Assignment-Centric Approach to Data-Path Synthesis for Application Specific VLSIs

by

Koji OHASHI

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Mineo Kaneko

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2003

Abstract

Most of the conventional high-level synthesis aim mainly to minimize the number of control steps and the number of functional units, and they first decide the schedule and the number of functional units by resource constrained scheduling or time constrained scheduling, which are followed by resource assignment. However, the connectivity between modules is also an important metric for VLSIs for its connection with wire complexity, transmission delay, power consumption, testability, etc. In order to handle interconnection-related metric more accurately, simultaneous scheduling and assignment approach and assignment-driven approach are proposed. In those approaches, we often encounter a scheduling problem with specified resource assignment, which becomes one of the most importance core tasks in high-level synthesis. We study a loop pipeline scheduling problem under given resource assignment.

In this thesis, (1) we treat both assignment of operations to functional units and assignment of data to registers, (2) we introduce disjunctive arcs with variable weights to scheduling graph for representing constraints induced by assignment specification, (3) we formulate the range of available value for each un-fixed variable, (4) we construct a branch-and-bound method incorporated with successive refinement of those ranges, (5) we present a heuristic method to find a schedule having the minimum iteration period based on the reduction of those ranges using sensitivity to iteration period, (6) we extend our variable disjunctive arc approach to assignment constrained scheduling for dependence graph with conditional branches, and (7) we derive a branch-and-bound method and a heuristic method incorporated with successive refinement of parameter space.

Finally, we have developed a high-level synthesis system based on a new strategy for exploring solution space, and demonstrated its high ability in synthesizing cost optimal data-paths, especially in reducing connection-relevant hardware resource. Most important feature of our assignment-centric approach is its ability to control connectivity between modules, and a higher level of design optimization considering transmission delay and transmission power can be achieved by incorporating floorplan into our system.

Acknowledgments

The author wishes to express his sincere gratitude to his principal advisor Professor Mineo Kaneko of Japan Advanced Institute of Science and Technology for his constant encouragement and kind guidance during this work. The author would like to thank his advisor Professor Yasushi Hibino of Japan Advanced Institute of Science and Technology for his helpful discussions and suggestions.

The author devotes his sincere thanks to Professor Tetsuo Asano of Japan Advanced Institute of Science and Technology, Associate Professor Kunihiko Hiraishi of Japan Advanced Institute of Science and Technology and Associate Professor Kazuhito Ito of Saitama University for their valuable suggestions and discussions.

The author would like to thank Associate Satoshi Tayu of Japan Advanced Institute of Science and Technology and Associate Yasuhiro Takashima for their helpful comments. The author would also wish to thank to all the members in Kaneko Laboratory.

Finally, the author sincerely thanks his parents, Taketoshi Ohashi, Teruko Ohashi, for their support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High-level synthesis is the task to transform an algorithm level behavioral description into register-transfer-level structural and behavioral descriptions, and it contains several interdependent sub-problems, such as scheduling, allocation and assignment (It is also called binding) [1]. Most of the conventional high-level synthesis aim mainly to minimize the number of control steps and the number of functional units, and they first decide the schedule and the number of functional units by resource constrained scheduling or time constrained scheduling, which are followed by resource assignment. However, the connectivity between modules is also an important metric for VLSIs for its connection with wire complexity, transmission delay, power consumption, testability, etc.

To evaluate interconnection related metric as closely as possible, assignment information would be indispensable. Our possible approaches are; (A)"scheduling followed by assignment, and repeat them if necessary", (B)"combine schedule-related variables and assignment-related variables to form a entire variable space, and both schedule and assignment are constructed gradually and concurrently", and (C)"combine schedule-related variables and assignment-related variables to form a entire variable space, and solve the combined problem by ILP, branch-and-bound, simulated annealing, etc." Many researchers would recognize that the first approach (A) is in-promising since it is hard to make decision on schedule with regarding interconnections which shall be fixed only after resource assignment. Approaches (B) and (C) are usually called "simultaneous (or concurrent) scheduling and assignment".

With respect to the second approach (B), if an efficient assignment constrained scheduler, which can work even for partial assignment, is available, we can extend the approach (B) to (B') "assignment is constructed gradually guided partly by assignment constrained scheduling". In this extended simultaneous scheduling and assignment approach, a schedule is not fixed at all. Instead, a schedule accommodates to an incrementally-varying partial assignment, and a partial schedule constructed in approach (B) is now considered as a solution instance of the assignment (partial assignment) constrained scheduling problem. As a result, it may probably provide us a larger number of decision alternatives, and we have more room for developing sophisticated and efficient heuristics to construct an assignment gradually in approach (B').

With respect to branch-and-bound, simulated annealing, etc. in the third approach (C), if an efficient assignment constrained scheduler, which can work even for partial assignment, is available, we can drastically reduce the solution space to be explored with a slight expense of optimality, because branching (in branch-and-bound) or neighbor

solutions (in simulated annealing) with respect to schedule variables are all collapsed by an assignment constrained scheduling.

As a result, we conclude that an assignment constrained scheduling becomes an important core task in both approaches (B) ((B')) and (C).

In this thesis, we treat the scheduling problem under specified operation to functional unit and data to register assignment. (A certain high-level synthesis system, into which the assignment constrained scheduling is incorporated, is not concerned.) We propose an assignment constrained loop pipeline scheduling method based on a scheduling graph with disjunctive arcs having variable weight. The disjunctive arc approach to scheduling problem is often used in shop scheduling problems. We can see other disjunctive arc approaches to the scheduling for data-path synthesis [27, 26, 23, 22, 25]. In [26], assignments are specified only for operations and data transfers, and optimum scheduling method is not discussed. The method proposed in [27] does not treat loop pipeline scheduling. Also, they proposed only a simple heuristic algorithm. In [22], the schedule analyzer transforms functional unit and register binding into precedence constraints (disjunctive arcs). However disjunctive arcs are introduced only for unambiguous sequentialization of operation and data lifetime, and the final schedule relies on "off-the-shelf (resource constraints, not binding constraints) scheduler". As the result in their approach, "the existence of a schedule is not strictly guaranteed."

In this thesis, by contrast, (1) we treat both assignment of operations to functional units and assignment of data to registers, (2) we introduce disjunctive arcs with variable weights to scheduling graph for representing constraints induced by assignment specification, (3) we examine the range of available value for each un-fixed variable, (4) we construct a branch-and-bound method incorporated with successive refinement of those ranges, (5) we present a heuristic method to find the minimum iteration period based on the reduction of those ranges using sensitivity to iteration period, (6) we extend our variable disjunctive arc approach to assignment constrained scheduling for dependence graph with conditional branches, and (7) we derive a branch-and-bound method and a simple heuristic method incorporated with successive refinement of parameter space.

The rest of this thesis is organized as follows. In Chapter 2, conventional approach, assignment-driven approach and several notations and terminology used in this thesis are described. Chapter 3 shows our variable disjunctive arc approach to assignment constrained scheduling and presents a branch-and-bound method to decide loop pipeline scheduling. In Chapter 4, an improved longest path length between vertices on a scheduling graph is presented and the assignment constrained heuristic scheduling algorithm is proposed. Chapter 5 proposes our extended variable disjunctive arc approach to assignment constrained scheduling for dependence graph with conditional branches. Chapter 6 presents application of our proposed scheduling method to data-path synthesis and shows experimental results. Finally, Chapter 7 is used for conclusion.

Chapter 2

High-Level Synthesis

2.1 Conventional High-Level Synthesis

In Computer Aided Design (CAD) and Design Automation (DA) of VLSIs, top-down transformation technology is widely used. High-level synthesis is the task to transform an algorithm level behavioral description into register-transfer-level structural and behavioral descriptions. An overview of high-level synthesis is shown in Fig. 2.1. To capture the behavioral specification, a dependence graph is widely used. On the other hand, a register-transfer-level structural description consists of functional units, register, and the other interconnection resource such as nets, buses, and multiplexers. High-level synthesis contains three major subtasks such as scheduling, assignment and allocation. Scheduling assigns operations of the behavioral description to control steps, each of which corresponds to a cycle of a system clock. Allocation chooses the type and the number of functional units and registers from the unit library. Assignment assigns operations to functional units, data to register, and data transfers to wires or busses [1].

Scheduling assigns operations in the behavioral description to control steps. The total number of functional units required in a control step corresponds to the number of operations scheduled to it. If more operations are scheduled to each control step, then more functional units are necessary. On the other hand, if a few operations are scheduled to each control step, then a few functional units are sufficient, but more control steps are necessary. Thus there exit two scheduling problems; time constrained scheduling problem and resource constrained scheduling problem. Time constrained scheduling problem is to minimize the hardware cost while all operations are scheduled to the fixed number of control steps. On the other hand, resource constrained scheduling problem is to minimize the control steps needed for executing all operations under a fixed amount of hardware.

For these two time constrained scheduling problem and resource constrained scheduling problem, Force-directed scheduling [2], List scheduling [3] and Integer linear programming approach [5] are known as typical scheduling methods. For time constrained scheduling problem, Force directed scheduling successively selects a single operation among all unassigned operations, and assigns it to one control step according to a kind of priority called "force". For resource-constrained scheduling problem, List scheduling proceeds form a control step to a control step, selects operations based on a certain priority function, and assigns them to current control step. Integer linear programming approaches for both problems have been formulated. Many modified versions of these scheduling methods also have been proposed for those two scheduling problems.

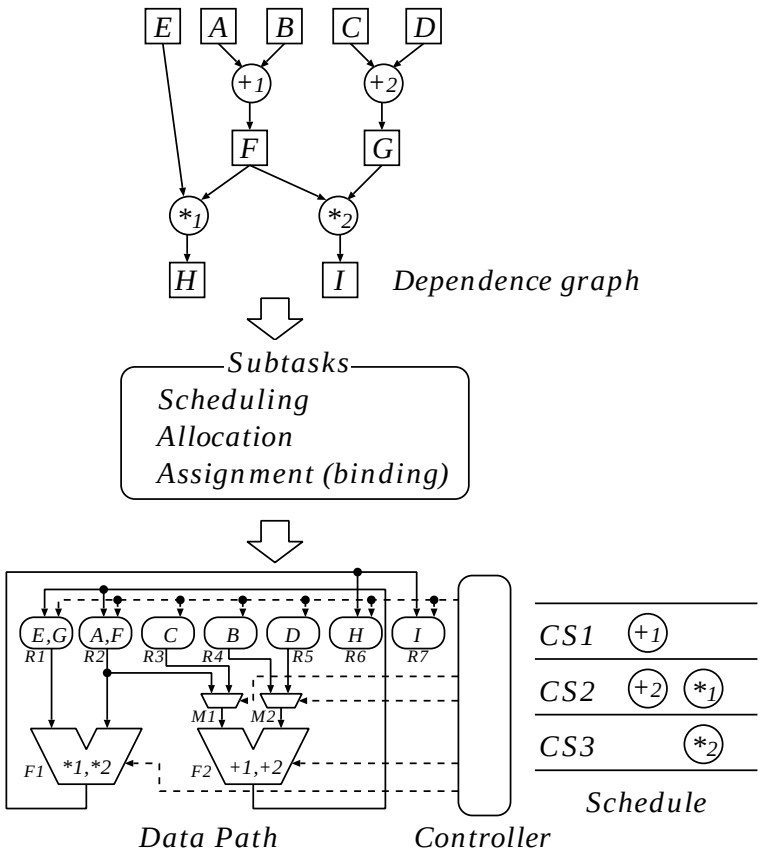

Figure 2.1: An overview of high-level synthesis.

Many works have been also done for pipeline scheduling discussed in this thesis. Sehwa [4] is the first system that incorporates pipeline scheduling into high-level synthesis. Scheduling algorithm is list-based while the operations in the list are ordered by a priority function called urgency. It can handle only functional pipeline (target algorithm contains only intra-iteration data dependency but not inter-iteration data dependency). In [5], the scheduling problem (functional pipelining) are tackled via ILP formulation. FAMOS [6] is based on the iterative improvement of schedule solution. Heuristics for exploring solution space is similar to the one which was originally proposed by Kernighan and Lin in their min-cut graph partitioning. FAMOS also handles only functional pipeline. To reduce the number of registers, the maximum density of live register variables is to be minimized. About interconnections, they focus on only bus cost. HAL [7] performs a time-constrained, functional pipelining scheduling using the force directed method. The issue of inter-iteration dependency has not been addressed. Force-directed scheduling was extended also in [8] for pipeline schedule. PLS [9] schedules operations into a state of an iteration at a time using a list-based algorithm. The paper addresses the minimization of the delay (computation time of one iteration). [10] proposes two-phase approach consisting of "as soon as possible pipelined scheduling" and "rescheduling". In [11], loops in an instance algorithm are extracted, and operations in those loops are scheduled first. Afterward, non-recursive nodes are scheduled. In [12], Improved Force-Directed Scheduling [8] is modified to handle "connectivity" maximization in high-level synthesis, where the "connectivity" is a metric introduced to estimate interconnection complexity.

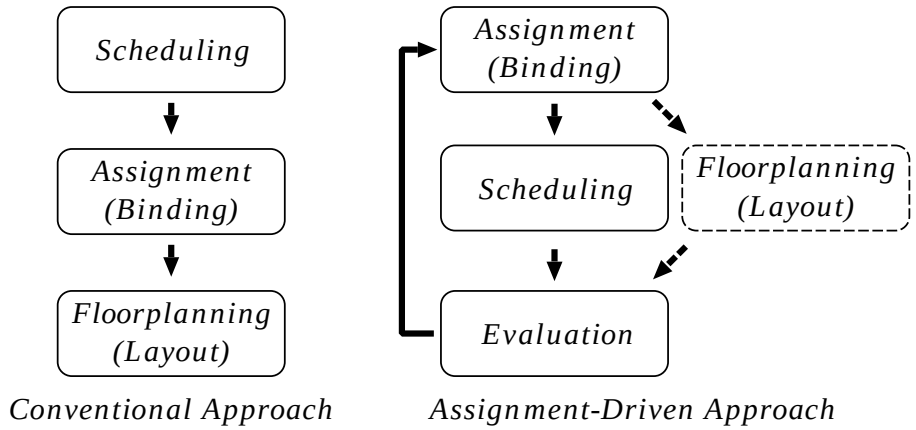

Figure 2.2: Conventional approach and assignment-driven approach.

In many conventional approaches, after scheduling, a data-path is constructed by allocation and assignment. Allocation determines the number and types of functional units and registers, which are used in the design, from unit library. Assignment assigns the operations, data, and data transfer in the scheduled dependence graph to functional units, register, and interconnection units, respectively. Thus assignment consists of functional unit assignment, register assignment, and interconnection assignment. There are three major approaches to solve the assignment problem; constructive approaches, decomposition approaches, and iterative approaches. Constructive approaches progressively construct a design. Decomposition approaches decompose the assignment problem into its constituent parts and solve each of them separately. Iterative approaches try to combine and interleave the solution of assignment sub-problems. Many researchers have proposed assignment methods based on one of those three approaches [13, 14, 15, 16, 29, 30].

2.2 Our New Approach to High-Level Synthesis

Most of the conventional high-level synthesis aim mainly to minimize the number of control steps and the number of functional units, and they first decide the schedule and the number of functional units by resource constrained scheduling or time constrained scheduling, which are followed by resource assignment. However, the connectivity between modules is also an important metric for VLSIs for its connection with wire complexity, transmission delay, power consumption, testability, etc. In the stepwise design; scheduling and resource assignment in this order, it may be hard to make a decision on operation schedule with regarding connectivity which shall be fixed only after resource assignment.

In order to handle interconnection-related metric more accurately, assignment-driven approach is proposed (see Fig. 2.2). In our assignment-driven approach, scheduling is driven by iteratively generated functional unit assignment and register assignment, and scheduling length or scheduling feasibility is evaluated for each resource assignment generated in assignment space exploration. In future, we are going to incorporate floorplanning into our system to realize high-level synthesis considering transmission delay. In this thesis, assignment constrained scheduling (which is to determine when operations are executed under specified operation to functional unit and data to register assignment),

which would become one of core task in our assignment-driven approach and simultaneous scheduling and assignment approaches, is treated. Especially pipeline scheduling for high-throughput digital signal processing is discussed.

2.3 Our Assignment Constrained Scheduling

In most of schedulers for high-level synthesis, hardware resource is treated as a constraint to be met or an objective to be minimized. While the number of functional units is evaluated exactly and the number of registers is evaluated fairly exactly, interconnections are hard to estimate in the scheduling phase. Hence, in some papers, only bus cost is estimated, or some approximated cost instead of the exact interconnections, such as "connectivity" in [12], is used for the purpose to minimize interconnection cost. However, interconnections between modules become an important metric for VLSIs for its connection with hardware cost, routability, transmission delay, power consumption, testability, etc. To treat interconnection-related metric more exactly, some researchers try to combine assignment with scheduling [17, 18, 19, 20, 21]. In [17], it treats only DAG scheduling and the treatment of register mapping is not clearly stated. In [18], assignment considered during scheduling is only the one of operations to functional units, and register assignment is done only after scheduling. Hence, interconnections are still approximately evaluated by "connectivity parameter". [19] also combines functional unit assignment for operations and scheduling, but not register assignment for variables. [20] and [21] focus mainly on the minimization of register requirements, and both try to solve the entire problem using an ILP approach, which is computationally too expensive.

To evaluate interconnection related metric as closely as possible, assignment information would be indispensable. Our possible approaches are; (A)"scheduling followed by assignment, and repeat them if necessary", (B)"combine schedule-related variables and assignment-related variables to form a entire variable space, and both schedule and assignment are constructed gradually and concurrently", and (C)"combine schedule-related variables and assignment-related variables to form a entire variable space, and solve the combined problem by ILP, branch-and-bound, simulated annealing, etc." Many researchers would recognize that the first approach (A) is in-promising since it is hard to make decision on schedule with regarding interconnections which shall be fixed only after resource assignment. Approaches (B) and (C) are usually called "simultaneous (or concurrent) scheduling and assignment (or binding)".

With respect to the second approach (B), if an efficient assignment constrained scheduler, which can work even for partial assignment, is available, we can extend the approach (B) to (B') "assignment is constructed gradually guided partly by assignment constrained scheduling". In this extended simultaneous scheduling and assignment approach, a schedule is not fixed at all. Instead, a schedule accommodates to an incrementally-varying partial assignment, and a partial schedule constructed in approach (B) is now considered as a solution instance of the assignment (partial assignment) constrained scheduling problem. As a result, it may probably provide us a larger number of decision alternatives, and we have more room for developing sophisticated and efficient heuristics to construct an assignment gradually in approach (B').

With respect to branch-and-bound, simulated annealing, etc. in the third approach (C), if an efficient assignment constrained scheduler, which can work even for partial

assignment, is available, we can drastically reduce the solution space to be explored with a slight expense of optimality, because branching (in branch-and-bound) or neighbor solutions (in simulated annealing) with respect to schedule variables are all collapsed by an assignment constrained scheduling.

As a result, we conclude that an assignment constrained scheduling becomes an important core task in both approaches (B) ((B')) and (C).

Up to now, we can find the following potential applications of assignment constrained scheduling. (Of course, its applications are not limited to the following): Application specific embedded processors are recently increasingly used in digital systems, and reconciliation of usage conflicts of the distributed registers as well as the functional unites in scheduling becomes a significant problem. Mesman et al. [22] proposed an efficient method for solving such problem by introducing a register constraint analysis before scheduling, in which register binding precedes scheduling. [23] takes the same approach, and treats minimization problem of spill code to resolve register conflict. In [24], module placement is incorporated into high-level synthesis, where the resource assignment is done first, followed by module placement and scheduling considering data transfer delay. [25] takes similar approach with assignment solution space exploration.

2.4 Terminology and Notations

Now we will define terminology and notations used throughout this thesis as follows.

- Dependence graph

  Computation algorithm is specified with a directed graph $G = (V_G, A_G)$ (Fig. 2.3(a)), where V_G is a union of a set V_O of operations and a set V_D of data, and A_G is a union of a set $A_O \subset V_O \times V_D$ (Cartesian product of two sets V_O and V_D) of arcs from operations to data and a set $A_I \subseteq V_D \times V_O$ of arcs from data to operations. An example of a dependence graph is shown in Fig. 2.3 Throughout this thesis, we use $p(v)$ $(s(v))$ to represent an immediate predecessor (successor) of a vertex v in a directed graph. If there exist plural predecessors (successors), we will use an appropriate suffix like $p_i(v)$ $(s_j(v))$ to identify each of them.

  The number of control steps, which is needed for executing each operation, is given by $e : V_O \to Z_+$, where a control step is a time unit. Each operation $o \in V_O$ has its unique output data $s(o) \in V_D$ with $(o, s(o)) \in A_O$, and each data $d \in V_D$ is used by operations $s_i(d) \in V_O$ with $(d, s_i(d)) \in A_I$ as input. Assuming that each operation in a dependence graph is executed repeatedly, each arc $(d, o) \in A_I$ is associated with its delay $D(d, o)$ which implies that d in kth ($k \in Z$) iteration (denoted as d^k) is used by o in $(k + D(d, o))$th iteration (i.e., $o^{k+D(d,o)}$).

- Resource assignment

  $\rho : V_O \to \mathcal{F}$ and $\xi : V_D \to \mathcal{R}$ are functional unit assignment and register assignment, respectively, where $\mathcal{F}$ is a set of functional units and $\mathcal{R}$ is a set of registers.

- Pipeline schedule

  Schedule is a mapping $\sigma : V_O \to Z$, where σ denotes the start control step of each operation in 0th iteration. Every operation is executed repeatedly with a common period T_r. That is, the execution of o_i^k in kth ($k \in Z$) iteration starts at $\sigma(o_i) + kT_r$.

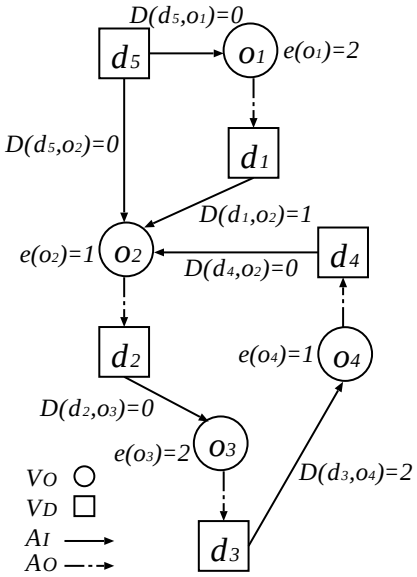

Figure 2.3: Dependence graph.

