การสังเคราะห์วงจรขั้นสูงโดยขั้นตอนวิธีเชิงวิวัฒน์

นายราชพร เขียนประสิทธิ์

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2545

ISBN 974-17-2529-9

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

HIGH-LEVEL CIRCUIT SYNTHESIS BY EVOLUTIONARY ALGORITHMS

Mr.Rachaporn Keinprasit

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic year 2002

ISBN 974-17-2529-9

Thesis Title	HIGH-LEVEL CIRCUIT SYNTHESIS BY EVOLUTIONARY ALGORITHMS
By	Mr.Rachaporn Keinprasit
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor Prabhas Chongstitvatana, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctor's Degree

.... Dean of Faculty of Engineering

(Professor Somsak Punyakeow, D.Eng.)

THESIS COMMITTEE

.... Chairman

(Associate Professor Somchai Prasitjutrakul, Ph.D.)

.... Thesis Advisor

(Associate Professor Prabhas Chongstitvatana, Ph.D.)

.... Member

(Pansak Siriruchatapong, Ph.D.)

.... Member

(Professor Chidchanok Lursinsap, Ph.D.)

.... Member

(Thit Siriboon, Ph.D.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ราชพร เขียนประสิทธิ์ : การสังเคราะห์วงจรขั้นสูงโดยขั้นตอนวิธีเชิงวิวัฒน์ . (HIGH-LEVEL CIRCUIT SYNTHESIS BY EVOLUTIONARY ALGORITHMS) อ. ที่ปรึกษา : รศ.ดร. ประภาส จงสถิตย์วัฒนา, 70 หน้า. ISBN 974-17-2529-9.

ในการวิจัยนี้ได้เสนอขั้นตอนวิธีแบบใหม่ที่มีชื่อว่า Ants on a Tree (AOT) ซึ่งมีพื้นฐานมาจากขั้นตอนวิธีแบบ Ant Colony Optimization โดยขั้นตอนวิธีแบบนี้สามารถเชื่อมโยงขั้นตอนวิธีหลายอย่างเข้าด้วยกันเพื่อช่วยแก้ปัญหาเดียวกันได้ ความสามารถของขั้นตอนวิธีนี้จะถูกสาธิตโดยการนำมาใช้แก้ปัญหาการสังเคราะห์วงจรระดับสูง การสังเคราะห์วงจรระดับสูงประกอบด้วยขั้นตอนการออกแบบหลายขั้นตอน และขั้นตอนวิธีหลายชนิดเพื่อแก้ปัญหาในแต่ละขั้น ขั้นตอนวิธี AOT สามารถที่จะนำขั้นตอนวิธีต่างๆมาช่วยในการกำหนดขอบเขตของการค้นหาคำตอบ และแปลงเป็นค่าถ่วงน้ำหนักเพื่อช่วยนำทางในการค้นหาคำตอบ ในระหว่างที่ทำการค้นหาคำตอบขั้นตอนวิธี AOT จะสร้างต้นไม้การตัดสินใจแบบพลวัตเพื่อช่วยในการค้นหา เทคนิคที่ใช้ในขั้นตอนวิธีแบบกลวิธีขยายและจำกัดเขตถูกนำมาช่วยในการค้นหาคำตอบ ปัญหาเกี่ยวกับขนาดของหน่วยเก็บข้อมูลจะไม่เกิดขึ้นในขั้นตอนวิธีแบบนี้ เนื่องจากขนาดของหน่วยเก็บข้อมูลถูกจำกัดโดยการระเหิดของ pheromone ที่สร้างขึ้นโดยมดซึ่งเป็นคุณสมบัติอย่างหนึ่งของขั้นตอนวิธีนี้

ขั้นตอนวิธี AOT ได้ถูกทดสอบโดยการนำไปใช้สังเคราะห์วงจรกรองคลื่นความถี่แบบ Elliptic และผลจากการทดสอบแสดงว่าขั้นตอนวิธี AOT สามารถนำไปใช้ในทางปฏิบัติได้ โดยการกำหนดปริมาณทรัพยากรในขั้นต้นของการออกแบบ ทำให้สามารถใช้ Fixed-resource Mobility เพื่อช่วยปรับให้ขั้นตอนวิธี AOT มีประสิทธิภาพมากยิ่งขึ้น

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต..
สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา.......................................
ปีการศึกษา 2545

4171820121 :MAJOR COMPUTER ENGINEERING
KEY WORD: ANT ALGORITHMS / HIGH-LEVEL SYNTHESIS / DATA PATH DESIGN / GENETIC ALGORITHMS / DYNAMIC NICHE SHARING

RACHAPORN KEINPRASIT : HIGH-LEVEL CIRCUIT SYNTHESIS BY EVOLUTIONARY ALGORITHMS. THESIS ADVISOR : ASSOC. PROF. PRABHAS CHONGSTITVATANA, Ph.D., 70 pp. ISBN 974-17-2529-9.

In this research an algorithm based on Ant Colony Optimization techniques called Ants on a Tree (AOT) is proposed. This algorithm can integrate many algorithms together to solve a single problem. The strength of AOT is demonstrated by solving a High-Level Synthesis problem. A High-Level Synthesis problem consists of many design steps and many algorithms to solve each of them. AOT can easily integrate these algorithms to limit the search space and use them as heuristic weights to guide the search. During the search, AOT generates a dynamic decision tree. A boosting technique similar to branch and bound algorithms is applied to guide the search in the decision tree. The storage explosion problem is eliminated by the evaporation of pheromone trail generated by ants, the inherent property of our search algorithm.

The algorithm was tested with the Elliptical Wave Filter (EWF) benchmark, and found that it is practically to be used. By allocating the resources at the early design state, the Fixed-resource Mobility could be integrated to further improve the performance of the algorithm.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Department	Computer Engineering	Student's signature ....
Field of study	Computer Engineering	Advisor's signature ....
Academic year	2002	

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Associate Professor Prabhas Chongstitvatana, Ph.D., for his valuable advice and continuous support.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CONTENTS

ABSTRACT (THAI) iv

ABSTRACT (ENGLISH) v

ACKNOWLEDGEMENT vi

CONTENTS vii

LIST OF TABLES x

LIST OF FIGURES xi

1 INTRODUCTION 1

2 HIGH-LEVEL SYNTHESIS 3

2.1 Overview 3

2.2 Heuristic Algorithms 3

2.2.1 Differential Equation Example 4

2.2.2 As Soon As Possible (ASAP), As Late As Possible (ALAP), and Mobility 5

2.2.3 Lower Bound Algorithms 6

2.2.4 Scheduling 7

2.2.5 Functional Unit Allocation and Assignment 13

2.2.6 Register Allocation and Assignment 13

2.2.7 Bus Allocation and Assignment 14

2.3 Integer Linear Programming 17

2.4 Genetic Algorithms 20

2.5 Conclusion 22

3 ANT COLONY OPTIMIZATION ALGORITHMS 23

3.1 overview 23

3.2 Ant System 25

3.3 Ant Colony System and Ant-Q 26

3.4 $\mathcal{MAX} - \mathcal{MIN}$ Ant System 27

3.5 Conclusion 27

4 ANTS ON A TREE ALGORITHM 28

4.1 Overview . . . 28

4.2 Construction of a decision path . . . 30

4.3 State transition rule . . . 30

4.4 Pheromone updating rule . . . 31

4.5 Path exploration . . . 32

4.6 Dynamic niche algorithm . . . 33

4.7 Boosting . . . 33

4.8 Conclusion . . . 34

5 HIGH-LEVEL SYNTHESIS BY AOT 35

5.1 Overview . . . 35

5.2 Resource Allocation . . . 37

5.3 Scheduling . . . 38

5.4 Functional unit assignment . . . 39

5.5 Register assignment . . . 39

5.6 Bus assignment . . . 40

5.7 Scheduling Example . . . 40

5.8 Conclusion . . . 42

6 EXPERIMENTS 43

6.1 Overview . . . 43

6.2 Differential Equation . . . 43

6.2.1 Setup . . . 45

6.2.2 Result . . . 46

6.3 Elliptical Wave Filter . . . 48

6.3.1 Setup . . . 49

6.3.2 Result . . . 51

6.4 Conclusion . . . 55

7 FIXED-RESOURCE MOBILITY 56

7.1 Overview . . . 56

7.2 Fixed-Resource Mobility . . . 56

7.3 An Experiment on EWF . . . 60

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

7.4 Conclusion 61

8 SUMMARY AND CONCLUSION 62

REFERENCES 65

BIOGRAPHY 70

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

LIST OF TABLES

6.1 Optimum solutions of Differential Equation Solver. 44

6.2 Cost of Resources. 45

6.3 Parameter list. 45

6.4 Experimental results of Differential Equation Solver. 46

6.5 EWF Optimal solutions from OASIC. 49

6.6 EWF solutions from PSGA Synthesis. 50

6.7 EWF solutions from GA Synthesis. 50

6.8 Cost of Resources. 50

6.9 Parameter list. 51

6.10 Experimental results and timing report. 52

6.11 Experimental results and timing report (without bus). 52

6.12 Experimental results of the 21 time-steps and two-cycle multiplier (problem No.3). 54

7.1 EWF Optimal Solutions from OASIC. 57

7.2 EWF Optimal Solutions (without bus) from GA Synthesis. 57

7.3 EWF Optimal Solutions from AOT. 58

7.4 EWF Optimal Solutions (without bus) from AOT. 58

7.5 EWF Optimal solutions from AOT, comparing to the Fixed-Resource Mobility 60

7.6 EWF Optimal solutions (without bus) from AOT, comparing to the Fixed-Resource Mobility 60

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

LIST OF FIGURES

2.1 Differential Equation. 4

2.2 Differential Equation CDFG. 5

2.3 ASAP Scheduling of the differential equation. 5

2.4 ALAP Scheduling of the differential equation. 6

2.5 Mobility of the differential equation. 6

2.6 List Scheduling of the differential equation. 9

2.7 Variable lifetimes of the ASAP Scheduling. 14

2.8 Variable lifetimes of the ALAP Scheduling. 15

2.9 Variable lifetimes of the List Scheduling. 15

2.10 Data transfer of the ASAP Scheduling. 16

2.11 Data transfer of the ALAP Scheduling. 16

2.12 Data transfer of the List Scheduling. 16

3.1 Ants' behavior. 23

4.1 Search trees composed of partial solutions in AOT. 28

5.1 Applying AOT to High-Level Synthesis. 36

5.2 Example CDFG. 41

5.3 Decision tree of the scheduling process. 41

6.1 Differential Equation. 43

6.2 Differential Equation CDFG. 44

6.3 Optimum Scheduling of the differential equation. 46

6.4 Variable lifetimes of the Optimum Scheduling. 47

6.5 Data transfers of the Optimum Scheduling. 47

6.6 Target architecture of the differential equation. 47

6.7 Elliptical wave filter. 48

6.8 Elliptical wave filter CDFG. 49

6.9 Solutions of the two-cycle multiplier. 53

6.10 Solutions of the pipelined multiplier. 53

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 1

INTRODUCTION

With the advance of the fabrication technology and the marketing demand, the complexity of Very Large Scale Integration (VLSI) is increasing everyday. Computer-aided design (CAD) tools play an important role in today's design process. Without these tools it is not possible to complete the design nor to meet today's time-to-market. While performance is the critical issue in the general purposed systems such as personal computers. The designers of embedded systems have to face many other issues. Power requirement, reliability, size, and integration are some of the examples. These requirements lead to the new research area called Hardware-Software Co-design (S. Parameswaran 1998), which aims to synthesis the target system on one or a few chips from the system description. The tools have to consider the cost of implementing each part of the system in both hardware and software to optimize them within the target contraints.

Digital signal processing (DSP) is one of widely applied technology, it will reside in almost every electronic equipment in the near future. Some of the applications which we can instantly think of are telecommunication, multimedia, information retrieval, and human interface. Most of DSP algorithms require many of add and multiply operations, they are the computation intensive algorithms. The common way to implement a DSP algorithm is composed of three steps. First, the critically computational loops are synthesized into hardware modules. Second, the remaining computations are compiled into software modules, which will be executed by an embedded processor. Then the flexibility of software is used to concert these modules together.

In this research we target the basic building block of the over all design process, the hardware implementation of the inner loop of the DSP algorithm. This topic is still interested by some researchers, the recent publication (E. Torbey and J. P. Knight 1999) is an example. While we focus on this basic building block the proposed algorithm is aimed for the integration of the whole process in mind.

In hardware synthesis, usually the algorithm or behavior of the target system will be described in a high-level language such as VHDL or Verilog. This description will be transformed into a control data flow graph (CDFG). Then a batch of algorithms will

transfer this CDFG into register-transfer level (RTL), consisted of a data path and a control unit. This process, as explained in (M. C. McFarland, A. C. Parker, and R. Camposano 1990), is called High-Level Synthesis (HLS). The objective of this research is to propose an algorithm to synthesize the data path. This problem is known to be a NP-Hard problem as stated in (M. C. McFarland, A. C. Parker, and R. Camposano 1990; C. H. Gebotys and M. I. Elmasry 1992), and (M.R. Gary and D.S. Johnson 1979).

The main contribution of our work is that the proposed algorithm makes use of many sources of existing knowledge in terms of algorithms and design rules and integrates them in a flexible way to solve hard real-world problems. We demonstrate its strength by solving a High-Level Synthesis problem.

In the next chapter, High-Level Synthesis is reviewed, followed by an overview of Ant Algorithms. Then the AOT algorithm is introduced. The experiments are described and the results are reported. An improvement of the algorithm by using the Fixed-Resource Mobility is demonstrated. Finally, the algorithm is summarized and discussed in the last chapter.

CHAPTER 2

HIGH-LEVEL SYNTHESIS

2.1 Overview

A High-Level Synthesis (M. C. McFarland, A. C. Parker, and R. Camposano 1990) is a process that consists of many steps. A behavioral description of the target system is successively transformed into a data path and a control unit.

Many algorithms were proposed to solve each synthesis step as can be found in the literature, such as (D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin 1992; P. Michel, U. Lauther, and P. Duzy 1992), and (G. De Micheli 1994). Most of them are heuristic algorithms and cannot guarantee to find the optimum solution. Even if we can get an optimum solution of each step, combining them together may not give us the optimum result. Some of these heuristic algorithms, from (A. Kumar, A. Kumar, and M. Balakrishnan 1995; A. Sharma and R. Jain 1993; A. Sharma and R. Jain 1994; S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997; J. M. Rabaey and M. Potkonjak 1994), (P. G. Paulin and J. P. Knight 1989b; W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, J. L. Van Meerbergen, and A. Van Der Werf 1995), and (F. J. Kurdahi and A. C. Parker 1987), will be reviewed in the next section. Integer Linear Programming (ILP) (C. H. Gebotys and M. I. Elmasry 1992), (M. Rim and R. Jain 1994; M. Langevin and E. Cerny 1993), and (S. Chaudhuri and R. A. Walker 1996) was applied to search for the optimal solution, but this requires exponential computation time. Genetic Algorithms (GA) (M. J. M. Heijligers and J. A. G. Jess 1995; M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995), and (E. Torbey and J. P. Knight 1999) were applied to trade off between the quality of the solution and the computation time. These works will be reviewed in the following sections.

2.2 Heuristic Algorithms

Many algorithms were proposed to solve each synthesis step as can be found in various literatures such as (D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin 1992; P. Michel, U. Lauther, and P. Duzy 1992), and (G. De Micheli 1994). Most of them are heuristic algorithms and can not guarantee the optimum solution. If we can

get an optimum solution for each step, combining these together may not give us the optimum data path.

High-level synthesis is a process which consists of many design steps. The list of common synthesis steps are as follows:

1. Resources Approximation by Lower-Bound determination.
2. Scheduling.
3. Functional unit Allocation and Assignment.
4. Register Allocation and Assignment.
5. Bus Allocation and Assignment.

