The number of all graph tasks.

The more fitness level of tj task tends to zero of the connection cost between processors tends towards zero too. If, therefore, the fitness level of a ti task is equal to zero, It turns out that all related tasks of ti are performed on the same processors. T has a direct relation with x; as T increases x decreases and vice versa.

In case the fitness level of a task is lower or equal to the threshold amount, then the head of the task gets penalized. Two positions are possible when penalizing a head:

1. The head might be in a position other than frontier position. In the case, penalizing makes it less important. How the head of task T7 is penalizes, is shown in Figure 8.

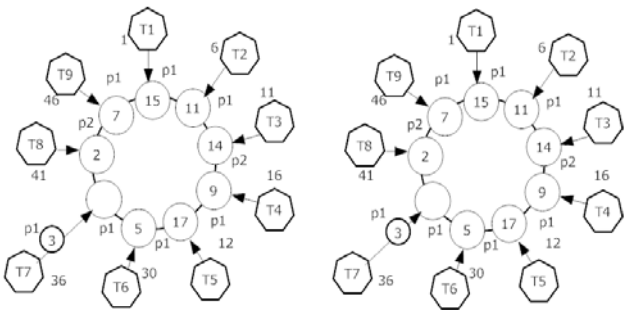
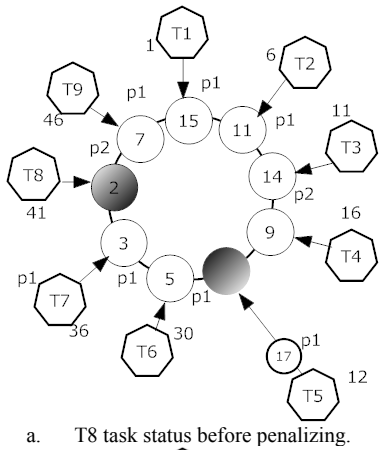


Figure 8. T7 task penalizing.

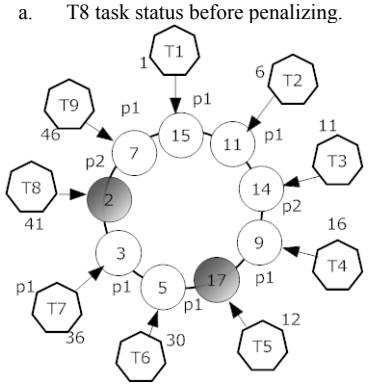
2. The head might be in frontier position. In that case, we look for a head in the graph that has the greatest reduction in the amount of FT when processors (the numbers attributed to heads) are changed. Now if the

found head is in the frontier position, the positions of the two heads are changed with each other and if otherwise, i.e. if the found head is not in the frontier position,

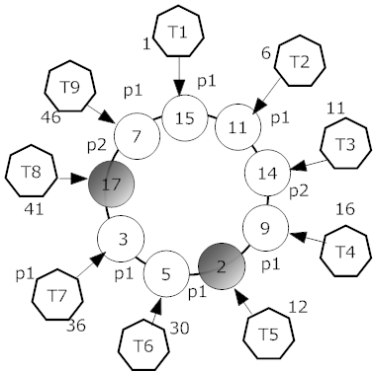
First the found head should be moved to its frontier position and then change occurs. Figure 9 shows how T8 task is penalized



a. T8 task status before penalizing.



b. T5 vertex transmits to border.



c. T8, T5 task interchanging.

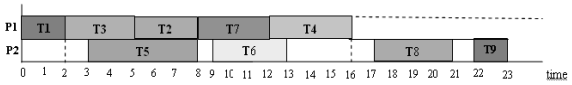
Figure 9. T8 task penalizing.

In this paper, the performance of the proposed algorithm is compared with well-known definite and indefinite algorithms. At first the proposed algorithm is simulated and evaluated on homogeneous platforms and then evaluated on heterogeneous platforms. Parameters that are used in the hybrid algorithm are shown in TABLE III.

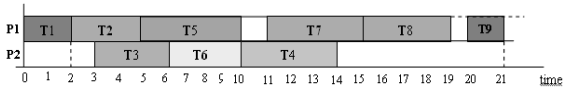
TABLE III.
HYBRID ALGORITHM PARAMETERS

Algorith	Memory Depth	Mutation Rate	Crossover Rate	Iteration	Population
GA	-	0.2	0.7	40	50
LAGA	4	0.2	0.7	20	50

First by observing the task graph in figure 6, results obtained from various algorithms and the proposed algorithm is displayed in figure 10.



a. Acquired Gantt chart of **GA** Algorithm



b. Acquired Gantt chart of **LGA** Algorithm

Figure 10. Comparison of proposed algorithm with other algorithms (for indicated graph in Figure 3).