- Lifetime

  The lifetime $\tau(o_i)$ of an operation o_i is given as

$$\tau(o_i) = \bigcup_{k \in Z} \tau(o_i^k),$$

  where,

$$\tau(o_i^k) \;\; = \;\; [\sigma(o_i) + kT_r,\ \sigma(o_i) + e(o_i) - 1 + kT_r],$$

  assuming $\tau(o_i^k) \cap \tau(o_i^{k+1}) = \emptyset,\ \forall k \in Z$.

  Similarly, the lifetime $\tau(d_i)$ of data d_i is given as

$$\tau(d_i) = \bigcup_{k \in Z} \tau(d_i^k)$$

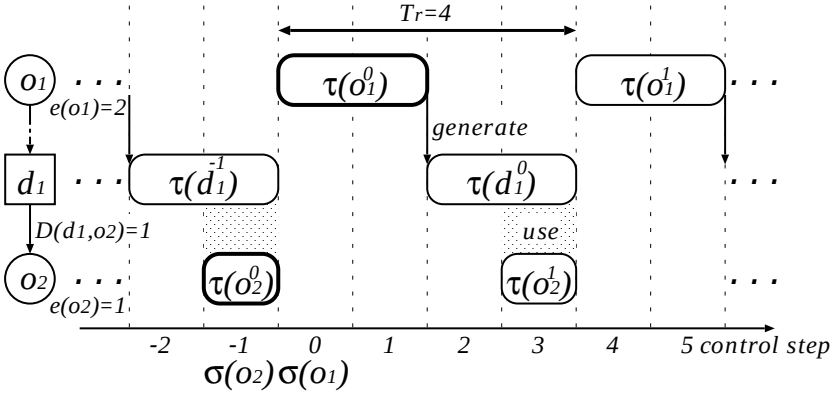

Figure 2.4: Lifetime chart for o_1, o_2, and d_1 in Fig. 2.3(a) under $\sigma(o_1) = 0$, $\sigma(o_2) = -1$ and $T_r = 4$.

with,

$$\tau(d_i^k) = \Big[\sigma(p(d_i)) + e(p(d_i)) + kT_r, \underset{j}{\mathrm{MAX}}[\sigma(s_j(d_i)) + e(s_j(d_i)) - 1 + (k + n_{ij})T_r]\Big],$$

assuming $\tau(d_i^k) \cap \tau(d_i^{k+1}) = \emptyset$, $\forall k \in Z$, where $n_{ij} = D(d_i, s_j(d_i))$. An example of lifetimes $\tau(o_1)$, $\tau(o_2)$, and $\tau(d_1)$ of operations o_1 and o_2 and data d_1 in Fig. 2.3(a) under $\sigma(o_1) = 0$, $\sigma(o_2) = -1$ and $T_r = 4$ is shown in Fig. 2.4.

Chapter 3

Assignment Constrained Loop Scheduling

3.1 Introduction

In this chapter, we will discuss assignment constrained loop pipeline scheduling method based on a scheduling graph with disjunctive arcs having variable weight. The disjunctive arc approach to scheduling problem is often used in shop scheduling problems. We can see other disjunctive arc approaches to the scheduling for data-path synthesis [27, 26, 23, 22, 25]. In [26], assignments are specified only for operations and data transfers, and optimum scheduling method is not discussed. The method proposed in [27] does not treat loop pipeline scheduling. Also, they proposed only a simple heuristic algorithm. In [22], the schedule analyzer transforms functional unit and register binding into precedence constraints (disjunctive arcs). However disjunctive arcs are introduced only for unambiguous sequentialization of operation and data lifetime, and the final schedule relies on "off-the-shelf (resource constraints, not binding constraints) scheduler". As the result in their approach, "the existence of a schedule is not strictly guaranteed."

By contrast, (1) we treat both assignment of operations to functional units and assignment of data to registers, (2) we introduce disjunctive arcs with variable weights to scheduling graph for representing constraints induced by assignment specification, (3) we examine the range of available value for each unknown variable, and construct a branch-and-bound method incorporated with successive refinement of those ranges.

We define assignment constrained loop scheduling problem as follows; Given a dependence graph G and resource assignment ρ and ξ, find the minimum iteration period T_r^* and a schedule σ^* for T_r^* such that;

1. In any control step, each functional unit is occupied at most one operation, and also each register is occupied at most one data. That is,

$$\forall o_i, o_j \in V_O,\ o_i \neq o_j,$$
$$\rho(o_i) = \rho(o_j) \ \Rightarrow\ \tau^*(o_i) \bigcap \tau^*(o_j) = \emptyset,$$

$$\forall d_i, d_j \in V_D,\ d_i \neq d_j,$$
$$\xi(d_i) = \xi(d_j) \ \Rightarrow\ \tau^*(d_i) \bigcap \tau^*(d_j) = \emptyset,$$

   where τ^* is τ for σ^*.

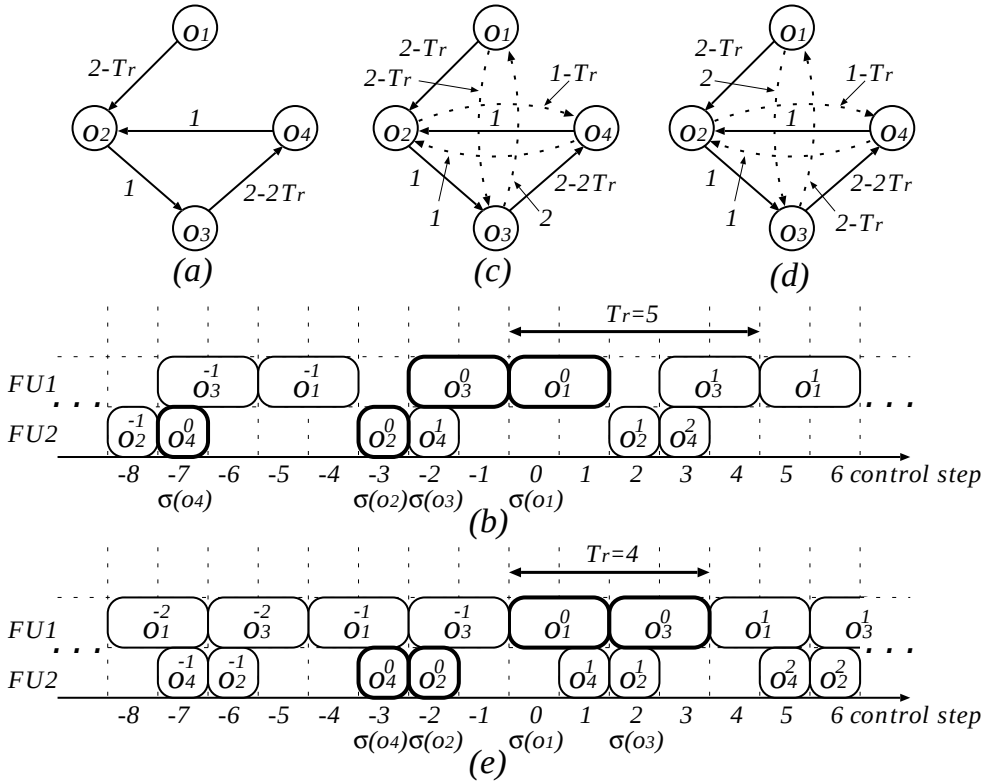

Figure 3.1: Precedence constraint graphs and schedules for the dependence graph in Fig. 2.3(a).

2. Any data cannot be used by any operation as input before it is generated. That is,

$$\forall d_i \in V_D,$$
$$\sigma^*(s_j(d_i)) + D(d_i, s_j(d_i))T_r^* \geq \sigma^*(p(d_i)) + e(p(d_i)).$$

3.2 Disjunctive Arc Approach

3.2.1 Demonstrative Example

To demonstrate the concept of our variable disjunctive arc approach, we will consider a schedule of the dependence graph in Fig. 2.3(a). Figure 3.1(a) shows the precedence constraint graph which is obtained from the input dependence graph by removing data vertices while keeping precedence relation between operations. Note that arc (o_i, o_j) with its weight w_{ij} constrains a schedule to satisfy

$$\sigma(o_j) \geq \sigma(o_i) + w_{ij}.$$

For example, the arc (o_3, o_4) comes from arcs (o_3, d_3) and (d_3, o_4) of the input dependence graph, and its weight $2 - 2T_r$ reflects $e(o_3) = 2$ and $D(d_3, o_4) = 2$. That is, since o_4 requires d_3 generated in 2 iteration before, o_4^k can start after o_3^{k-2} is finished for every $k \in Z$. Since the schedule function σ is defined for executions in 0th iteration,

$$\sigma(o_4) + kT_r \geq \sigma(o_3) + (k-2)T_r + e(o_3), \quad \forall k \in Z,$$

and it reduces to the constraint

$$\sigma(o_4) \geq \sigma(o_3) + 2 - 2T_r.$$

Thus the arc (o_3, o_4) has its weight $2 - 2T_r$.

Now we assume the resource assignment for operations $\rho(o_1) = \rho(o_3) = FU1$ and $\rho(o_2) = \rho(o_4) = FU2$. Figure 3.1(b) shows an example of a schedule under the specified functional unit assignment. From this schedule, we can extract execution sequences $o_1^k o_3^{k+1} o_1^{k+1}$ and $o_2^k o_4^{k+1} o_2^{k+1}$, and it is worthwhile to note that these sequences essentially limit the iteration period to no smaller than $T_r = 5$. That is, the sequence $o_1^k o_3^{k+1} o_1^{k+1}$ (their lifetimes should not overlap) constrains a schedule to satisfy

$$\begin{aligned} \sigma(o_3) &\geq \sigma(o_1) + e(o_1) - T_r \\ \sigma(o_1) &\geq \sigma(o_3) + e(o_3). \end{aligned}$$

These two precedence constraints are represented by adding arcs (o_1, o_3) with its weight $2 - T_r$ and (o_3, o_1) with 2 to the precedence constraint graph in Fig. 3.1(a). Figure 3.1(c) shows the resultant precedence constraint graph which includes not only additional arcs (o_1, o_3) and (o_3, o_1) but also the other additional arcs (o_2, o_4) and (o_4, o_2) due to the sequence $o_2^k o_4^{k+1} o_2^{k+1}$ on $FU2$.

Since operations with precedence constraints can be scheduled if and only if the corresponding precedence constraint graph contains no positive cycle, the minimum possible T_r is 5 for Fig. 3.1(c). To find a schedule with smaller T_r, we need to find better operation sequences on $FU1$ and $FU2$. If we employ $o_1^k o_3^k o_1^{k+1}$ instead of $o_1^k o_3^{k+1} o_1^{k+1}$, the corresponding precedence constraint graph with additional arcs is obtained as shown in Fig. 3.1(d), and its minimum possible T_r is 4. Figure 3.1(e) shows a schedule under $T_r = 4$, which is obtained by computing longest path lengths from some reference vertex to all other vertices (ASAP scheduling) on the resultant precedence constraint graph in Fig. 3.1(d).

More generally for this particular instance, the problem to find a schedule with the minimum iteration period T_r^* (under the specified functional unit assignment) is reduced to find the best sequences $o_1^k o_3^{k+F_{1,3}} o_1^{k+1}$ and $o_2^k o_4^{k+F_{2,4}} o_2^{k+1}$ (only $F_{1,3}$ and $F_{2,4}$ are variables to be fixed), and $F_{1,3} = 0$ and $F_{2,4} = 1$ are the solution which give $T_r^* = 4$.

3.2.2 Scheduling Graph

We will call a graph which represents precedence constraints between operations as a "scheduling graph" $G_S = (V_S = V_O, A_S)$, where a set of arcs A_S is associated with two functions $E_S : A_S \to Z$ and $D_S : A_S \to Z$. When the iteration period T_r is specified, the weight of each arc $(o_i, o_j) \in A_S$ is computed as $E_S(o_i, o_j) - D_S(o_i, o_j)T_r$, and the constraints for a feasible schedule are given as,

$$\begin{aligned} &\forall (o_i, o_j) \in A_S, \\ &\sigma(o_j) \geq \sigma(o_i) + E_S(o_i, o_j) - D_S(o_i, o_j)T_r. \end{aligned}$$

It is clear that a scheduling graph has a feasible schedule under T_r if and only if the graph contains no positive cycles.

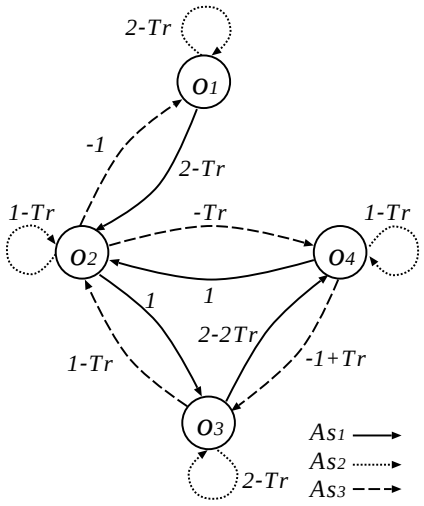

Figure 3.2: The scheduling graph constructed from the dependence graph in Fig. 2.3.

A scheduling graph, which is called an "initial scheduling graph" and denoted as G_{S0}, is first constructed as

$$\begin{aligned}
A_S &= A_{S1} \cup A_{S2} \cup A_{S3}, \\
A_{S1} &= \{(o_i, o_j) | (s(o_i), o_j) \in A_I\}, \\
&\quad E_S(o_i, o_j) = e(o_i), D_S(o_i, o_j) = D(s(o_i), o_j), \\
A_{S2} &= \{(o_i, o_i) | o_i \in V_O\}, \\
&\quad E_S(o_i, o_i) = e(o_i), D_S(o_i, o_i) = 1, \\
A_{S3} &= \{(s_j(d_i), p(d_i)) | d_i \in V_D\}, \\
&\quad E_S(s_j(d_i), p(d_i)) = e(s_j(d_i)) - e(p(d_i)), \\
&\quad D_S(s_j(d_i), p(d_i)) = -D(p(d_i), s_j(d_i)) + 1.
\end{aligned}$$

A_{S1} is a set of constraint arcs which reflects precedence constraints of the given dependence graph G, while A_{S2} and A_{S3} are sets of constraint arcs to avoid overlap between successive lifetimes of each operation and data, respectively, (i.e. $\tau(o_i^k) \cap \tau(o_i^{k+1}) = \emptyset$, $\tau(d_i^k) \cap \tau(d_i^{k+1}) = \emptyset$, $\forall k \in Z$). Fig. 3.2 shows the initial scheduling graph constructed from the dependence graph in Fig. 2.3.

Lemma 1 *If a dependence graph G is connected, then its corresponding initial scheduling graph G_{S_0} is strongly connected.*

We often consider a schedule with a specified reference vertex $v \in V_S$ in which $\sigma(v) = 0$ is retained, and we denote it as $\sigma_v : V_S \to Z$. Let $L(T_r, v, w)$ be the longest path length from $v \in V_O$ to $w \in V_O$ under iteration period T_r on G_S. Then we can have the following lemma.

Lemma 2 *For any choice of a reference node v,*

$$L(T_r, v, w) \leq \sigma_v(w) \leq -L(T_r, w, v).$$

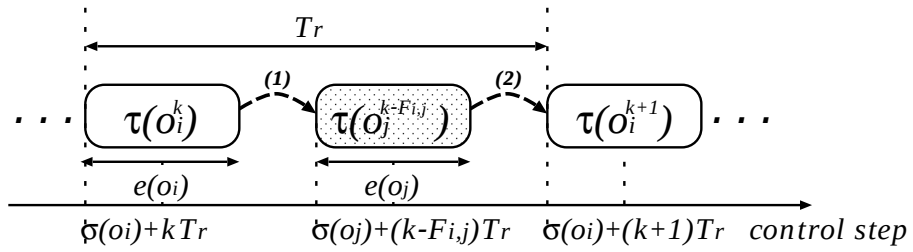

Figure 3.3: Collision-free schedule of o_i and o_j assigned to the same functional unit.

3.2.3 Constraints from Resource Assignment

If two operations o_i and o_j in G are assigned to the same functional unit (i.e. $\rho(o_i) = \rho(o_j)$), then the lifetimes $\tau(o_i)$ and $\tau(o_j)$ should not overlap (see Fig. 3.3), which is satisfied if and only if there exists an integer $F_{i,j}$ such that;

(1) The lifetime of kth execution of the operation o_i (i.e. $\tau(o_i^k)$) precedes the lifetime of $(k - F_{i,j})$th execution of the operation o_j (i.e. $\tau(o_j^{k-F_{i,j}})$).

(2) At the same time, the lifetime of $(k - F_{i,j})$th execution of the operation o_j (i.e. $\tau(o_j^{k-F_{i,j}})$) precedes the lifetime of $(k+1)$th execution of the operation (i.e. $\tau(o_i^{k+1})$).

From above two constraints, we obtain the following inequalities.

$$\begin{aligned} \sigma(o_j) + (k - F_{i,j})T_r &\geq \sigma(o_i) + e(o_i) + kT_r \\ \sigma(o_j) &\geq \sigma(o_i) + e(o_i) + F_{i,j}T_r. \end{aligned} \tag{3.1}$$

$$\begin{aligned} \sigma(o_i) + (k + 1)T_r &\geq \sigma(o_j) + e(o_j) + (k - F_{i,j})T_r \\ \sigma(o_i) &\geq \sigma(o_j) + e(o_j) - (1 + F_{i,j})T_r. \end{aligned} \tag{3.2}$$

On the scheduling graph G_S, these two constraints can be represented by adding a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$\begin{aligned} E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = -F_{i,j}, \\ E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1 + F_{i,j}. \end{aligned}$$

We call these arcs "disjunctive arcs". An example of these disjunctive arcs is shown in Fig. 3.4.

Similarly, if two data d_i and d_j which are generated by operations o_i and o_j, respectively, are assigned to the same register, then the lifetimes $\tau(d_i)$ and $\tau(d_j)$ should not overlap (see Fig. 3.5), which is satisfied if and only if there exists an integer $R_{i,j}$ such that;

(1) The lifetime of the data d_i which is generated by kth execution of operation o_i (i.e. $\tau(d_i^k)$) precedes the lifetime of the data d_j which is generated by $(k - R_{i,j})$th execution of operation o_j (i.e. $\tau(d_j^{k-R_{i,j}})$).

(2) At the same time, the lifetime of the data d_j which is generated by $(k - R_{i,j})$th execution of operation o_j (i.e. $\tau(d_j^{k-R_{i,j}})$) precedes the lifetime of the data d_i which is generated by $(k + 1)$th execution of o_i (i.e. $\tau(d_i^{k+1})$).

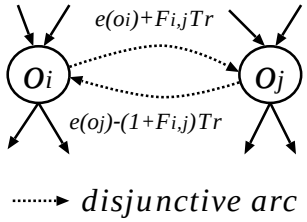

Figure 3.4: Pairs of disjunctive arcs induced by functional unit assignment.

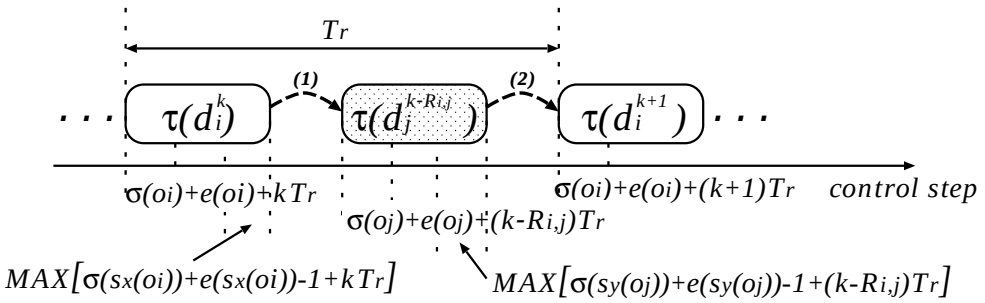

Figure 3.5: Collision-free schedule of d_i and d_j assigned to the same register.

When operation $s_x(o_i)$ uses d_i as its input and $t_x = D_S(o_i, s_x(o_i))$, and at the same time operation $s_y(o_j)$ which uses d_j as its input and $u_y = D_S(o_j, s_y(o_j))$, we obtain the following inequalities from the above two constraints.

$$\begin{aligned} &\sigma(o_j) + e(o_j) - 1 + (k - R_{i,j})T_r \\ &\quad \geq \ \underset{x}{\mathrm{MAX}}\,[\sigma(s_x(o_i)) + e(s_x(o_i)) - 1 + (k + t_x)T_r] \\ &\sigma(o_j) \\ &\quad \geq \ \underset{x}{\mathrm{MAX}}\,[\sigma(s_x(o_i)) + e(s_x(o_i)) - e(o_j) + (t_x + R_{i,j})T_r]\,. \end{aligned} \tag{3.3}$$

$$\begin{aligned} &\sigma(o_i) + e(o_i) - 1 + (k + 1)T_r \\ &\quad \geq \ \underset{y}{\mathrm{MAX}}\,[\sigma(s_y(o_j)) + e(s_y(o_j)) - 1 + (k - R_{i,j} + u_y)T_r] \\ &\sigma(o_i) \\ &\quad \geq \ \underset{y}{\mathrm{MAX}}\,[\sigma(s_y(o_j)) + e(s_y(o_j)) - e(o_i) - (1 + R_{i,j} - u_y)T_r]\,. \end{aligned} \tag{3.4}$$

On G_S, these constraints are represented by adding disjunctive arcs $(s_x(o_i), o_j)$ for every operation $s_x(o_i)$ which uses d_i as its input, and $(s_y(o_j), o_i)$ for every operation $s_y(o_j)$ which uses d_j as its input, and their weight functions are given as,

$$\begin{aligned} E_S(s_x(o_i), o_j) &= e(s_x(o_i)) - e(o_j),\ D_S(s_x(o_i), o_j) = -t_x - R_{i,j}, \\ E_S(s_y(o_j), o_i) &= e(s_y(o_j)) - e(o_i),\ D_S(s_y(o_j), o_i) = -u_y + 1 + R_{i,j}, \end{aligned}$$

where $t_x = D_S(o_i, s_x(o_i))$, $u_y = D_S(o_j, s_y(o_j))$, and $R_{i,j}$ is an unknown integer to be fixed. Fig. 3.6 shows an example of these disjunctive arcs.