The next subsections are organized as follows. The differential equation solver algorithm which is used as an example is presented in section 2.2.1. The As Soon As Possible (ASAP), As Late As Possible (ALAP), and Mobility are presented in section 2.2.2. Then the synthesis steps which were described previously are reviewed in more details in section 2.2.3, 2.2.4, 2.2.5, 2.2.6, and 2.2.7.

2.2.1 Differential Equation Example

$$y'' + 3xy' + 3y = 0$$

```
While (x<a) repeat:
        x1= x+dx;
        u1= u-(3*x*u*dx)-(3*y*dx);
        y1= y+(u*dx);
        x = x1; y = y1; u = u1;
end;
```

Figure 2.1: Differential Equation.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

In this paper the Differential Equation is used as an example. It was also used in many papers. Fig. 2.1 shows the equation and the algorithm to solve it. Fig. 2.2 shows the control data flow graph (CDFG) of the inner loop of the algorithm.

This CDFG has 5 multiply operations, 2 add operations, 2 subtract operations, and 1 compare operation. It also has 9 variables, 3 of them (x, y, and u) are used to

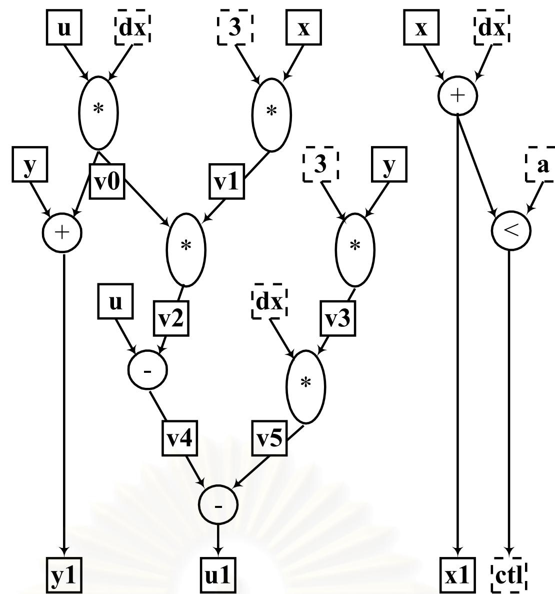

Figure 2.2: Differential Equation CDFG.

keep values for the next loop. The constant values 3, a, and dx are shown in the dashed boxes. For multiply operations they were assigned to use multi-cycle multipliers with execution time equal to two cycles. For other operations they were assigned with the functional units with one execution cycle. In this example the target execution time is 7 control steps.

2.2.2 As Soon As Possible (ASAP), As Late As Possible (ALAP), and Mobility

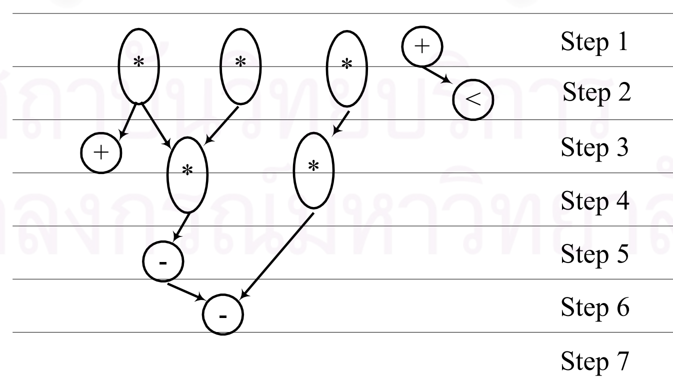

Figure 2.3: ASAP Scheduling of the differential equation.

ASAP and ALAP are the earliest and the latest time-step of an operation to be scheduled. These values were computed from the critical path of each operation by

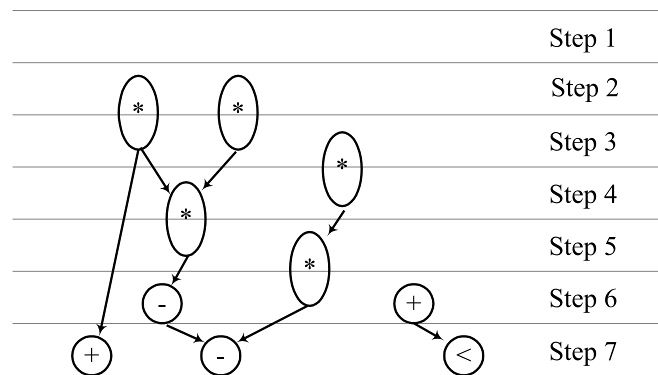

Figure 2.4: ALAP Scheduling of the differential equation.

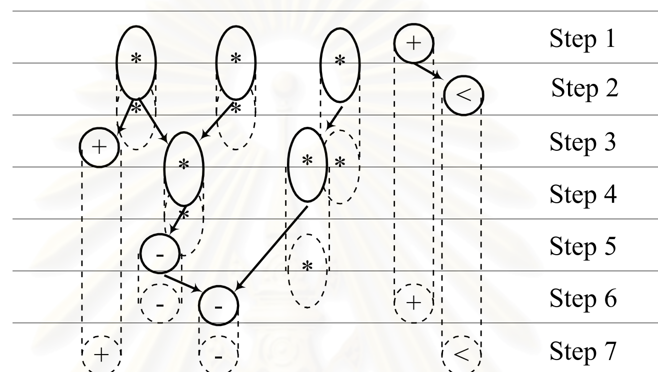

Figure 2.5: Mobility of the differential equation.

assuming that there were unlimited resources. ASAP and ALAP are very useful, it is a fundamental step in many synthesis algorithms. Fig. 2.3 and Fig. 2.4 show the exmaple of ASAP and ALAP scheduling.

Mobility of an operation is a number of cycles between ASAP and ALAP as shown in Fig. 2.5. Mobilities are used in many scheduling algorithms. They can be used to approximate the search space.

2.2.3 Lower Bound Algorithms

In the synthesis algorithms, we use lower bounds to reduce the search space. These include the lower bounds on number of time-steps for execution, the number of FUs, the number of registers, and the number of buses. There are many algorithms which were developed for finding lower bounds. Most of the algorithms were based on the time-frame and the list scheduling algorithms. Time-frame is a range of time-step. Since the lower bounds are computed before the scheduling process, we do not know the exact time-step of each operation. One way to solve this problem is to average the

resources on all possible time-frames from the first time-step to the maximum time-step. A resource will be counted in a time-frame if it is guaranteed by the mobility of the operation (range from ASAP to ALAP). Algorithms based on time-frame will have complexity at least equal to the square of the number of time-steps, because all possible time-frames have to be considered. The algorithms that fall in to this category are (A. Kumar, A. Kumar, and M. Balakrishnan 1995; A. Sharma and R. Jain 1993; A. Sharma and R. Jain 1994), and (S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997).

For the algorithms, which based on list scheduling, the priority functions are an important issue. The common priority functions are mobility and ALAP of operations. The algorithms that fall in to this category are (M. Rim and R. Jain 1994) and (M. Langevin and E. Cerny 1993).

Another way to compute the lower bound is to formulate the problem into integer linear programming (ILP) problem. After that, some constraints (usually the precedence constraints) were relaxed. Then the relaxed ILP problem will be solved by some polynomial time algorithms. The algorithms that fall in to this category are (M. Rim and R. Jain 1994; M. Langevin and E. Cerny 1993), and (S. Chaudhuri and R. A. Walker 1996).

In some cases, we can estimate a lower bound from some simple features of the problem. For example the number of registers must be at least equal to the number of loop variables, or the number of buses must be at least equal to the number of inputs and outputs of the FUs which must be transfered in the same time-step of the same type (C. H. Gebotys and M. I. Elmasry 1992).

Algorithms in (S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997) and (J. M. Rabaey and M. Potkonjak 1994) can compute lower bound for all type of resources (FU, register, and bus or interconnection), which are very useful for this research.

2.2.4 Scheduling

Scheduling is a process that assigns each operation in the CDFG to a time-step. There are many algorithms proposed for this design step and it is the most critical step in the synthesis process. Most of the remaining steps will be directly effected by the result from scheduling. Two main types of scheduling are the time- constrained scheduling and resource-constrained scheduling. The time-constrained scheduling tries

to minimize the resoureces used within fixed execution cycles. While the resource-constrained scheduling tries to minimize the execution cycles for the fixed resources. Next some of the well-known algorithms will be briefly review as follows.

1. List Scheduling Algorithm.
2. Force-Directed Scheduling (FDS) Algorithm.
3. Force-Directed List Scheduling (FDLS) Algorithm.
4. Branch and Bound Algorithm.

List Scheduling

List Scheduling is a resource-constrained scheduling. The first step of List Scheduling is to specify the hardware constraints, which usually are the number of functional units for the simple case. Then from the first time-step the ready operations will be assigned, and the process is repeated until all operations were assigned. If in any time-step there are more ready operations than the available hardware reources, we have to decide to defer some operations. Which operations to defer depends on the priority of each operation. The process of list scheduling is as follows.

1. Assign current time-step, $Ts = 1$.
2. For each resource type t
   (a) Determine the list of ready operations of type t, R.
   (b) Determine amount of avaiable resources of type t, n.
   (c) if $|R| \leq n$ assign all the operations in the list R to the time-step Ts.
   (d) else select n operations from the list R and assign them to the time-step Ts.
3. $Ts = Ts + 1$.
4. Repeat step 2 until all operations were assigned.

The priorities to select n operations from the list R can be calculated from mobility or ALAP value of the operation. Fig. 2.6 shows the result from list scheduling with two multipliers, one adder, one subtracter, and one comparator. In the first time-step,

we can see that there are three multiply operations were ready to be scheduled, as in the ASAP scheduling (Fig. 2.3). But we have to schedule them with only two multipliers. So we have to select only two of them. In this case, if we use the mobilities of the operations as the priorities, the operation $v3$ will be deferred, and we will get the result as in the Fig. 2.6. Otherwise, we will not be able to schedule the CDFG in 7 time-steps.

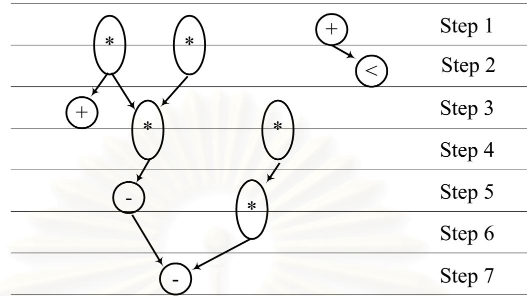

Figure 2.6: List Scheduling of the differential equation.

Force-Directed Scheduling

The Force-Directed Scheduling was introduced in (P. G. Paulin and J. P. Knight 1989b) and (P. G. Paulin and J. P. Knight 1989a). This algorithm was refined later by Verhaegh *et al.* in (W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, J. L. Van Meerbergen, and A. Van Der Werf 1995). As stated in (P. G. Paulin and J. P. Knight 1989b) that the intent of this algorithm is to reduce the hardware resources by balancing the concurrency of the operations assigned to them but without lengthening the total execution time. It is a time-constrained scheduling.

This algorithm is an iterative algorithm. Each iteration composes of three steps. First step is to evaluate the time-frame for each operation by ASAP and ALAP. Next step is to calculate the distribution graphs (DG), which is the summation of the propabilities for each type of operation for each time-step. The DGs indicate the concurrency of similar operations.

$$DG(i) = \sum_{Opntype} Prob(Opn, i). \tag{2.1}$$

In this Eq. 2.1, i indicates a time-step, $Prob(Opn, i)$ is the probability of an operation in the time-step i.

The third step is to calculate the Forces. Forces is the change of the average of the distribution graphs in the time-frame, which were effected by the scheduling

assignment. Suppose the time-frame was changed from $[t,b]$ to $[nt,nb]$, $Force(nt,nb)$ could be calculated as in the Eq. 2.2.

$$Force(nt,nb) = \sum_{i=nt}^{nb}[DG(i)/nb - nt + 1)] - \sum_{i=t}^{b}[DG(i)/b - t + 1)]. \qquad (2.2)$$

If there is an assignment of an operation to a time-step, the time-frame of the operation will be reduced to 1, we called this a self-Force. For the predecessors and the successors of the scheduled operation, their time-frames may be effected by this scheduling. These forces were called precessor Forces and successor Forces. The total Force of a scheduling assignment is the summation of self-Force, predecessor Forces, and successor Forces.

The assignment which has the lowest Force will be selected. The process of Force-Directed Scheduling is as follows.

1. Evaluate time-frames.
   (a) Find ASAP.
   (b) Find ALAP.
2. Update distribution graphs.
3. For each operation and its feasible time-step.
   (a) Calculate self-Force.
   (b) If there are predecessor operations, add predecessor Forces.
   (c) If there are successor operations, add successor Forces.
4. Schedule the operation to the time-step with the lowest force.
5. Repeat step 1 until all operations were scheduled.

The detail of this algorithm can be found in (P. G. Paulin and J. P. Knight 1989b).

Force-Directed List Scheduling

This scheduling algorithm is based on list scheduling algorithm. In this algorithm the force, as in Force-Directed Scheduling, is used as a priority function. So when there

are more ready operations than the available resources, the algorithm will defer the operations which will produce the lowest Force.

Branch and Bound Scheduling

Branch and bound technique is used in many scheduling algorithms, (S. Y. Ohm and C. S. Jhon 1992) and (P.-Y. Hsiao, G.-M. Wu, and J. Y. Su 1998) are the examples.

For the Lower bound directed Scheduling (LBS) algorithm in (S. Y. Ohm and C. S. Jhon 1992), the initial node of the decision tree is the ASAP scheduling. Each branched-off node will have an operation defers its scheduling time-step by one. For each node, two values have to be computed. The first value is the cost of the scheduling. The second value is the lower-bound cost of its branch. This lower-bound can be computed by using the time-frame technique as mentioned before. While traversing in the decision tree, the algorithm will keep track of the best explored schedule, the lowest cost. If the lower-bound cost of any node is equal to or greater than the best scheduling, it will be pruned off.

This algorithm uses the depth-first search strategy. It selects the next node to be explored by using the branch candidate set (BCS). The BCS is the list of operations in a time-step with the most concurrency for each operator type. The process of this algorithm is as follows.

LBS

1. Initialize initial state, $I =$ ASAP scheduling.

2. Initialize minimum cost state, $M = I$.

3. $Findbest(I)$.

Findbest(S)

1. If $cost(S) = LBcost(M)$ optimum solution found terminate.

2. If $LBcost(S) \geq cost(M)$ pruned this node.

3. If $cost(S) < cost(M)$ $M = S$.

4. Construct BCS

5. For each operation in BCS

   (a) $N = S$ which having the selected operation deferred by one time-step.

   (b) $Findbest(N)$

In (P.-Y. Hsiao, G.-M. Wu, and J. Y. Su 1998), the MPT-based branch-and-bound algorithm, each node represents the schedule assignment of an operation. Each node in the same level of the tree will represent the assignment of the same operation. The sequence of operations for scheduling is started from the critical path, follows by the paths related to the critical path and the rest of paths. In each path the leaf node is scheduling first.

To select a node for exploring, the algorithm uses six priority rules as follows.