IV. EXPERIMENTAL RESULTS

The proposed LAGA-based high-level synthesis system has been implemented in the C language. The extended high level synthesis tool accepts ANSI C programs and generates RTL specifications in verilog.

To perform a qualitative assessment of algorithm of LAGA in the high-level synthesis, it was tested on a number of DSP benchmarks drawn from high-level synthesis literature.

For all the benchmarks tested, the synthesized designs were assumed to operate with a clock period of 20 ns. We used a 0.35- CMOS module library, where ALUs, multipliers, registers, and multiplexers are implemented as hard macro cells (cells having fixed aspect ratio and pin locations). The ALUs have a propagations delay of 6.5 ns, and multipliers have a propagations delay of 15 ns. We assume that the area cost and delay of a pipelined multiplier are the same as those of a nonpipelined multiplier, respectively.

In all the experiments, the size of the GA population was set to 100, the crossover probability was 0.90, and the mutation probability set to 0.20. Each of the GA runs was stopped after 10000 fitness evaluations. Since GA algorithms are stochastic algorithms, ten independent runs with different random number seeds were performed for each of the benchmark problem instances, and the best solution found by the GA in each of the ten runs was recorded.

We compared our results with those obtained from four different scheduling techniques commonly used in high-level synthesis, namely, the GA scheduling , ALAP scheduling , force-directed (FDS) scheduling , and simultaneous scheduling, allocation, and binding (SAM) technique . These scheduling algorithms were tested in a traditional high-level synthesis framework that performs the three synthesis subtasks of scheduling, allocation, and binding independently. The goal of this comparison was twofold: 1) to verify the performance gains from concurrently performing scheduling and allocation, over a traditional synthesis flow that carries out these subtasks independently and 2) to use the performance of these scheduling techniques as a baseline to compare our results. The same benchmarks are used for comparing the results.

For each of the benchmark problems, a design-space exploration was performed by setting different values to the weights (and) corresponding area constraints. Results are shown in TABLE IV and TABLE V.

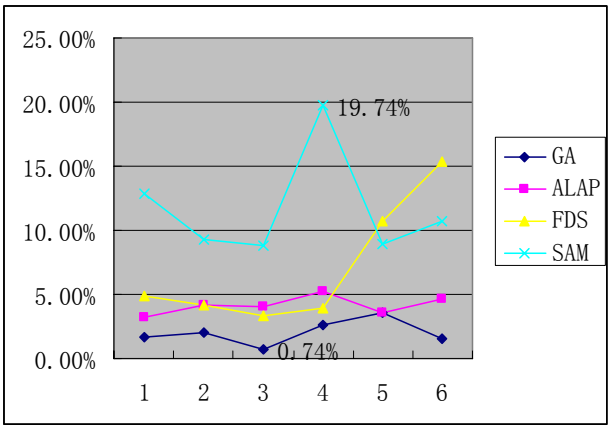
In TABLE IV, the chip latency in bold indicate the Fastest designs for each areas value. From the table, it can be seen that LAGA algorithm finds better solutions than those of the other four scheduling techniques, for all the benchmarks tested.

TABLE IV.
COMPARISON OF OUR LAGA-BASED METHOD WITH OTHER SCHEDULING ALGORITHMS

Benchmark Example	Area Constraint	Chip Latency(ns)				
		LAGA	GA	ALAP	FDS	SAM
IIR	6.0mm^2	62	63	65	66	70
FIR	5.0 mm^2	97	99	101	101	106
EWF	4.5 mm^2	272	274	283	281	296
ARF	10.0 mm^2	76	78	80	79	91
DCT	12 mm^2	56	58	58	62	61
FDCT	18 mm^2	65	66	68	75	72

TABLE V shows the improvement of the LAGA-based solutions over those of the other four scheduling techniques. The average improvements range from 0.74% (compared to the GA method) to 19.74% (compared to the FDS method).

TABLE V.
INCREASE PROPORTION OF THE LAGA-BASED SOLUTIONS OVER OTHERS



The experimental results show that the LAGA algorithm to complete scheduling and allocation concurrently in high level synthesis is very effective, especially under area constraint. It can obtain a better delay performance than other algorithms.