A variable $F_{i,j}$ is introduced into G_S (initially $G_{S_0} = G_S$) for every pair of operations o_i and o_j which are assigned to the same functional unit. A variable $R_{i,j}$ is also introduced

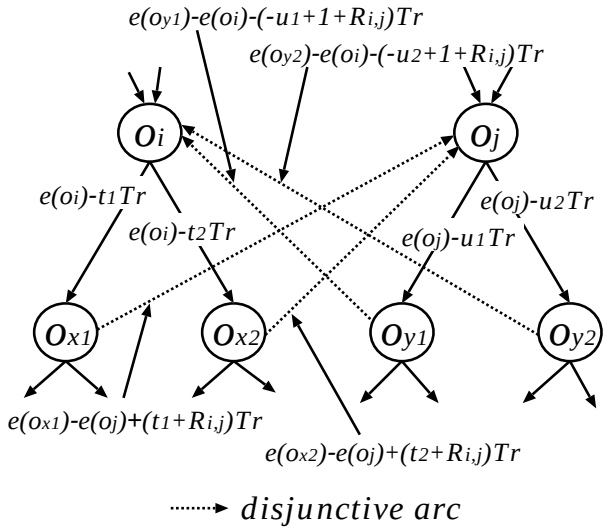

Figure 3.6: Pairs of disjunctive arcs induced by register assignment.

into G_S for every pair of data d_i (generated by o_i) and d_j (generated by o_j) which are assigned to the same register. We denote the set of all variables $F_{i,j}$s as $\boldsymbol{F}$ and the set of all variables $R_{i,j}$s as $\boldsymbol{R}$.

Theorem 1 *Under given resource assignment and iteration period T_r, the input algorithm is schedulable if and only if there exist $\Sigma : \boldsymbol{F} \cup \boldsymbol{R} \to Z$ and the resultant scheduling graph contains no positive cycles.*

Corollary 1 *If $\Sigma : \boldsymbol{F} \cup \boldsymbol{R} \to Z$ is given, and the resultant scheduling graph contains no positive cycles under the iteration period T_r, a schedule $\sigma : V_O \to Z$ can be computed in polynomial time of $|V_O|^3$.*

3.2.4 Ranges of Variables to Be Fixed

We investigate the ranges of variables $\boldsymbol{F} \cup \boldsymbol{R}$. In the following, we remember that the initial scheduling graph G_{S_0} is strongly connected from Lemma 1.

We consider the variable $F_{i,j} \in \boldsymbol{F}$ on the disjunctive arcs derived from operation resource assignment $\rho(o_i) = \rho(o_j)$. When we consider a scheduling σ_{o_v} with a reference node o_v, two inequalities

$$F_{i,j} \leq \frac{\sigma_{o_v}(o_j) - \sigma_{o_v}(o_i) - e(o_i)}{T_r} \tag{3.5}$$

and

$$\frac{-\sigma_{o_v}(o_i) + \sigma_{o_v}(o_j) + e(o_j)}{T_r} - 1 \leq F_{i,j} \tag{3.6}$$

hold from inequalities (3.1) and (3.2), respectively. Further, from Lemma 2,

$$\frac{\sigma_{o_v}(o_j) - \sigma_{o_v}(o_i) - e(o_i)}{T_r} \leq \frac{-L(T_r, o_j, o_v) - L(T_r, o_v, o_i) - e(o_i)}{T_r} \tag{3.7}$$

and,

$$\frac{L(T_r, o_i, o_v) + L(T_r, o_v, o_j) + e(o_j)}{T_r} - 1 \leq \frac{-\sigma_{o_v}(o_i) + \sigma_{o_v}(o_j) + e(o_j)}{T_r} - 1 \tag{3.8}$$

hold.

Since inequalities (3.5), (3.6), (3.8) and (3.7) hold for any selection of the reference node, we can obtain the following inequalities.

$$\begin{aligned} \underset{o_v \in V_O}{\mathrm{MAX}} \left[\frac{L(T_r, o_i, o_v) + L(T_r, o_v, o_j) + e(o_j)}{T_r} - 1 \right] \leq \\ F_{i,j} \leq \underset{o_v \in V_O}{\mathrm{MIN}} \left[\frac{-L(T_r, o_j, o_v) - L(T_r, o_v, o_i) - e(o_i)}{T_r} \right] \end{aligned} \tag{3.9}$$

On the other hand, for the data d_i which is generated by the operation o_i and is used by the operation $s_x(o_i)$ and the data d_j which is generated by the operation o_j and is used by the operation $s_y(o_j)$, the variable $R_{i,j}$ on the disjunctive arcs is introduced when $\xi(d_a) = \xi(d_f)$. Similarly, we consider a scheduling σ_{o_v} with a reference node o_v, two inequalities

$$R_{i,j} \leq \underset{x}{\mathrm{MAX}} \left[\frac{\sigma_{o_v}(o_j) - \sigma_{o_v}(s_x(o_i)) - e(s_x(o_i)) + e(o_j)}{T_r} - t_x \right] \tag{3.10}$$

and

$$\mathrm{MAX} \left[\frac{-\sigma_{o_v}(o_i) + \sigma_{o_v}(s_y(o_j)) + e(s_y(oj)) - e(o_i)}{T_r} + u_y - 1 \right] \leq R_{i,j} \tag{3.11}$$

hold from inequalities (3.3) and (3.4), respectively. Further, from Lemma 2,

$$\begin{aligned} \underset{x}{\mathrm{MAX}} \left[\frac{\sigma_{o_v}(o_j) - \sigma_{o_v}(s_x(o_i)) - e(s_x(o_i)) + e(o_j)}{T_r} - t_x \right] \\ \leq \underset{x}{\mathrm{MAX}} \left[\frac{-L(T_r, o_j, o_v) - L(T_r, o_v, s_x(o_i)) - e(s_x(o_i)) + e(o_j)}{T_r} - t_x \right] \end{aligned} \tag{3.12}$$

and,

$$\begin{aligned} \underset{y}{\mathrm{MAX}} \left[\frac{L(T_r, o_i, o_v) + L(T_r, o_v, s_y(oj)) + e(s_y(oj)) - e(o_i)}{T_r} + u_y - 1 \right] \\ \leq \mathrm{MAX} \left[\frac{-\sigma_{o_v}(o_i) + \sigma_{o_v}(s_y(o_j)) + e(s_y(oj)) - e(o_i)}{T_r} + u_y - 1 \right] \end{aligned} \tag{3.13}$$

hold.

Since inequalities (3.10), (3.11), (3.13) and (3.12) hold for any selection of the reference node, we can obtain the following inequalities.

$$\begin{aligned} \underset{o_v \in V_O}{\mathrm{MAX}} \ \underset{y}{\mathrm{MAX}} \left[\frac{L(T_r, o_i, o_v) + L(T_r, o_v, s_y(oj)) + e(s_y(oj)) - e(o_i)}{T_r} + u_y - 1 \right] \leq \\ R_{i,j} \leq \underset{o_v \in V_O}{\mathrm{MIN}} \ \underset{x}{\mathrm{MIN}} \left[\frac{-L(T_r, o_j, o_v) - L(T_r, o_v, s_x(o_i)) - e(s_x(o_i)) + e(o_j)}{T_r} - t_x \right]. \end{aligned} \tag{3.14}$$

Then since for any selection of the reference node, we have

$$L(T_r, o_x, o_v) + L(T_r, o_v, o_y) \leq L(T_r, o_x, o_y),$$

inequalites (3.9) and (3.14) can be simplified further, and finally the following theorem is obtained.

Theorem 2 *A relation between the iteration period T_r and the possible ranges of $F_{i,j}$ and $R_{i,j}$ are given without loss of optimality as follows;*

$$\frac{L(T_r, o_i, o_j)+e(o_j)}{T_r} - 1 \leq F_{i,j} \leq \frac{-L(T_r, o_j, o_i)-e(o_i)}{T_r},$$

$$\underset{y}{\mathrm{MAX}} \left[\frac{L(T_r, o_i, s_y(o_j))+e(s_y(oj))-e(o_i)}{T_r} + u_y - 1 \right] \leq$$
$$R_{i,j} \leq \underset{x}{\mathrm{MIN}} \left[\frac{-L(T_r, o_j, s_x(o_i))-e(s_x(o_i))+e(o_j)}{T_r} - t_x \right],$$

where $L(T_r, v, w)$ is the longest path length from node v to node w on scheduling graph G_S under the iteration period T_r.

Note that the possible range only guarantees that there is no solution outside of this range. Since the effect of other disjunctive arcs with un-fixed variable weight is not included exactly in the evaluation of the possible range for a variable concerned, the possible range dose not guarantee a solution except the case that only one variable remains un-fixed and the other are fixed.

3.3 Branch-and-Bound Method for Exact Solution

We show the scheduling (iteration period constraint scheduling) algorithm based on branch-and-bound exploration of the solution space for Σ. The outline of the algorithm is described in Fig. 3.7. An initial solution space for Σ is formed from a set of possible integers (possible range), which are calculated by using Theorem 2, for every unknown variables ($\boldsymbol{F} \cup \boldsymbol{R}$). Also these ranges are updated using Theorem 2 to increase bounding opportunities, whenever a branching is proceeded. Once a feasible $\Sigma : \boldsymbol{F} \cup \boldsymbol{R} \rightarrow Z$ (i.e. the corresponding scheduling graph G_S contains no positive cycles) is found, the scheduling σ is obtained by calculating the longest path lengths from a reference node to all nodes on G_S.

In our variable disjunctive arc approach, a variable arises for each pair of operations which are assigned to the same functional unit and data which are assigned to the same register. Thus the number of variables is the largest when all the operations are assigned to one functional unit and all the data are assigned to one register. When we let N be the number of operation nodes, $|\boldsymbol{F} \cup \boldsymbol{R}|$ is at most N^2. Moreover if we let f be the maximum number of integers in a ranges over all variables in $\boldsymbol{F} \cup \boldsymbol{R}$, the total number of integer assignments to $\boldsymbol{F} \cup \boldsymbol{R}$ is at most $f^{|\boldsymbol{F} \cup \boldsymbol{R}|} = f^{N^2}$. The time complexity of seeking the longest path length is $\mathcal{O}(N^3)$ if we use Floyd's algorithm. Therefore the time complexity of this algorithm is $\mathcal{O}(N^3 f^{N^2})$. However, in practice, our approach dose not

SCHEDULING(G, ρ, ξ, T_r)

1. Construct the initial scheduling graph G_{S_0} from G.
2. Construct variable list $\boldsymbol{F} \cup \boldsymbol{R}$ from ρ and ξ.
3. **if** (BAB($G_{S_0}, \boldsymbol{F} \cup \boldsymbol{R}, T_r$) == 1) "SUCCESS"
   else "FAIL"

BAB($G_S, \boldsymbol{F} \cup \boldsymbol{R}, T_r$)

Step 1: Calculate the longest path length of every pair of nodes on scheduling graph G_S. (if a positive cycle is detected, **return(0)**).

Step 2: Calculate the range of remained variables in $\boldsymbol{F} \cup \boldsymbol{R}$.

Step 3: **if** (There exists no un-fixed variables)

print the longest path length from reference node to all other nodes, and **return(1)**

else if (There exists an un-fixed variable whose range contains no integer value)

return(0)

else if (There exists un-fixed variables whose range contains exactly one integer value)

for(each un-fixed variable whose range contains exactly one integer value)

fix the value of the un-fixed variable to the integer value contained in its range, and add the corresponding disjunctive arcs to G_S. back to Step 1.

else

Select one un-fixed variable
for(each integer contained in its range)
3-1. fix the value, add the corresponding disjunctive arcs to G_S
3-2. **if** (BAB($G_S, \boldsymbol{F} \cup \boldsymbol{R}, T_r$) == 1) **return(1)**
3-3. delete disjunctive arcs added in step 3-1.
return(0)

Figure 3.7: Scheduling algorithm based on branch-and-bound exploration.

spend excessive run-time since the solution space Σ is explored efficiently. Since possible ranges of un-fixed variables may be reduce by fixing other variables, possible ranges of un-fixed variables are updated whenever a variable is fixed for some value in branching procedure.

3.4 Conclusion

In this chapter, we have proposed an approach using parametric scheduling graph with disjunctive arcs generated from the specified assignment information (operation to functional unit assignments and data to register assignments) in loop pipeline scheduling problem. After transforming our assignment constrained scheduling problem into the problem to assign integers to disjunctive arcs, we have derived a branch-and-bound solution method with successive refinement of parameter space.

Chapter 4

Heuristic Scheduling Based on Sensitivity to Iteration Period

4.1 Introduction

In Chapter 3, disjunctive arc with variable weight has been introduced, and the assignment constrained scheduling problem has been transformed into the problem to assign integers to variables on disjunctive arcs. However their solution is based on the branch-and-bound search and hence time consuming.

In this chapter, we will discuss an improved longest path length between nodes in a scheduling graph, by which we can improve ASAP schedule and ALAP schedule under resource assignment. And also we will discuss a heuristic assignment constrained loop pipeline scheduling method based on a scheduling graph with disjunctive arcs having variable weight. Our proposed method is based on the evaluation of possible range of each variable on a disjunctive arc, and we propose a heuristic method to narrow these ranges based on the sensitivity to iteration period.

4.2 Improvement of Longest Path Length

When we compute the longest path length $L(T_r, x, y)$ from x to y under T_r, we can only use arcs in the initial scheduling graph and some disjunctive arcs whose variables ($F_{i,j}$ or $R_{i,j}$) have been fixed as integers. If we can make use of disjunctive arcs with un-fixed variables more aggressively, we may obtain tighter evaluation on the longest path length.

Let $P_{x(ij)y}$ and $P_{x(ji)y}$ be the longest paths from x to y, each of which contains exactly one disjunctive arc (o_i, o_j) or (o_j, o_i) having an un-fixed variable $F_{i,j}$ (see Fig. 4.1), and let $\tilde{L}_{F_{i,j}}(T_r, x, y)$ be an improved longest path length from x to y considering un-fixed variable $F_{i,j}$. Then,

$$\begin{aligned}
\tilde{L}_{F_{i,j}}(T_r, x, y) &\geq L(T_r, x, o_i) + e(o_i) + F_{i,j}T_r + L(T_r, o_j, y), \\
\tilde{L}_{F_{i,j}}(T_r, x, y) &\geq L(T_r, x, o_j) + e(o_j) - (1 + F_{i,j})T_r + L(T_r, o_i, y).
\end{aligned}$$

While we do not know the value of $F_{i,j}$, but we know that $F_{i,j}$ must have some integer value. Hence we can find the minimum integer $\tilde{L}_{F_{i,j}}(T_r, x, y)$ satisfying the above two inequalities simultaneously among all possible integers (given by Theorem 2) for $F_{i,j}$, and

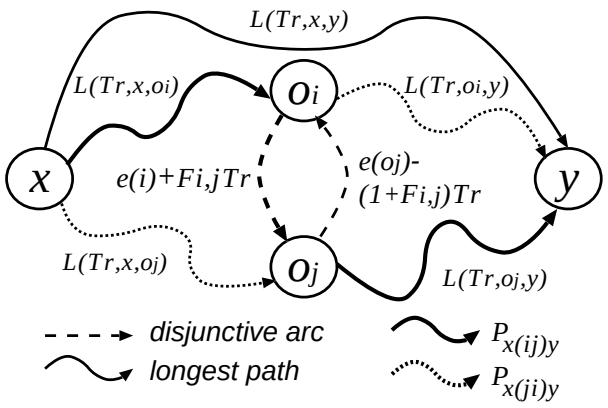

Figure 4.1: Longest paths $P_{x(ij)y}$ and $P_{x(ji)y}$ containing exactly one disjunctive arc having an un-fixed variable $F_{i,j}$.

let it be $\tilde{L}^{min}_{F_{i,j}}(T_r, x, y)$. Finally we can improve the longest path length to $\overline{L}$ as follows,

$$\overline{L}(T_r, x, y) \;=\; \mathrm{MAX}\left\{ \underset{X \in \boldsymbol{F} \cup \boldsymbol{R}}{\mathrm{MAX}} \tilde{L}^{min}_{X}(T_r, x, y),\; L(T_r, x, y) \right\}.$$

Theorem 3 *For any choice of a reference node v,*

$$L(T_r, v, w) \le \overline{L}(T_r, v, w) \le \sigma_v(w) \le -\overline{L}(T_r, w, v) \le -L(T_r, w, v).$$

Note that we can use improved $\overline{L}(T_r, x, y)$ instead of $L(T_r, x, y)$ in Theorem 2 without loss of optimality.

Example:

Consider scheduling graph shown in Fig. 4.2(a). Assume that operations o_a and o_b are assigned to one functional unit ($FU1$), operations o_c and o_d are assigned to another functional unit ($FU2$), and the numbers of control steps for executing each operation are $e(o_a) = e(o_b) = 1$ and $e(o_c) = e(o_d) = 2$. Now we consider the iteration period $T_r = 5$. Longest paths from o_a to o_b, from o_b to o_a, from o_c to o_d and from o_d to o_c are $o_a o_c o_b$, $o_b o_c o_a$, $o_c o_b o_d$ and $o_d o_a o_c$, respectively, and their lengths are all -2. We have possible ranges of $F_{a,b}$ and $F_{c,d}$ as follows,

$$-\frac{6}{5} \le F_{a,b} \le \frac{1}{5}, \quad -1 \le F_{c,d} \le 0.$$

If we choose $F_{a,b} = 0$ from possible integers $\{-1, 0\}$ for $F_{a,b}$, then the scheduling fails since, whichever -1 or 0 $F_{c,d}$ is fixed, the resultant scheduling graph contains positive cycles. Also, if we choose $F_{c,d} = 0$ and $F_{a,b} = -1$ or 0, the scheduling fails.

Consider longest paths $P_{o_b(cd)o_a} = o_b o_c o_d o_a$ and $P_{o_b(dc)o_a} = o_b o_d o_c o_a$ from o_b to o_a considering disjunctive arcs having un-fixed variable $F_{c,d}$. Since their path lengths are $5F_{c,d}$ for $P_{o_b(cd)o_a}$ and $-5(1+F_{c,d})$ for $P_{o_b(dc)o_a}$, and from the possible range of $F_{c,d}$, $5F_{c,d} < -5(1 + F_{c,d}) = 0$ in case $F_{c,d} = -1$, and $-5(1 + F_{c,d}) < 5F_{c,d} = 0$ in case $F_{c,d} = 0$. So we obtain improved longest path length $\overline{L}(5, o_b, o_a) = 0$. Similarly we obtain improved longest path lengths $\overline{L}(5, o_d, o_c) = -1$ using $P_{o_d(ab)o_c} = o_d o_a o_b o_c$ for $F_{a,b} = 0$ and $P_{o_d(ba)o_c} =$

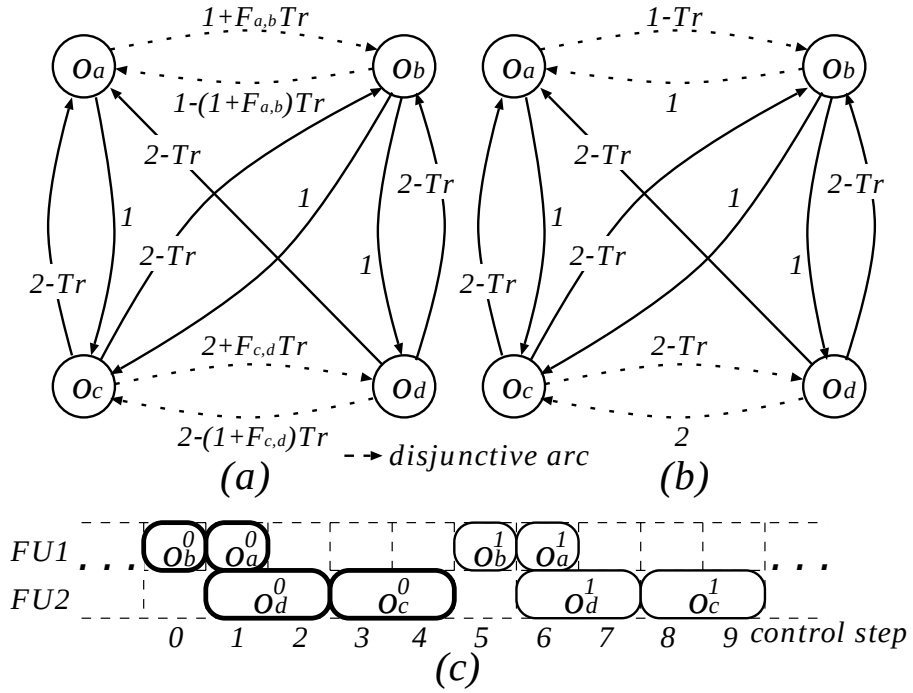

Figure 4.2: Scheduling graphs and a schedule under $\rho(o_a) = \rho(o_b)$, $\rho(o_c) = \rho(o_d)$, and $T_r = 5$.

$o_d o_b o_a o_c$ for $F_{a,b} = -1$, (and $\overline{L}(5, o_a, o_b) = \overline{L}(5, o_c, o_d) = -2$). When we use $\overline{L}(5, o_b, o_a)$ and $\overline{L}(5, o_d, o_c)$ instead of $L(5, o_b, o_a)$ and $L(5, o_d, o_c)$, the possible ranges of $F_{a,b}$ and $F_{c,d}$ are reduced without loss of optimality as follows,

$$-\frac{6}{5} \leq F_{a,b} \leq -\frac{1}{5}, \quad -1 \leq F_{c,d} \leq -\frac{1}{5}.$$

Then, single possible integers $F_{a,b} = -1$ and $F_{c,d} = -1$ are obtained. The resultant scheduling graph with disjunctive arcs (variables are fixed as integers) is shown in Fig. 4.2(b). As the result, we obtain a schedule (Fig. 4.2(c)) under $T_r = 5$ by applying ASAP scheduling to the scheduling graph in Fig. 4.2(b).