1. Precedence constraint.

2. Lowest cost matrix among the current candidates.

3. Minimum value of MPT items.

4. Backtrack among ancestor, until the lowest-cost ancestor matrix can be found.

5. Backtrack among ancestor, until the lowest MPT value can be found.

6. Latest time-step.

The first rule is obvious, since we need the solution which does not violated the precedence constraint.

$$M_{i,j} = \begin{bmatrix} m_{11} & m_{21} & \cdot\ \cdot & m_{k1} \\ m_{12} & m_{22} & \cdot\ \cdot & m_{k2} \\ . & . & .\ \ . & . \\ m_{1n} & m_{2n} & \cdot\ \cdot & m_{kn} \\ M_{opn(1)} & M_{opn(2)} & \cdot\ \cdot & M_{opn(k)} \end{bmatrix} \tag{2.3}$$

The cost matrix could be described as in Eq. 2.3. In this equation k is the total number of types of all operations. n is the maximum number of time-step. m_{ij} indicates amount of type i operations already assigned to the time-step j. $M_{opn(i)}$ is the maximum number of m_{ij} in the i column. The last row of the matrix $M_{i,j}$ indicates

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

the cost of the current scheduling.

$$M_{opn(i)} = \max_{j=1}^{n} m_{ij} \tag{2.4}$$

$$MPT_{opn(type)}(j) = \sum_{opn(type)} Poss(opn, j) \tag{2.5}$$

$$Poss(opn, j) = \begin{cases} 1 & \text{the mobility of } opn \text{ involves time-step } j \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The third value is the MPT value. It can be described as in Eq. 2.6. It is used to find the best time-step for scheduling. If these priorities could not select a node from the curent candidates to explore, the algorithm backtracks and compares the priorities values of their ancestors. When the above five priorities is not work , the last thing to do is to explore the latest time-step.

2.2.5 Functional Unit Allocation and Assignment

Functional unit allocation and assignment is a process that allocates and assigns each operation in the CDFG to the allocated functional units. If it is a resource-constrained scheduling then functional units are allocated before scheduling. For time-constrained scheduling, after the scheduling, we will know the assigned time-step of each operation as in Fig. 2.3, 2.4, and 2.6 then the number of functional units for each type can be directly counted. If there is any case that one operation can be mapped to more than one operators, we need an algorithm for functional unit assignment. For example, an assignment can be done in a sequence from the earliest to the latest time-step to reduce the infeasible search. All assignments (functional unit, register, and bus) are closely related to each other. Optimum solution in one area may not be an overall optimum solution. We can transform these problems into clique partitioning problems, but clique partitioning problem itself is an NP-Completed problem.

2.2.6 Register Allocation and Assignment

Register allocation and assignment is a process that allocates enough registers and assigns each variable in the CDFG to allocated registers. From the scheduled operations

we can form the variable lifetime table as in Fig. 2.7, 2.8, and 2.9. The lower-bound on number of registers can be determined by the maximum number of overlap variable register lifetimes, for example the ASAP Scheduling will required as least 6 registers as we can see from Fig. 2.7 in control step number 3. Kurdahi *et al.* (F. J. Kurdahi and A. C. Parker 1987) shows how to adopt the Left-Edge algorithm, which was invented for the channel routing in VLSI design, to this problem. But this polynomial time algorithm can not give us an optimum solution in some case. The loop variable is an example that the Left-Edge algorithm can not be applied. This problem can be transformed in to the clique partitioning problem (C. J. Tseng and D. P. Siewiorek 1983) by forming a graph, which vertices are the set of variables and each pair of the vertices will be connected by an edge if their lifetimes do not overlap. But this clique partitioning problem has an exponential time complexity.

A problem as in (L. Stok 1992) which should be mentioned here is about the overlap of the lifetime between the old loop variable an the new generated loop variable. If the scheduled CDFG produces a new loop value before its last used of old value, the new loop value can not be kept in the same register. For example in the ASAP Scheduling as in Fig. 2.7, this problem occurs with the old loop variable x and the new loop variable $x1$. The x variable is needed in control step 2 but the $x1$ variable is defined in the same step as marked by the dashed lines. So the $x1$ value will have to be stored in another register and later transfers to the old loop register for the next execution loop.

x y u xl yl ul v0 v1 v2 v3 v4 v5

3	Step 1
4	Step 2
6	Step 3
6	Step 4
5	Step 5
4	Step 6
3	Step 7
3	Step 1

Figure 2.7: Variable lifetimes of the ASAP Scheduling.

2.2.7 Bus Allocation and Assignment

Bus allocation and assignment is a process that allocates buses and assigns each data transfer in the CDFG to the allocated buses. From the scheduled CDFG we can

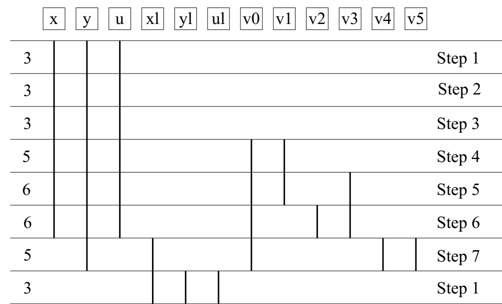

Figure 2.8: Variable lifetimes of the ALAP Scheduling.

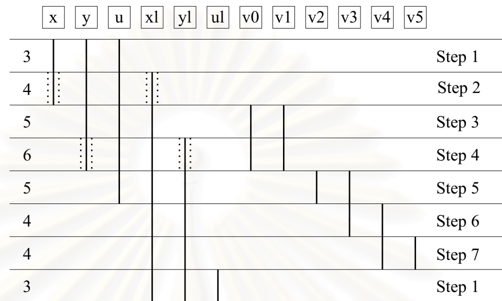

Figure 2.9: Variable lifetimes of the List Scheduling.

form the data transfer table as in Fig. 2.10, 2.11, and 2.12. The lower-bound on number of buses can be determined by the maximum number of parallel data transfers for all control steps , for example the ASAP Scheduling will required as least 10 buses as in Fig. 2.10 in control step number 2. As in (C. J. Tseng and D. P. Siewiorek 1983), this problem can be transformed in to the clique partitioning problem by forming a graph, which vertices are the set of data transfers and each pair of the vertices will be connected by an edge if both transfers are in the different control step or have the same data source.

The problem about the overlapped loop variable lifetime can be solved by assigning the new value data transfer in the same or after the last use control step of old value. As in Fig. 2.10 and 2.12 the $x1$ value can be transferred to the old loop register in the same last use control step, and shares the bus with the feeding of $x1$ value to the comparator. For the case of y variable in Fig. 2.9, another data transfer is required as shown in Fig. 2.12 as a separate transfer in step 4.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

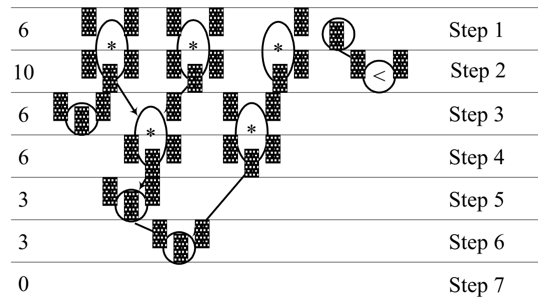

Figure 2.10: Data transfer of the ASAP Scheduling.

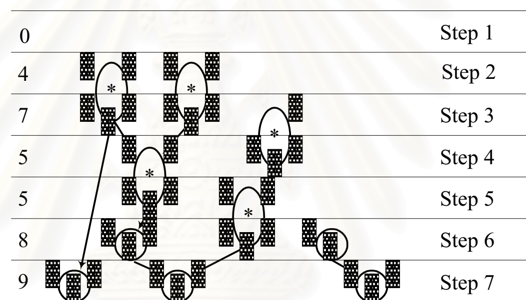

Figure 2.11: Data transfer of the ALAP Scheduling.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

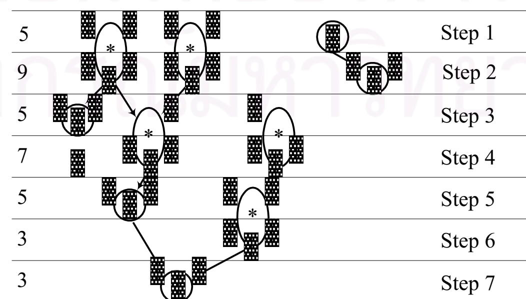

Figure 2.12: Data transfer of the List Scheduling.

2.3 Integer Linear Programming

An important technique that should be mentioned here is an integer linear programming (ILP) technique. In this technique, the constraints in the problem have to be formulated into equations, then these equations will be solved by the algorithms for solving the integer linear programming problem. Normally, these techniques still require an exponential computation time. Many papers (C. H. Gebotys and M. I. Elmasry 1992; C. H. Gebotys and M. I. Elmasry 1993; C. H. Gebotys and M. I. Elmasry 1991) and (C. H. Gebotys and M. I. Elmasry 1990) were published by Gebotys *et al.* and they are very wellknown, Some of them are used as a target references.

An example of these techniques (C. H. Gebotys and M. I. Elmasry 1992) is presented here. In this section the following terminology is used:

k a code operation.

$k_1 \prec k_2$ represents the partial order between k_1 and k_2.

$x_{j,k} = 1$ represents the assignment of operation k to the time-step j ($j \in Z$, set of intergers).

$In(k)$ number of distinct inputs (≥ 1) to k.

$Out(k)$ number of distinct outputs ($= 1$).

$j_z = R(k_z)$ means that $asap(k_z) \leq j_z \leq alap(k_z)$.

$k_z \in op(C_z, L_z)$ means that operation k_z requires C_z time-steps to produce an output data value and can accept a new input data every L_z time-steps.

The number of functional unit of type $i = I_i$. $k \in i$ means that code operation k is implementable by the functional unit type i (or $i \in op(C, L)$).

R number of registers ($R \in Z$).

B number of buses ($B \in Z$).

The first constraint is the assignment constraint. This equation (2.7) ensures that each operation will be assigned once.

$$\sum_{j \in R(k)} x_{j,k} = 1, \ \forall k \tag{2.7}$$

The second constraint is the precedence constraint. Eq. 2.8 ensures that k_2 is scheduling before k_1 when $k_2 \prec k_1$. In this equation only one operation could be

assigned in the overlap mobility region. Since operation k_1 has to be assigned after the completed time of operation k_2, so the mobility of k_2 has to be biased by its execution time C_2.

$$\sum_{\substack{j_1 \leq j \\ j_1 \in R(k_1)}} x_{j_1,k_1} + \sum_{\substack{j-(C_2-1) \leq j_2 \\ j_2 \in R(k_2)}} x_{j_2,k_2} \leq 1,\ \forall\ k_2 \prec k_1,\ j \in R(k_1) \cap (R(k_2) + C_2 - 1). \quad (2.8)$$

The third constraint is the functional unit constraint. Eq. 2.9 ensures that no more than I_i functional units is required. For each time-step j the number of operations which has the same type i and were assigned in the range $[j, j + L - 1]$ is limited to I_i.

$$\sum_{\substack{k \in i \\ j \in R(k)}} \sum_{j_1=j}^{j_1=j+(L-1)} x_{j_1,k} \leq I_i,\ \forall\ j,\ i \in op(C, L). \quad (2.9)$$

The fourth constraint is the register constraint. Eq. 2.10 ensures that no more than R registers is required. The term $\sum_{j_1 \leq j-(C_n-1),\ j_1 \in R(k_n)} x_{j_1,k_n}$ means the operation is completed before or at j. The term $\sum_{j_2 > j,\ j_3 \in R(k_e),\ k_n \prec k_e} x_{j_2,k_e}$ means the operation is started after j. The term $\sum_{j_3 \leq j,\ j_3 \in R(k_e),\ k_n \prec k_e} x_{j_3,k_e}$ means the operation is started before or at j. While the term $\sum_{j_4 \leq j-(C_n-1),\ j_4 \in R(k_n)} x_{j_4,k_n}$ means the operation is completed after j. The number of registers is computed by counting the number of arcs $k_n \prec k_e$ that cross the time-step j. The number of arcs that cross time-step j are computed by dividing the number of heads and tails of the arcs that cross j by 2. In the equation, if an arc crosses j then the head and the tail will be counted by two. If an arc does not cross j then the head and the tail will cancel each other to zero.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

$$\sum_{k_n} \left(\sum_{\substack{j_1 \leq j-(C_n-1) \\ j_1 \in R(k_n)}} x_{j_1,k_n} + \sum_{\substack{j_2 > j \\ j_3 \in R(k_e) \\ k_n \prec k_e}} x_{j_2,k_e} - \sum_{\substack{j_3 \leq j \\ j_3 \in R(k_e) \\ k_n \prec k_e}} x_{j_3,k_e} - \sum_{\substack{j_4 \leq j-(C_n-1) \\ j_4 \in R(k_n)}} x_{j_4,k_n} \right) \leq 2R, \quad (2.10)$$

$\forall\ j$, and for all maximum sets of arcs $(k_n \prec k_e)$ that cross j each with unique heads (k_n).

The fifth constraint is the bus constraint. Eq. 2.11 ensures that no more than B registers is required. The equation counts the number of inputs and outputs for each

time-step.

$$\sum_{\substack{k \\ j \in R(k)}} (In(k))x_{j,k} + \sum_{\substack{k_1 \in R(j_1) \\ j_1 = j-(C_1-1)}} (Out(k_1))x_{j_1,k_1} \leq B, \ \forall\ j. \tag{2.11}$$

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

2.4 Genetic Algorithms

In 1975, Holland (J. H. Holland 1975) published the concept of Genetic Algorithms (GA). It was successfully applied to many optimization problems. GA optimizes problems by encoding the problem into strings called chromosomes. The first population of the chromosomes is randomly generated. The subsequent generations are obtained by selection, crossover, and mutation.

Selection is a process to select chromosomes. The probabilities of selection bias by the fitness values, which were computed from a fitness function. The fitness function is a function to compute the quality of a chromosome. Roulette wheel selection is a selection, which probability of each chromosome to be selected is a direct proportion of its fitness value.

Crossover is a process that takes two chromosomes as inputs, and generates two new chromosomes. It begins by generating a random crossover point, which is a point where two chromosomes are cut and swaps between each other. For example, if we have chromosome $A = [1, 2, 3, 4, 5, 6]$ and chromosome $B = [a, b, c, d, e, f]$. The crossover point was randomly selected and equal to 2. The crossover will produce two new chromosome $C = [1, 2, c, d, e, f]$ and chromosome $D = [a, b, 3, 4, 5, 6]$.

Mutation is a process to perturb each element in the chromosome. The intention of these crossover and mutation is to generate new promising chromosomes as in the nature. The process of the Simple Genetic Algorithm (SGA) is as follows.

1. Initialize the population, the first set of chromosomes, P.
2. Set generation, $g = 0$.
3. $g = g + 1$.
4. Compute fitness of every chromosome.
5. $Q = \phi$.
6. For $|P|/2$ loops
   (a) Select 2 chromosomes from P, by roulette wheel selection.
   (b) Crossover the selected chromosomes.
   (c) Mutate the crossovered chromosomes.

(d) Add the mutated chromosomes to Q.

7. $P = Q$.

8. repeat step 3 until $g \geq maxgen$.

There were many works that use Genetic Algorithm to solve High-Level Synthesis problems (M. J. M. Heijligers and J. A. G. Jess 1995; M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995), and (E. Torbey and J. P. Knight 1999). We will review each of them in the following paragraphs.

In (M. J. M. Heijligers and J. A. G. Jess 1995), they used a modified version of list scheduling algorithm. The chromosome is composed of a permutation of operations as a priority list. To add a resource constraint, outputs from a lower bound algorithm are used as minimum resources. Then a list of extra resources is added to the chromosome. So while the algorithm searches for the best priority list, it also searches for a set of the optimum resources. One problem of this method is that it is possible to have more than one chromosomes produce the same schedule. This many to one mapping between chromosomes and the schedule leads to the inefficiency of the search. Another problem is that the list scheduling algorithm, for transforming from a chromosome to a schedule, may excluded the optimum solution in some case. They created a new algorithm called Topological Sorted Scheduling to overcome this problem.

In (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995), the chromosome encoding has almost the same concept as in (M. J. M. Heijligers and J. A. G. Jess 1995). A chromosome is composed of work remaining number for each operation and the number of functional units for each type. The work remaining number is the worst case delay on the longest path of the node to the output. These values were initialised in the chromosomes and then perturbed by a random value. They were used as a priority in the list-scheduling-like algorithm called Most-Work-Remaining (MWR) heuristic algorithm. About the number of functional units, in this algorithm they use absoulute numbers of resources encode in the chromosomes, instead of the numbers, which were biased from the lower bound solution.