V. CONCLUSION

In this paper, a new method LAGA is used in high level synthesis to deal with scheduling and allocation simultaneously. It can produce area and performance optimized designs. This algorithm utilizes Genetic Algorithm and Learning Automata methods sequentially to search for the mode space. It can find the Solutions quickly by using Genetic Algorithm and Learning Automata sequentially in search process.

The method is simulated on a number of DSP benchmarks. It can succeed in obtaining optimal solutions. The same problems have been also solved in a general way. The experimental results indicate that LAGA algorithm is very effective in high-level synthesis, especially under the area constraint.

REFERENCES

[1] I. Das. A preference ordering among various Pareto optimal alternatives. Structural and Multidisciplinary Optimization, 18(1):30–35, Aug. 1999.

[2] M. Palesi and T. Givargis, “Multi-objective design space exploration using genetic algorithms,” in CODES ’02: Proceedings of the tenth international symposium on Hardware/software codesign. Estes Park, Colorado: ACM, pp. 67–72. 2002,

[3] M. Laumanns, L. Thiele, and E. Zitzler, “An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method,” European Journal of Operational Research, vol. 169, no. 3, pp. 932–942, Mar. 2006.

[4] R. J. Cloutier and D. E. Thomas, “The combination of scheduling, allocation and mapping in a single algorithm,” in Proc. 27th Design Automation Conf., pp. 71–76, Jun. 1990.

[5] X. Yao and T. Higuchi, “Promises and challenges of evolvable hardware,” IEEE Trans. Syst., Man, Cybern., pt. C, vol. 29, no. 1, pp. 87–97, Feb. 1999.

[6] S.M. Logesh, D.S. Harish Ram, M.C. Bhuvaneswari, Multi-objective Optimization of Power, Area and Delay during High-Level Synthesis of DFG’s - A Genetic Algorithm Approach, Electronics Computer Technology (ICECT), 2011 3rd International Conference on, pages 108 – 112, 8-10 April 2011.

[7] J. C. Gallagher, S. Vigraham, and G. Kramer, “A family of compact genetic algorithms for intrinsic evolvable hardware,” IEEE Trans. Evol. Comput., vol. 8, no. 2, pp. 1–126, Apr. 2004.

[8] G. Ascia, V. Catania, and M. Palesi, “A GA-based design space exploration framework for parameterized system-on-a-chip platform,” IEEE Trans. Evol. Comput., vol. 8, no. 4, pp. 329–346, Aug. 2004.

[9] Fujimoto, N.; Hagihara, K.; A comparison among grid scheduling algorithms for independent coarse-grained tasks. International Symposium on Applications and the Internet Workshops, SAINT 2004, Pages: 674 - 680, 2004.

[10] Zhang, Yaping; Peng, Haoyu; Wang, Zonghui; Shi, Jiaoying; Research and Implementation of Distributed Simulation and Parallel Rendering System Based on Grid. 2010 First International Conference on Networking and Distributed Computing (ICNDC), Pages: 3 - 7, 2010.

[11] Yanfang Fu; Yan Fan; Scheduling Method of Resource for Simulation System Based on Grid. Second International Conference on Computer Modeling and Simulation, ICCMS ’10, Pages: 226 - 229, 2010.

[12] Liu, Hanbing; Su, Hongyi; Zhan, Shouyi; Chai, Xundong; Zhang, Yabin; Hou, Baocun; Guo, Linqin; Fan, Shuai. Resource scheduling based on dynamic dependence injection in virtualization-based simulation grid. 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Pages: 396 - 401 , 2010.

[13] Huang Hai, Tian Lei, Wu Wei, Sun Songlin, Jing Xiaojun. Genetic algorithm for scheduling of interactive tasks in simulation grid. Proceedings of 2010 WASE International

Conference on Information Engineering, ICIE 2010, v 1, Pages 30-33, 2010.
[14] Wei Hongtao. Task Management and Scheduling Methods for Grid-Computing-based Simulation, Doctor's thesis, National University of Defense Technology, 2005.
[15] B. M. Pangrle and D. D. Gajski, "Slicer: A state synthesizer for intelligent silicon compilation," in Proc. Int. Conf. Computer-Aided Des., pp. 536–541, 1987.
[16] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," IEEE Trans. Comput.-Aided Des., vol. 8, no. 6, pp. 661–679, 1989.
[17] A. C. Parker, J. T. Pizarro, and M. Mlinar, "Maha: A program for datapath synthesis," in Proc. 23rd ACM/IEEE Design Automation Conf., pp. 461–466., 1986,
[18] R. Camposano, "Path-based scheduling for synthesis," IEEE Trans. Comput.-Aided Des., vol. 10, pp. 85–93, 1991.
[19] R. K. Brayton, R. Camposano, G. De Micheli, R. Otten, and J. van Eijndhoven, "The Yorktown silicon compiler system," in Silicon Compilation, D. Gajski, Ed. Reading, MA: Addison-Wesley, pp. 204–310, 1988.
[20] S. G. Ara′ ujo, A. C. Mesquita, and A. Pedroza. Optimized Datapath Design by Evolutionary Computation. In Proceedings of the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03), 30 June - 2 July 2003, Calgary, Alberta, Canada, pages6–9, 2003.
[21] F. Ferrandi, P. L. Lanzi, G. Palermo, C. Pilato, D. Sciuto, and A. Tumeo. An evolutionary approach to area-time optimization of FPGA designs. Embedded Computer Systems: Architectures, Modeling and Simulation, 2007. IC-SAMOS 2007. International Conference on, pages 145–152, 16-19 July 2007.
[22] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Spark: a high-level synthesis framework for applying parallelizing compiler transformations. VLSI Design, 2003. Proceedings. 16th International Conference on, pages 461–466, 4-8 Jan. 2003.
[23] V. Krishnan and S. Katkoori. A genetic algorithm for the design space exploration of datapaths during high-level synthesis'. IEEE Trans. Evolutionary Computation, 10(3):213–229, 2006.
[24] C. Mandal, P. Chakrabarti, and S. Ghose. Design space exploration for data path synthesis. Proc. of the 10-th International Conference on VLSI Design, pages 166–170, 1996.
[25] C. Mandal, P. P. Chakrabarti, and S. Ghose. GABIND: a GA approach to allocation and binding for the high-level synthesis of data paths. IEEE Trans. Very Large Scale Integr. Syst., 8(6):747–750, 2000.
[26] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In Proc. CODES'02 Workshop, pages 67–72, New York, NY, USA, 2002. ACM.
[27] P. G. Paulin and J. P. Knight. Force-directed scheduling in automated data path synthesis. In Design Automation Conference, 1987.
[28] C. Pilato, G. Palermo, A. Tumeo, F. Ferrandi, P. L. Lanzi, and D. Sciuto. Fitness inheritance in evolutionary and multi-objective highlevel synthesis. In IEEE Proceedings of CEC 2007 - Congress on Evolutionary Computation, 2007.
[29] G. Papa, J. silc, Scheduling algorithms based on genetic approach, Proc. 4'h Conference on Neural networks and their applications, Zakopane, Poland, pp. 469-474, May 1999.
[30] G. Papa, J. silc, The Use of Genetic Algorithm in the Integrated Circuit Design, Proc. 2nd Intl. Workshop on Design, Test and Applications WDTA'99, Dubrovnik, Croatia , pp. 73-76 June 1999.
[31] G. Papa, J. silc, Using Simulated Annealing and Genetic Algorithm in the Automated Synthesis of Digital Systems, In Mastorakis (ed.): Recent advances in circuits and systems, World Scientific, pp. 377-381, 1998.

Huijing Yang received the B.S. degree in Computer Science and Technology from Heilongjiang University in 2002. And she received the M.S. degree in Computer Software and Theory from Harbin Engineering University in 2005. Her primary research area focused on IC design and Computer Architecture. Currently she is a Lecturer of the School of Software, Harbin University of Science and Technology, Harbin, China.

Chunying Wang was born in Jilin Province, China, in 1977 .She received the B.S. degree in Computer Science and Technology from Northeast Forestry University, China in 2000, and M.S. degree in Computer Software and Theory from Technology from Harbin University of Science and Technology, China in 2003. She has been working in Harbin University of Science and Technology from 2005. Currently, she is a in school of Software. Her major research interests include wireless network and network security.

Ning Du was born in Heilongjiang Province, China, in 1979 .She received the B.S. degree in Computer Science and Technology from Harbin University of Science and Technology, China in 2003 , and M.S. degree in Computer Software and Theory from Technology from Harbin University of Science and Technology, China in 2006. She has been working in Harbin University of Science and Technology from 2006. Currently, she is a in college of Software. Her major research interests include wireless network and network security.