4.3 Heuristic Scheduling Based on Sensitivity to Iteration Period

Our final goal is to find the minimum iteration period T_r^*, but, at the same time, we also need to find integer for variables $\boldsymbol{F} \cup \boldsymbol{R}$ to confirm the feasibility of T_r^*. Our basic strategy is the alternate updates of the iteration period and the solution for variables $\boldsymbol{F} \cup \boldsymbol{R}$. Note that, when variables in $\boldsymbol{F} \cup \boldsymbol{R}$ are all fixed, the corresponding minimum iteration period T_r can be computed in polynomial time with respect to the number of operations. However, even if the iteration period T_r is fixed, the problem to find solution for $\boldsymbol{F} \cup \boldsymbol{R}$ to meet the iteration period T_r (or less) is $\mathcal{NP}$-complete [28]. Also the solution for $\boldsymbol{F} \cup \boldsymbol{R}$ is not always unique.

4.3.1 Heuristics to Find Solution for $F \cup R$

If we ignore the possibility that the longest path between two vertices may change depending on the value of iteration period T, the lower bounds F^{lower}_{i,j,T_r} and R^{lower}_{i,j,T_r} and the upper bounds F^{upper}_{i,j,T_r} and R^{upper}_{i,j,T_r} of the possible ranges for $F_{i,j}$ and $R_{i,j}$ are approximated as functions of the iteration period T as follows,

$$\begin{aligned} F^{lower}_{i,j,T_r}(T) &= \frac{\sum_{e \in P_{o_i o_j}} [E_S(e) - D_S(e)T] + e(o_j)}{T} - 1, \\ F^{upper}_{i,j,T_r}(T) &= \frac{-\sum_{e \in P_{o_j o_i}} [E_S(e) - D_S(e)T] - e(o_i)}{T}, \end{aligned}$$

$$\begin{aligned} R^{lower}_{i,j,T_r}(T) &= \underset{y}{\mathrm{MAX}} \left\lceil \frac{\sum_{e \in P_{o_i s y(o_j)}} [E_S(e) - D_S(e)T] + e_{yi}}{T} + u_y - 1 \right\rceil, \\ R^{upper}_{i,j,T_r}(T) &= \underset{x}{\mathrm{MIN}} \left\lceil \frac{-\sum_{e \in P_{o_j s x(o_i)}} [E_S(e) - D_S(e)T] - e_{xj}}{T} - t_x \right\rceil. \end{aligned}$$

where P_{xy} is a longest path from x to y under some specific iteration period $T = T_r$. Note that the third suffix T_r of $F^{lower}_{i,j,T_r}(T)$, etc., represents that the function (approximation) is derived from the longest path under the iteration period $T = T_r$. Since we have

$$\begin{aligned} &\sum_{e \in P_{xy}} E_S(e) \\ &= E_S(x, z_1) + E_S(z_1, z_2) + \cdots + E_S(z_n, y) \\ &\geq e(x) - e(z_1) + e(z_1) - e(z_2) + \cdots + e(z_n) - e(y) \\ &= e(x) - e(y), \end{aligned}$$

we can obtain

$$\begin{aligned} \sum_{e \in P_{o_i o_j}} E_S(e) + e(o_j) &\geq 0, \\ \sum_{e \in P_{o_j o_i}} E_S(e) + e(o_i) &\geq 0, \\ \sum_{e \in P_{o_i s y(o_j)}} E_S(e) + e_{yi} &\geq 0, \\ \sum_{e \in P_{o_j s x(o_i)}} E_S(e) + e_{xj} &\geq 0. \end{aligned}$$

Then, we can show the following monotonic property.

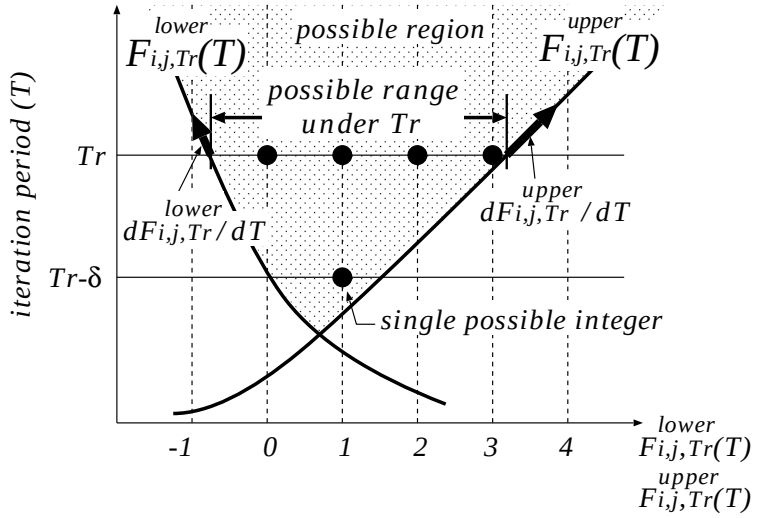

Figure 4.3: Priority functions $\Delta(F_{i,j}) = \delta$ and $\Gamma(F_{i,j}) = dF^{upper}_{i,j,T_r}/dT - dF^{lower}_{i,j,T_r}/dT$.

Lemma 3 *The possible range is reduced, as the iteration period becomes smaller. That is, for* $\delta > 0$,

$$\begin{aligned}
F^{lower}_{i,j,T_r}(T_r) \leq\ & F^{lower}_{i,j,T_r}(T_r-\delta) \leq F^{lower}_{i,j,T_r-\delta}(T_r-\delta),\\
F^{upper}_{i,j,T_r-\delta}(T_r-\delta) \leq\ & F^{upper}_{i,j,T_r}(T_r-\delta) \leq F^{upper}_{i,j,T_r}(T_r),\\
R^{lower}_{i,j,T_r}(T_r) \leq\ & R^{lower}_{i,j,T_r}(T_r-\delta) \leq R^{lower}_{i,j,T_r-\delta}(T_r-\delta),\\
R^{upper}_{i,j,T_r-\delta}(T_r-\delta) \leq\ & R^{upper}_{i,j,T_r}(T_r-\delta) \leq R^{upper}_{i,j,T_r}(T_r).
\end{aligned}$$

If G_S *contains positive cycles under* $T = T_r - \delta$, $F^{lower}_{i,j,T_r-\delta}(T_r-\delta)$ *and* $R^{lower}_{i,j,T_r-\delta}(T_r-\delta)$ *and* $F^{upper}_{i,j,T_r-\delta}(T_r-\delta)$ *and* $R^{upper}_{i,j,T_r-\delta}(T_r-\delta)$ *are assumed to be* ∞ *and* $-\infty$, *respectively.*

Note that decisions are needed when possible ranges of some un-fixed variables in $\boldsymbol{F}\cup\boldsymbol{R}$ remain containing plural integers. Our strategy employed here is to reduce those possible ranges by estimating ranges at an iteration period smaller than T_r. We can evaluate how fast the possible range is reduced as the iteration period decreases by (1) the minimum δ such that no more than one possible integer is contained in the resultant possible range under $T = T_r - \delta$ (let such minimum δ be $\Delta(X)$ for $X \in \boldsymbol{F}\cup\boldsymbol{R}$, and if there is no such δ, $\Delta(X)$ is defined as $-\infty$), and (2) the differential coefficient of the width of possible range with respect to the iteration period, $\Gamma(X) \triangleq d(X^{upper} - X^{lower})/dT$ for $X \in \boldsymbol{F}\cup\boldsymbol{R}$ (Fig. 4.3).

The routine HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) is the one to find the solution for $\boldsymbol{F}\cup\boldsymbol{R}$ under given iteration period T. In HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$), the possible ranges under T are calculated by using Theorem 2 for every un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R}$. If there are un-fixed variables each range of which contains exactly one integer, then we fix them, add the corresponding disjunctive arcs on G_S, and update the other ranges. When the possible ranges of all un-fixed variables contain more than one integer, we find one variable X (an element in $\boldsymbol{F} \cup \boldsymbol{R}$, and un-fixed) which has maximum $\Gamma(X)$ among the minimum $\Delta(X)$, and fix X as the integer which is contained in the reduced range under the iteration period $T - \Delta(X)$. Note that the reduced range under the iteration period $T - \Delta(X)$ might not

contain exactly one possible integer in two cases. (C1) If the reduced range under the iteration period $T - \Delta(X)$ contains more than one integer (i.e., $\Delta(X) = \infty$), X is fixed as $\lfloor (X^{upper} - X^{lower})/2 \rfloor$. (C2) If $\Delta(X) = 0_+$ (arbitrary small value greater than 0) and the reduced range under the iteration period $T - \Delta(X)$ contains no integer (i.e., X^{upper} and X^{lower} are integers and $X^{upper} = X^{lower} + 1$), X is fixed as X^{upper} if the absolute value of the differential coefficient of $X^{upper}(T)$ with respect to T is smaller than that of $X^{lower}(T)$ and otherwise X^{lower}. For more details, please refer to Fig. 4.4.

4.3.2 Overall Scheduler

Fig. 4.4 shows the outline of the proposed assignment constrained heuristic scheduling based on the alternate updates of the iteration period and the solution for $\boldsymbol{F} \cup \boldsymbol{R}$, and the imaginary decrement of the iteration period with Δ, Γ priority in the latter. In the algorithm, we begin with HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) for the upper bound of the iteration period ($T = T_r^{max}$). Once every variable is fixed so that the resultant scheduling graph G_S contains no positive cycles, we calculate the minimum iteration period T_r' on G_S, and update T_r as T_r'. Executing HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) repeatedly with updating T_r as T_r' until it returns "FAIL", we obtain the minimum iteration period T_r under given G, ρ, and ξ. When σ is required, we apply ASAP scheduling under the T_r using the solution for $\boldsymbol{F} \cup \boldsymbol{R}$ obtained in the last HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) which returns "SUCCESS".

The time complexity of HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) with respect to the number of operations N is evaluated as follows. Using Floyd's algorithm, the time complexity for computing longest path lengths of all pairs of vertices is $\mathcal{O}(N^3)$. Step 1 is computed in $\mathcal{O}(N^3 + |\boldsymbol{F} \cup \boldsymbol{R}|^2)$ (including the time complexity for computing improved longest path lengths). Step 3, 4, and 5 are computed in $\mathcal{O}(|\boldsymbol{F} \cup \boldsymbol{R}|)$. The overall time complexity of HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) is $\mathcal{O}(|\boldsymbol{F} \cup \boldsymbol{R}| \cdot (N^3 + |\boldsymbol{F} \cup \boldsymbol{R}|^2))$ (or $\mathcal{O}(N^6)$ since $|\boldsymbol{F} \cup \boldsymbol{R}| = \mathcal{O}(N^2)$). If we let T_r^{max} be the upper bound of the iteration period, then totally, the overall time complexity of our assignment constrained heuristic scheduling algorithm is $\mathcal{O}(T_r^{max} \cdot N^6)$, since HS($G_S$,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) may be repeated $\mathcal{O}(T_r^{max})$ times. However, since feasible integers are assigned to $\boldsymbol{F} \cup \boldsymbol{R}$ by using decision to be made at a smaller iteration period, T_r reduces fast in the repetition of HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) and the number of repetition HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$) is very small in practice.

4.4 Experiments

The performance of our heuristic scheduling algorithm is evaluated using 1000 different resource assignments for the fifth-order elliptic wave digital filter benchmark (which contains 26 additions, 8 multiplications, 35 variables and 8 constants), and 100 different resource assignments for the modified fifth-order elliptic wave digital filter so that two original iterations are blocked as a new single iteration. Note that the latter contains double numbers of operations and data (i.e., it contains 52 additions, 16 multiplications, 69 variables and 8 constants). The proposed scheduling algorithm is implemented using C programming language on a 1GHz Pentium III personal computer.

Table 4.1 lists the numbers of instances as entries of the matrix whose column corresponds to the difference between the minimum iteration period T_r' calculated using the solution for $\boldsymbol{F} \cup \boldsymbol{R}$ obtained in HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) and the optimum solution obtained by branch-and-bound method and whose row corresponds to the turn of HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$)

SCHEDULING(G,ρ,ξ)

1. $T_r \leftarrow T_r^{max} + 1$ (T_r^{max} is the upper bound of the iteration period).
2. Construct the initial scheduling graph G_{S_0} from G.
3. Construct variable list $\boldsymbol{F} \cup \boldsymbol{R}$ from ρ and ξ.
4. **while** (HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) = "SUCCESS")

   Using the solution for $\boldsymbol{F} \cup \boldsymbol{R}$ obtained in HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$), calculate minimum iteration period T_r'.

   $T_r \leftarrow T_r'$.
5. **if** ($T_r = T_r^{max} + 1$)

   return(∞). /* No feasible schedule */

   else

   return(T_r). /* T_r minimum iteration period */

   /* Apply ASAP scheduling under the T_r, when σ is required. */

HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R}$,$T$)

Step 1: Calculate the longest path length of every pair of vertices on G_S under the iteration period T. If positive cycles are detected, **return("FAIL")**.

Step 2: If there is no un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R}$, **return("SUCCESS")**.

Step 3: Calculate the possible ranges, Δ, and Γ of remained variables in $\boldsymbol{F} \cup \boldsymbol{R}$. If some range contains no integer, **return("FAIL")**.

Step 4: If there are un-fixed variables whose ranges contain exactly one integer, then fix them, add the corresponding disjunctive arcs in G_S, and return to Step 1.

Step 5: For the un-fixed variable $X \in \boldsymbol{F} \cup \boldsymbol{R}$ which has the maximum $\Gamma(X)$ among the minimum $\Delta(X)$, fix X as one integer which is contained in the reduced range under $T - \Delta(X)$, exceptionally $\lfloor (X^{upper} - X^{lower})/2 \rfloor$ in case (C1), or X^{upper} or X^{lower} in case (C2), add the corresponding disjunctive arcs in G_S, and return to Step 1.

Figure 4.4: Outline of our assignment constrained heuristic scheduling algorithm.

Table 4.1: Results of our scheduling.

elliptic wave filter (original)						
method	Heuristics					B&B
difference	+0	+1	+2	+3	+4	-
1st iteration	891	76(-5)	28(-11)	5	0	average
2nd iteration	16	0	0	0	0	6.69
3rd iteration	0	0	0	0	0	iterations
final result	907	71	17	5	0	
runtime[s] (shortest, longest, average)	0.002, 0.070, 0.017					0.019, 955.0, 12.20

elliptic wave filter (double)						
method	Heuristics					B&B
difference	+0	+1	+2	+3	+4	-
1st iteration	79	18	1	0	2(-2)	average
2nd iteration	2	0	0	0	0	15.12
3rd iteration	0	0	0	0	0	iterations
final result	81	18	1	0	0	
runtime[s] (shortest, longest, average)	0.18, 0.35, 0.19					1.30, 16979, 576.5

repetition. The number which appears in ith iteration row, $+j$ column denotes the number of instances (among 1000 or 100) for which T_r' = "optimum iteration period $+j$" is obtained in ith turn of HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) repetition. The number in parentheses denotes the number of instances whose solutions are refined in the next turn of HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$). As we can see from this table, HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R}$,$T_r - 1$) is repeated at most 3 times (including the final one to decide no solution for a smaller iteration period) until the procedure terminates, while branch-and-bound method needs to update T_r 6.63 or 15.12 times in average. As for the quality of solutions, our proposed heuristic method produces optimum solutions for at least 80% of test instances. Table 4.2 shows solution distribution generated by our heuristic method for original elliptic wave filter. The last row of each sub-table shows average runtimes of our proposed heuristic method and of branch-and-bound method, both of which are for one input instance. The proposed heuristic method runs about 700$\sim$3000 times faster than the exact solution method on average.

In our heuristic method, the possible range of un-fixed variable $X \in \boldsymbol{F} \cup \boldsymbol{R}$ under smaller iteration period than the one now considered is evaluated, and X is fixed as an integer which is contained in its approximated range. From the sampling inspection of our in-optimum solutions, we have confirmed that our misdecisions occur in the following cases. (a) If, depending on the value of iteration period, the longest paths change, then the approximated possible range is different from the actual possible range and X would be fixed as an invalid integer which is not contained in its actual possible range. (b) If $\Delta(X) = 0_+$ and the absolute value of the differential coefficient of $X^{upper}(T)$ is equal to one of $X^{lower}(T)$, X is always fixed as X^{lower} without any justification.

If the assignment is limited only for operations to functional units and assignment of data to register is free, an incorrect decision on sequentialization of two operations assigned to the same functional unit leads not so much increase of scheduling length

Table 4.2: Results of our scheduling for original elliptic wave filter in Table 4.1.

Heuristic	35																		5
	34																	20	-
	33															1	28	-	-
	32												2	2	11	51	-	-	-
	31													10	69	-	-	-	-
	30											1	18	83	-	-	-	-	-
	29											16	114	-	-	-	-	-	-
	28											131	-	-	-	-	-	-	-
	27							1	3		34	-	-	-	-	-	-	-	-
	26						2	5		29	-	-	-	-	-	-	-	-	-
	25						8		59	-	-	-	-	-	-	-	-	-	-
	24					2	6	64	-	-	-	-	-	-	-	-	-	-	-
	23					2	78	-	-	-	-	-	-	-	-	-	-	-	-
	22				1	48	-	-	-	-	-	-	-	-	-	-	-	-	-
	21					48	-	-	-	-	-	-	-	-	-	-	-	-	-
	20			2	31	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	19			12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	18			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	17	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Exact T_r		17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

Table 4.3: Comparison of our scheduling with HCS.

difference	+0	+1	+2	+3	+4	+5	+6	+7	+8
Proposed method	922	67	11	0	0	0	0	0	0
HCS	403	193	176	96	64	41	24	3	0

and iteration period, since the lifetime of each operation is fixed and relatively small (in case of our schedule example, 1 and 2). On the other hand, if the assignment of data to register is specified, which is our case, an incorrect sequentialization of two data (incorrect sequentialization of two operations which generate these data) may lead much increase of scheduling length and iteration period, since the lifetime of each data is un-fixed in advance and may become much longer than a lifetime of an operation. That is thought as a reason why in-optimum solutions having larger difference in iteration period compared with optimum solution are generated occasionally.

Table 4.3 shows the comparison of our proposed method with other method with respect to the fifth-order elliptic wave digital filter benchmark. Unfortunately, authors could not find any scheduler for the same problem of ours. So we implemented another heuristics (HCS) combining unambiguous sequentializations and ASAP-like sequentializations. For an un-fixed variable $F_{i,j}$ (or $R_{i,j}$) $\in \boldsymbol{F} \cup \boldsymbol{R}$, ASAP-like sequentializations are the sequences o_i and o_j based on two longest path lengths $L(Tr, v, o_i)$ and $L(Tr, v, o_j)$ for a reference node v. That is, if $L(Tr, v, o_i)$ is shorter than $L(Tr, v, o_j)$, then operation o_i (data d_i) precedes operation o_j (data d_j), and otherwise operation o_j (data d_j) precedes

operation o_i (data d_i). The ASAP-like sequentializations are applied for $X_{i,j} \in \boldsymbol{F} \cup \boldsymbol{R}$ having the maximum $|L(Tr, v, o_i) - L(Tr, v, o_j)|$ when possible ranges of some un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R}$ remain containing plural integers. The construction of Table 4.3 is similar to Table 4.1, but it shows only final results. For Table 4.3, 1000 different assignments, which are different to those used for Table 4.1, are generated, and hence the result of proposed method is different to the one in Table 4.1.

4.5 Conclusion

Evaluating possible ranges of variables to be fixed, we have proposed an improved longest path length between vertices in a scheduling graph, by which we can improve possible ranges of variables. We have proposed also a heuristic method to find the minimum iteration period under specified resource assignment. Experimental results for both the fifth-order elliptic wave filter benchmark and its modified version in which the number of operations and data is doubled show the proposed heuristic method runs about 700$\sim$3000 times faster than the exact solution method, while it can find the optimum iteration periods in most cases.

Chapter 5

Assignment Constrained Schedule with Conditional Branches

5.1 Introduction

Operations in a computation algorithm may vary depending on the input data or intermediate results of the algorithm. In that case, only when a certain condition holds, some operations are executed and data are generated by these operations. The operation whose execution depends on a condition is called a conditional operation, and the data generated by the operation is called a conditional data. Figure 5.1(a) shows an example of a computation algorithm including conditional operations and data. In this algorithm, data b is added to data y if data x is greater than data a ($x > a$), or data c is subtracted from data z otherwise ($x \leq a$). Operation 1 compares the input data x with the other input data a, and determines if a condition holds. We call it a decision operation. If the condition holds (i.e., the condition $x > a$ is true), then operation 2 (i.e. an addition) is executed and data $i(= x + a)$ is generated. On the other hand, if the condition does not hold (i.e., the condition $x > a$ is false), then operation 3 (i.e. a subtraction) is executed and data $j(= z - c)$ is generated. Let two operations (data) each of which is executed on the condition exactly opposite to the other be said mutually exclusive to the other operation (data). For example, operation 3 is the mutually exclusive operation of operation 2 and data j is the mutually exclusive data of data i. If operations 2 and 3 are assigned to the same functional unit, since these operations are mutually exclusive to each other, it is possible to schedule operations 2 and 3 to the control steps such that these lifetimes overlap as shown in Fig. 5.1(b). After the operation 1, operation 2 is executed if the condition holds, or operation 3 is executed if the condition does not hold. Note that, only after the condition is resolved, the conditional operations, which are mutually exclusive and are assigned to the same functional unit, can be scheduled to the control step such that these lifetimes overlap.

If the lifetimes of mutually exclusive operations which are assigned to the same functional unit do not overlap or these operations are assigned to a different functional unit to each other, then these operations can be scheduled before the decision operation is completed. Examples of the schedules for operations of a computation algorithm in Fig. 5.1(a) are shown in Fig. 5.1(c) and (d), respectively. In these schedules, both operation 2 and operation 3 are executed and one of the results (i.e. data i and data j) is selected based on the condition resolved by the decision operation 1. This is possible since there are no

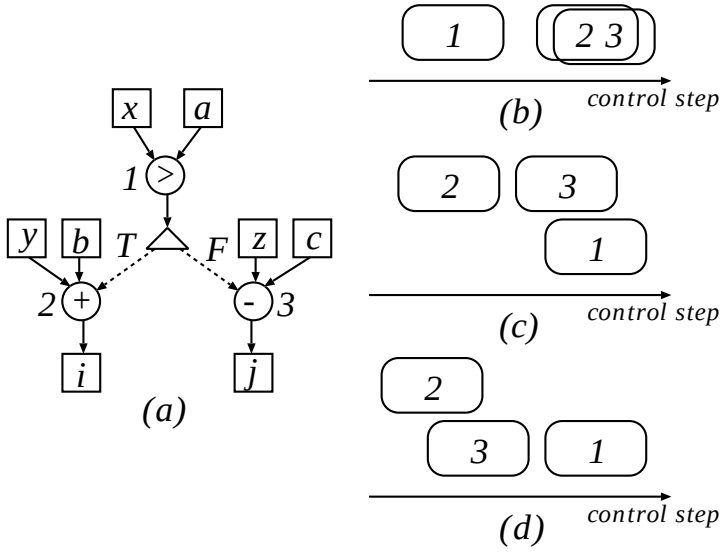

Figure 5.1: A computation algorithm with a conditional branch and the schedules for operations.

data dependencies and therefore no precedence constraints from the decision operation to the conditional operations.