Knight *et al.*, the authors of Force-Directed Scheduling (P. G. Paulin and J. P. Knight 1989b), were also interested in GA, and they had done some researches on this topic (E. Torbey and J. P. Knight 1999; E. Torbey and J. P. Knight 1998) and (R. S.

Martin and J. P. Knight 1994). In (E. Torbey and J. P. Knight 1999), a chromosome is composed of the number of functional units for each type and the relative time-step to the mobility range of each operation. The storage is handled by inserting a storage operator into the CDFG at every edge. So the storage can be scheduled and bind at the same time, also different type of storage units can be included in the optimization process. This algorithm also reports the fastest execution time on the EWF benchmark.

The main contribution of these people is to find a technique to encode or decode the chromosome with feasible solutions, which prevents the GA from wasting the computation capability in evaluating the infeasible solutions.

2.5 Conclusion

For the Heuristic algorithm, the advantage is the speed of the algorithm, which usually is in the $O(n^2)$ complexity. One of the disadvantages is that it can not guarantee the optimality of the solution. Many heuristic steps have to be integrated together, for the High-Level Synthesis process, which may lead to poor solution.

For ILP, the strong point is that it can guarantee the optimal solution. The disadvantage is that it has exponential time complexity. It is difficult to form the efficient constrained equations, in order to improve the performance.

For GA, the advantage is that we could trade-off between the quality and processing time. To get a good performance algorithm, it requires a good chromosome encoding, and a good genetic operators.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 3

ANT COLONY OPTIMIZATION ALGORITHMS

3.1 overview

In this section, we give an overview and the development of the Ant Algorithms. Ant Algorithms were recently developed by Dorigo *et al.* as in (A. Colorni, M. Dorigo and V. Maniezzo 1991; M. Dorigo and A. Colorni 1996; L. M. Gambardella and M. Dorigo 1995), and (M. Dorigo and L. M. Gambardella 1997) to solve the travelling salesman problems (TSP) and later were expanded to solve other problems. The algorithms are based on the natural behavior of an ant colony, which uses a pheromone as its communication medium. While travelling, the pheromone is left along the way as a trail, and ants will use the information (pheromone) as their guidance. By this behavior, ants can find the shortest path as will be explained as follows:-

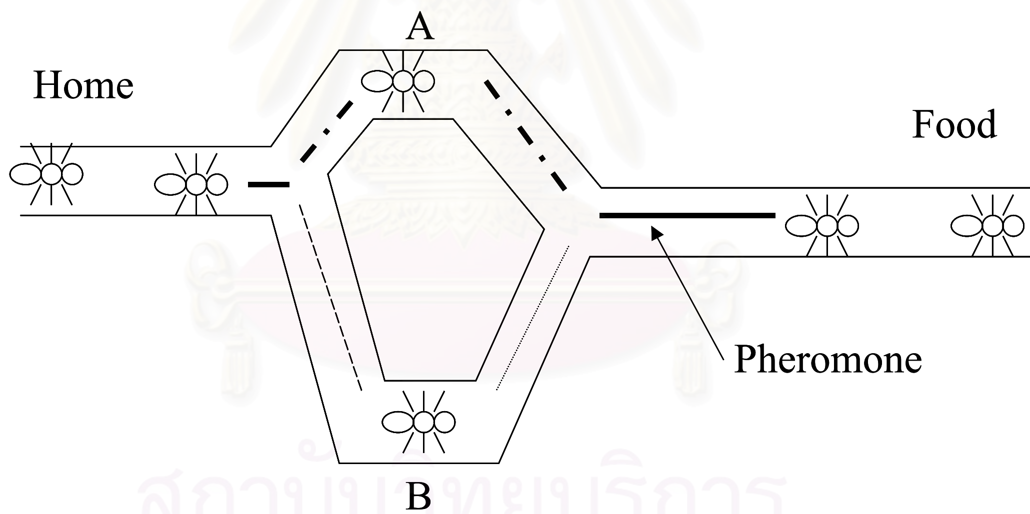

Figure 3.1: Ants' behavior.

1. Suppose ants have to travel from home to the food and travel back to their home as in the Fig. 3.1.

2. When ants get to the decision point, first when there is no pheromone, they choose the path randomly. As a result the number of ants should divide to path A and path B equally.

3. Suppose all ants walk at the same speed and they deposite the same amount of pheromone on the ground.

4. Since path A is shorter than path B, after a while the amount of pheromone on path A should be higher than path B.

5. More ants prefer path A since there is more pheromone. Then the pheromone gets even higher.

6. Less ants choose path B and by the evaporation, the pheromone on path B get even lower.

7. Most ants travel on path A, the shorter path.

Next, an example of applying this behavior to the travelling saleman problem is explained. The travelling salesman problem is the problem of a salesman who wants to find a shortest possible path. The path begins from his hometown through a given set of customer cities and returns back to his home. One of the possible implementations of the algorithm should be as follows:-

1. Initialize parameters such as the number of ants, pheromone weights, and heuristic weights. In TSP, a heuristic weight is inverse proportion to a distance between two cities.

2. Each ant constructs a tour by randomly selecting the unvisited cities. The selection is biased by the pheromone weights and the heuristic weights between the current city and the next possible cities. It will repeat the selection until all the cities are visited.

3. To simulate the pheromone evaporation, the pheromone on every path between any two cities is decreased by a proportion.

4. The distance of each tour is calculated, and the pheromone along the tour is increased by the inverse proportion of the tour length.

5. Repeat step 2 through step 4 until the termination condition is met.

The termination condition usually is the number of iterations or the CPU-time. Normally, as the number of iteration is increased, the tour length will be shorter and the pheromone along the best path will be higher than the pheromone in other paths.

This algorithm randomly selects the next nodes by the bias from the pheromone levels. This behavior (step 2) is called the state transition rule. After all tours are finished, the pheromone on every edge is decreased (step 3) to simulate the pheromone evaporation. The pheromone along the tour is increased (step 4) to simulate the pheromone that the ants have deposited along the tour. These behaviors (step 3 and 4) are called the pheromone updating rule. These two rules play an important role in ACO algorithms.

3.2 Ant System

Ant System (M. Dorigo and A. Colorni 1996) was the first algorithm introduced. The state transition rule of this algorithm is "make a decision among all possible choices, choose an action probabilistically proportional to the pheromone level in combination with the heuristic weights." The state transition rule of Ant System is as in Eq. (3.1).

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{u \in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

$p_k(r,s)$ is the probability that ant k in node r chooses to move to node s. $J_k(r)$ is the set of the unvisited nodes by ant k from the node r. $\eta(r,s)$ is the heuristic weight on edge (r,s) ,which is set to $1/\delta$. δ is the distance between node r and node s. $\tau(r,s)$ is the pheromone on edge (r,s). β is a parametrer to emphasize on the better values.

The pheromone updating rule of Ant System is as in Eq. (3.2). All ants are allowed to update their trails.

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \sum_{k=1}^{m}\Delta\tau_k(r,s) \tag{3.2}$$

$$\text{where } \Delta\tau_k(r,s) = \begin{cases} 1/L_k & \text{if } (r,s) \in \text{tour done by ant } k \\ 0 & \text{otherwise.} \end{cases}$$

α is a pheromone decay parameter, it is set to $0 < \alpha < 1$. L_k is the length tour performed by ant k, and m is the number of ants.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

3.3 Ant Colony System and Ant-Q

Ant-Q (L. M. Gambardella and M. Dorigo 1995) and Ant Colony System (ACS) (M. Dorigo and L. M. Gambardella 1997) are very similar algorithms. Ant-Q algorithm is inspired by Q-learning, a type of Reinforcement Learning algorithm (L. P. Kaelbling, M. L. Littman, and A. W. Moore 1996). The ACS is almost the same as Ant-Q but uses less computation. Their state transition rules are almost the same as in the Ant System algorithm except that they are heavily biased by the maximum pheromone level and heuristic weights. The state transition rule is as in Eq. (3.3).

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}\} & \text{if } q \leq q_0 \quad \text{(exploitation)} \\ S & \text{otherwise} \quad \text{(biased exploration)} \end{cases} \tag{3.3}$$

Where arg max is a function to select the maximum item in the set, q is a random number uniformly distribute in [0 .. 1], q_0 is a parameter ($0 \leq q_0 \leq 1$), and S is a random variable selected according to the probability distribution given in Eq. 3.1. The parameter q_0 determines the relative importance of exploitation versus exploration. Usually q_0 is set to a rather high value. For example, it is set to 0.9. If q is less than or equal to 0.9, ant chooses exploitation (best edge). If q is higher than 0.9, ant chooses exploration (random edge).

There are two types of the pheromone updating rules in this algorithm, the global pheromone updating rule and the local pheromone updating rule. For the global pheromone updating rule only the best solution is allowed to increase the pheromone level. The global pheromone updating rule is as in Eq. (3.4).

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau_{gb}(r,s) \tag{3.4}$$

$$\text{where } \Delta\tau_{gb}(r,s) = \begin{cases} (L_{gb})^{-1} & \text{if } (r,s) \in \text{global-best-tour} \\ 0 & \text{otherwise.} \end{cases}$$

For the local pheromone updating rule, every ant will decrease the pheromone level as it travels. This is done in order to prevent another ant from taking the same route. The pheromone updating rule is as in Eq. (3.5).

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\Delta\tau(r,s) \tag{3.5}$$

ρ is set in the range $0 < \rho < 1$. $\Delta\tau(r,s)$ is set to a constant parameter τ_0 in ACS, while it is set to a more complicate equation in Ant-Q.

3.4 $\mathcal{MAX} - \mathcal{MIN}$ Ant System

$\mathcal{MAX} - \mathcal{MIN}$ Ant System (T. Stützle and H. Hoos 1997) and ACS use the same state transition rule, as in Eq. 3.3. The main difference between the $\mathcal{MAX} - \mathcal{MIN}$ Ant System and ACS is that the pheromone level is controlled in a limited range $[\tau_{min}, \tau_{max}]$, or $\forall\tau(r,s)\ \tau_{min} \leq \tau(r,s) \leq \tau_{max}$. The improvement of this algorithm is to prevent ants from following the same route which caused search stagnation or premature convergence. The pheromone updating rule is as in Eq. (3.6).

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \Delta\tau_{best}(r,s) \tag{3.6}$$

$$\text{where } \Delta\tau_{best}(r,s) = \begin{cases} (L_{best})^{-1} & \text{if } (r,s) \in \text{best-tour} \\ 0 & \text{otherwise.} \end{cases}$$

3.5 Conclusion

All the improvements of these algorithms address two issues. First, they try to reduce random behavior of the algorithms, by using q_0 (ACS, Ant-Q, and $\mathcal{MAX} - \mathcal{MIN}$ Ant System) to make the searchs more direct. Second, to avoid premature convergence, ants are forced to search for new solutions. By decreasing the pheromone level as the ants travelling (ACS), it will reduce the probability for other ants to follow the same route. By limit the pheromone level ($\mathcal{MAX} - \mathcal{MIN}$ Ant System), there is always a chance for every possible choice. A good review of ACO can be found in (T. Stützle and M. Dorigo 1999).

CHAPTER 4

ANTS ON A TREE ALGORITHM

4.1 Overview

A solution of a High-Level Synthesis problem is composed of many parts, such as scheduling of the operations, allocation of functional units, and assignment of the operations. In normal practice a designer has to use many algorithms to find each part of the solution. A designer applies algorithms in sequence as shown in Fig. 4.1(a). In this figure, a circle indicates the problem to be solved, while an arrow indicates the selected partial solution. This partial solution effects the rest of the problem. Different partial solutions turn the problem into different problems. A different choice between alternative partial solutions leads to a different solution as shown in Fig. 4.1(b). Incorporating constraints on the solution, a search tree will look like Fig. 4.1(c). A problem arises as to which alternative we should consider first, since the number of alternatives grows exponentially with the size of the problem.

Instead of using only one result from each algorithm at each design step, we use each design algorithm to weight each possible result and an ACO is applied to optimize for the solution. We call this process the Ants On a Tree (AOT) algorithm.

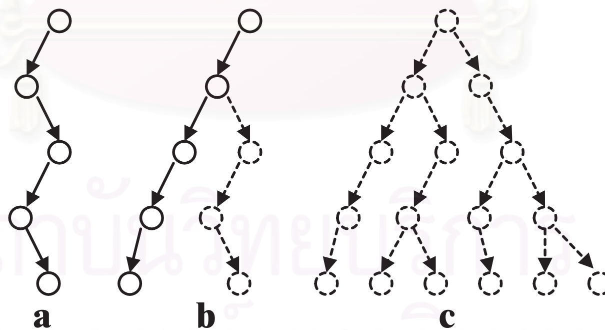

Figure 4.1: Search trees composed of partial solutions in AOT.

The fundamental concept in AOT is to use a structure called a decision tree to keep track of all the problem states that the system had reached. This is similar to the TOGAPS (C. P. Ravikumar and V. Saxena 1996), which maintains the search space for GA in a tree structure. There are some researchers who are interested in finding different structures to generate promising solutions as in (M. Pelikan, D. E. Goldberg, and E. Cantú-Paz 1999). In AOT, each ant searches for a solution and keeps its trail in

a decision path, which is a path in the decision tree. Next, we define the decision tree and the decision path.

A decision tree is a structure that represents a decision space for a design problem (T. M. Mitchell 1997). It keeps track of the knowledge found so far for the problem. Each node in the tree represents a design problem state. Each edge represents a choice that has been made. There is one root node, which represents the initial state of the design problem. A path from the root node to a leaf node is called a decision path. A decision path is a record of the decisions, which were made for a design.

The first node of the decision path is the starting point of the design process. From the first node, a decision is made in selecting a choice from all of the possible choices and the problem state transits to another state (node) in the design problem. This process is repeated until a complete design is found or the search reaches a dead end.

AOT is an optimization algorithm. The process of the AOT Algorithm is as follows:

1. Initialize paths for each ant by an initial design state.
2. For each ant, construct a decision path by means of the state transition rule.
3. Evaluate the cost function of each path. If the cost is satisfactory, then terminate.
4. Decrease the pheromone level of every edge in the tree to simulate the pheromone evaporation.
5. For each path, update the pheromone level according to the pheromone updating rule.
6. Prune each subtree that has an infeasible solution.
7. Prune each subtree that has a zero pheromone level.
8. Prune each subtree that has a cost higher than or equal to the best cost.
9. Use the Dynamic Niche Sharing process to find local minima of cost, which are called peaks.

10. Initialize the path for each ant in the next iteration by path exploration from the peaks.

11. Go to step 2.

The key procedures for this algorithm are the decision path construction, state transition rule, pheromone updating rule, path exploration, dynamic niche algorithm, and boosting. We will explain each one in turn.

4.2 Construction of a decision path

Construction of a decision path is a process in which a sequence of selections is made. This process begins from the first node, which is an initial design state. The possible choices are listed by the synthesis algorithm. Each possible choice is assigned a heuristic weight and a pheromone weight. A choice is made according to the state transition rule, which will be described next. Then the algorithm gets to a new state of the design problem. This process continues until the design is completed. While constructing the decision path, the decision tree is updated.

4.3 State transition rule

A state transition rule is a rule that is applied in order to advance from one design step to the next step by selecting a choice from the possible weighted choice list.

If there is any choice left, the next choice will be selected by the probability as in Eq. (4.1):

$$p(r,s) = \begin{cases} \frac{[h(r,s)]}{\sum_{u \in U(r)}[h(r,u)]} & \text{if } s \in U(r) \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

$p(r,s)$ is the probability that an ant in node r chooses to move to node s. $h(r,s)$ is the heuristic weight on edge (r,s). $U(r)$ is the set of the unvisited next nodes from the node r.

After all choices have been selected, then a choice will be randomly selected using the biases from pheromone level as in Eq. (4.2):

$$p(r,s) = \begin{cases} \frac{[\tau(r,s)+1]}{\sum_{u \in J(r)}[\tau(r,u)+1]} & \text{if } s \in J(r) \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

$\tau(r,s)$ is the pheromone on edge (r,s). $J(r)$ is the set of the next nodes, including the nodes that were pruned by pheromone evaporation, excluding the nodes that were pruned by the best cost and infeasible solutions, from the node r.

This state transition rule is different from those of ACO algorithms, because in ACO the selection is based on a linear combination of heuristic weights and pheromone levels, while in AOT the heuristic weights are used until all choices have been selected and then the pheromone levels are considered. The advantage of our method is that the determination of heuristic weights is independent of the pheromone level. This is very important for a problem such as High-Level Synthesis, which consists of many heuristic algorithms, while the same pheromone updating rule is used in every state.

Because the pheromone in AOT can be set to zero, the state transition rule is modified as in Eq. (4.2). Comparing this state transition rule with the state transition rule in the $\mathcal{MAX}-\mathcal{MIN}$ Ant System (T. Stützle and H. Hoos 1997), the actual pheromone level is limited to the range $[1, \tau_{max}+1]$.

Another modification in our algorithm is that the Q_0 parameter, which is used in the ACO state transition rule, was eliminated. This decision is influenced by the path exploration, which will be explained later.

4.4 Pheromone updating rule

The pheromone updating rule is the rule for calculating the new level of the pheromone. In this algorithm, we use the technique modified from the $\mathcal{MAX}-\mathcal{MIN}$ Ant System (T. Stützle and H. Hoos 1997). The pheromone level is also used to prevent the algorithm from storage explosion. In each iteration, the pheromone level of every edge in the tree is decreased by a fixed value α to simulate evaporation as in Eq. 4.3. Then for each path, the pheromone level will be set to a fixed value τ_{max} as in Eq. 4.4 .

$$\tau(r,s) \leftarrow \tau(r,s) - \alpha \tag{4.3}$$

$$\tau(r,s) \leftarrow \tau_{max} \tag{4.4}$$

The storage explosion problem is eliminated by this pheromone updating rule. In this algorithm the storage depends on the number of nodes in the decision tree. We will demonstrate an upper bound on the number of nodes in the tree to prove that the storage explosion problem is eliminated. In each loop of the algorithm the number of nodes is increased while each ant is constructing a decision path, and is decreased when the tree is pruned. The pheromone level on an edge that links to a node controls the lifetime of that node. Once a new node is created and is added to the tree by the decision path construction, the pheromone level is set to τ_{max}. In every loop of the algorithm, the pheromone level is decreased by α until it gets to zero and the node is pruned, unless the node is selected again. If the node is selected again its pheromone level is set to τ_{max}.

In the case that while constructing the decision paths, only new nodes are created and are added to the tree, no old node in the tree is selected, and the maximum number of nodes in the tree N_{max} can be computed as in Eq. (4.5):

$$N_{max} = P \cdot L_{max} \cdot \frac{\tau_{max}}{\alpha}, \tag{4.5}$$

where P is the number of paths in each iteration or loop, which equals the number of ants. L_{max} is the maximum path length, which can be computed for each problem. The maximum number of nodes generated in each iteration is equal to $P \cdot L_{max}$. The maximum node lifetime is equal to τ_{max}/α.

For the case that some of the old nodes are selected, the maximum node lifetime for the selected old nodes is equal to τ_{max}/α. No new node is created for each selected old node. From these facts, we conclude that N_{max} in Eq. (4.5) still holds as an upper bound. Hence, the number of nodes in the decision tree is bounded and a storage explosion, as occurs in reinforcement learning (L. P. Kaelbling, M. L. Littman, and A. W. Moore 1996), is avoided.

If the pheromone evaporation rate α is set to zero the decision tree will continue to grow and AOT will approximate an exhaustive search algorithm.

4.5 Path exploration

Path exploration is a process to initialize a new path from the selected path. The process begins by selecting a random exploration point in the path. Then a path

from the root to the exploration point is copied to a new ant as an initial path. This is similar to the use of the Q_0 parameter in the ACO state transition rule, in which Q_0 is a probability to preserve the maximum pheromone level path. The ACO state transition rule serves both roles, for exploitation and exploration, but AOT path exploration serves exploitation while the state transition rule serves exploration.

From the state transition rule in Eq. (4.1), one may think of AOT as an exhaustive search algorithm, but this is not the case, since some of the unvisited nodes might not be selected. When an ant randomly selects an exploration point that is located after a branch of the unvisited nodes, these nodes will not be explored by the ant.

4.6 Dynamic niche algorithm

The dynamic niche algorithm (B. L. Miller and D. E. Goldberg 1996) is an algorithm first applied to genetic algorithms to overcome the premature convergence problem and to explore the search space for finding niche answers. We applied it to AOT to prevent search stagnation, which is found in ACO. For each iteration, instead of exploiting only the global best or the iteration best path, every peak generated by the niche algorithm is used. In GA, the idea of niche sharing is to the share the fitness (cost) of chromosomes between their neighborhoods. Two chromosomes are said to be in the same neighborhood if the distance between them is less than some specific number. In AOT two paths are said to be in the same neighborhood if they share the same node within the path length L_{dis}, which is equal to half of the average path length of each iteration. The process of the modified Dynamic niche algorithm is as follows.

1. Sort the set of all the decision paths, P, in decreasing order.
2. Initialize the set of peaks, $Q = \phi$.
3. For each decision path, p, from the set P, in the sorting order
    (a) if p is not a neighborhood of any path in Q add p into Q.

4.7 Boosting

A boosting algorithm is applied in step 8. The cost of each decision path is checked against the minimum cost found so far. Any path with higher or equal cost

will be eliminated. This technique is similar to branch and bound algorithms as used in (P.-Y. Hsiao, G.-M. Wu, and J. Y. Su 1998) and (S. Y. Ohm and C. S. Jhon 1992). It reduces the search space and improves the stability of the algorithm. To effectively prune the tree, it is better to know the cost at an early state of the decision path, so that the computation effort is not wasted on the subtree, which cannot give a better solution. This will be an issue that we have to consider when we select the design algorithms. The synthesis algorithms in the next section will also reflect this idea.

4.8 Conclusion

The AOT is introduced. It is a modified version of ACO. The major change is that it is applied to search a tree, callled the decision tree. Some of the rules in ACO algorithm are modified. A newly introduced operator, the path exploration is aimed to replace q_0 which is used in ACS, Ant-Q, and $\mathcal{MAX}-\mathcal{MIN}$ Ant System) to make the searchs more direct. The state transition rule is also modified. The biased probabilistic rule is applied by using the heuristic weights first. The pheromone levels are used after all the choices are selected. It was modified in this way because the heuristic weights may be the resultes from many heuristic algorihtms. The main purpose of this pheromone updating rule is to protect the algorithm from stroage explosion. It was adopted from the pheromone updating rule of $\mathcal{MAX}-\mathcal{MIN}$ Ant System.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 5

HIGH-LEVEL SYNTHESIS BY AOT

5.1 Overview

To apply the AOT to High-Level Synthesis, the synthesis algorithms are integrated into the decision path construction process. At each problem state, the synthesis algorithm is used to list the possible partial solutions and assign a weight to each of them. The state transition rule is used to select one of the possible partial solutions. When the new partial solution is integrated to the old solution, the problem state is advanced to the next problem state.

In this implementation, for each problem state the algorithm searches a partial solution for each part of the final solution. The sequence is as follows:

1. Find the number of resources for each type (Allocation).

2. Find a time-step for each operation (Scheduling).

3. Find a functional unit for each operation (Functional unit assignment).

4. Find a register for each operation (Register assignment).

5. Find a bus for each input of the functional unit (Bus assignment).

These are the normal techniques found in various literatures such as (D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin 1992; P. Michel, U. Lauther, and P. Duzy 1992), and (G. De Micheli 1994). Most of them are heuristic algorithms and can not guarantee the optimum solution. Combining many of them together arbitrarily is unlikely to get the optimum solution. We had modified them to match with the AOT algorithm. Whenever a decision has to be made and there is not enough information, the design algorithm will list the possible choices with heuristic weights and let AOT selects one of them.

The number of each functional unit type is bounded by a lower bound and upper bound, which can be assigned by a human to limit the search to be only in the area of interest. The lower bound numbers from a good algorithm will improve the performance of the AOT.

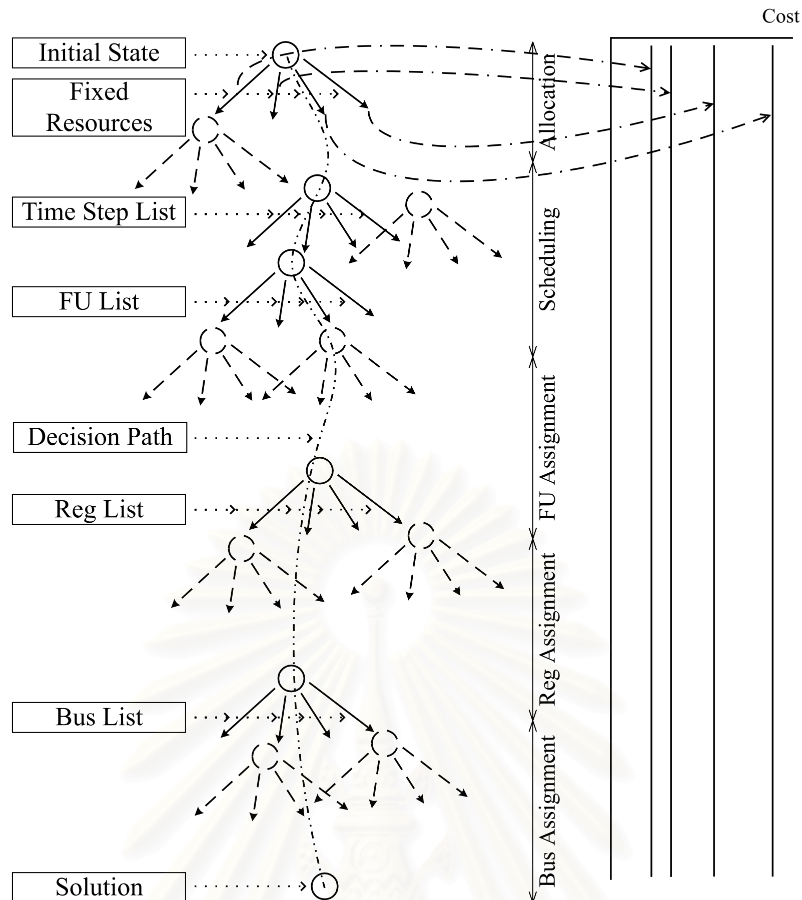

Figure 5.1: Applying AOT to High-Level Synthesis.

In the scheduling process, instead of using normal ASAP and ALAP time-steps to bound the possible time-steps, the constraints on the resources are combined and fixed before scheduling process. For example, while the ASAP time-step of an operation is computed, the required functional unit for this operation is checked for availability. If it is not available the next possible time-step is considered.

The possible hardware components to be assigned are listed from the available hardware components. These hardware components must be available in the appropriate time-step, which is consistent with the scheduling time-step.

The heuristics for weighted assignment are as follows:

1. Prefer the time-step that has fewer operations and less data transfer.

2. Prefer the register that connects to the same functional unit output.

3. Prefer the bus that connects to the same functional unit output.

One may notice that some of these heuristics are inspired by (P. G. Paulin and J. P.

Knight 1989b).

Next, we will explain in more details for each synthesis algorithm.

5.2 Resource Allocation

The resource allocation process could be explained as follows:-

1. Execution time allocation.

   List the possible number of time-steps.

   (a) Check the list with the lower bound algorithm.

   (b) Use AOT to select the number of time-step.

2. Functional unit (FU) allocation.

   For each operation type:

   (a) List the possible number of FUs.

   (b) Use AOT to select the number of FU

3. Register allocation.

   List the possible number of registers.

   (a) Check the list with the lower bound algorithm.

   (b) Use AOT to select the number of register.

4. Bus allocation.

   List the possible number of buses.

   (a) Check the list with the lower bound algorithm.

   (b) Use AOT to select the number of bus.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Resource allocation is a process that allocates a number of resources, which will be used in the later process to construct the register transfer level design. This process starts from allocation of the number of functional units for each operation type. After all the functional units were allocated, the number of time-steps for execution will be allocated. Then the number of registers and buses will be allocated. In allocation process we used some lower bound algorithms, as will be explained in the later section, to reduce the search space.

At the end of this step, all of the resources were allocated, this means that AOT will know the cost for each design whether it is feasible or not. The cost information will help AOT to prune the decision tree after it found a feasible solution. The next steps are scheduling and assignment which function as verifying the feasibility of the already allocated resources. After completing these steps a complete RTL circuit is generated. Pruning the decision tree at the earlier decision step will reduce the search space dramatically.

5.3 Scheduling

The scheduling process could be explained as follows:-

1. Find the ASAP (as soon as possible time-step) and ALAP (as late as possible time-step) of each operation.
2. Find the least mobility operation (computed from ASAP and ALAP).
3. Find the heuristic value for each time-step in the mobility range.
4. Use AOT to select a time-step in the range ASAP to ALAP.
5. Repeat the first step until all the operations were scheduled.

Scheduling is a process that assigns each operation in the CDFG to a time-step. At each step, the algorithm will check for mobility of each operation. Then the least mobility operation will be selected for scheduling first. AOT will assign this operation to a time-step in the mobility range. A heuristic behind this is to reduce the search space by testing the critical operation first. If it is an infeasible design the decision tree will be pruned.

The heuristic value of assigning an operation k to the time-step i is as in the Eq. 5.1.

$$h(r, s) = \frac{(1.2 \cdot C_{max} - C_i)^2}{\sum_{j \in R_k} (1.2 \cdot C_{max} - C_j)^2}. \tag{5.1}$$

Where C_i is the number of operations which were already assigned to the time-step i. C_{max} is the maximum number of operations for all the time-steps in the ASAP and ALAP range. R_k is the list of time-steps in mobility range of the operation k. $h(r, s)$

is the heuristic weight on edge (r, s) of the decision tree, while the r is the current node and s is the node after operation k was assigned to time-step i.

The number of functional units, which were fixed in the alllcation process, could be used to tighten the mobility.

5.4 Functional unit assignment

The functional unit assignment process could be explained as follows:-

1. Sort the operations in the ascending order of the scheduled time-steps.
2. For each operation:
   (a) Find the list of available FUs.
   (b) Use AOT to assign the operation to a FU.

Functional unit assignment is a process that assigns each operation in the CDFG to the allocated functional units. After the scheduling, we will know the assigned time-step of each operation. The assignment will be done in a sequence from the earliest to the latest time-step to reduce the infeasible search.

5.5 Register assignment

The register assignment process could be explained as follows:-

1. Sort the operations (each output of an operation will be associated with a variable) in the descending order of the longest to the shortest variable life. If there are operations with equal variable life cycles, sort the earlier operation first.
2. For each operation:
   (a) Find the list of available registers.
   (b) Use AOT to assign the operation to a register.

Register assignment is a process that assigns each operation in the CDFG to allocated registers. By giving a priority to the longer variable life, the assignment has a chance to work on the critical variable first (as in the scheduling process).

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

5.6 Bus assignment

The bus assignment process could be explained as follows:-

1. Find the maximum parallel data transfer time-step.
2. Assign each input of FUs and registers, which is active in that time-step, to a bus.
3. For each first input of FUs, which is not assigned:
   (a) Find the list of available buses.
   (b) Use AOT to assign the input to a bus.
4. For each second input of FUs, which is not assigned:
   (a) Find the list of available buses.
   (b) Use AOT to assign the input to a bus.
5. For each input of registers, which is not assigned:
   (a) Find the list of available buses.
   (b) Use AOT to assign the input to a bus.