In this chapter, we will discuss assignment constrained scheduling for iterative algorithms with conditional branches. Many researchers have proposed scheduling methods for computation algorithms with conditional branches [31, 32, 33, 34, 35]. However, all these scheduling methods are time constrained scheduling or resource constrained scheduling. We propose an approach using a scheduling graph with disjunctive arcs generated form the specified assignment information. Especially, we consider disjunctive arcs to obtain a feasible schedule for mutually exclusive operations to functional units assignment and mutually exclusive data to registers assignment, and derive a branch-and-bound method and a simple heuristic method incorporated with successive refinement of parameter space.

5.2 Preliminaries

5.2.1 Dependence Graph with Conditional Branches

Although we define a dependence graph representing an iterative algorithm in Chapter 2, in this chapter we redefine a dependence graph to represent an iterative algorithm with conditional branches.

A computation algorithm with conditional branches is specified with a directed graph $G = (V_G, A_G)$, which is called "dependence graph with conditional branches". V_G is a union of a set V_O of operations, a set V_D of data, a set V_F of fork nodes, and a set V_J of join nodes. A_G is a union of a set A_D of arcs which represent a data dependency and are called data dependency arcs, and a set A_C of arcs which imply the operations adjoin to ones are conditionally executed and are called control dependency arcs. An example of a dependence graph with conditional branches is shown in Fig 5.2(a). The number of control steps, which is needed for executing each operation, is given by $e : V_O \rightarrow Z_+$,

where a control step is a time unit. We assume that the number of control steps which is needed at a fork node or a join node is 0.

Each fork node $f \in V_F$ implies the start of conditional branch, and has one incoming data dependency arc and two outgoing control dependency arcs. The branch to be executed is determined by the operation, which is an immediate predecessor of a fork node, and it is called the decision operation of the branch. For example, operation o_1 is the decision operation of branch f_1 in Fig. 5.2. If the condition is true, then operations o_2 is executed and data d_2 is generated and used, and if the condition is false, then operations o_3 is executed and data d_3 is generated and used. Note that a control dependency arc does not imply data dependences from the fork node to the conditional operations.

Each join node $j \in V_J$ implies the conditional operation flows from a fork node join to a single operation flow, and has two incoming data dependency arcs and one outgoing data dependency arc. Thus each fork node has its corresponding join node. For example in Fig. 5.2, two data flows which branch at the fork node f_1 join at the join node j_1 when either conditional data d_2 (generated by conditional operation o_2) or conditional data d_3 (generated by conditional operation o_3) is used.

Assuming that each operation in a dependence graph is executed repeatedly, each data dependency arc (d, o) from data d to operation o is associated with its delay $D(d, o)$ which implies that d in kth ($k \in Z$) iteration (denoted as d^k) is used by o in $(k + D(d, o))$th iteration (i.e., $o^{k+D(d,o)}$). For example in Fig. 5.2, data d_4 (d_4^k) which is generated by operation o_4 (o_4^k) is used by operation o_1 in the next iteration (o_1^{k+1}). A dependency arc (j, o) from join node j to operation o also have its delay $D(j, o)$. The delay $D(j, o)$ implies that data d^k before join node j is used by $o^{k+D(j,o)}$ (neglecting the join node j which is contained in a path from data d to operation o). Note that we use delay $D(j, o)$ of a data dependency arc from join node j to operation o identified by delay $D(d, o)$ of one from data d to operation o.

5.2.2 Problem Formulation

The input and output of assignment constrained scheduling problem with conditional branches are represented as follows.

Input;

- Dependence graph with conditional branches $G = (V_G, A_G)$:
- Resource assignment ρ, ξ:

Output;

- Pipeline schedule σ:

  The following constraints must be satisfied.

  1. A schedule satisfies the precedence constraints specified by execution time of operations, data dependency arcs and their delay in G.
  2. If two operations (data), which are not mutually exclusive to each other, are assigned to the same functional unit (register), then their lifetimes do not overlap.
  3. If two operations (data), which are mutually exclusive to each other, are assigned to the same functional unit (register), then

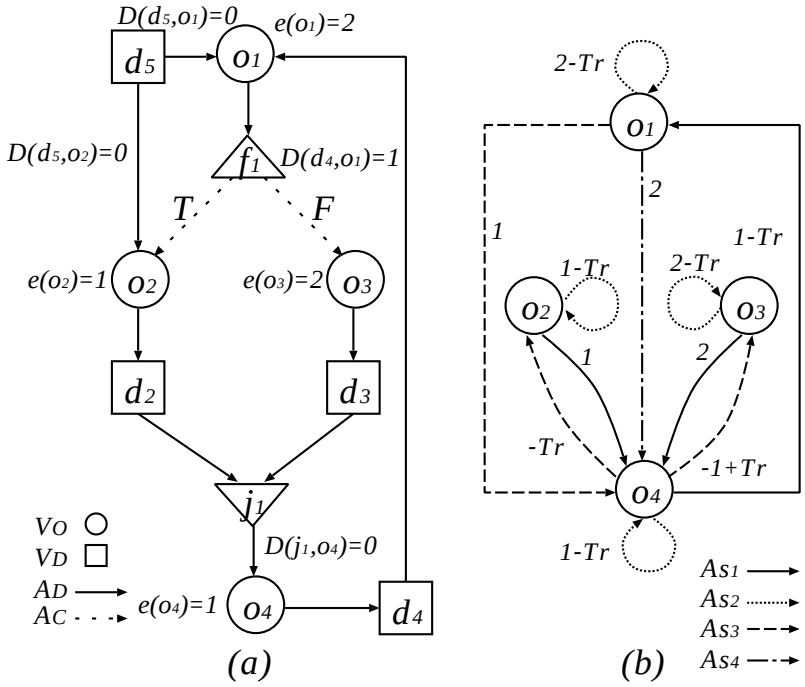

Figure 5.2: Dependence graph with conditional branches and its initial scheduling graph.

a. their lifetimes do not overlap,
b. or either of these operations (data) is executed (stored) after the decision operation is completed.

5.2.3 Scheduling Graph

For a dependence graph with conditional branches, we construct a scheduling graph like one in Chapter 3.

$$
\begin{aligned}
A_{S1} &= \{(p(d), s_i(d))|(p(d), s_i(d)) \in V_D\}, \\
&\quad E_S(p(d), s_i(d)) = e(p(d)), D_S(p(d), s_i(d)) = D(d, s_i(d)), \\
A_{S2} &= \{(o_i, o_i)|o_i \in V_O\}, \\
&\quad E_S(o_i, o_i) = e(o_i), D_S(o_i, o_i) = 1, \\
A_{S3} &= \{(s_j(d_i), p(d_i))|d_i \in V_D\}, \\
&\quad E_S(s_j(d_i), p(d_i)) = e(s_j(d_i)) - e(p(d_i)), \\
&\quad D_S(s_j(d_i), p(d_i)) = -D(p(d_i), s_j(d_i)) + 1.
\end{aligned}
$$

A_{S1} is a set of constraint arcs which reflects precedence constraints of the given dependence graph G, while A_{S2} and A_{S3} are sets of constraint arcs to avoid overlap between successive lifetimes of each operation and data, respectively, (i.e. $\tau(o_i^k) \cap \tau(o_i^{k+1}) = \emptyset$, $\tau(d_i^k) \cap \tau(d_i^{k+1}) = \emptyset$, $\forall k \in Z$).

Moreover there exists does not exist a data dependency between operation $p(f)$ immediately preceding fork node f and operation $s(j(f))$ succeeding its corresponding join node $j(f)$ but a precedence constraint between these operations. A set A_{S4} of arcs which

reflect these precedence constraints are represented as follow.

$$\begin{aligned} A_{S4} &= \{(p(f), s(j(f)) | f \in V_F\}, \\ & E_S(p(d), s_i(d)) = e(p(f)), D_S(p(f), s(j(f))) = D(j(f), s(j(f))). \end{aligned}$$

A scheduling graph $G_S = (V_S = V_O, A_S = A_{S1} \cup A_{S2} \cup A_{S3} \cup A_{S4})$ is called "initial scheduling graph" for a dependence graph with conditional branches. Fig 5.2(b) shows the initial scheduling graph constructed from the dependence graph with conditional branches in Fig. 5.2(a).

5.3 Disjunctive Arc Approach

In this chapter, we propose an approach using a scheduling graph with variable disjunctive arcs generated from assignment information in a loop pipeline scheduling problem for a dependency graph with conditional branches. Especially, if two operations (data), which are mutually exclusive to each other, are assigned to the same functional unit (register), then it should be satisfied either that their lifetimes do not overlap or that either of the operations (data) is executed (stored) after the decision operation is completed. Section 5.3.1 and Section 5.3.2 present disjunctive arc approach in two cases to obtain a feasible schedule, respectively.

5.3.1 Collision-Free of Lifetimes

If two operations o_i and o_j in G, which are mutually exclusive to each other, are assigned to the same functional unit (i.e. $\rho(o_i) = \rho(o_j)$), then one feasible solution is that the lifetimes $\tau(o_i)$ and $\tau(o_j)$ are not overlap, which is satisfied if and only if there exists an integer $F_{i,j}$ such that;

(1) $\tau(o_i^k)$ precedes $\tau(o_j^{k-F_{i,j}})$, that is,

$$\sigma(o_j) + (k - F_{i,j})T_r \quad \geq \quad \sigma(o_i) + e(o_i) + kT_r.$$

(2) $\tau(o_j^{k-F_{i,j}})$ precedes $\tau(o_i^{k+1})$, that is,

$$\sigma(o_i) + (k+1)T_r \quad \geq \quad \sigma(o_j) + e(o_j) + (k - F_{i,j})T_r.$$

On the scheduling graph G_S, these two constraints can be represented by adding a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$\begin{aligned} E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = -F_{i,j}, \\ E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1 + F_{i,j}. \end{aligned}$$

Fig. 5.3(a) shows an example of these disjunctive arcs.

Similarly, if two data d_i and d_j which are generated by mutually exclusive operations o_i and o_j, respectively, are assigned to the same register, then one feasible solution is that the lifetimes $\tau(d_i)$ and $\tau(d_j)$ are not overlap. On G_S, these constraints are represented by adding disjunctive arcs $(s_x(o_i), o_j)$ for every operation $s_x(o_i)$ which uses d_i as its input,

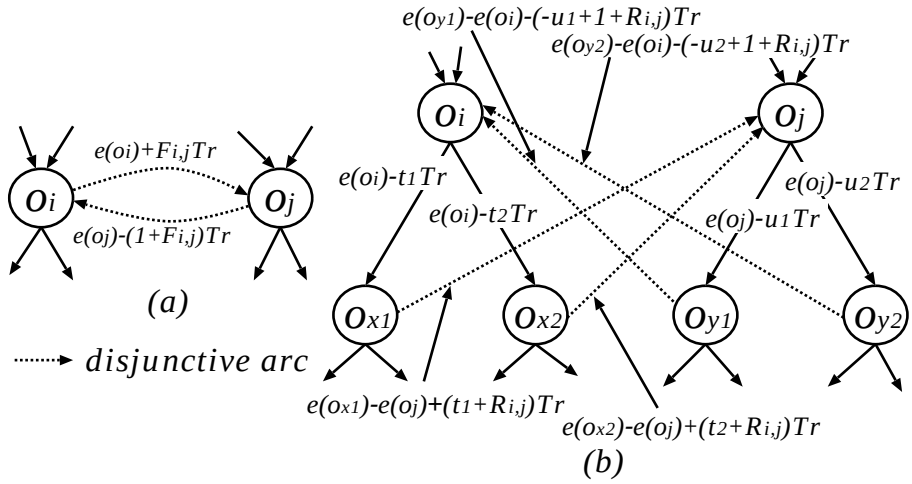

Figure 5.3: Pairs of disjunctive arcs induced by functional unit assignment (a) and by register assignment (b).

and $(s_y(o_j), o_i)$ for every operation $s_y(o_j)$ which uses d_j as its input, and their weight functions are given as,

$$E_S(s_x(o_i), o_j) = e(s_x(o_i)) - e(o_j),\ D_S(s_x(o_i), o_j) = -t_x - R_{i,j},$$
$$E_S(s_y(o_j), o_i) = e(s_y(o_j)) - e(o_i),\ D_S(s_y(o_j), o_i) = -u_y + 1 + R_{i,j},$$

where $t_x = D_S(o_i, s_x(o_i))$, $u_y = D_S(o_j, s_y(o_j))$, and $R_{i,j}$ is an unknown integer to be fixed (Fig. 5.3(b)).

Note that, in a dependence graph with conditional branches, data dependency arcs between conditional operations (which are executed depending on the result of a decision operation) contain no delay. Thus if two mutually exclusive operations o_i and o_j assigned to the same functional unit, the lifetimes should not overlap by either of integers $\{-1, 0\}$ for $F_{i,j}$. Similarly if two mutually exclusive data d_i and d_j assigned to the same register, the lifetimes should not overlap by either of integers $\{-1, 0\}$ for $R_{i,j}$.

5.3.2 Execution after Decision Operation

If two operations o_i and o_j, which are mutually exclusive to each other, are assigned to the same functional unit (i.e. $\rho(o_i) = \rho(o_j)$), the other feasible solution is that either of the following constraints must be satisfied;

(1) The lifetime of kth execution of the decision operation o_d (i.e. $\tau(o_d^k)$), which decides which operation is executed, precedes the lifetime of kth execution of the operation o_i (i.e. $\tau(o_i^k)$), and the lifetimes of kth executions of these two operations o_i and o_j (i.e. $\tau(o_i^k)$ and $\tau(o_j^k)$) precede the lifetimes of $(k+1)$th executions of these two operations o_i and o_j (i.e. $\tau(o_i^{k+1})$ and $\tau(o_j^{k+1})$) (see Fig. 5.4(1)).

(2) The lifetime of kth execution of the decision operation o_d (i.e. $\tau(o_d^k)$), which decides which operation is executed, precedes the lifetime of kth execution of the operation o_j (i.e. $\tau(o_j^k)$), and the lifetimes of kth executions of these two operations o_i and

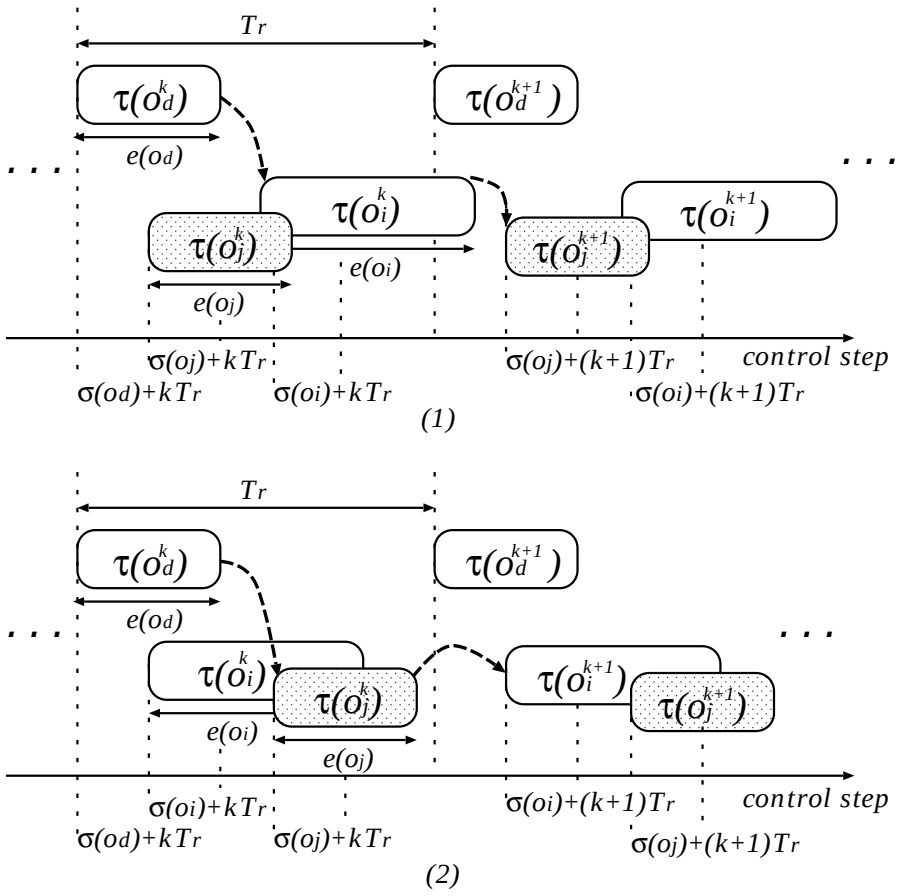

Figure 5.4: Precedence constraints between conditional operations assigned to the same functional unit and the corresponding decision operation.

o_j (i.e. $\tau(o_i^k)$ and $\tau(o_j^k)$) precede the lifetimes of $(k+1)$th executions of these two operations o_i and o_j (i.e. $\tau(o_i^{k+1})$ and $\tau(o_j^{k+1})$) (see Fig. 5.4(2)).

From above two constrains, we obtain the following inequalities, respectively.

(1)

$$\begin{aligned} \sigma(o_i) + kT_r &\geq \sigma(o_d) + e(o_d) + kT_r \\ \sigma(o_i) &\geq \sigma(o_d) + e(o_d). \end{aligned} \tag{5.1}$$

$$\begin{aligned} \mathrm{MIN}\{\sigma(o_i), \sigma(o_j)\} + (k+1)T_r &\geq \mathrm{MAX}\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\} + kT_r \\ \mathrm{MIN}\{\sigma(o_i), \sigma(o_j)\} &\geq \mathrm{MAX}\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\} - T_r. \end{aligned} \tag{5.2}$$

(2)

$$\begin{aligned} \sigma(o_j) + kT_r &\geq \sigma(o_d) + e(o_d) + kT_r \\ \sigma(o_j) &\geq \sigma(o_d) + e(o_d). \end{aligned} \tag{5.3}$$

$$\begin{aligned} \mathrm{MIN}\{\sigma(o_i), \sigma(o_j)\} + (k+1)T_r &\geq \mathrm{MAX}\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\} + kT_r \\ \mathrm{MIN}\{\sigma(o_i), \sigma(o_j)\} &\geq \mathrm{MAX}\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\} - T_r. \end{aligned} \tag{5.4}$$

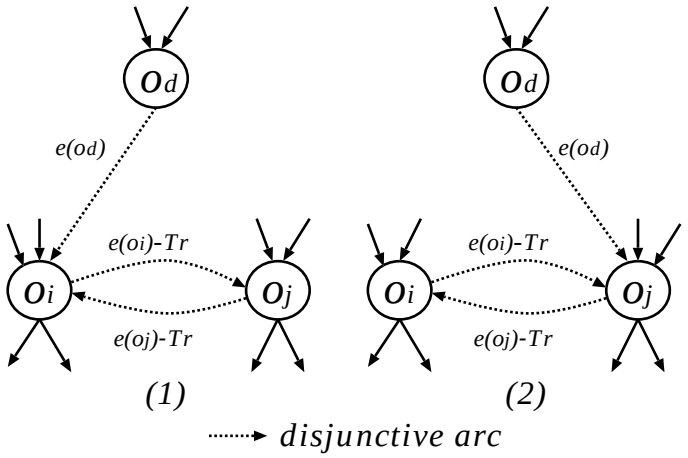

Figure 5.5: Disjunctive arcs to execute after the decision operation o_d induced by functional unit assignment.

On the scheduling graph G_S, these two constraints are represented by adding disjunctive arcs as following, respectively.

(1) The constraint is respected by adding (o_d, o_i) with

$$E_S(o_d, o_i) = e(o_d)$$

and a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = 1,$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1.$$

(2) The constraint is respected by adding (o_d, o_j) with

$$E_S(o_d, o_j) = e(o_d)$$

and a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = 1,$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1.$$

Examples of disjunctive arcs representing these two constraints (1) and (2), respectively, are shown in Fig. 5.5.

Similarly, if two data d_i and d_j, which are generated by mutually exclusive operations o_i and o_j, respectively, are assigned to the same register (i.e. $\xi(d_i) = \xi(d_j)$), then the satisfaction of either of the following constraints may derive feasible solution.

(1) The lifetime of kth execution of the decision operation o_d (i.e. $\tau(o_d^k)$), which decides which data is used, precedes the lifetime of data d_i which is generated by kth execution of the operation o_i (i.e. $\tau(d_i^k)$). And the lifetimes of these two data d_i and d_j which are generated by kth executions of these two operations o_i and o_j, respectively, (i.e. $\tau(d_i^k)$ and $\tau(d_j^k)$) precede the lifetimes of these two data d_i and d_j which are generated by $(k+1)$th executions of these two operations o_i and o_j, respectively, (i.e. $\tau(d_i^{k+1})$ and $\tau(d_j^{k+1})$).

(2) The lifetime of kth execution of the decision operation o_d (i.e. $\tau(o_d^k)$), which decides which data is used, precedes the lifetime of data d_j which is generated by kth execution of the operation o_j (i.e. $\tau(d_j^k)$). And the lifetimes of these two data d_i and d_j which are generated by kth executions of these two operations o_i and o_j, respectively, (i.e. $\tau(d_i^k)$ and $\tau(d_j^k)$) precede the lifetimes of these two data d_i and d_j which are generated by $(k+1)$th executions of these two operations o_i and o_j, respectively, (i.e. $\tau(d_i^{k+1})$ and $\tau(d_j^{k+1})$).

When operation $s_x(o_i)$ uses d_i as its input and $t_x = D_S(o_i, s_x(o_i))$, and at the same time operation $s_y(o_j)$ which uses d_j as its input and $u_y = D_S(o_j, s_y(o_j))$, we obtain the following inequalities from the above two constraints.

(1)

$$\begin{aligned} \sigma(o_i) + e(o_i) + kT_r &\geq \sigma(o_d) + e(o_d) + kT_r \\ \sigma(o_i) &\geq \sigma(o_d) + e(o_d) - e(o_i). \end{aligned} \tag{5.5}$$

$$\begin{aligned} &\mathrm{MIN}\left\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\right\} + (k+1)T_r \\ &\quad \geq \mathrm{MAX}\left\{\underset{x}{\mathrm{MAX}}\left\{\sigma(s_x(o_i)) + e(s_x(o_i)) + txT_r\right\}, \right. \\ &\qquad\qquad \left. \underset{y}{\mathrm{MAX}}\left\{\sigma(s_y(o_j)) + e(s_y(o_j)) + u_yT_r\right\}\right\} + kT_r \\ &\mathrm{MIN}\left\{\sigma(o_i), \sigma(o_j)\right\} \\ &\quad \geq \mathrm{MAX}\left\{\underset{x}{\mathrm{MAX}}\left\{\sigma(s_x(o_i)) + e(s_x(o_i)) - e(o_j) + (tx-1)T_r\right\}, \right. \\ &\qquad\qquad \left. \underset{y}{\mathrm{MAX}}\left\{\sigma(s_y(o_j)) + e(s_y(o_j)) - e(o_i) + (u_y-1)T_r\right\}\right\}. \end{aligned} \tag{5.6}$$

(2)

$$\begin{aligned} \sigma(o_j) + e(o_j) + kT_r &\geq \sigma(o_d) + e(o_d) + kT_r \\ \sigma(o_j) &\geq \sigma(o_d) + e(o_d) - e(o_j). \end{aligned} \tag{5.7}$$

$$\begin{aligned} &\mathrm{MIN}\left\{\sigma(o_i) + e(o_i), \sigma(o_j) + e(o_j)\right\} + (k+1)T_r \\ &\quad \geq \mathrm{MAX}\left\{\underset{x}{\mathrm{MAX}}\left\{\sigma(s_x(o_i)) + e(s_x(o_i)) + txT_r\right\}, \right. \\ &\qquad\qquad \left. \underset{y}{\mathrm{MAX}}\left\{\sigma(s_y(o_j)) + e(s_y(o_j)) + u_yT_r\right\}\right\} + kT_r \\ &\mathrm{MIN}\left\{\sigma(o_i), \sigma(o_j)\right\} \\ &\quad \geq \mathrm{MAX}\left\{\underset{x}{\mathrm{MAX}}\left\{\sigma(s_x(o_i)) + e(s_x(o_i)) - e(o_j) + (tx-1)T_r\right\}, \right. \\ &\qquad\qquad \left. \underset{y}{\mathrm{MAX}}\left\{\sigma(s_y(o_j)) + e(s_y(o_j)) - e(o_i) + (u_y-1)T_r\right\}\right\}. \end{aligned} \tag{5.8}$$

On G_S, these two constraints are represented by adding disjunctive arcs as following, respectively.

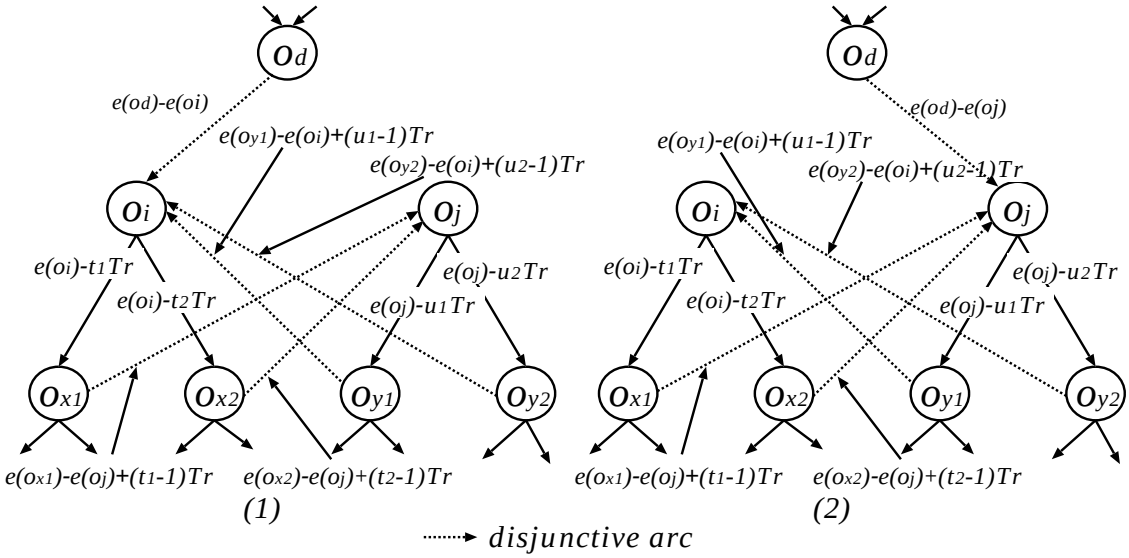

Figure 5.6: Disjunctive arcs to execute after the decision operation o_d induced by register assignment

(1) The constraint is represented by adding arc (o_d, o_i) with

$$E_S(o_d, o_i) = e(o_d) - e(o_i)$$

and pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$ with

$$E_S(s_x(o_i), o_j) = e(s_x(o_i)) - e(o_j),\ D_S(s_x(o_i), o_j) = -t_x + 1,$$
$$E_S(s_y(o_j), o_i) = e(s_y(o_j)) - e(o_i),\ D_S(s_y(o_j), o_i) = -u_y + 1.$$

(2) The constraint is represented by adding arc (o_d, o_j) with

$$E_S(o_d, o_j) = e(o_d) - e(o_j)$$

and pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$ with

$$E_S(s_x(o_i), o_j) = e(s_x(o_i)) - e(o_j),\ D_S(s_x(o_i), o_j) = -t_x + 1,$$
$$E_S(s_y(o_j), o_i) = e(s_y(o_j)) - e(o_i),\ D_S(s_y(o_j), o_i) = -u_y + 1.$$

Examples of disjunctive arcs representing these two constraints (1) and (2), respectively, are shown in Fig. 5.6.

5.3.3 Problem Transformation

A variable $F_{i,j}$ is introduced into G_S for every pair of operations o_i and o_j which are assigned to the same functional unit. A variable $R_{i,j}$ is also introduced into G_S for every pair of data d_i (generated by o_i) and d_j (generated by o_j) which are assigned to the same register. Especially, A variable $F'_{i,j}$ ($R'_{i,j}$) is used for every pair of mutual exclusive operations o_i and o_j (mutual exclusive data d_i and d_j). We denote the set of all variables

$F_{i,j}$s as $\boldsymbol{F}$, the set of all variables $F'_{i,j}$s as $\boldsymbol{F'}$, the set of all variables $R_{i,j}$s as $\boldsymbol{R}$ and the set of all variables $R'_{i,j}$s as $\boldsymbol{R'}$.

Finally, our assignment constrained scheduling problem is transformed into the problem to determine which disjunctive arcs are added so that the resultant scheduling graph contains no positive cycles;

- for two mutually exclusive operations o_i and o_j assigned to the same functional unit,

  F1. a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = -F'_{i,j},$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1 + F'_{i,j}.$$

  where $F'_{i,j}$ is fixed as a single integer.

  F2. a arc (o_d, o_i) and a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_d, o_i) = e(o_d),\ D_S(o_d, o_i) = 0,$$
$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = 1,$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1.$$

  F3. a arc (o_d, o_j) and a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_d, o_j) = e(o_d),\ D_S(o_d, o_j) = 0,$$
$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = 1,$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1.$$

- for the other two operations o_i and o_j assigned to the same functional unit,

  F1. a pair of arcs (o_i, o_j) and (o_j, o_i) with

$$E_S(o_i, o_j) = e(o_i),\ D_S(o_i, o_j) = -F_{i,j},$$
$$E_S(o_j, o_i) = e(o_j),\ D_S(o_j, o_i) = 1 + F_{i,j}.$$

  where $F_{i,j}$ is fixed as a single integer.

- for two mutually exclusive data d_i and d_j assigned to the same register,

  R1. pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$,

$$E_S(s_x(o_i), o_j) = e_{xj},\ D_S(s_x(o_i), o_j) = -t_x - R'_{i,j},$$
$$E_S(s_y(o_j), o_i) = e_{yi},\ D_S(s_y(o_j), o_i) = -u_y + 1 + R'_{i,j},$$

  where $R'_{i,j}$ is fixed as a single integer.

  R2. a arc (o_d, o_i) and pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$ with

$$E_S(o_d, o_i) = e(o_d) - e(o_i),\ D_S(o_d, o_i) = 0,$$
$$E_S(s_x(o_i), o_j) = e_{xj},\ D_S(s_x(o_i), o_j) = -t_x + 1,$$
$$E_S(s_y(o_j), o_i) = e_{yi},\ D_S(s_y(o_j), o_i) = -u_y + 1,$$

  R3. a arc (o_d, o_j) and pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$ with

$$E_S(o_d, o_j) = e(o_d) - e(o_j),\ D_S(o_d, o_j) = 0,$$
$$E_S(s_x(o_i), o_j) = e_{xj},\ D_S(s_x(o_i), o_j) = -t_x + 1,$$
$$E_S(s_y(o_j), o_i) = e_{yi},\ D_S(s_y(o_j), o_i) = -u_y + 1,$$

- for the other two data d_i and d_j assigned to the same register,
  R1. pairs of arcs $(s_x(o_i), o_j)$ and $(s_y(o_j), o_i)$,

$$E_S(s_x(o_i), o_j) = e_{xj},\ D_S(s_x(o_i), o_j) = -t_x - R_{i,j},$$
$$E_S(s_y(o_j), o_i) = e_{yi},\ D_S(s_y(o_j), o_i) = -u_y + 1 + R_{i,j},$$

  where $R_{i,j}$ is fixed as a single integer.

Note that, when the candidate disjunctive arcs F2 (R2) for variable $F'_{i,j}(R'_{i,j}) \in \boldsymbol{F'}(\boldsymbol{R'})$ are added in the scheduling graph G_S, variable $F'_{i,j}(R'_{i,j})$ is assumed to be fixed as ∞. Similarly, when the candidate disjunctive arcs F3 (R3) for variable $F'_{i,j}(R'_{i,j}) \in \boldsymbol{F'}(\boldsymbol{R'})$ are added in the scheduling graph G_S, variable $F'_{i,j}(R'_{i,j})$ is assumed to be fixed as $-\infty$.

5.4 Scheduling Algorithm for Exact Solution

We show the scheduling (iteration period constraint scheduling) algorithm for iterative algorithm with conditional branches based on branch-and-bound exploration of the solution space for Σ. The outline of the algorithm is described in Fig. 5.7. An initial solution space for Σ is formed from a set of possible integers (possible range), which are calculated by using Theorem 2, for every unknown variables $(\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'})$. Also these ranges are updated using Theorem 2 to increase bounding opportunities, whenever a branching is proceeded. Once the corresponding scheduling graph G_S which contains no positive cycles is found, the scheduling σ is obtained by calculating the longest path lengths from a reference node to all nodes on G_S.

In our disjunctive arc approach, a variable arises for each pair of operations which are assigned to the same functional unit and data which are assigned to the same register. Thus the number of variables is the largest when all the operations are assigned to one functional unit and all the data are assigned to one register. When we let N be the number of operation nodes, $|\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}|$ is at most N^2. Moreover if we let f be the maximum number of integers in a ranges over all variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$, the total number of integer assignments to $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ is at most $(f+2)^{|\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}|} = (f+2)^{N^2}$. The time complexity of seeking the longest path length is $\mathcal{O}(N^3)$ if we use Floyd's algorithm. Therefore the time complexity of this algorithm is $\mathcal{O}(N^3(f+2)^{N^2})$.

5.5 Simple Heuristic Method

As we have mentioned in chapter 4, our final goal is to find the minimum iteration period T_r^*, but we also need to find integer for variables $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ to confirm the feasibility of T_r^*. Our basic strategy is the alternate updates of the iteration period and the solution for variables $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$.

5.5.1 Heuristics to Find Solution for $\boldsymbol{F'} \cup \boldsymbol{R'}$

In chapter 4, we have proposed heuristics to find a solution for variables $\boldsymbol{F} \cup \boldsymbol{R}$ based on the reduction of possible ranges using sensitivity to iteration period. However, this heuristics is not applied directly to $\boldsymbol{F'} \cup \boldsymbol{R'}$, since we choose only one among disjunctive arcs with a variable (F1 or R1) and disjunctive arcs without any variable (F2 and F3 or

SCHEDULING(G, ρ, ξ, T_r)

1. Construct initial scheduling graph G_{S_0} from G.
2. Construct variable list $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ from ρ and ξ.
3. **if** (BAB$(G_{S_0}, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) "SUCCESS"
   else "FAIL"

BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r)$

Step 1: Calculate the longest path length of every pair of nodes on scheduling graph G_S If positive cycles are detected, **return(0)**.

Step 2: If there is no un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$, print the longest path lengths from the reference node to all other nodes, and **return(1)**.

Step 3: Calculate the possible ranges of remained variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$. If some range contains no integer in $\boldsymbol{F} \cup \boldsymbol{R}$, **return(0)**.

Step 4: If there are un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R}$ whose ranges contain exactly one integer, then fix them, add the corresponding disjunctive arcs (F1 or R1) in G_S, and go to Step 1.

Step 5: If, for $F'_{i,j}$ (or $R'_{i,j}$) $\in \boldsymbol{F'}$ (or $\boldsymbol{R'}$), operation o_i (or data d_i) is executed (stored) after the decision operation o_d is completed in the same iteration period, then fix ∞, add the corresponding disjunctive arcs F2 (or R2) in G_S, and go to Step 1.

Step 6: If, for $F'_{i,j}$ (or $R'_{i,j}$) $\in \boldsymbol{F'}$ (or $\boldsymbol{R'}$) and operation o_j (or data d_j) is executed (stored) after the decision operation o_d is completed in the same iteration period, then fix $-\infty$, add the corresponding disjunctive arcs F3 (or R3) in G_S, and go to Step 1.

Step 7: Select one un-fixed variable $X \in \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$.

for (each integer contained in its range)

- 7-1. fix the value, and add the corresponding disjunctive arcs (F1 or R1) in G_S.
- 7-2. **if** (BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) **return(1)**.
- 7-3. delete disjunctive arcs added in Step 7-1.

if $(X \in \boldsymbol{F'})$

- 7-4. fix ∞, and add the candidate disjunctive arcs (F2) in G_S.
- 7-5. **if** (BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) **return(1)**.
- 7-6. delete disjunctive arcs added in Step 7-4.
- 7-7. fix ∞, and add the candidate disjunctive arcs (F3) in G_S.
- 7-8. **if** (BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) **return(1)**.
- 7-9. delete disjunctive arcs added in Step 7-7.

else if $(X \in \boldsymbol{R'})$

- 7-4. fix ∞, and add the candidate disjunctive arcs (R2) in G_S.
- 7-5. **if** (BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) **return(1)**.
- 7-6. delete disjunctive arcs added in Step 7-4.
- 7-7. fix ∞, and add the candidate disjunctive arcs (R3) in G_S.
- 7-8. **if** (BAB$(G_S, \boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}, T_r) == 1$) **return(1)**.
- 7-9. delete disjunctive arcs added in Step 7-7.

return(0)

Figure 5.7: Outline of our assignment constrained scheduling algorithm for an iterative algorithm with conditional branches

R2 and R3), and the latter is independent of possible integers contained in the possible range.

To choose one heuristically among several candidates for a variable in $\boldsymbol{F'} \cup \boldsymbol{R'}$, one-step look-ahead (or 1-opt) strategy will be adopted here. For each un-fixed variable $X \in \boldsymbol{F'} \cup \boldsymbol{R'}$, we temporary add one of candidate disjunctive arcs and calculate the minimum iteration period T_r^{min}. If the scheduling graph becomes to contain positive cycles after the addition of candidate disjunctive arcs, its corresponding minimum iteration period T_r^{min} is set to ∞. Basically we are going to find the minimum T_r^{min} over all un-fixed variable in $\boldsymbol{F'} \cup \boldsymbol{R'}$ and all their candidate disjunctive arcs and to adopt its corresponding choice.

The routine HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T$) is the one to find the solution for $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ under given iteration period T. In HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T$), the possible ranges under T are calculated by using Theorem 2 for every un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$. If there are un-fixed variables for $\boldsymbol{F} \cup \boldsymbol{R}$ each range of which contains exactly one integer, then we fix them without any further qualification, add the corresponding disjunctive arcs on G_S, and update the other ranges. On the other hand, for un-fixed variables in $\boldsymbol{F'} \cup \boldsymbol{R'}$, if there is exactly one candidate disjunctive arc having the minimum iteration period $T_r^{min} \neq \infty$ (i.e., the other candidate disjunctive arcs have the minimum iteration period $T_r^{min} = \infty$) for one variable $X \in \boldsymbol{F'} \cup \boldsymbol{R'}$, then we fix X corresponding to the finite T_r^{min} without any further qualification, add the disjunctive arcs on G_S, and update the other ranges. When it turns to the situation where the above two types of inevitable decisions are not applied to any un-fixed variables, it makes a heuristic decision. That is, with respect to $\boldsymbol{F'} \cup \boldsymbol{R'}$, we find the minimum T_r^{min} over all un-fixed variables in $\boldsymbol{F'} \cup \boldsymbol{R'}$ and all their candidate disjunctive arcs, and fix one variable to an appropriate value according to the found minimum T_r^{min}. With respect to $\boldsymbol{F} \cup \boldsymbol{R}$, we find one variable X (un-fixed element in $\boldsymbol{F} \cup \boldsymbol{R}$) which has maximum $\Gamma(X)$ among the minimum $\Delta(X)$, and fix X as the integer which is contained in the reduced range under the iteration period $T - \Delta(X)$.

For more details, please refer to Fig. 5.8.

5.5.2 Overall Scheduler

Fig. 5.8 shows the outline of the proposed assignment constrained heuristic scheduling based on the alternate updates of the iteration period and the solution for $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$, and the imaginary decrement of the iteration period with Δ, Γ priority in the latter. In the algorithm, we begin with HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T$) for the upper bound of the iteration period ($T = T_r^{max}$). Once every variable is fixed so that the resultant scheduling graph G_S contains no positive cycles, we calculate the minimum iteration period T_r' on G_S, and update T_r as T_r'. Executing HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) repeatedly with updating T_r as T_r' until it returns "FAIL", we obtain the minimum iteration period T_r under given G, ρ, and ξ. When σ is required, we apply ASAP scheduling under the T_r using the solution for $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ obtained in the last HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) which returns "SUCCESS".

5.6 Experiments

The performance of our heuristic scheduling algorithm is evaluated using 1000 different resource assignments for dependence graph with conditional branches, which contains 13

SCHEDULING(G,ρ,ξ)

1. $T_r \leftarrow T_r^{max} + 1$ (T_r^{max} is the upper bound of the iteration period).
2. Construct the initial scheduling graph G_{S_0} from G.
3. Construct variable list $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ from ρ and ξ.
4. **while** (HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) = "SUCCESS")

   Using the solution for $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ obtained in HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$), calculate minimum iteration period T_r'.

   $T_r \leftarrow T_r'$.
5. **if** ($T_r = T_r^{max} + 1$)

   return(∞). /* No feasible schedule */

   else

   return(T_r). /* T_r minimum iteration period */

   /* Apply ASAP scheduling under the T_r, when σ is required. */

HS(G_S,$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T$)

Step 1: Calculate the longest path length of every pair of vertices on G_S under the iteration period T. If positive cycles are detected, **return("FAIL")**.

Step 2: If there is no un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$, **return("SUCCESS")**.

Step 3: Calculate the possible ranges, Δ, and Γ of remained variables in $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$. If some range contains no integer, **return("FAIL")**.

Step 4: If there are un-fixed variables in $\boldsymbol{F} \cup \boldsymbol{R}$ whose ranges contain exactly one integer, then fix them, add the corresponding disjunctive arcs (F1 or R1) in G_S, and go to Step 1.

Step 5: If, for $F_{i,j}'$ (or $R_{i,j}'$) $\in \boldsymbol{F'}$ (or $\boldsymbol{R'}$), operation o_i (or data d_i) is executed (stored) after the decision operation o_d is completed in the same iteration period, then fix ∞, add the corresponding disjunctive arcs F2 (or R2) in G_S, and go to Step 1.

Step 6: If, for $F_{i,j}'$ (or $R_{i,j}'$) $\in \boldsymbol{F'}$ (or $\boldsymbol{R'}$) and operation o_j (or data d_j) is executed (stored) after the decision operation o_d is completed in the same iteration period, then fix $-\infty$, add the corresponding disjunctive arcs F3 (or R3) in G_S, and go to Step 1.

Step 7: For the candidate disjunctive arcs having the minimum iteration period T_r^{min} in un-fixed variables $\boldsymbol{F'} \cup \boldsymbol{R'}$, the corresponding variable is fixed as the corresponding integer, add the corresponding disjunctive arcs in G_S, and return to Step 1.

Step 8: For the un-fixed variable $X \in \boldsymbol{F} \cup \boldsymbol{R}$ which has the maximum $\Gamma(X)$ among the minimum $\Delta(X)$, fix X as one integer which is contained in the reduced range under $T - \Delta(X)$, exceptionally $\lfloor (X^{upper} - X^{lower})/2 \rfloor$ in case (C1), or X^{upper} or X^{lower} in case (C2), add the corresponding disjunctive arcs in G_S, and return to Step 1.

Figure 5.8: Outline of our assignment constrained heuristic scheduling algorithm.

Table 5.1: Results of our scheduling.

method	Heuristics								B&B
difference	+0	+1	+2	+3	+4	+5	+6	NS	-
1st iteration	646	178	91	39	16	10	2	18	average
2nd iteration	118	34	3	1	2	0	0	0	4.47
3rd iteration	17	1	0	0	0	0	0	0	iterations
4th iteration	1	0	0	0	0	0	0	0	
5th iteration	0	0	0	0	0	0	0	0	
final result	782	97	44	33	16	8	2	18	
runtime[s]	0.068								1.132

additions, 4 multiplications, 15 variables, 2 fork nodes and 2 join nodes. The proposed scheduling algorithm is implemented using C programming language on a 1GHz Pentium III personal computer.

Table 5.1 lists the numbers of instances as entries of the matrix whose column corresponds to the difference between the minimum iteration period T_r' calculated using the solution for $\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$ obtained in HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) and the optimum solution obtained by branch-and-bound method and whose row corresponds to the turn of HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) repetition. The number which appears in ith iteration row, $+j$ column denotes the number of instances among 1000 for which T_r' = "optimum iteration period $+j$" is obtained in ith turn of HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) repetition. NS column denotes the number of infeasible schedules. As we can see from this table, HS(G_{S_0},$\boldsymbol{F} \cup \boldsymbol{R} \cup \boldsymbol{F'} \cup \boldsymbol{R'}$,$T_r - 1$) is repeated at most 5 times (including the final one to decide no solution for a smaller iteration period) until the procedure terminates, while branch-and-bound method needs to update T_r 4.47 times in average. As for the quality of solutions, our heuristic method produces optimum solutions for at least 78% of test instances. Table 5.2 shows solution distribution generated by our heuristic method. The last row of each sub-table shows average runtimes of our proposed heuristic method and of branch-and-bound method, both of which are for one input instance. The proposed heuristic method runs about 16 times faster than the exact solution method on average.

However, no schedules are obtained for 18 instances among 1000, for which, feasible schedules are obtained by using the exact solution method. An analysis of the feasible schedule for assignment constrained scheduling is left for a future problem.

5.7 Conclusion

In this chapter, we have discussed a loop pipeline scheduling problem for a dependence graph with conditional branches, and have extended our variable disjunctive arc approach using a scheduling graph with disjunctive arcs generated from the specified assignment information. After transforming our assignment constrained scheduling problem into the problem to determine which disjunctive arcs are added, we have proposed a branch-and-bound method and a simple heuristic method incorporated with successive refinement of parameter space.

Table 5.2: Iteration period distribution of our scheduling.

Heuristic	NS			2	2		1	1		5		3		4		
	21															1
	20									2		1		1		9
	19									8	1	3		4	1	-
	18									13	5	17	1	15	-	-
	17									15	3	12	3	-	-	-
	16									15	4	28	-	-	-	-
	15									37	16	-	-	-	-	-
	14							1	1	284	-	-	-	-	-	-
	13					2			6	-	-	-	-	-	-	-
	12			1			2	7	-	-	-	-	-	-	-	-
	11			1		1	23	-	-	-	-	-	-	-	-	-
	10		7	3	14	78	-	-	-	-	-	-	-	-	-	-
	9		4	12	92	-	-	-	-	-	-	-	-	-	-	-
	8		7	125	-	-	-	-	-	-	-	-	-	-	-	-
	7	1	88	-	-	-	-	-	-	-	-	-	-	-	-	-
	6	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Exact T_r		6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Chapter 6

Assignment-Driven Approach Based Data-Path Synthesis System

6.1 Introduction

Most of the conventional high-level synthesis aim mainly to minimize the number of control steps and the number of functional units, and they first decide the schedule and the number of functional units by resource constrained scheduling or time constrained scheduling, which are followed by resource assignment. However, the connectivity between modules is also an important metric for VLSIs for its connection with wire complexity, transmission delay, power consumption, testability, etc. In the stepwise design; scheduling and resource assignment in this order, it may be hard to make a decision on operation schedule with regarding connectivity, which shall be fixed only after resource assignment.

In contrast to the conventional approaches starting the scheduling following by the resource assignment, we are going to present assignment-driven approach in order to handle interconnection-related metric more accurately in this chapter. The number of interconnections is considered as the objective to be minimized, while the scheduling feasibility is treated as one of design constraint together with the number of available modules and iteration period. We incorporate the proposed scheduling method into Simulated Annealing (SA) exploration of assignment solution space. Each assignment solution visited in SA is a complete resource assignment, and each assignment solution is evaluated in its scheduling feasibility and the number of interconnections. Syntheses of data-path with the reduced number of interconnections are demonstrated by experimental results for a number of synthesis examples.

6.2 SA Based Assignment Space Exploration

As an application of the proposed scheduling under given resource assignment, we incorporate our scheduler into data-path synthesis based on assignment space exploration, which can respect connectivity between modules (functional units and registers) explicitly throughout the synthesis process.

We will treat multiplexer-type architecture, which consists of adders, non-pipelined multipliers, ALUs, registers, multiplexers and interconnections between modules (or terminals). Operation chaining is not considered here. We will consider data-path synthesis

SA Based Assignment Space Exploration

$\rho_i,\ \xi_i \leftarrow$ initial assignment
$\sigma_i \leftarrow$ initial schedule
$\rho_{best},\ \xi_{best},\ \sigma_{best} \leftarrow \rho_i,\ \xi_i,\ \sigma_i$
while stopping criteria are not satisfied **do**
  $\rho_j,\ \xi_j \leftarrow$ neighbor assignment
  $c \leftarrow$ cost(ρ_j, ξ_j) - cost(ρ_i, ξ_i)
  $y \leftarrow \min(1, \exp(-c/T))$
  $r \leftarrow$ random(0,1)
  call SCHEDULING$(G, \rho_j, \xi_j, T_r))$
  if scheduling feasibility & $(r < y)$ **then**
    $\rho_i,\ \xi_i,\ \sigma_i \leftarrow \rho_j,\ \xi_j,\ \sigma_j$
    if cost$(\rho_{best}, \xi_{best}) >$ cost(ρ_i, ξ_i) **then**
      $\rho_{best},\ \xi_{best},\ \sigma_{best} \leftarrow \rho_i,\ \xi_i,\ \sigma_i$
  update temperature

Figure 6.1: Outline of our SA approach.

problem to find the data-path with minimum number of point-to-point interconnections under given set of available modules (functional units and registers) $\mathcal{F}, \mathcal{R}$, and iteration period T_r. We adopt the resource assignment space exploration using Simulated Annealing (SA). Figure 6.1 shows the outline of SA based assignment space exploration. Each assignment solution visited in SA is a complete resource assignment (ρ, ξ), and each assignment solution is evaluated in its scheduling feasibility ("SUCCESS" or "FAIL" by SCHEDULING$(G, \rho, \xi, T_r))$ under given T_r and the number of point-to-point interconnections (by cost(ρ, ξ)). Neighbor assignment solution is generated by any of the following operations;

(1) change assignment of a single operation,

(2) change assignment of a single data,

(3) exchange assignment of two operations,

(4) exchange assignment of two data.

And it is accepted stochastically depending on evaluations of current and neighbor assignment solutions. Note that if an assignment solution is evaluated in no feasible schedule ("FAIL" by SCHEDULING(G, ρ, ξ, T_r)), we never accept the solution in our SA approach.

If an assignment solution is accepted, the number of interconnections, cost(ρ_i, ξ_i), is compared with the minimum number of interconnections so far, cost$(\rho_{best}, \xi_{best})$. If the cost$(\rho_i, \xi_i)$ is better than the minimum cost$(\rho_{best}, \xi_{best})$, the best assignment solution $(\rho_{best}, \xi_{best})$ is updated. Thus we can always obtain the best explored assignment solution while maintaining the ability to escape form local minima.

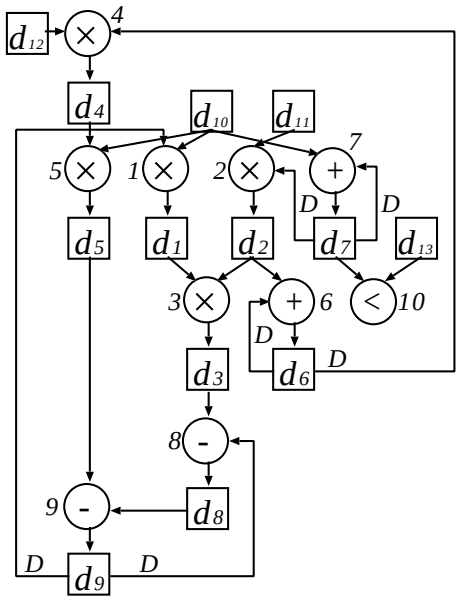

Figure 6.2: Differential equation benchmark

Table 6.1: Experimental results for differential equation ($T_r = 4$)

System	T_r	FU	R	Mx	Mi	ME	#CON
Ours	4	$1+, 1-, 1<, 2\times$	5	3	7	4	16
HAL	4	$1+, 1-, 1<, 2\times$	5	4	10	6	-
SE	4	$1+, 1-, 1<, 2\times$	5	4	9	5	-

6.3 Synthesis Examples

The proposed method is implemented using C program language on a 1GHz Pentium III personal computer. We present experimental results for a number of synthesis examples from the literature for comparison with results reported by other systems.

6.3.1 Differential Equation Example

Differential equation benchmark, which contains 5 additions, 5 multiplications, 10 variables and 4 constants (Fig. 6.2), is used as an input instance of data-path synthesis. First,

Table 6.2: Experimental results for differential equation

System	T_r	FU	R	Mx	Mi	ME	#CON
Ours	6	1ALUs,2×	6	5	11	6	17
	7	1ALUs,2×	5	2	9	7	14
	10	1ALUs,1×	5	4	10	6	15

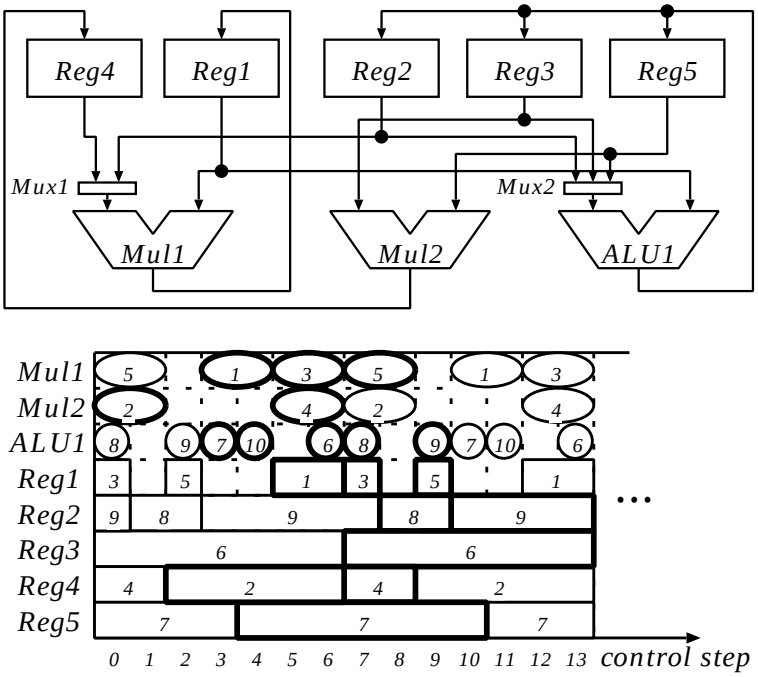

Figure 6.3: Data-path and schedule under $T_r = 7$

to compare with other systems we assume iteration period $T_r = 4$, and that an addition, a subtraction and a comparison are performed by an adder, a subtractor and a comparator, respectively, in one control step, and a multiplication is performed by a multiplier in one control step. However, since other systems enable to use operation chaining, additions, subtractions and comparisons are performed by adders, subtractors and comparators, respectively, in half a control step. The numbers of functional units and register are given as 1 adder, 1 subtracter, 1 comparator, 2 multipliers and 5 registers, respectively.

Table 6.1 shows the results of our proposed system, together with results of other systems SE and HAL quoted from [29]. In the table, FU and R show the numbers of functional units and registers, respectively. Mx, Mi, and ME show the number of multiplexers, multiplexer's inputs, and equivalent two-input single-output multiplexers respectively. #COM shows the number of point-to-point interconnections. Note that we eliminate the interconnection between constant data (multiplier) and input terminals of functional units in Mx, Mi, ME, and #CON. As we can see from this table, our system provides solutions with reduced numbers of multiplexers, multiplexer's inputs and equivalent two-input single-output multiplexers compared to best numbers of them from other systems. In addition, we obtained the minimum number of point-to-point interconnections and it took less than 1 second for this example.

Next, we assume that an addition, a subtraction and a comparison are performed by ALUs in one control step, respectively, and a multiplication is performed by a multiplier in two control steps. Table 6.2 shows the results of our system for three specifications of iteration period and the numbers of functional units and registers. We obtained the minimum number of point-to-point interconnections for all instances. Under each of specifications, it takes less than 1 second to obtain the best data-path and the best schedule. Figure 6.3 shows the resultant data-path and schedule under iteration period

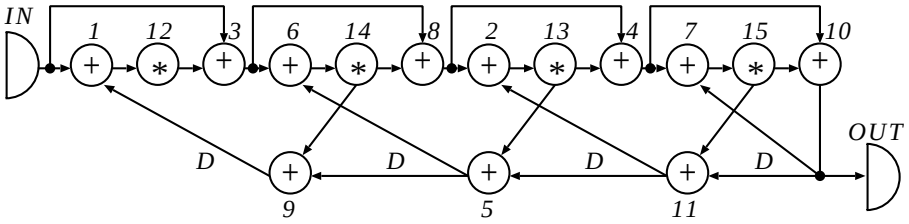

Figure 6.4: Four-order all-pole lattice filter

Table 6.3: Experimental results for four-order all pole lattice filter

System	T_r	FU	R	Mx	Mi	ME	#CON	runtime[s]
Ours	8	$3+, 2\times$	7	4	9	5	21	15(4761)
	9	$2+, 2\times$	6	5	11	6	19	18(4100)
	10	$2+, 1\times$	6	3	7	4	16	18(1336)
	11	$1+, 1\times$	7	3	9	6	17	19(202)

$T_r = 7$, 2 multipliers, 1 ALUs, and 5 registers.

6.3.2 Four-Order All-Pole Lattice Filter Example

Four-order all-pole lattice filter, which contains 11 additions, 4 multiplications, 16 variables and 4 constants (Fig. 6.4), is used as an input instance of second data-path synthesis. We assume that an addition is performed on an adder in one control step, and a multiplication is performed on a (non-pipelined) multiplier in two control steps. In SA, the temperature (T) which controls the probability of an uphill move ($\exp(-c/T)$) is scheduled with the form $T_i = 0.98 \times T_{i-1}$ from $T_0 = 15$ until it reaches 0.05, and in each temperature 500 moves are attempted.

Table 6.3 shows the results of our system for several specification of iteration period and the numbers of functional units and registers. As the results of differential equation benchmark, we obtained the minimum number of point-to-point interconnections for all instances. The last column shows the runtimes of our system. The number in parentheses denotes the runtime of branch-and-bound based assignment space exploration, which generate optimal solutions for data-path synthesis problem. From this column, we can see our system runs 17.5 seconds on average. In addition, our system runs about 140 times faster than the exact solution method on average. Figure 6.5 shows the resultant data-path, which contains 3 adders 2 multiplier, 8 registers, 4 multiplexers and 21 interconnections, and schedule under iteration period $T_r = 8$.

6.3.3 Fifth-Order Elliptic Wave Filter Example

Fifth-order elliptic wave filter, which contains 26 additions, 8 multiplications, 35 variables and 8 constants (Fig. 6.6), is used as an input instance of third data-path synthesis. We assume that an addition is performed on an adder in one control step, and a multiplication is performed on a (non-pipelined) multiplier in two control steps. In SA, the temperature

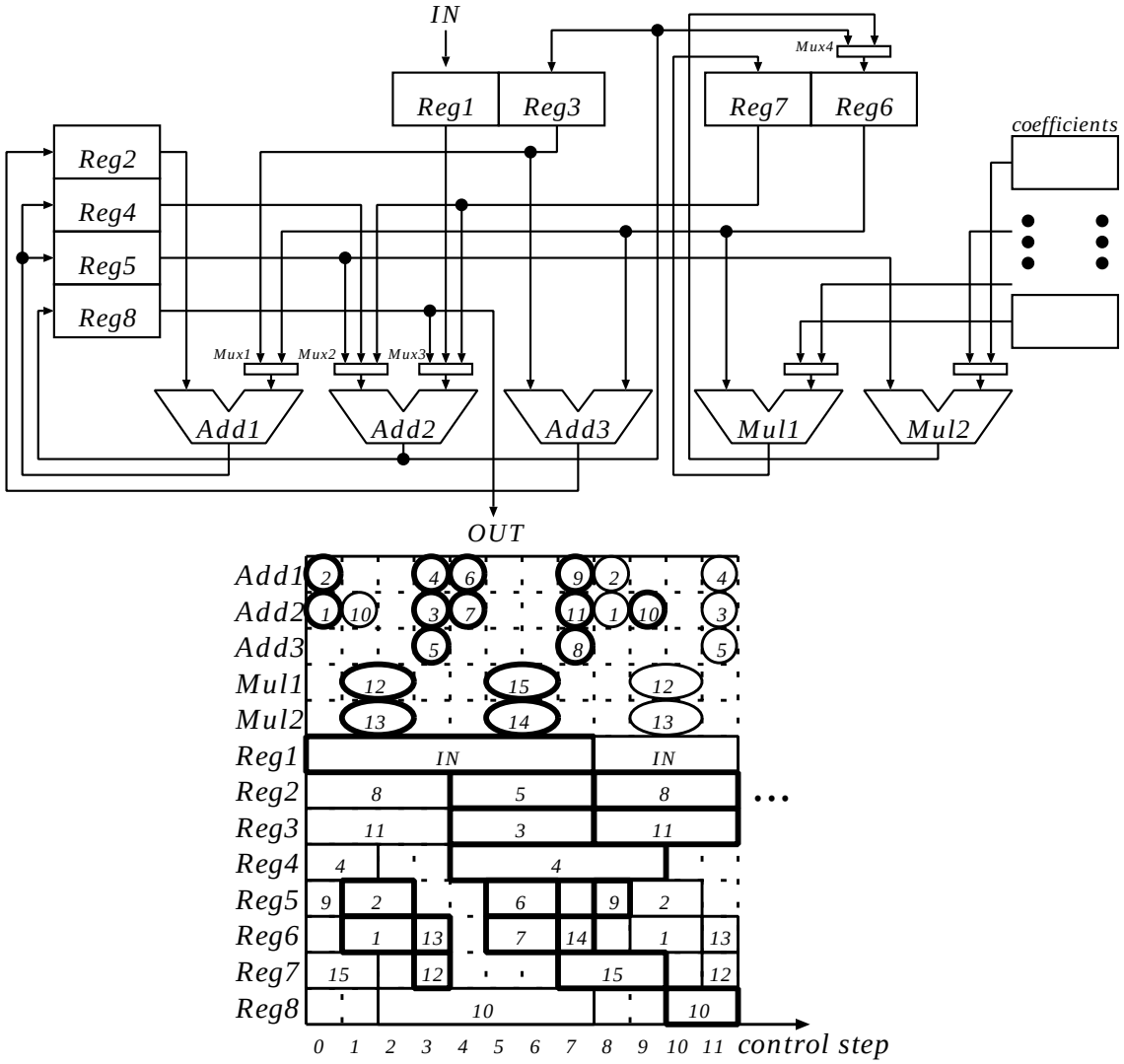

Figure 6.5: Data-path and schedule under $T_r = 8$

(T) which controls the probability of an uphill move $(\exp(-c/T))$ is scheduled with the form $T_i = 0.98 \times T_{i-1}$ from $T_0 = 15$ until it reaches 0.05, and in each temperature 5000 moves are attempted.

Table 6.4 shows the results of our system, together with results of other systems, for several specification of iteration period and the numbers of functional units and registers. Experimental results are grouped by iteration period. As we can see from this table, our system provide solutions with $12.5\% \sim 25\%$ reduced numbers of multiplexers, multiplexer's inputs and equivalent two-input single-output multiplexers compared to best numbers of them from other systems in first and second groups (i.e., iteration periods T_r are 21 and 19). Results in third and fourth groups (i.e., iteration periods T_r are 17 and 16) are the best solutions, with respect to not only the number of multiplexers but also the number of FUs under given T_r, that ever appeared in literatures. Figure 6.7 shows the resultant data-path and schedule under iteration period $T_r = 16$. The average runtime of our system is about 60 minutes for each input instance.

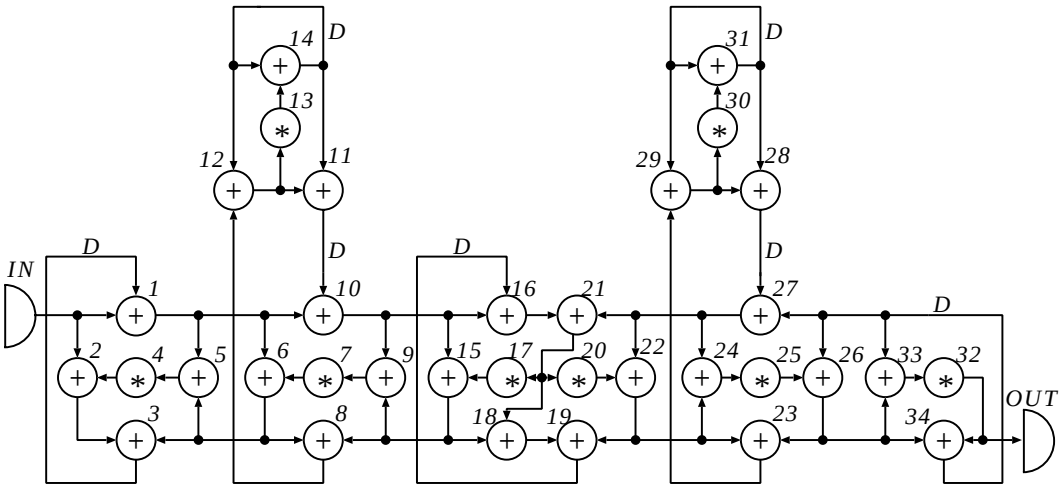

Figure 6.6: Fifth-order elliptic wave filter

Table 6.4: Experimental results for fifth-order elliptic wave filter

System	T_r	FU	R	Mx	Mi	ME	#CON	runtime[m]
Ours	21	$2+, 1\times$	11	7	21	14	29	77
SE	21	$2+, 1\times$	11	8	24	16	-	-
SPLICER	21	$2+, 1\times$	-	9	43	34	-	-
MABAL	21	$2+, 2\times$	11	13	43	30	-	-
Ours	19	$2+, 2\times$	10	9	24	15	31	64
SE	19	$2+, 2\times$	10	11	31	20	-	-
HAL	19	$2+, 2\times$	12	-	29	-	-	-
EMUCS	19	$2+, 2\times$	12	12	34	22	-	-
Ours	17	$2+, 2\times$	11	8	23	15	31	60
Ours	16	$3+, 2\times$	11	12	31	19	37	49

6.3.4 Dependence Graph with Conditional Branches Example

Finally, dependence graph with conditional branches in [35], which contains 13 additions, 4 multiplications, 15 variables, 2 fork nodes and 2 join nodes (see Fig 6.8), is used as an input instance of data-path synthesis. In SA, the temperature (T) which controls the probability of an uphill move ($\exp(-c/T)$) is scheduled with the form $T_i = 0.98 \times T_{i-1}$ from $T_0 = 15$ until it reaches 0.05, and in each temperature 1000 moves are attempted.

Table 6.5 shows the results of our system for several specification of iteration period and the numbers of functional units and registers. As we can see from this table, we obtained the minimum number of point-to-point interconnections for all instances. We found optimal solutions for all instances by using branch-and-bound based assignment space exploration. The number in parentheses denotes the runtime of branch-and-bound based assignment space exploration. Our system runs about 140 seconds on average. In addition, our system runs about 700 times faster than the exact solution method on average. Figure 6.9 shows the resultant data-path and schedule under iteration period

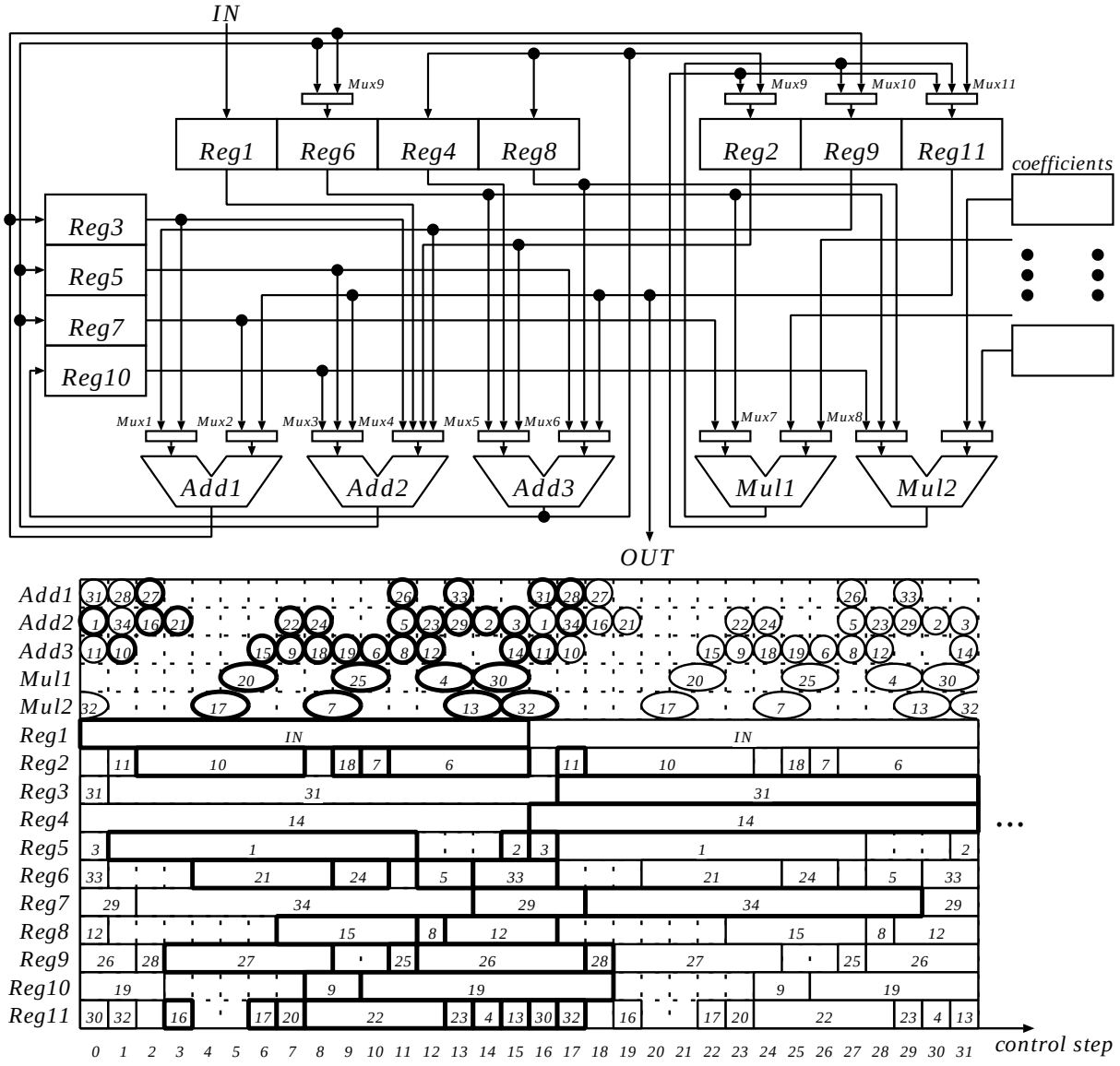

Figure 6.7: Data-path and schedule under $T_r = 16$

$T_r = 4$.

From the stochastic nature of SA and incompleteness of annealing schedule, the quality of a final solution tends to vary on each run. However, we have confirmed that most of solutions, which are obtained from multiple runs, are the minimum ones.

6.4 Conclusion

In this chapter, as an application of the proposed scheduling method, it is incorporated with Simulated-Annealing based exploration of assignment solution space. Experimental results for fifth-order elliptic wave filter show that our system provides data-paths with the 12.5% $\sim$ 25% reduced number of multiplexer's, multiplexer's input, and equivalent two-input single-output multiplexers compared to results of others systems. Also, for

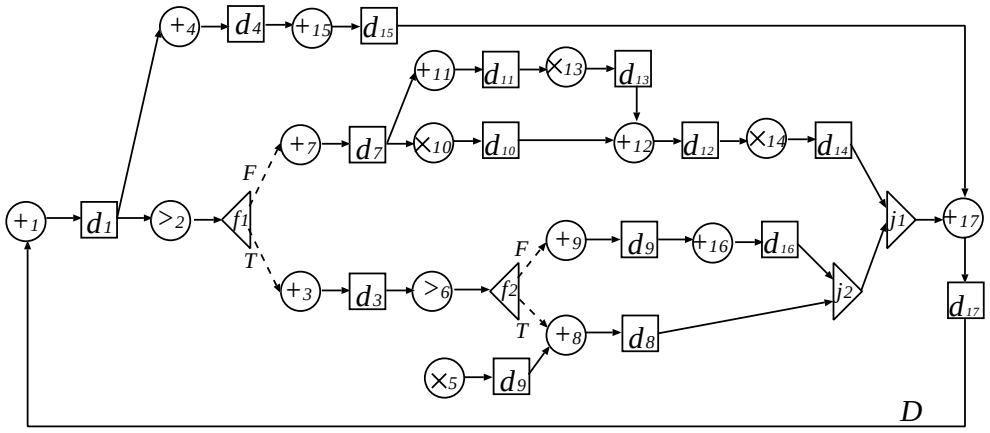

Figure 6.8: Dependence graph with conditional branches in [35].

Table 6.5: Experimental results.

System	T_r	FU	R	Mx	Mi	ME	#CON	runtime[s]
Ours	9	2 ALUs	3	0	0	0	7	177(40)
	8	2 ALUs	3	0	0	0	7	129(336)
	7	3 ALUs	3	0	0	0	8	114(1769)
	6	3 ALUs	3	1	2	1	9	101(8003)
	5	4 ALUs	4	3	6	3	12	80(248070)
	4	5 ALUs	4	2	4	2	13	71(244580)

dependence graph with conditional branches, syntheses of data-path with the minimum number of interconnections are demonstrated.

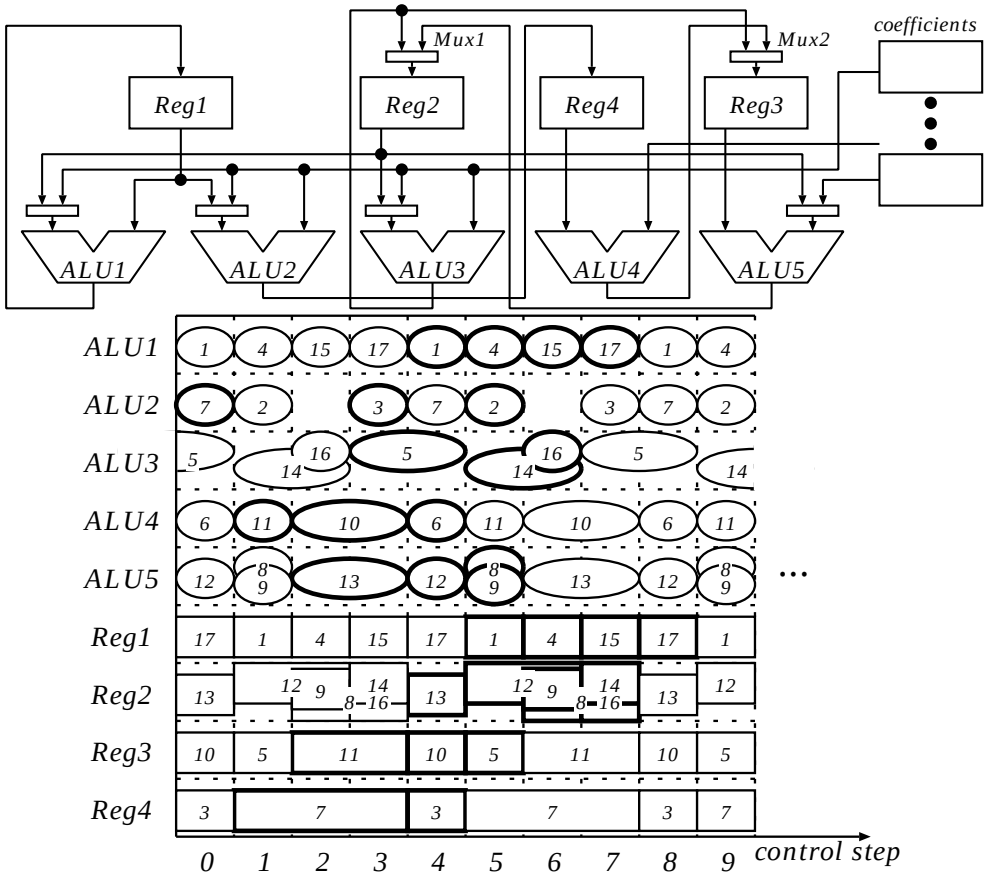

Figure 6.9: Data-path and schedule under $T_r = 4$

Chapter 7

Conclusion

Most of the conventional high-level synthesis aim mainly to minimize the number of control steps and the number of functional units, and they first decide the schedule and the number of functional units by resource constrained scheduling or time constrained scheduling, which are followed by resource assignment. However, the connectivity between modules is also an important metric for VLSIs for its connection with wire complexity, transmission delay, power consumption, testability, etc. In order to handle interconnection-related metric more accurately, simultaneous scheduling and assignment approach and assignment-driven approach are proposed. In those approaches, we often encounter a scheduling problem with specified resource assignment, which becomes the most importance core task.

In this thesis, we have proposed an approach using a parametric scheduling graph with disjunctive arcs generated from the specified assignment information (operation to functional unit assignments and data to register assignments) in loop pipeline scheduling problem. After transforming our assignment constrained scheduling problem into the problem to assign integer to disjunctive arcs and evaluating the possible range of variables to be fixed, we have derived a branch-and-bound method incorporated with successive refinement of parameter space. We have proposed also a heuristic method based on the sensitivity with respect to the iteration period to find the minimum iteration period. Experimental results show that our heuristic method can find solution quickly, while keeping the quality of solution.

To treat assignment constrained loop pipeline scheduling for a dependence graph with conditional branches, we have extended our variable disjunctive arc approach. Similarly, after transforming the scheduling problem into the problem to determine which disjunctive arcs are added, we have derived a branch-and-bound method and a simple heuristic method incorporated with successive refinement of parameter space.

As a result, for a dependence graph representing all control constructs (i.e., precedence constraints, loops and conditional branches), we have proposed assignment constrained scheduling method. As an application of the proposed scheduling method, it is incorporated with Simulated-Annealing based exploration of assignment solution space. Experimental results show that our system provides syntheses of data-path with the reduced number of interconnections.

For practical applications, we need to develop an efficient heuristics for the exploration of assignment solution space concurrent with assignment constrained pipeline scheduling. The incorporation of floorplanning into our assignment-centric approach is also an impor-

tant future work.

Bibliography

[1] P. Michel, U. Lauther, P. Duzy, "The synthesis approach to digital system design," Kluwer Academic, 1992.

[2] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.8, no.6, pp.661–679, 1989.

[3] R. Camposano, W. Rosenstiel, "Synthesizing circuits from behavioral descriptions," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.8, no.2, pp.171–180, 1989.

[4] N. Park and A.C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.7, no.3, pp.356–370, 1988.

[5] C.-T. Hwang, J.-H. Lee, Y.-C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.10, no.4, pp.464-475. 1991.

[6] I.-C. Park, C.-M. Kyung, "FAMOS: An efficient Scheduling Algorithm for High-Level Synthesis," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.10 No.12, pp.1437-1448, 1993.

[7] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.8, no.6, pp.661-679, 1989.

[8] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, J.L. van Meerbergen, A. van der Werf, "Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.14, no.8, pp.945-960, 1995.

[9] C.-T. Hwang, Y.-C. Hsu, Y.-L. Lin, "PLS: A Scheduler for Pipeline Synthesis," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.12, no.12, pp.1279-1286, 1993.

[10] T.-F. Lee, C.-H.Wu, Y.-L. Lin, D.D. Gajski, "A Transformation-Based Method for Loop Folding," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.13, no.4, pp.439-450, 1994.

[11] C.-Y. Wang, K.K. Parhi, "High-Level DSP Synthesis Using Concurrent Transformation, Scheduling, and Allocation," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.14, no.3, pp.274-294, 1995.

[12] N. Togawa, M. Yanagisawa, T. Ohtsuki, "A Fast Scheduling Algorithm Based on Gradual Time-Frame Reduction for Datapath Synthesis," IEICE Trans. Fundamentals, vol.E81-A, no.6, pp.1231-1241, June 1998.

[13] K. Kucukcakar, A.C. Parker, "Data path tradeoff using MABAL," Proc. 27th ACM/IEEE Design Automation Conference, pp.511-516, 1990.

[14] F.J. Kurdahi, A.C. Parker, "REAL: A program for register allocation," Proc. 24th ACM/IEEE Design Automation Conference, pp.511-516, 1987.

[15] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, Y.-C. Hsu, "Data path allocation based on bipartite weighted matching," Proc. 27th ACM/IEEE Design Automation Conference, pp.499-504, 1990.

[16] F.-S. Tsay, Y.-C Hsu, "Data path construction and refinement," In Digest of Technical Papers, ICCAD, pp.308–311, 1990.

[17] R.J. Cloutier, D.E. Thomas, "The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm," Proc. 27th ACM/IEEE Design Automation Conference, pp.71-76, 1990.

[18] K. Bazargan, S. Ogrenci, M. Sarrafzadeh, "Integrating Scheduling and Physical Design into a Coherent Compilation Cycle for Reconfigurable Computing Architectures," DAC2001, 2001.

[19] A. Doboli, R. Vemuri, "Integrated High-Level Synthesis and Power-Net Routing for Digital Design under Switching Noise Constraints," DAC2001, 2001.

[20] A.E. Eichenberger, E.S. Davidson, S.G. Abraham, "Optimum modulo schedules for minimum register requirements," Proc. Int. Conf. on Supercomputing, pp.31-40, 1995.

[21] R. Govindarajan, E.R. Altman, G.R. Gao, "Minimizing register requirements under resource-constrained rate-optimal software pipelining," Proc. 27th Int. Symp. on Microarchitecture, pp.85-94, 1994.

[22] B. Mesman, M. Strik, A.H. Timmer, J.L. van Meerbergen, J.A.G. Jess, "A Constraint Driven Approach to Loop Pipelining and Register Binding," Proc. IEEE DATE, pp.377–383, 1998.

[23] T. Watanabe, N. Ishiura, M. Yamaguchi, "A Code Generation Method for Datapath Oriented Codesign of Application Specific DSPs," The 13th Workshop on Circuit and Systems in Karuizawa, pp.539–544, 2000.

[24] D. Kim, J. Jung, S. Lee, J. Jeon, K. Choi, "Behavior-to-Placement RTL Synthesis with Performance-Driven Placement," Proc. ICCAD2001, 2001.

[25] K.Oohashi, M.Kaneko, S.Tayu, "Assignment-Space Exploration Approach to Concurrent Data-Path/Floorplan Synthesis," Proc. ICCD2000, pp.370-375, 2000.

[26] K. Ito, H. Kunieda, "VLSI System Compiler for Digital Signal Processing: Modulation and Synchronization," IEEE Trans. Circuits & Syst., vol.38, No.4, pp.422–433, 1991.

[27] T. Kim, K.-S. Chung, C.L. Liu, "A Stepwise Refinement Synthesis of Digital Systems for Testability Enhancement," IEICE Trans. Fundamentals, Vol.E82-A, No.6, pp.1070–1081, 1999.

[28] T. Yorozuya, K. Ohashi, M. Kaneko, "Assignment-Driven Loop Pipeline Scheduling and Its Application to Data-Path Synthesis," IEICE Trans. Fundamentals, Vol.E85-A, No.4, pp.819–826, April 2002.

[29] Tai A.Ly, Jack T.Mowchenko, "Applying Simulated Evolution to High Level Synthesis", IEEE Trans. Computer Aided Design of Integrated Circuits and Systems, vol.12, No.3, pp.389-408, 1993.

[30] B.M.Pangril, "Splicer: A Heuristic Approach to Connectivity Binding", Proc. 25th ACM/IEEE Design Automation Conference, pp.536-541,1988.

[31] A.C. Parker, J.T. Pizarro, and M. Mlinar, "MAHA: A program for datapath synthesis," Proc. 23rd DAC, pp461-446, 1986.

[32] H. Ishiwata, N. Togawa, M. Sato, T. Ohtsuki, "A Time-Constrained Scheduling Algorithm for CDFG with conditional branches," IEICE Technical Report, VLD95-133, 1996.

[33] T. Kim, N. Yonezawa, J.W.S. Liu, G.L.Liu, "A scheduling algorithm for conditional resource sharing-A hierarchical reduction approach," IEEE Trans., CAD, Vol.13, No.4, pp425-438, 1994.

[34] A. Yamada, T. Yamazaki, N. Ishiura, I. Shirakawa, T. Kambe, "Datapath scheduling for behavioral description with conditional branches," IEICE Trans., Vol. E77-A, No. 12, pp1999-2009, 1994.

[35] K. Ito, T. Kawasaki, "An Overlapped Scheduling Method for an Iterative Processing Algorithm with Conditional Operations," IEICE Trans., Fundamentals, Vol. E81-A, No.3, pp429-438, 1998.

Publications

[1] Koji Ohashi, Mineo Kaneko, "Loop Pipeline Scheduling for Assignment Constrained Iteration Period Minimization", IEICE Trans. Fundamentals, 2002. (submitted)

[2] Koji Ohashi, Mineo Kaneko, "Heuristic Assignment-Driven Scheduling for Data-Path Synthesis", Proc. IEEE International Symposium on Circuits and Systems, pp.702-706, 2002.5.

[3] Toshiyuki Yorozuya, Koji Ohashi, Mineo Kaneko, "Assignment-Driven Loop Pipeline Scheduling and Its Application to Data-Path Synthesis", IEICE Trans. Fundamentals, Vol.E85-A, No.4, pp.819-826, 2002.4.

[4] Koji Ohashi, Mineo Kaneko, "Minimization of Iteration Period in Assignment-Constrained schedule for Data-Path Synthesis", The 15th Workshop on Circuits and Systems in Karuizawa, pp.441-446, 2002.4.

[5] Koji Ohashi, Mineo Kaneko, "Assignment-Driven Heuristic Scheduling Based on Sensitivity to Iteration Period for Datapath Synthesis", Technical Report of IEICE, VLD2001-105, Vol.101, NO.467, pp.97-102, 2001.11.

[6] Toshiyuki Yorozuya, Koji Ohashi, Mineo Kaneko, "Assignment-Driven Loop Pipelining and Its Application to High Level Synthesis", Proc. of the Workshop on Synthesis And System Integration of MIxed Technologies, pp.135-142, 2001.10.

[7] Toshiyuki Yorozuya, Koji Ohashi, Mineo Kaneko, "Assignment-Driven Loop Pipelining and Its Application to High Level Synthesis", The 14th Workshop on Circuits and Systems in Karuizawa, pp.393-398, 2001.4. (in Japanese)

[8] Toshiyuki Yorozuya, Koji Ohashi, Mineo Kaneko, "Assignment-Driven Loop Pipelining and Its Application to High Level Synthesis", Technical Report of IEICE, VLD2000-141, Vol.100, NO.646, pp.43-48, 2001.3. (in Japanese)

[9] Koji Ohashi, Mineo Kaneko, Satoshi Tayu, "Assignment-Space Exploration Approach to Concurrent Data-Path/Floorplan Synthesis", Proc. International Conference on Computer Design, pp.370-375, 2000.9.

[10] Koji Ohashi, Mineo Kaneko, Satoshi Tayu, "Assignment-Drive Approach to Data-Path Synthesis Incorporated with Floorplanning", The 13th Workshop on Circuits and Systems in Karuizawa, pp.251-256, 2000.4.

[11] Koji Ohashi, Mineo Kaneko, Satoshi Tayu, "Assignment-Drive Approach to Data-Path Synthesis Incorporated with Floorplanning", Technical Report of IEICE, VLD99-117, Vol.99, NO.659, pp.1-8, 2000.3. (in Japanese)