Bus assignment is a process that assign each operation in the CDFG to the allocated buses. Since the target datapath architecture allows only one bus per input (will be explained in later section), we will work in sequence for each input of each FU (each FU has two inputs) and then the input of each register. By checking for the maximum parallel data transfer time-step, we assign a bus for each input, which is active in that time-step first. Since all of them are active in the same time-step they can not share the same bus.

5.7 Scheduling Example

Next, an application of AOT to scheduling process is demonstrated. Only the construction of a decision path is explained, because the other steps in AOT are not effected if the application is changed. The CDFG in Fig. 5.2 is used as an example. In this CDFG, it consists of 3 operations of the same functional unit type and it is constrained to be scheduling in 3 time-steps. In scheduling process, each operation will

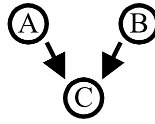

Figure 5.2: Example CDFG.

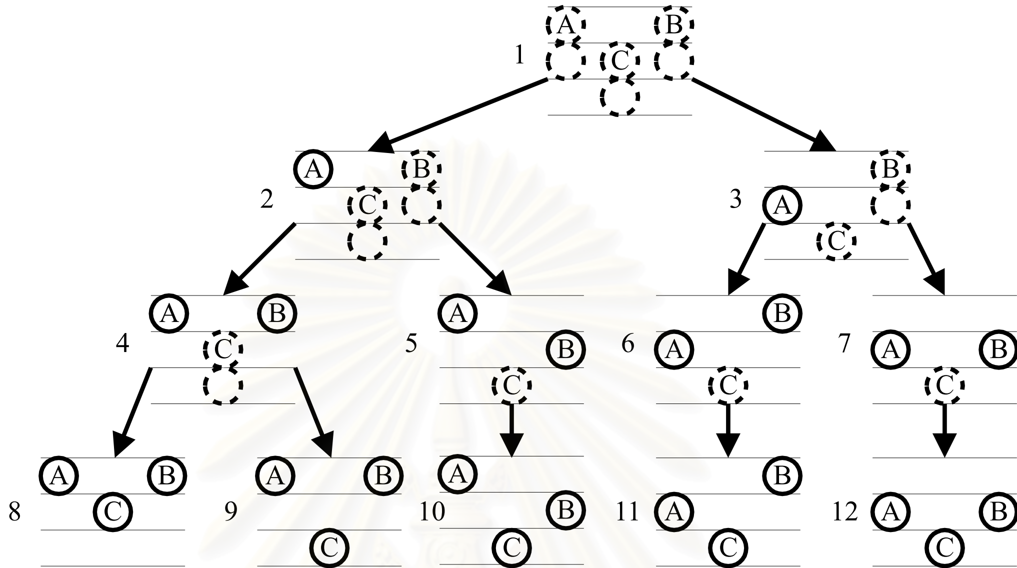

Figure 5.3: Decision tree of the scheduling process.

be assigned to a time-step in the ASAP and ALAP range as the first node in the Fig. 5.3.

If the operation A was assigned to the first time-step, then the state of scheduling process will be as in the node 2 of the Fig. 5.3. At this state, there are two choices to assign operation B, which are the first time-step (node 4) and the second time-step (node 5). For simplicity, the heuristic weight Eq. 5.1 was modified to Eq. 5.2

$$h(r, s) = (C_{max} - C_i) \cdot a + b. \qquad (5.2)$$

Where C_i is the number of operations which were already assigned to the time-step i. C_{max} is the maximum number of operations for all the time-steps in the ASAP and ALAP range. a is a factor to bias for the less assigned time-step. b is a constant to give a chance for the highest assigned time-step. For this experiment, we choose $a = 2$ and $b = 1$.

In this case, for the first time-step C_i is equal to 1 and for the second time-step

C_i is equal to 0, then C_{max} is equal to 1. The heuristic weight for assigning operation B to the first time-step ($h(2,4)$) is $(1-1)*2+1=1$, and the heuristic weight for assigning operation B to the second time-step ($h(2,5)$) is $(1-0)*2+1=3$. According to the state transition rule, the probability to assign to the first time-step ($p(2,4)$) is $1/(1+3)=0.25$, and the probability to assign to the second time-step ($p(2,5)$) is $3/(1+3)=0.75$. So there is a higher probability to select the second time-step, which is according to the heuristic in the previous paragraph. Since this is the first decision path so the heuristic weights are used to select the next node, but in other situations the pheromone weights may be used as in the state transition rule.

This process is repeated until all the operations are assigned or there is an infeasible assignment. Then the construction of a decision path in the AOT algorithm is completed.

5.8 Conclusion

This chapter explains how to synthesis a data-path by AOT. A normal synthesis process is used. It is composed of resource allocation, scheduling, functional unit assignment, register assignment, and bus assignment. Sequence of decisions in the systhesis problem are arranged into the decision tree. The procedure of AOT is applied to search for the solutions. Some heuristic knowledge is converted to the heuristic weights. They are used to bias in the state transition rule.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 6

EXPERIMENTS

6.1 Overview

In these experiments, we use the target architecture as in (P. Michel, U. Lauther, and P. Duzy 1992), which consists of combinational functional units (except the pipeline functional unit), distributed registers, multiplexers or unidirectional buses. All registers use the same clock edge. We also place the constraint that each register or functional unit has only one bus per input as in (C. H. Gebotys and M. I. Elmasry 1992). This means that even a connection from an output to an input will be counted as a bus because it has to be routed in a channel (D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin 1992). For loop registers, we ensure that the input loop registers will be valid until their last use. The output loop registers will be valid until the end of the loop. The input loop register is the same register as the output loop register for the same variable as mentioned in (L. Stok 1992).

The experiments were carried out on a personal computer with the Athlon processor running at 850 MHz and 256 Mbytes of memory.

6.2 Differential Equation

$$y'' + 3xy' + 3y = 0$$

```
While (x<a) repeat:
        x1= x+dx;
        u1= u-(3*x*u*dx)-(3*y*dx);
        y1= y+(u*dx);
        x = x1; y = y1; u = u1;
end;
```

Figure 6.1: Differential Equation.

The Differential Equation, which is used as the example in the previous chapter, is used in this experiment. The equation (Fig. 6.1) and the control data flow graph (Fig. 6.2) are repeated here for convenience. This CDFG has 5 multiply operations, 2

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

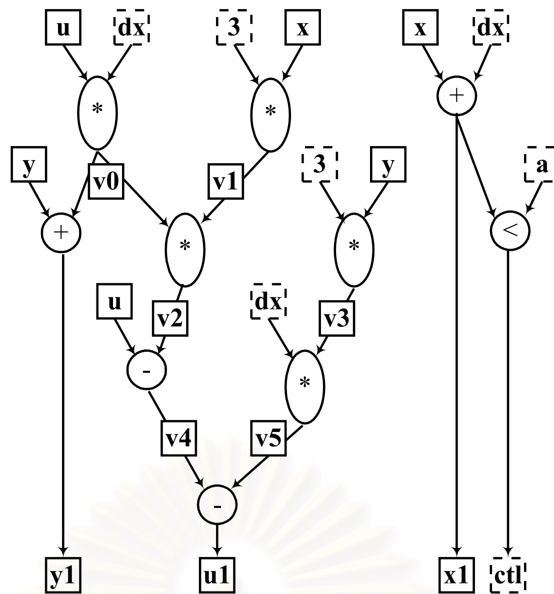

Figure 6.2: Differential Equation CDFG.

add operations, 2 subtract operations, and 1 compare operation. It also has 9 variables, 3 of them (x, y, and u) are used to keep values for the next loop. The constant values 3, a, and dx are shown in the dashed boxes.

The optimum solutions from the paper (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995) are shown in the Table 6.1.

Table 6.1: Optimum solutions of Differential Equation Solver.

Solution No.	Time-steps	Adders	Subtractors	Comparators	ALU	Single-cycle Multipliers	Two-cycle Multipliers	Pipelined Multipliers	Registers
1	4	1	1	1		1			5
2	6				1			2	5
3	7				1			1	5
4	7	1	1	1			2		5

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

6.2.1 Setup

In this experiment, the same set of functional units as in (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995) are used, but in this experiment buses are also generated. We ran 4 experiments with resources as follows:

1. 4 time-steps with adder, subtractor, comparator, and single-cycle multiplier.
2. 6 time-steps with ALU and pipelined multiplier.
3. 7 time-steps with ALU and pipelined multiplier.
4. 7 time-steps with adder, subtractor, comparator, and two-cycle multiplier.

Costs of the resources were obtained from (C. H. Gebotys and M. I. Elmasry 1992). Except the cost of bus, which is reduced to 10, in order to get the same optimum solutions as in (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995). These are shown in Table 6.2.

Table 6.2: Cost of Resources.

Resources	Cost/unit
Time-step	50
Single-cycle adder	50
Single-cycle subtractor	50
Single-cycle comparator	50
Single-cycle ALU	250
Single-cycle multiplier	250
Two-cycle multiplier	250
Pipelined multiplier	250
Register	15
Bus	10

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

The experiment was carried out with the parameter list in Table 6.3.

Table 6.3: Parameter list.

Parameter	Value
Number of ants	10
Maximum pheromone level	5.0
Pheromone evaporation rate	0.1
Number of runs	30

Table 6.4: Experimental results of Differential Equation Solver.

Solution No.	Time-steps	Adders	Subtractors	Comparators	ALU	Single-cycle Multipliers	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	4	1	1	1		2			5	8	15.1	10.34	0.3	0.46
2	6				1			2	5	6	1.77	0.42	0.03	0.18
3	7				1			1	5	6	1.13	0.58	0.03	0.18
4	7	1	1	1			2		5	7	45.43	22.84	1.43	0.88

6.2.2 Result

The result of this experiment is reported in Table 6.4. The optimum solution of the problem No. 4, which is the example in the chapter 2, is shown in Fig. 6.3, Fig. 6.4, Fig. 6.5, and Fig. 6.6. The optimum data path requires two multipliers, one adder, one subtracter, one comparator, five registers, and seven buses.

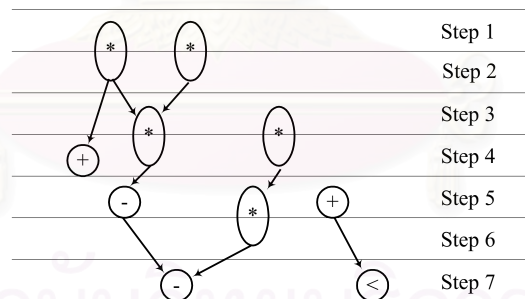

Figure 6.3: Optimum Scheduling of the differential equation.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

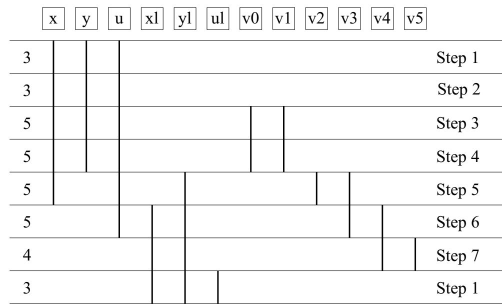

Figure 6.4: Variable lifetimes of the Optimum Scheduling.

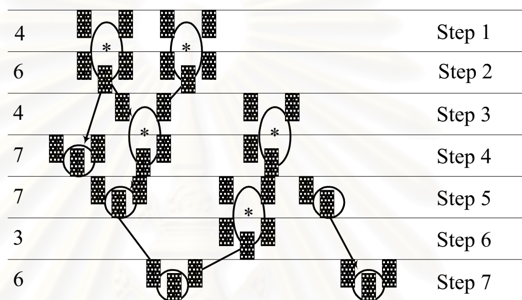

Figure 6.5: Data transfers of the Optimum Scheduling.

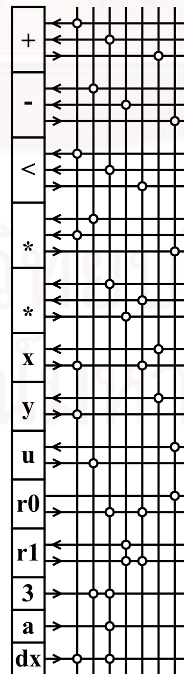

Figure 6.6: Target architecture of the differential equation.

6.3 Elliptical Wave Filter

The Elliptical Wave Filter is selected as the benchmark for testing in this research. It was used as a benchmark in many papers (P. Michel, U. Lauther, and P. Duzy 1992; P. G. Paulin and J. P. Knight 1989b; E. Torbey and J. P. Knight 1999; C. H. Gebotys and M. I. Elmasry 1992), and (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995).

The algorithm of The Elliptical Wave Filter is presented in Fig. 6.7. The CDFG of the elliptical wave filter which was scheduled for 19 control steps and pipelined multiplier is shown in Fig. 6.8. The optimal solutions from (C. H. Gebotys and M. I. Elmasry 1992) are given in the Table. 6.5. While the soultions from (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995) and (E. Torbey and J. P. Knight 1999) are present in Table 6.6 and Table 6.7.

```
process elliptic(inp, sv2, sv13, sv18, sv26, sv33,
        sv38, sv39, reset, over)

        /* State Variables */

        inout port inp[SIZE];
        inout port sv2[SIZE];
        inout port sv13[SIZE];
        inout port sv18[SIZE];
        inout port sv26[SIZE];
        inout port sv33[SIZE];
        inout port sv38[SIZE];
        inout port sv39[SIZE];
        in port reset;
        out port over;

[
        /* weight registers */

        register rega[SIZE] ;
        register regb[SIZE] ;
        register regc[SIZE] ;
        register regd[SIZE] ;
        register rege[SIZE] ;
        register regf[SIZE] ;
        register regg[SIZE] ;
        register regh[SIZE] ;

        /* internal variables */

        boolean inpi[SIZE];
        boolean outpi[SIZE];
        boolean sv2i[SIZE];
        boolean sv13i[SIZE];
        boolean sv18i[SIZE];
        boolean sv26i[SIZE];
        boolean sv33i[SIZE];
        boolean sv38i[SIZE];
        boolean sv39i[SIZE];

        /* Temporary.Results */

        boolean op3[SIZE];
        boolean op32[SIZE];
        boolean op12[SIZE];
        boolean op20[SIZE];
        boolean op25[SIZE];
        boolean op21[SIZE];
        boolean op24[SIZE];
        boolean op19[SIZE];
        boolean op27[SIZE];
        boolean op11[SIZE];
        boolean op22[SIZE];
        boolean op29[SIZE];
        boolean op9[SIZE];
        boolean op30[SIZE];
        boolean op8[SIZE];
        boolean op31[SIZE];
        boolean op7[SIZE];
        boolean op10[SIZE];
        boolean op28[SIZE];
        boolean op41[SIZE];
        boolean op6[SIZE];
        boolean op15[SIZE];
        boolean op35[SIZE];
        boolean op40[SIZE];
        boolean op4[SIZE];
        boolean op16[SIZE];
        boolean op36[SIZE];
        /* instruction.Execution */

        if (reset) {
                load rega=2;
                load regb=2;
                load regc=2;
                load regd=2;
                load rege=2;
                load regf=2;
                load regg=2;
                load regh=2;
        } else {
                inpi = read(inp);
                sv2i = read(sv2);
                op3 = inpi + sv2i;
                sv33i = read(sv33);
                sv39i = read(sv39);
                op32 = sv33i + sv39i;
                sv13i  = read(sv13);
                op12 = op3 + sv13i;
                sv26i = read(sv26);
                op20 = op12 + sv26i;
                op25 = op20 + op32;
                op21 = op25 * rega;
                op24 = op25 * regb;
                op19 = op12 + op21;
                op27 = op24 + op32;
                op11 = op12 + op19;
                op22 = op19 + op25;
                op29 = op27 + op32;
                op9 = op11 * regc;
                sv26i = op22 + op27;
                write sv26 = sv26i;
                op30 = op29 * regd;
                op8 = op3 + op9;
                op31 = op30 + sv39i;
                op7 = op3 + op8;
                op10 = op8 + op19;
                op28 = op27 + op31;
                op41 = op31 + sv39i;
                op6 = op7 * rege;
                sv18i = read(sv18);
                op15 = op10 + sv18i;
                sv38i = read(sv38);
                op35 = sv38i + op28;
                outpi = op41 * regf;
                op4 = inpi + op6;
                op16 = op15 * regg;
                op36 = op35 * regh;
                sv39i = op31 + outpi;
                write sv39 = sv39i;
                sv2i = op4 + op8;
                write sv2 = sv2i;
                sv18i = op16 + sv18i;
                write sv18 = sv18i;
                sv38i = sv38i + op36;
                write sv38 = sv38i;
                sv13i = op15 + sv18i;
                write sv13 = sv13i;
                sv33i = sv38i + op35;
                write sv33 = sv33i;
        };
        write over=1;
        write over=0;
]/*elliptic*/
END_OF_FILE
```

Figure 6.7: Elliptical wave filter.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

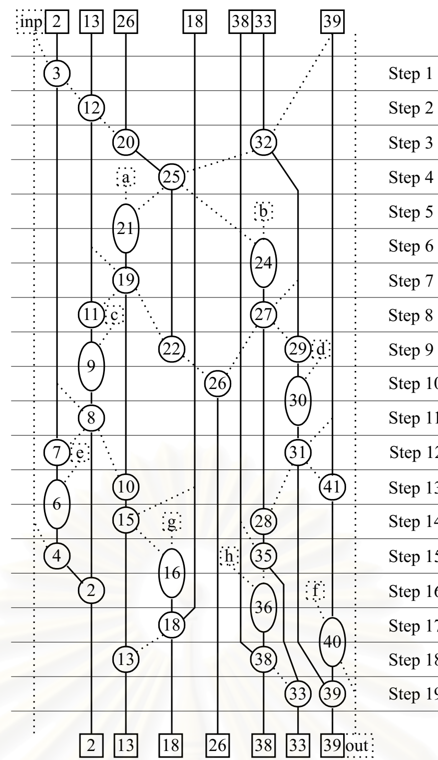

Figure 6.8: Elliptical wave filter CDFG.

Table 6.5: EWF Optimal solutions from OASIC.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	CPU Time (Sec.)
1	17	3	3		10	10	30
2	18	2	2		10	8	180
3	21	2	1		9	7	1800
4	17	3		2	10	10	30
5	18	3		1	10	9	180
6	19	2		1	9	7	348

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

6.3.1 Setup

In this experiment, we use a lower bound of the bus from (C. H. Gebotys and M. I. Elmasry 1992), which is the number of single-cycle functional units of one type multiplied by 3. For the lower bound on the number of registers, we use the number of loop registers. For the upper bound, the resources from the ASAP scheduling are used.

Table 6.6: EWF solutions from PSGA Synthesis.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	CPU Time (Sec.)
1	17	3	3		10	-	10.0
2	18	2+1	2		10	-	10.2
3	21	2	1		9+1	-	10.3
4	17	3		2	10	-	10.0
5	18	3		1	10	-	10.25
6	19	2		1	9	-	10.25

Table 6.7: EWF solutions from GA Synthesis.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	CPU Time (Sec.)
1	17	3	3		10	-	3.6
2	18	2	2		10	-	11.0
3	21	2	1		9	-	66.8
4	17	3		2	10	-	-
5	18	3		1	10	-	-
6	19	2		1	9	-	-

Costs of the resources were obtained from (C. H. Gebotys and M. I. Elmasry 1992). These are shown in Table 6.8.

Table 6.8: Cost of Resources.

Resources	Cost/Unit
Time-step	50
Single-cycle adder	50
Two-cycle multiplier	250
Pipelined multiplier	250
Register	15
Bus	100

The target cost was taken from (C. H. Gebotys and M. I. Elmasry 1992). The experiment was carried out with the parameter list in Table 6.9.

Table 6.9: Parameter list.

Parameter	Value
Number of ants	100
Maximum pheromone level	5.0
Pheromone evaporation rate	0.1
Number of runs	30

We ran 6 experiments with resources as follows:

1. 17 time steps with single-cycle adder and two-cycle multiplier.
2. 18 time steps with single-cycle adder and two-cycle multiplier.
3. 21 time steps with single-cycle adder and two-cycle multiplier.
4. 17 time steps with single-cycle adder and pipelined multiplier.
5. 18 time steps with single-cycle adder and pipelined multiplier.
6. 19 time steps with single-cycle adder and pipelined multiplier.

6.3.2 Result

The results of these experiments are reported in Table 6.10. The data-paths of these results are shown in Fig. 6.9 and Fig. 6.10. If the sysnthesis is used for the Digital Signal Processor then buses are not required to be systhesized. Because usually in the processor, only the number of functional units and the number of registers are limited. Number of buses are available enough for all the maximum possible transfers in one cycle. For the solutions without buses, the results are reported in Table 6.11. Please be noted that the CPU times report in these tables are used to show the practicality of the algorithm for the current technology and to compare between the solutions. Since the platform is different from the experiments in other papers.

Table 6.10: Experimental results and timing report.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	17	3	3		10	10	1	0	0.27	0.44
2	18	2	2		10	8	3.67	1.11	3.17	1.37
3	21	2	1		9	7	45.67	35.99	57.83	63.55
4	17	3		2	10	10	1	0	0.13	0.34
5	18	3		1	10	9	3.4	2.09	3.33	2.56
6	19	2		1	9	7	20.83	8.31	23.07	9.96

Table 6.11: Experimental results and timing report (without bus).

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	17	3	3		10	-	1	0	0.2	0.4
2	18	2	2		10	-	1.6	0.61	1.1	0.47
3	21	2	1		9	-	4.17	2.07	3.6	2.09
4	17	3		2	10	-	1	0	0.07	0.25
5	18	3		1	10	-	1	0	0.1	0.3
6	19	2		1	9	-	1	0	0.4	0.49

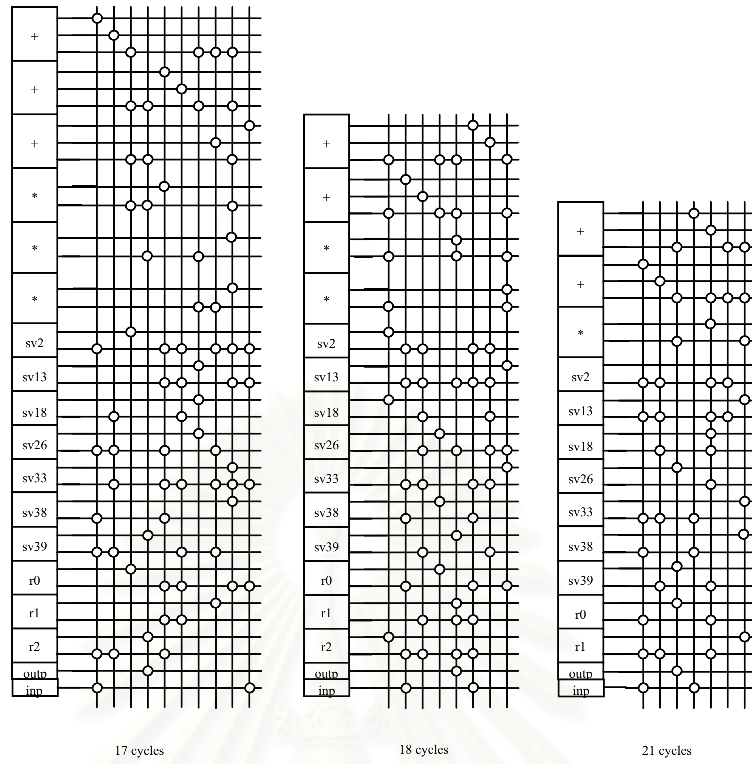

Figure 6.9: Solutions of the two-cycle multiplier.

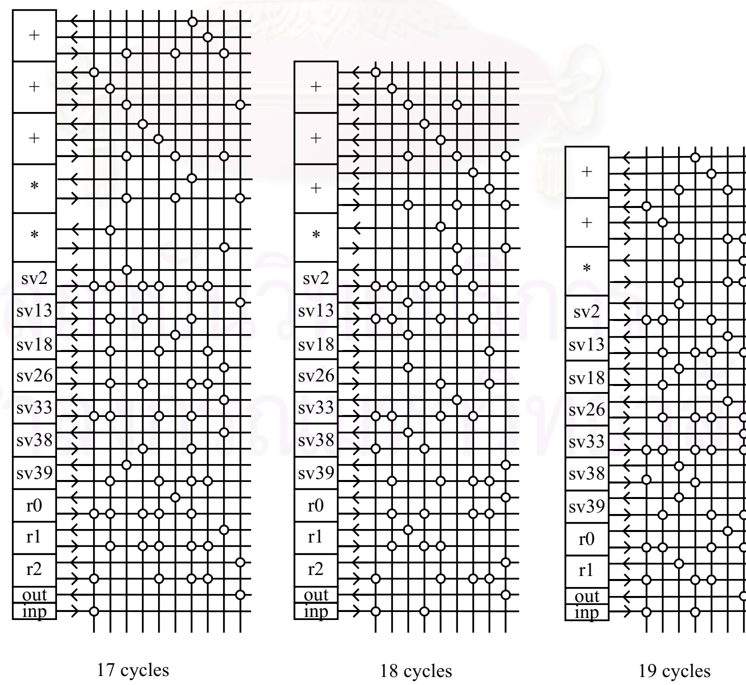

Figure 6.10: Solutions of the pipelined multiplier.

From Table 6.10, the result of 21 time-steps and two-cycle multiplier (problem No.3) is the slowest to converge. This problem has the highest mobility compared with other problems. Since the critical path of the CDFG is 17 time-steps, then 21 time-steps is about 25 % more than the critical path; therefore the search space of this problem is the largest for these problems. To improve the performance, the experiment was repeated with the tighter lower bounds from (S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997). The result is shown in Table 6.12.

Table 6.12: Experimental results of the 21 time-steps and two-cycle multiplier (problem No.3).

Problem	Iteration		Time (Sec)	
	Mean	s.d.	Mean	s.d.
Result from Table 6.10	45.67	35.99	57.83	63.55
Result with Lower Bound	20.03	8.86	25.3	15.06
Near optimal result	6.27	2.84	6.7	3.3
Near optimul result with Lower Bound	4.93	2.57	5.17	2.73

With the results shown in Table 6.12, we can see that with these lower bounds, the optimal solution can be found in about half of both iterations and execution time. In the case where the mobility is high there will be many nodes in the decision tree to be explored. For a path with high resources the system can find a solution easily. To improve the performance, lower bounds of resources should be tightened to eliminate the search among infeasible solutions caused by paths with low resources.

As Torbey and Knight stated in their paper (E. Torbey and J. P. Knight 1999), "the algorithm does not have to find the optimum, only a good engineering solution. The optimum requires an exponential search." To show the efficiency of the algorithm in finding a near optimal solution, the experiment was carried out with one additional register solution. The result is reported in Table 6.12, which shows that the number of iterations and the computational time of the near optimal solution was reduced to less than one sixth. If the lower bounds (S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997) were used the result got even better as shown in Table 6.12.

6.4 Conclusion

AOT can find all of the optimal solutions as in (C. H. Gebotys and M. I. Elmasry 1992). For all the problems, the design with 21 time-steps and two-cycle multiplier (problem No.3) seems to be hardest problem for all the algorithms (C. H. Gebotys and M. I. Elmasry 1992) (E. Torbey and J. P. Knight 1999) (AOT). In the next chapter, the algorithms to address this problem is introduced. Lower bounds should be used to speed up the algorithm. If the optimum solution is not the goal, a trade-off between quality and search time could be done.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 7

FIXED-RESOURCE MOBILITY

7.1 Overview

From the observation of all the results presented in the last chapter, one could easily see a common characteristic of those algorithms. In Table 7.1, results from the experiments of Gebotys *et al.* (C. H. Gebotys and M. I. Elmasry 1992), the computational time is increased as the number of total scheduling time-steps increases. These happend in both the experiments with two-cycle multiplier and pipelined multiplier. The results in Table 7.2, from Torbey *et al.* (E. Torbey and J. P. Knight 1999), and the results from AOT in Tabel 7.3 and 7.4 also have the same characteristic. Please note that, the results from Dhodhi *et al.* (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995) do not show that same characteristic, this may cause from the fact that these solutions are not the optimal solutions.

When the number of total scheduling time-steps increases, usually the mobilities of all the operations also increase. If we calculate the size of search space from mobilities of all operations, then we could understand why these algorithms require more CPU time. Another related issue to the increasing number of total scheduling time-steps is that it will require less functional units. In order to reduce the complexity, we fix the number of resources first and then use it to tighten up the mobilities.

In the next section the Fixed-Resource Mobility is introduced. Then the results from the experiments on the EWF are reported and compare to the results in the last chapter. The conclusion is presented in the last section.

7.2 Fixed-Resource Mobility

ASAP (as soon as possible) and ALAP (as late as possible) are the earliest and the latest time step of an operation to be scheduled. These values were computed from the critical path of each operation by assuming that there were unlimited resources. ASAP and ALAP are very useful, it is a fundamental step in many synthesis algorithms. We develop tighter bounds, the fixed-resource ASAP and fixed-resource ALAP, by including the resource constraints. These values can be used instead of ASAP and ALAP in most

Table 7.1: EWF Optimal Solutions from OASIC.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	CPU Time (Sec.)
1	17	3	3		10	10	30
2	18	2	2		10	8	180
3	21	2	1		9	7	1800
4	17	3		2	10	10	30
5	18	3		1	10	9	180
6	19	2		1	9	7	348

Table 7.2: EWF Optimal Solutions (without bus) from GA Synthesis.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	CPU Time (Sec.)
1	17	3	3		10	-	3.6
2	18	2	2		10	-	11.0
3	21	2	1		9	-	66.8
4	17	3		2	10	-	-
5	18	3		1	10	-	-
6	19	2		1	9	-	-

cases. Here, we use them to compute a tighter mobility for scheduling. It was mentioned in (C.-T Hwang, J.-H Lee, and Y.-C Hsu 1991) that the tighter mobility can be computed by dividing the number of operations of the same type to be executed by the number of functional units of that type. Our algorithms will be tighter in some cases but also have more computation complexity.

Next, the algorithms to compute fixed-resource ASAP and fixed-resource ALAP are explained. These values consider not only the critical path but also the fixed number of functional units for a CDFG. These algorithms are also applicable to a partially scheduled CDFG.

Table 7.3: EWF Optimal Solutions from AOT.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	17	3	3		10	10	1	0	0.27	0.44
2	18	2	2		10	8	3.67	1.11	3.17	1.37
3	21	2	1		9	7	45.67	35.99	57.83	63.55
4	17	3		2	10	10	1	0	0.13	0.34
5	18	3		1	10	9	3.4	2.09	3.33	2.56
6	19	2		1	9	7	20.83	8.31	23.07	9.96

Table 7.4: EWF Optimal Solutions (without bus) from AOT.

Solution No.	Time-steps	Adders	Two-cycle Multipliers	Pipelined Multipliers	Registers	Buses	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	17	3	3		10	-	1	0	0.2	0.4
2	18	2	2		10	-	1.6	0.61	1.1	0.47
3	21	2	1		9	-	4.17	2.07	3.6	2.09
4	17	3		2	10	-	1	0	0.07	0.25
5	18	3		1	10	-	1	0	0.1	0.3
6	19	2		1	9	-	1	0	0.4	0.49

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

The algorithm for the fixed-resource ASAP of operation x:

1. Find P, the set of all predecessor operations of operation x, excluding the scheduled operations.
2. Construct a resource allocation table of all the scheduled operations.
3. Sort the operations in set P in the ascending order of the ASAP time-step of each operation.
4. For each operation y in set P, by the sorted order, assign it to the earliest

possible time-step in the resource allocation table. The time-step has to satisfy both the ASAP time-step of operation y and the resource constraints but ignore the precedence constraints.

5. The fixed-resource ASAP of operation x is equal to the latest complete time-step of all the operations in set P plus 1.

The proof of this algorithm is trivial. Since it needs to make sure that the fixed-resource ASAP is less than or equal to the earliest feasible time-step. By assigning the operators in the ascending order of ASAP values, the operations of each type will be packed in the tightest manner. Because the operations of the same type have the same latency and execution time, so another order of assignment will only extend the assigned time-steps. If this assignment is feasible, then the fixed-resource ASAP will be the earliest feasible time-step. If this assignment is not feasible, then the fixed-resource ASAP will be less than the earliest feasible time-step.

The algorithm for the fixed-resource ALAP of operation x:

1. Find S, the set of all successor operations of operation x, excluding the scheduled operations.

2. Construct a resource allocation table of all the scheduled operations.

3. Sort the operations in set S in the descending order of the ALAP time-step of each operation.

4. For each operation y in set S, by the sorted order, assign it to the latest possible time-step in the resource allocation table. The time-step has to satisfy both the ALAP time-step of operation y and the resource constraints but ignore the precedence constraints.

5. The fixed-resource ALAP of operation x is equal to the earliest time-step of all the operations in set S minus the number of execution time-steps of operation x.

The complexity of this algorithm is $O(n)$. If we want to compute for all the operations then it is $O(n^2)$. To get the best performance, the algorithms should be executed in the right order, so the computed time-steps could be used in the next calculation.

7.3 An Experiment on EWF

The experiments on EWF are repeated, but this time the fixed-resource mobility were integrated with AOT algorithm. The results are reported in Table 7.5 and 7.6.

Table 7.5: EWF Optimal solutions from AOT, comparing to the Fixed-Resource Mobility

	Normal Mobility				Fixed-Resource Mobility			
Solution No.	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	1	0	0.27	0.44	1	0	0.17	0.37
2	3.67	1.11	3.17	1.37	1.07	0.25	0.4	0.55
3	45.67	35.99	57.83	63.55	13.63	24.08	33.17	65.4
4	1	0	0.13	0.34	1	0	0.2	0.4
5	3.4	2.09	3.33	2.56	2.83	1.07	3.57	1.75
6	20.83	8.31	23.07	9.96	4.7	1.88	6.9	3.51

Table 7.6: EWF Optimal solutions (without bus) from AOT, comparing to the Fixed-Resource Mobility

	Normal Mobility				Fixed-Resource Mobility			
Solution No.	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)	Iteration (mean)	Iteration (s.d.)	CPU Time (Sec.)(mean)	CPU Time (s.d.)
1	1	0	0.2	0.4	1	0	0.07	0.25
2	1.6	0.61	1.1	0.47	1	0	0.1	0.3
3	4.17	2.07	3.6	2.09	1.77	0.76	2.43	1.36
4	1	0	0.07	0.25	1	0	0.13	0.34
5	1	0	0.1	0.3	1	0	0.17	0.37
6	1	0	0.4	0.49	1	0	0.37	0.48

7.4 Conclusion

From the Table 7.5 and Table 7.6, the number of iteration is lower when the fixed-resource mobility was integrated with AOT. Since the complexity of the fixed-resource mobility is higher than the normal mobility, so the improvement of the CPU time is not seen clearly in all solutions, except the solution no. 3 and 6.

From the results, it can be seen that the algorithms can be used to accelerate the computational time. Usually if the resource is tighter the execution time-step has to be higher, the mobility will be higher, due to the longer execution time-steps. But with the Fixed-Resource Mobility, the mobility could be tighten, and the computational time is reduced. This algorithms should be good for digital signal processor, which usually has a few functional units.

CHAPTER 8

SUMMARY AND CONCLUSION

In this research an algorithm based on Ant Colony Optimization techniques called Ants on a Tree (AOT) is introduced. This algorithm can integrate many algorithms together to solve a single problem. The strength of AOT is demonstrated by solving a High-Level Synthesis problem. A High-Level Synthesis problem consists of many design steps and many algorithms to solve each of them. AOT can easily integrate these algorithms to limit the search space and use them as heuristic weights to guide the search.

In the chapter 2, the High-Level Synthesis Algorithms are reviewed. The strenght and weakness for each kind of algorithms is pointed out. For the Heuristic algorithm, the advantage is the speed of the algorithm, which usually is in the $O(n^2)$ complexity. The weakness is that it can not guarantee the optimality of the solution. With the real design process, many heuristic algorithms have to be used, and the quality solution is difficult to achieve. A lot of human interventions are needed. In the case of ILP, the strong point is that it can guarantee to get the optimal solution. The disadvantage is that it has exponential time complexity. It is difficult to form the efficient constrained equations. Another related issue for the ILP is that, when some of the constraints are relaxed, it could be used as a good bound for the search space. For GA, the advantage is that the quality and the processing time could be traded off. Good chromosome encoding and genetic operators are required to get the better performance.

ACO is introduced in chapter 3. The common steps of the algorithm is explained. Two issues to improve the performance of the algorithm are addressed. First, they try to reduce random behavior of the algorithms, by using q_0 (ACS, Ant-Q, and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System) to make the search more direct. Second, to avoid premature convergence, ants are forced to search for new solutions. By decreasing the pheromone level as the ants travel (ACS), it will reduce the probability for other ants to follow the same route. By limiting the pheromone level ($\mathcal{MAX}$-$\mathcal{MIN}$ Ant System), there is always a chance for every possible choice.

In chapter 4, AOT is introduced. The major difference from ACO is that it is applied to search a tree, called the decision tree. Some of the rules in ACO algorithm

are modified. A newly introduced operator, the path exploration is aimed to replace q_0, which is used in ACS, Ant-Q, and $\mathcal{MAX}-\mathcal{MIN}$ Ant System. The state transition rule is also modified. The biased probabilistic rule is applied by using the heuristic weights first. The pheromone levels are used after all the choices are exhausted. It was modified in this way because the heuristic weights may be the results from many heuristic algorihtms. The main purpose of this pheromone updating rule is to protect the algorithm from storage explosion. It was adopted from the pheromone updating rule of $\mathcal{MAX}-\mathcal{MIN}$ Ant System.

The chapter 5 explains how to synthesis a data-path by AOT. A normal synthesis process is used. It is composed of resource allocation, scheduling, functional unit assignment, register assignment, and bus assignment. Sequence of decisions in the systhesis problem are arranged into the decision tree. The procedures of AOT are applied to search for the solutions. Some heuristic knowledge are converted to the heuristic weights. They are used to biased in the state transition rule.

Expriments in chapter 6 show the performance of the algorithm. AOT can find all of the optimal solutions as in (C. H. Gebotys and M. I. Elmasry 1992). For all the problems, the design with 21 time-steps and two-cycle multiplier (problem No.3) seems to be hardest problem for all the algorithms.

To improve the performance of AOT further, in the chapter 7, a new algorithm is introduced and includes into the AOT algorithm. From the results, the performance gain could be clearly seen. Usually if the resources is tighter, the mobility will be higher, due to the longer execution time-steps. But with the Fixed-Resource Mobility, the mobility could be tighten, and the computational time is reduced. This algorithms should be applicable for digital signal processor, which usually has a few functional units.

จุฬาลงกรณ์มหาวิทยาลัย

The contributions of this work are as follows:

1. The main contribution of our work is that the proposed algorithm can make use of many sources of existing knowledge in terms of algorithms and design rules and integrates them in a flexible way to solve hard real-world problems in High-Level Synthesis.

2. The proposed algorithm differs from ILP and other heuristic algorithms that it is in the class of evolutionary algorithms, therefore it generates multiple solutions

instead of one solution, it also continuously improves the solutions so the quality of the solution and the time to find the solution can be trade off. The ability to trade off quality and time is important for real-world practical design problems.

3. It is easy to include heuristic algorithms into AOT by using the output of a heuristic as the initial heuristic weights, which is the characteristic of Ant algorithms.

4. We compare AOT to GA. In GA we have to find good encoding of the solutions and the proper genetic operators. These steps are not trivial, because improper encoding method may lead to infeasible solutions (which we have to recognize them) and to inefficient mapping (many to one). Many people who adopted the GA have to find the new encoding and new genetic operators as (E. Torbey and J. P. Knight 1999; M. J. M. Heijligers and J. A. G. Jess 1995), and (M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995), and it will be more difficult if that problem is composed of many steps.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

REFERENCES

A. Colorni, M. Dorigo and V. Maniezzo 1991. Distributed optimization by ant colonies. In Proc. European Conference on Artificial Life, pp. 134–142.

A. Kumar, A. Kumar, and M. Balakrishnan 1995. Heuristic search based approach to scheduling, allocation and binding in Data Path Synthesis. In Proceedings of 8th International Conference on VLSI Design, pp. 75–80.

A. Sharma and R. Jain 1993. Estimating architecture resources and performance for high-level synthesis application. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 1(2): 175–190.

A. Sharma and R. Jain 1994. Register estimation from behavioral specifications. In Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 576–580.

B. L. Miller and D. E. Goldberg 1996. Genetic Algorithms with Dynamic Niche Sharing for Multi-modal Function Optimization. In Proceeding of IEEE International Conference on Evolutionary Computation.

C. H. Gebotys and M. I. Elmasry 1990. A global optimization approach for architectural synthesis. IEEE Conferences on Computer-Aided Design: 258–261.

C. H. Gebotys and M. I. Elmasry 1991. Simultaneous scheduling and allocation for cost constrained optimal architectural synthesis. IEEE journal on Solid State Circuits: 2–6.

C. H. Gebotys and M. I. Elmasry 1992. Optimal Synthesis of High-Performance Architectures. IEEE journal on Solid State Circuits 27(3): 389–397.

C. H. Gebotys and M. I. Elmasry 1993. Global optimization approach for architectural synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 12(9): 1266–1278.

C. J. Tseng and D. P. Siewiorek 1983. Facet: A Procedure for the automated synthesis of digital systems. In Proceedings of ACM/IEEE Conference on Design Automation, pp. 566–572.

C. P. Ravikumar and V. Saxena 1996. Synthesis of testable pipelined datapaths using genetic search. In Proceedings of Ninth International Conference on VLSI Design, pp. 205–210.

C.-T Hwang, J.-H Lee, and Y.-C Hsu 1991. A formal approach to the scheduling problem in high level synthesis. IEEE Transactions on Computer Aided Design 10(4): 464–475.

D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin 1992. High-Level Synthesis: Introduction to Chip and System Design. Kluwer Academic publishers.

E. Torbey and J. P. Knight 1998. High-Level Synthesis of Digital Circuits using Genetic Algorithms. In Proceedings of IEEE Conferences on Evolutionary Computation, pp. 224–229.

E. Torbey and J. P. Knight 1999. Performing Scheduling and Storage Optimization Simultaneously Using Genetic Algorithms. In Proceedings of IEEE Midwest Symposium on Circuits and Systems, pp. 282–287.

F. J. Kurdahi and A. C. Parker 1987. REAL: A Program for REagister ALlocation. In Proceedings of ACM/IEEE Conference on Design Automation, pp. 210–215.

G. De Micheli 1994. Synthesis and Optimization of Digital Circuits. Mcgraw-Hill.

J. H. Holland 1975. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan.

J. M. Rabaey and M. Potkonjak 1994. Estimating implementation bounds for real time DSP application specific circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13(6): 669–683.

L. M. Gambardella and M. Dorigo 1995. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In Proceedings of the 12th. International Machine Learning Conference, pp. 252–260. Morgan Kaufmann.

L. P. Kaelbling, M. L. Littman, and A. W. Moore 1996. Reinforcement Learning: A Survey. journal of Artificial Intelligence Research 4: 237–285.

L. Stok 1992. Transfer Free Register Allocation in Cyclic Data Flow Graphs. In Proceedings of IEEE 3rd European Conference on Design Automation, pp. 181–185.

M. C. McFarland, A. C. Parker, and R. Camposano 1990. The High-Level Synthesis of Digital Systems. Proceedings of the IEEE 78(2): 301–318.

M. Dorigo and A. Colorni 1996. The Ant System: Optimization by a colony of cooperating agents. IEEE Transaction on Systems, Man and Cybernetics-Part B 26(1): 1–13.

M. Dorigo and L. M. Gambardella 1997. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transaction on Evolutionary Computation 1(1).

M. J. M. Heijligers and J. A. G. Jess 1995. High-Level Synthesis Scheduling and Allocation using Genetic Algorithms based on Constructive Topological Scheduling Techniques. In Proceedings of IEEE International Conference on Evolutionary Computation, pp. 56–61.

M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker 1995. Datapath Synthesis Using a Problem-Space Genetic Algorithm. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 14(8): 934–944.

M. Langevin and E. Cerny 1993. A recursive technique for computing lower-bound performance of schedules. In Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 16–20.

M. Pelikan, D. E. Goldberg, and E. Cantú-Paz 1999. BOA: The Bayesian Optimization Algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 525–532.

M. Rim and R. Jain 1994. Lower-bound performance estimation for the high-level synthesis scheduling problem. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13(4): 451–458.

M.R. Gary and D.S. Johnson 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman.

P. G. Paulin and J. P. Knight 1989a. Algorithms for high-level synthesis. IEEE Design & Test of Computers 6(6): 18–31.

P. G. Paulin and J. P. Knight 1989b. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. IEEE Transaction on Computer-Aided Design 8(6): 661–679.

P. Michel, U. Lauther, and P. Duzy 1992. The Synthesis Approach to Digital System Design. Kluwer Academic publishers.

P.-Y. Hsiao, G.-M. Wu, and J. Y. Su 1998. MPT-based branch-and-bound strategy for scheduling problem in high-level synthesis. In Proceedings of IEE Computer and Digital Techniques, Vol. 145, pp. 425–432.

R. S. Martin and J. P. Knight 1994. PASSOS: A Different Approach for Assignment and Scheduling for Power, Area and Speed Optimization in High-Level Synthesis. In Proceedings of IEEE Conference on Circuits and Systems, pp. 339–342.

S. Chaudhuri and R. A. Walker 1996. Computing lower bounds on functional units before scheduling. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 4(2): 273–279.

S. Parameswaran 1998. HW-SW Co-Synthesis: The Present and The Future. In Proceedings of the Asia and South Pacific Design Automation Conference, pp. 19–22.

S. Y. Ohm and C. S. Jhon 1992. A Branch-and-bound Method For The Optimal Scheduling. In Proceedings of IEEE Custom Integrated Circuits Conference, pp. 8.6.1–8.6.4.

S. Y. Ohm, F. J. Kurdahi, and N. D. Dutt 1997. A unified lower bound estimation technique for high-level synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 16(5): 458–472.

T. M. Mitchell 1997. Machine Learning. McGraw-Hill.

T. Stützle and H. Hoos 1997. The $\mathcal{MAX}-\mathcal{MIN}$ Ant System and Local Sear for the Traveling Salesman Problem. In Proceedings of IEEE International Conference on Evolutionary Computation, pp. 309–314.

T. Stützle and M. Dorigo 1999. ACO Algorithms for the Traveling Salesman Problem. In Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithm, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Application. John Wiley & Sons.

W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, J. L. Van Meerbergen, and A. Van Der Werf 1995. Improved Force-Directed Scheduling in

high-throughput digital signal processing. IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems 14(8): 945–960.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

BIOGRAPHY

Name	Rachaporn Keinprasit
Sex	Male
Date of Birth	March 28, 1965
Marital Status	Single
Work	Researcher
Work address	Wireline Communications Section, Telecommunications and Network Research and Development Division, National Electronics and Computer Technology Center (NECTEC), National Science and Technology Development Agency (NSTDA), Ministry of Science and Technology, 112 Thailand Science Park, Phahonyothin Rd., Klong Luang, Pathumthani Thailand 12120

Education

2003	Ph.D. in Computer Engineering, Chulalongkorn University
1990	M.Eng. in Control System, Kesetsart University
1986	B.Eng. in Communications and Electronics, Kesetsart University

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย