

Chapter 5

Various Models

5.1 Introduction

The last chapter is concerned with retiming and unfolding on unit-time DFGs. Pipelined schedules for general-time DFGs are produced by allowing node splitting. Based on the same concept, pipelined schedules with fractional starting times can be generated by allowing a general-time node to be split into fractional parts. Since such a fractional retiming is hard to interpret, this chapter develops an algorithm to generate integral or fractional schedules without generating retiming functions as an intermediate step.

There has been much work on finding the rate-optimal static schedule with the minimum unfolding factor under different assumptions [38, 78, 65]. This chapter tries to clarify this problem by using two kinds of timing models and two types of implementations. We derive the minimum rate-optimal unfolding factor for each of the four combinations, and design an efficient unified polynomial-time algorithm. Moreover, we will prove general theorems for static scheduling where the rate-optimal scheduling is just a special case. From our results we know the relationship between an unfolding factor and its corresponding minimum iteration period for static scheduling and how to get such a schedule.

Two timing models are considered. If a scheduled computation can start at any rational number, we say this model is a *fractional model*; otherwise is an *integral-grid*

model. Basically the fractional model is much simpler than the integral-grid model, because it does not have the integral constraints. Therefore we will first focus on the integral-grid model, and then we can easily apply our results to the fractional model. Two design styles are studied: *pipelined* and *non-pipelined* processors, as introduced in Section 2.7. The software-pipelining problem solved in Section 4.4 uses pipelined design and the integral model. We show how to find a rate-optimal static schedule with the minimum unfolding factor under pipelined implementation and non-pipelined implementation for the two timing models.

The next section introduces the two timing models and some properties of the two design styles. An efficient polynomial-time algorithm is presented in Section 5.3, which can be applied to all four combinations of models. This algorithm is particularly efficient because it works on the original *DFG* instead of the large unfolded *DFG*. In Section 5.4 we show our results on the integral models for pipelined and non-pipelined designs.

For the pipelined design and integral timing model, the results from Chapter 4 directly lead to the minimum rate-optimal unfolding factors. By using the techniques developed for the pipelined design, we are able to prove that the minimum rate-optimal unfolding factor for non-pipelined design is $\left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil \cdot \rho$, where $\max_v t(v)$ is the maximum computation time over all nodes, and σ and ρ are the numerator and the denominator of the irreducible form of the iteration bound $B(G)$. From this result, we conclude that the minimum unfolding factor shown in [39] is incorrect. Section 5.5 describes the results on the fractional model. We show the minimum rate-optimal unfolding factor is just 1 for pipelined design and is ρ for non-pipelined design. In addition to providing minimum rate-optimal unfolding factors and algorithms, the contribution of this work is to give a general technique which can be generalized to other scheduling problems under different design situations, and can be easily combined with other structural transformation techniques.

Many results for unit-time DFGs also hold for scheduling general-time DFGs under the integral timing model and pipelined implementation. For clarity, we recast them in this chapter so that the reader can compare them with the results of other models.

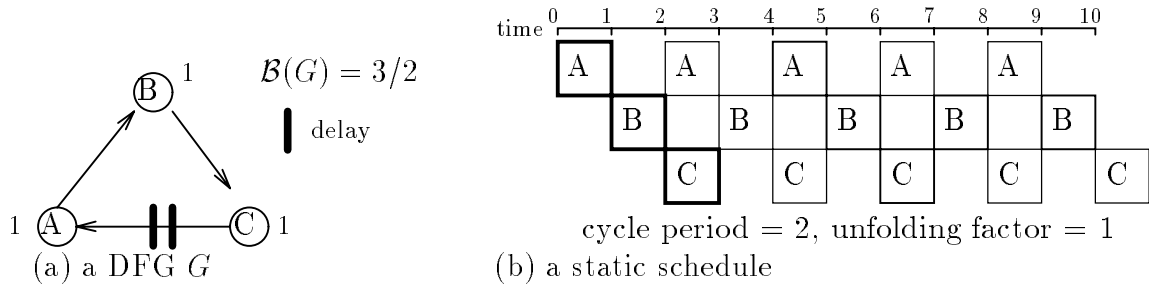


Figure 5.1: A simple DFG

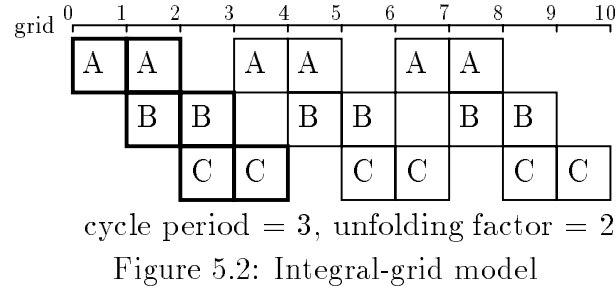
5.2 Various Models and Design Styles

Consider a data-flow graph $G = (V, E, d, t)$ as defined in Definition 2.1. A *schedule* is represented by a function $S : V \times N \rightarrow R$. The starting time of node v in the i -th iteration is $S(v, i)$. A schedule is *legal* if for every edge $u \xrightarrow{e} v$ and iteration i , we have $S(u, i) + t(u) \leq S(v, i + d(e))$. A schedule is said to have an *unfolding factor* f and a *cycle period* c if $S(v, i + f) = S(v, i) + c$ for every v in V and iteration i . Thus, such a schedule can be represented by the partial schedule of the first f iterations. A new instance of this partial schedule of f iterations can be initiated for every interval of length c to form a legal complete schedule. Moreover, if in every instantiation, an operation of the partial schedule is assigned to the same processor, this type of schedule is called a *static schedule* or a *processor-static schedule*.

If a schedule does not have a repeating pattern, there does not exist an unfolding factor and a cycle period for this schedule. We only consider schedules with a repeating pattern. Note that in this chapter we organize the repeating pattern of a schedule as the ordinary model, as depicted in Figure 2.7. There is no distinction of the prologue and epilogue.

Under different timing or architecture models, there are several ways of implementing a static schedule. We classify the implementation styles below, and present one unified algorithm to solve the scheduling problems under different models in Section 5.3. The minimum rate-optimal unfolding factors under different models are also characterized in Sections 5.4 and 5.5.

The measurement of computation time and cycle period is in terms of a pre-defined *time unit*, which may be a machine cycle or a clock cycle under different models.

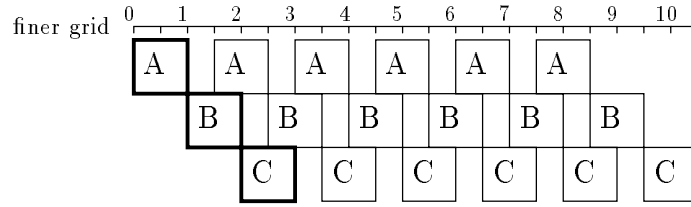


Without loss of generality, we assume that every node has an integral computation time. There are two models for the timing style of scheduling.

Integral-grid model We imagine there is an integral grid (time unit) for the schedule. An operation can only be issued in the beginning of a time unit. Under this model, a schedule has an integral cycle period.

Fractional Model (Gridless Model) The starting time of a node can be any fractional number, i.e. the schedule can have an infinitely fine grid, and an operation can be issued at any time. Under this model, a schedule may have a fractional cycle period. Actually, we will show that a grid of size $1/\rho$ gives schedules as good as gridless schedules, where ρ is the denominator of $\mathcal{B}(G)$ in its irreducible form.

Consider the DFG in Figure 5.1 with iteration bound $\mathcal{B}(G) = 3/2$. Figures 5.2 and 5.3 show a schedule under the integral-grid model and a schedule under the fractional model, respectively, where the first instances of the static schedules are highlighted by boxes with thicker boundaries. These two static schedules have the same iteration period, $3/2$. In order to achieve rate-optimality, the static schedule under the integral-grid model consists of 2 iterations, while that under the fractional model contains one iteration. Usually, a static schedule is stored as a program code where each instruction consists a set of nodes to be issued. An instruction is executed at each clock tick. The costs of storing a static schedule can be measured by the number of instructions in the program code. Under the fractional model as shown, the clock ticks at every $1/2$ time unit. Thus, the cost of storing the integral-grid and the fractional models are the same.



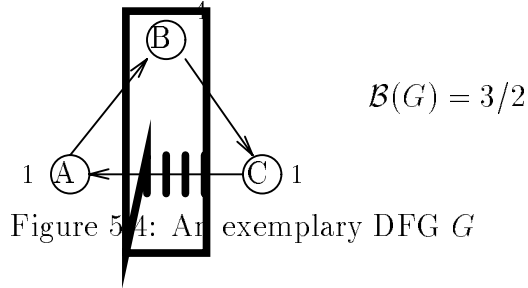
cycle period = 1.5, unfolding factor = 1

Figure 5.3: Fractional Grid Model

From Chapter 2, we know that a pipelined schedule of a general-time DFG corresponds to a retiming on the general-time DFG where nodes can be split into integral parts. Under the fractional timing model, a pipelined schedule of a general-time DFG also corresponds to a retiming on the general-time DFG where nodes can be split into fractional parts. For example, in Figure 5.3 the repeating part of the schedule, marked by the box with dashed boundary, corresponds to a retiming on the DFG in Figure 5.1-(a) such that node B is split into two half-unit parts.

Under every timing model, there are the two design styles, pipelined and non-pipelined design, as described in Section 2.7. For a static schedule with the unfolding factor f under the non-pipelined design, the $(i + f)$ -th copy of a node cannot start execution before the i -th copy has finished execution. Thus, a schedule with unfolding factor f can be implemented by using non-pipelined processors if for every v in V and positive integer i , we have $S(v, i + f) \geq S(v, i) + t(v)$. While, for the pipelined design there is no such restriction.

Next, we derive the condition for a static schedule to be implemented under non-pipelined design. All the arguments are true for both the integral and fractional timing models. A new copy of a node assigned to the same processor starts execution every c time units in a static schedule of cycle period c . Thus, the cycle period must be larger than the computation time of a node. In order to improve the execution rate, one can execute several copies of a node with long computation time simultaneously. Hence, the schedule will have a larger cycle period, but the iteration period can be reduced. The following shows that as long as the cycle period of a schedule is no less than $\max_v t(v)$, it can be implemented by a static schedule under the non-pipelined



design.

Lemma 5.2.1 *Given a schedule S with cycle period $c \geq \max_v t(v)$, this schedule can be realized as a static schedule under the non-pipelined design.*

Proof: Assume that S has unfolding factor f . We can simply assign every copy of every node to a distinct processor. Since $t(v) \leq c$ for every v , we have $S(v, i + f) = S(v, i) + c \geq S(v, i) + t(v)$. Therefore, S is a static schedule under the non-pipelined design. \square

Actually, the inequality $c \geq \max_v t(v)$ is also a necessary condition for a static schedule to be implemented under non-pipelined design.

Lemma 5.2.2 *Any static schedule S under non-pipelined design has a cycle period c where $c \geq \max_v t(v)$.*

Proof: Assume that S has unfolding factor f . For every node v and iteration i , we have $S(v, i + f) \geq S(v, i) + t(v)$. Since S is repeated for every period of c , we know $S(v, i + f) = S(v, i) + c$. Thus, c must be no less than the computation time of a node. Therefore, we have $c \geq \max_v t(v)$. \square

The above lemmas give the necessary and sufficient condition for a static schedule to be implemented under the non-pipelined design.

Theorem 5.1 *Let G be a DFG, and S be a legal schedule with cycle period c and unfolding factor f . The static schedule S can be implemented under the non-pipelined design if and only if $c \geq \max_v t(v)$.*

Proof: This theorem follows from Lemma 5.2.1 and Lemma 5.2.2. \square

Thus, there are four combinations of implementations of static schedules. The following example, which is used throughout the rest of this chapter, shows that the

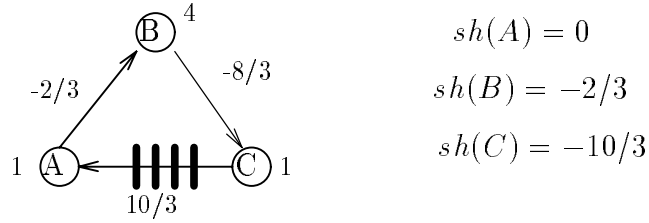


Figure 5.5: The scheduling graph G^s with $c/f = 3/2$

minimum rate-optimal unfolding factor obtained may be different under these four combinations. The exemplary DFG is shown in Figure 5.4, and its iteration bound $\mathcal{B}(G)$ is $3/2$. The minimum rate-optimal unfolding factors derived from theorems in this chapter for these four combinations are as follows. The derivations of unfolding factors and the corresponding schedules will be explained in this chapter.

Min Rate-optimal Unfolding Factor	Timing Models	
	Fractional	Integral
Pipelined Design	1	2
Nonpipelined Design	3	4

5.3 Unified Scheduling Algorithm

We present an algorithm to find a schedule for given cycle period c and unfolding factor f under the fractional and integral models. Throughout this section, no constraint is put on the cycle period c and the maximum computation time $\max_v t(v)$. Therefore, the pipelined design is considered. Nevertheless, the same algorithm can be used for the non-pipelined design as long as Theorem 5.1 is satisfied.

For a DFG $G = (V, E, d, t)$ and given c and f , we define a modified graph with different weights on edges: scheduling graph $G^s = (V, E, w)$ where $w(e) = d(e) - t(u) \cdot f/c$ for every edge $u \xrightarrow{e} v$ in E . This modified graph was used in Section 4.4 to compute legal retimings when modeling general-time DFGs as unit-time ones. Here, we use the modified graph to compute legal schedules for different models.

The next lemma relates the lower bound $\mathcal{B}(G)$ to the existence of negative-weight cycles in G^s .

Lemma 5.3.1 *Let G be a general-time DFG, c a (fractional) cycle period, and f an unfolding factor. The scheduling graph G^s contains no cycles having negative weights if and only if $c/f \geq \mathcal{B}(G)$.*

Proof: Although the cycle period c can be fractional and G is a general-time DFG, the proof of this lemma is analogous to that of Lemma 4.2.2 for a unit-time DFG. \square

Assume that there is no negative-weight cycle in the scheduling graph G^s . We add a node v_0 and directed edges from v_0 to every other node with zero weight in G^s . Let $sh(v)$ be the length of the shortest path from v_0 to v in the scheduling graph G^s . The values of $sh(v)$ can be computed by any single-source shortest path algorithm. It is easy to observe that for every node v we have $sh(v) \leq 0$, and there exists a node u in V such that $sh(u) = 0$.

The following theorem obtains legal fractional and integral schedules from the scheduling graph.

Theorem 5.2 *Let G be a general-time DFG, c a cycle period, f an unfolding factor. Assuming that there is no negative-weight cycle in the corresponding scheduling graph G^s , let $sh(v)$ be the length of the shortest path from v_0 to v .*

(a) $S^f(v, i) = -sh(v) \cdot c/f + i \cdot c/f$ for every v and i is a legal schedule under the fractional model.

(b) $S^i(v, i) = \lceil -sh(v) \cdot c/f + i \cdot c/f \rceil$ for every v and i is a legal schedule under the integral grid model.

Proof: Consider any edge $u \xrightarrow{e} v$ and iteration i . We claim that $S^f(u, i) + t(u) \leq S^f(v, i + d(e))$. For every edge $u \xrightarrow{e} v$ in G^s , we have $sh(u) + d(e) - t(u) \cdot f/c \geq sh(v)$. The following is derived.

$$\begin{aligned} -sh(u) \cdot c/f + t(u) &\leq -sh(v) \cdot c/f + d(e) \cdot c/f \\ -sh(u) \cdot c/f + i \cdot c/f + t(u) &\leq -sh(v) \cdot c/f + i \cdot c/f + d(e) \cdot c/f \end{aligned}$$

Thus, we know $S^f(u, i) + t(u) \leq S^f(v, i + d(e))$, and S^f is a legal schedule.

Since $t(u)$ is integral, we have

$$\lceil -sh(u) \cdot c/f + i \cdot c/f \rceil + t(u) \leq \lceil -sh(v) \cdot c/f + i \cdot c/f + d(e) \cdot c/f \rceil.$$

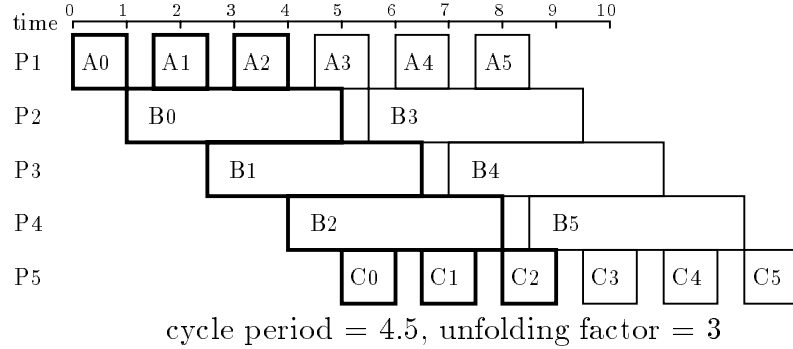


Figure 5.6: Fractional model and nonpipelined implementation

Therefore, we know $S^i(u, i) + t(u) \leq S^i(v, i + d(e))$, and S^i is a legal schedule. \square

Note that another integral schedule can be obtained by substituting the *floor* operation for the *ceiling* operation. Since there exists a node with $sh(v) = 0$, the schedule S^f and S^i both start from time 0. Consider the DFG in Figure 5.4 as an example. The iteration bound $\mathcal{B}(G)$ of this DFG is $3/2$. The scheduling graph G^s is depicted in Figure 5.5¹, where the number beside an edge e is weight $w(e)$. We first compute all the $sh(v)$ for every v in V , as shown in Figure 5.5. For $c = 4.5$ and $f = 3$, we obtain the fractional schedule in Figure 5.6, where $S^f(A, 0) = 0$, $S^f(B, 0) = 1$, and $S^f(C, 0) = 5$. And, $S^f(A, 1) = 0 + c/f = 1.5$, $S^f(B, 1) = 2.5$, $S^f(C, 1) = 6.5$.

The following lemma shows that the schedules S^f and S^i have unfolding factor f and cycle period c .

Lemma 5.3.2 *The schedules S^f and S^i have the following properties: For every node v and every positive integer i ,*

- (a) $S^f(v, i + f) = S^f(v, i) + c$.
- (b) $S^i(v, i + f) = S^i(v, i) + c$.

Proof: For the fractional schedule S^f , we have

$$\begin{aligned} S^f(v, i + f) &= -sh(v) \cdot c/f + (i + f) \cdot c/f \\ &= -sh(v) \cdot c/f + i \cdot c/f + c \\ &= S^f(v, i) + c. \end{aligned}$$

¹Notice that the scheduling graph G^s is the same for finding schedules of the same rate, i.e. the same c/f ratio. All rate-optimal schedules found in this chapter for the exemplary DFG will use the same scheduling graph in Figure 5.5.

For the integral schedule S^i , we have

$$\begin{aligned}
 S^i(v, i + f) &= \lceil -sh(v) \cdot c/f + (i + f) \cdot c/f \rceil \\
 &= \lceil -sh(v) \cdot c/f + i \cdot c/f + c \rceil \\
 &= \lceil -sh(v) \cdot c/f + i \cdot c/f \rceil + c \\
 &= S^i(v, i) + c.
 \end{aligned}$$

□

The entire scheduling algorithm is shown in Figure 5.7. The single-source shortest path algorithm introduced in [75] is adopted. If there is a cycle with negative weight in graph G^s , the algorithm will detect it; otherwise, the algorithm will compute the shortest paths.

It takes time $O(|V| |E|)$ to compute $sh(v)$, and time $O(f |V|)$ to generate a schedule with unfolding factor f . The time complexity of the scheduling algorithm is $O((|E| + f) |V|)$. This algorithm is particularly efficient because our scheduling algorithm for finding schedules with unfolding factor f works on the scheduling graph G^s , the size of which is the same as the original DFG. We do not unfold the DFGs while computing schedules.

The following theorem provides the necessary and sufficient condition for the existence of a schedule with cycle period c and unfolding factor f .

Theorem 5.3 *Let G be a general-time DFG, c a (fractional) cycle period, and f an unfolding factor. $c/f \geq \mathcal{B}(G)$ if and only if there exists a legal fractional schedule with unfolding factor f and cycle period c .*

Proof: Although the cycle period c can be fractional and G is a general-time DFG, the proof of this lemma is analogous to that of Lemma 4.2.3 for a unit-time DFG.

Since $\mathcal{B}(G)$ is the lower bound of iteration period, we know every legal schedule must satisfy the inequality $c/f \geq \mathcal{B}(G)$. Therefore, the if part is proved. Assume that $c/f \geq \mathcal{B}(G)$. From Lemma 5.3.1, we know there is no cycles having negative weights in G^s . Thus, we can find the value $sh(v)$ for every node v by the shortest path algorithm. From these values, we can generate a legal fractional schedule with

Scheduling Algorithm**Input :** a DFG G , an unfolding factor f and a cycle period c .**Output :** A schedule for the first f iterations such that every f iterations can start in an interval of length c .**begin** /* Q is a first-in first-out queue. $pass$ is used to prevent the algorithm being trapped in negative cycles. */

for every node $v \in V$ **do begin**
 $sh(v) \leftarrow 0$; PUSH v to Q ;

end

$pass \leftarrow 0$;

$last \leftarrow$ the last element in Q ;

while Q is not empty **and** $pass < |V|$ **do begin**

 Pop v from Q

for every edge $e = (v, w) \in E$ **do begin**

$sh(w) \leftarrow \min\{sh(v) + d(e) - t(v) \cdot f/c, sh(w)\}$

if w is not in Q **then** PUSH w to Q ;

end

if $v = last$ **then begin**

$pass \leftarrow pass + 1$;

$last \leftarrow$ the last element in Q ;

end

end

if Q is empty **then**

for $i = 0$ **to** $f - 1$ **do**

for every $v \in V$ **do**

$S^f(v, i) \leftarrow -sh(v) \cdot c/f + i \cdot c/f$

$S^i(v, i) \leftarrow \lceil -sh(v) \cdot c/f + i \cdot c/f \rceil$

else "No such schedule exists."

end

Figure 5.7: Scheduling Algorithm

unfolding factor f and cycle period c from Lemma 5.2-(a). Therefore, the only-if part is proved. \square

By using Lemma 5.2-(b) as a legal integral schedule, similar result can be obtained for the integral model.

Theorem 5.4 *Let G be a general-time DFG, c a (integral) cycle period, and f an unfolding factor. $c/f \geq \mathcal{B}(G)$ if and only if there exists a legal integral schedule with unfolding factor f and cycle period c .*

5.4 Integral-Grid Model

All legal schedules can be implemented as static schedules by using pipelined processors. In Chapter 4 we modeled a general-time DFG as a unit-time DFG to produce schedules for a pipelined design. Analogously, the minimum rate-optimal unfolding factors for general-time DFGs under the pipelined implementation can be derived.

Theorem 5.5 *Let G be a general-time DFG and ρ the denominator of $\mathcal{B}(G)$ in its irreducible form. The minimum rate-optimal unfolding factor under the integral model and pipelined implementation is ρ .*

Consider the example in Figure 5.4. Since $\mathcal{B}(G)$ is $3/2$, we know ρ is 2. Figure 5.8 shows a rate-optimal schedule with unfolding factor 2. From Theorem 5.5, we know that 2 is the minimum rate-optimal unfolding factor under the integral model. Figure 5.9 shows the corresponding static schedule.

Suppose we want to find a rate-optimal schedule. We can first find the iteration bound $\mathcal{B}(G)$ in its irreducible form. The iteration bound for a DFG can be found in time $O(|V||E|\log|V|)$ [46]. Then, we can use the scheduling algorithm to find a schedule with unfolding factor f and cycle period c in time $O((|E| + f)|V|)$.

It is possible that this minimum rate-optimal unfolding factor is still too large for the design requirement in some cases. The designer may want to trade off between an unfolding factor and the corresponding minimum iteration period. For an unfolding factor f , the minimum feasible c from the inequality $c/f \geq \mathcal{B}(G)$ is $MCP_p(f) = \lceil f \cdot \mathcal{B}(G) \rceil$. Thus, all pairs of f and $MCP_p(f)$ are easily generated, and the designer

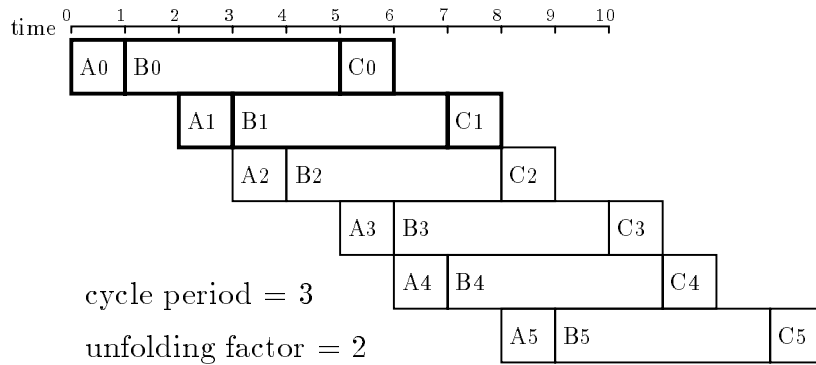


Figure 5.8: Integral model and pipelined implementation: time table

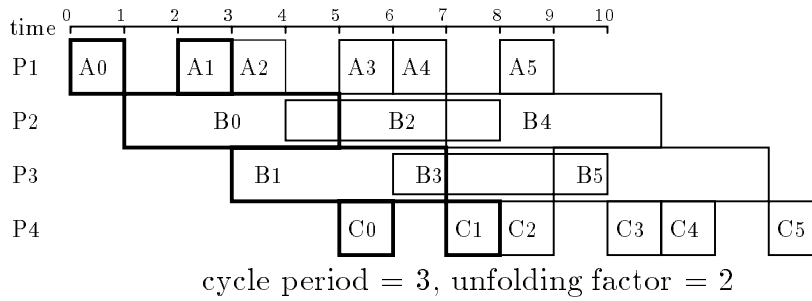


Figure 5.9: Integral model and pipelined implementation: static schedule

may choose the suitable pair according to the design requirement. We summarize this in the following corollary.

Corollary 5.4.1 *Let G be a DFG and f a given unfolding factor. The pipelined schedule using unfolding factor f has the minimum cycle period*

$$MCP_p(f) = \lceil f \cdot \mathcal{B}(G) \rceil.$$

The larger the unfolding factor is, the larger will be the memory used to store the static schedule. Under design requirements, we usually have a limit on the amount of memory available. The following corollary finds the unfolding factor and cycle period which gives the minimum iteration period, i.e. the average computation time per iteration.

Corollary 5.4.2 *Let G be a DFG and F the maximum unfolding factor feasible under resource requirements. The pipelined schedule with minimum iteration period using*

an unfolding factor not exceeding F has unfolding factor \hat{f} and cycle period \hat{c} , where \hat{c}/\hat{f} is the irreducible form of the value

$$\min_{1 \leq f \leq F} \frac{[f \cdot \mathcal{B}(G)]}{f}.$$

For example, for a DFG with $\mathcal{B}(G) = 15/11$, the minimum rate-optimal unfolding factor is 11. But, the maximum feasible unfolding factor under design requirements is only 6. From the above corollary, we know the minimum iteration period is obtained from $\hat{f} = 5$ and $\hat{c} = 7$.

Next we discuss the static schedules for non-pipelined processors under the integral model. The following theorem combines the requirements for rate-optimality (Theorem 5.4) and non-pipelined design (Theorem 5.1) to derive the rate-optimal minimum unfolding factor.

Theorem 5.6 *Let σ and ρ be the numerator and denominator of the iteration bound $\mathcal{B}(G)$ in its irreducible form. Set*

$$\hat{f} = \left\lceil \frac{\frac{\max_v t(v)}{\mathcal{B}(G)}}{\rho} \right\rceil \cdot \rho = \left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil \cdot \rho.$$

(a) *There exists a rate-optimal static schedule with unfolding factor \hat{f} under non-pipelined design.*

(b) *\hat{f} is the minimum rate-optimal unfolding factor to deliver a static schedule under non-pipelined design.*

Proof: In a rate-optimal schedule with unfolding factor \hat{f} , the cycle period \hat{c} is $\hat{f} \cdot \mathcal{B}(G)$. From Theorem 5.4, we know that there exists a legal schedule with cycle period \hat{c} and unfolding factor \hat{f} . From the definition of \hat{f} , we know that $\hat{f} \geq \max_v t(v)/\mathcal{B}(G)$. Thus, we have $\hat{c} \geq \max_v t(v)$. From Theorem 5.1, we know that this rate-optimal schedule is a static schedule under non-pipelined design.

Next, we show that \hat{f} is the minimum rate-optimal unfolding factor. For any rate-optimal unfolding factor f smaller than \hat{f} , the cycle period c is $f \cdot \mathcal{B}(G)$. We consider the following two cases:

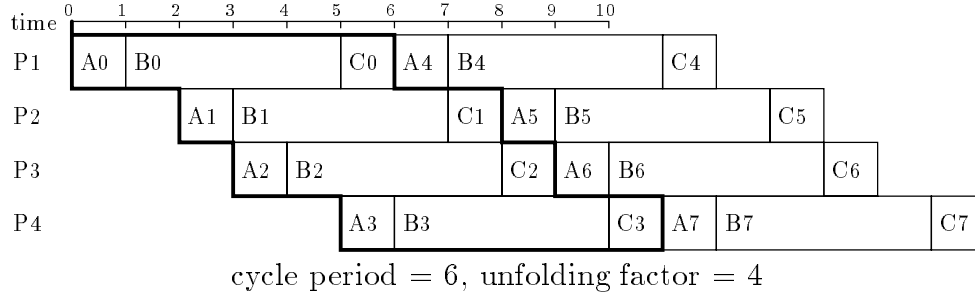


Figure 5.10: Integral model and non-pipelined implementation

- $f < \frac{\max_v t(v)}{\mathcal{B}(G)}$.

Since the cycle period c is smaller than $\max_v t(v)$, From Theorem 5.1, there does not exist any non-pipelined schedule for unfolding factor f .

- $\frac{\max_v t(v)}{\mathcal{B}(G)} \leq f < \hat{f}$.

A valid schedule with unfolding factor f under the integral model must have an integral cycle period c . Since $c = f \cdot \sigma / \rho$ is integral, there exists an integer k such that $f = k \rho$. We have

$$\frac{\max_v t(v)}{\sigma / \rho} \leq k \rho < \left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil \cdot \rho, \text{ or}$$

$$\frac{\max_v t(v)}{\sigma} \leq k < \left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil$$

Thus, we know that in this range of f , no integral k exists such that the cycle period c is integral.

Therefore, \hat{f} is the minimum unfolding factor to deliver a rate-optimal static schedule under non-pipelined implementation. \square

Our scheduling algorithm can find a schedule S^i . From Theorem 5.1, this schedule is a static schedule under non-pipelined design. Consider the example in Figure 5.4. The iteration bound $\mathcal{B}(G)$ is $3/2$, and $\max_v t(v)$ is 4. From Theorem 5.6, we derive the minimum rate-optimal unfolding factor as

$$\hat{f} = \left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil \cdot \rho = \left\lceil \frac{4}{3} \right\rceil \cdot 2 = 4.$$

From the scheduling algorithm in Figure 5.7, we obtain $S^i(A, 0) = 0$, $S^i(B, 0) = 1$, $S^i(C, 0) = 5$, $S^i(A, 1) = \lceil 3/2 \rceil = 2$, $S^i(B, 1) = \lceil 1 + 3/2 \rceil = 3$, and $S^i(C, 1) = \lceil 5 + 3/2 \rceil = 7$. Figure 5.10 shows the rate-optimal schedule with unfolding factor 4. This schedule is static under non-pipelined design, where the operation B5 starts after B1 is finished.

In practice, the minimum unfolding factor to achieve rate-optimal non-pipelined design may be too large to be realistic. It is necessary to find a schedule of the highest rate achievable under practical design requirements.

Lemma 5.4.3 *Let f be a given unfolding factor. The static schedule under non-pipelined design using unfolding factor f has the minimum cycle period $MCP_n(f)$, where*

$$MCP_n(f) = \begin{cases} \max_v t(v) & \text{if } \max_v t(v) \geq f \cdot \mathcal{B}(G), \\ \lceil f \cdot \mathcal{B}(G) \rceil & \text{otherwise.} \end{cases}$$

Proof: In order to have a non-pipelined implementation of cycle period c and unfolding factor f , the following inequalities have to be satisfied: $c/f \geq \mathcal{B}(G)$ (Theorem 5.4) and $c \geq \max_v t(v)$ (Theorem 5.1). It is easy to show that $MCP_n(f)$ is the minimum cycle period satisfying these inequalities in both cases. \square

For example, given a DFG with $\mathcal{B}(G) = 15/11$, $f = 5$, if $\max_v t(v) = 8$, we have $MCP_n(5)$ is 8; if $\max_v t(v) = 6$, we have $MCP_n(5)$ is 7.

Theorem 5.7 *Let F be the maximum unfolding factor feasible under resource constraints. The non-pipelined schedule with the minimum iteration period using an unfolding factor not exceeding F has minimum unfolding factor \hat{f} and cycle period \hat{c} . The values of \hat{f} and \hat{c} can be found as follows.*

- (a) *If $\max_v t(v) \geq F \cdot \mathcal{B}(G)$, we have $\hat{f} = F$ and $\hat{c} = \max_v t(v)$.*
- (b) *If $\max_v t(v) < F \cdot \mathcal{B}(G)$, the minimum iteration period \hat{c}/\hat{f} is the irreducible form of the value*

$$\min\left\{\frac{\max_v t(v)}{\left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil}, \min_{\frac{\max_v t(v)}{\mathcal{B}(G)} < f \leq F} \frac{\lceil f \cdot \mathcal{B}(G) \rceil}{f}\right\}.$$

Proof: We need to prove that $MCP_n(F)/F$ is the minimum iteration period achievable and F is the minimum unfolding factor capable of achieving the iteration period $MCP_n(F)/F$.

Consider an unfolding factor f smaller than F . In case that $\max_v t(v) \geq F \cdot \mathcal{B}(G)$, we know $\max_v t(v) \geq f \cdot \mathcal{B}(G)$ and $MCP_n(f) = MCP_n(F)$. Thus, the iteration period achieved by unfolding factor f is $MCP_n(F)/f$, which is larger than $MCP_n(F)/F$. The unfolding factor f could not achieve the minimum iteration period.

In case that $\max_v t(v) < F \cdot \mathcal{B}(G)$, we have two cases:

- $\max_v t(v) \geq f \cdot \mathcal{B}(G)$.

The larger the f , the better the iteration period achieved. The best iteration period for all f such that $\max_v t(v) \geq f \cdot \mathcal{B}(G)$ is $\hat{c}/\hat{f} = \max_v t(v) / \left\lfloor \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rfloor$.

- $\max_v t(v) < f \cdot \mathcal{B}(G)$.

Under this case, the minimum cycle period for non-pipelined design is $\lceil f \cdot \mathcal{B}(G) \rceil / f$, which the same as that for pipelined design. Since the minimum iteration period for every unfolding factor has to be examined, the second term of the minimum function is served for this purpose.

Therefore, the minimum iteration period under all cases is the minimum within the resource constraints. \square

For example, given a DFG with $\mathcal{B}(G) = 15/11$, $F = 6$, and $\max_v t(v) = 4$, we have $\max_v t(v) = 4 < 6 \cdot 15/11$. Thus, we use the formula in **(b)**. The first term is $\frac{4}{\left\lfloor \frac{4}{15/11} \right\rfloor} = 4/2 = 2$; the second term is $\min_{2 < f \leq 6} \lceil f \cdot 15/11 \rceil / f = 7/5$. Therefore, $\hat{c} = 7$ and $\hat{f} = 5$.

5.5 Fractional Model

Under some implementation models, the starting time of a node can be any fractional number, i.e. the schedule can have an arbitrarily fine grid. We can reduce the unfolding factor by using a finer time slice.

We use our algorithm to find a schedule with a fractional cycle period c and an (integral) unfolding factor f . First, we find the smallest integer ρ such that ρc is integral, i.e. ρ is the denominator of c in its irreducible form. Then, we use a finer time slice: $1/\rho$ time unit is a time slice. All the computation times in the DFG are modified to refer to time slices. Thus, we have a new DFG $G' = (V, E, d, t')$, where $t'(v) = \rho \cdot t(v)$. The lower bound of the new DFG becomes $\mathcal{B}(G') = \rho \cdot \mathcal{B}(G)$. Thus, we can use the theorems for the integral model to compute rate-optimal unfolding factors.

By using the fractional model, we can reduce the unfolding factor caused by the iteration bound $\mathcal{B}(G)$. The new iteration bound is $\rho \cdot \mathcal{B}(G)$. If we choose the right value for ρ , we can reduce the denominator of the iteration bound by a factor of ρ . Thus, under pipelined design, the rate-optimal unfolding factor is reduced by a factor of ρ . Theoretically, we can always choose ρ to be the denominator of $\mathcal{B}(G)$ in its irreducible form so that a static schedule without unfolding can be found. For the non-pipelined design, Theorem 5.1 gives the necessary and sufficient condition on the existence of a static schedule.

However, the fractional model does not relieve the inequality for non-pipelined design $c \geq \max_v t(v)$.

The minimum rate-optimal unfolding factor for pipelined and non-pipelined design under the fractional model are given in Theorem 5.8 and Theorem 5.9.

Theorem 5.8 *Under the fractional model and pipelined design, there always exists a rate-optimal pipelined static schedule without unfolding.*

Proof: Let σ and ρ be the numerator and denominator of the iteration bound $\mathcal{B}(G)$ in its irreducible form. Let $1/\rho$ be the size of a time slice. We apply Theorem 5.5 on the modified DFG G' . The iteration bound of G' , $\mathcal{B}(G')$, is $\rho \mathcal{B}(G) = \sigma$, which is an integer. Thus, the numerator and denominator of $\mathcal{B}(G')$ are σ and 1, respectively. Since the minimum rate-optimal unfolding factor is 1, there exists a rate-optimal schedule without unfolding. \square

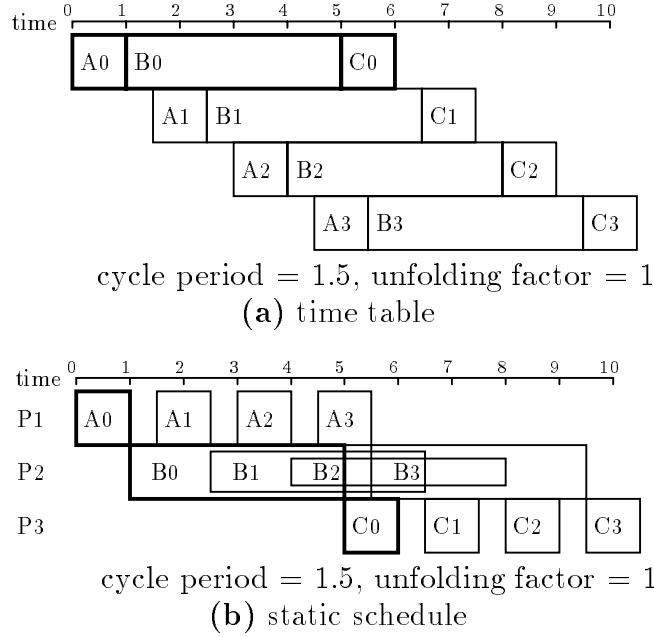


Figure 5.11: Fractional model and pipelined implementation

Theorem 5.9 *The minimum unfolding factor to deliver a rate-optimal static schedule under non-pipelined implementation on the fractional model is*

$$\hat{f} = \left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil.$$

Proof: Let σ and ρ be the numerator and denominator of the iteration bound $\mathcal{B}(G)$ in its irreducible form. Let $1/\rho$ be the size of a time slice. We apply Theorem 5.6 on the modified DFG G' . As shown in Theorem 5.8, the numerator and denominator of $\mathcal{B}(G')$ are σ and 1, respectively. Thus, the minimum unfolding factor \hat{f} is

$$\left\lceil \frac{\max_v \rho t(v)}{\sigma} \right\rceil \cdot 1 = \left\lceil \frac{\max_v t(v)}{\sigma/\rho} \right\rceil = \left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil.$$

□

Consider the example in Figure 5.4. Under the fractional model and pipelined design, Theorem 5.8 shows that there always exists a rate-optimal schedule without unfolding. The rate-optimal schedule obtained by our scheduling algorithm is shown in Figure 5.11. Under the fractional model and non-pipelined design, we derive the

rate-optimal minimum unfolding factor from Theorem 5.9 is

$$\hat{f} = \left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil = \left\lceil \frac{4}{3/2} \right\rceil = 3.$$

Figure 5.6 shows the rate-optimal schedule with unfolding factor 3 obtained by our scheduling algorithm.

We still measure the cycle period in terms of time units, instead of time slices. The minimum cycle periods for a given unfolding factor under the fractional model are similar to the ones under the integral model for either pipelined or non-pipelined design.

Lemma 5.5.1 *Let G be a DFG and f be a given unfolding factor.*

(a) *The pipelined schedule using unfolding factor f has the minimum cycle period $MCP_p(f) = f \cdot \mathcal{B}(G)$.*

(b) *The non-pipelined schedule using unfolding factor f has the minimum cycle period*

$$MCP_n(f) = \begin{cases} \max_v t(v) & \text{if } \max_v t(v) \geq f \cdot \mathcal{B}(G), \\ f \cdot \mathcal{B}(G) & \text{otherwise.} \end{cases}$$

Proof: From Corollary 5.4.1, for pipelined design the minimum cycle period in terms of time slices is $\lceil f \cdot \rho \mathcal{B}(G) \rceil$. Since $\rho \mathcal{B}(G)$ is integral, the minimum cycle period in terms of time slices is $\rho \cdot f \mathcal{B}(G)$, which is $f \cdot \mathcal{B}(G)$ in terms of time units. Similarly, from Lemma 5.4.3, we can derive the minimum cycle period in terms of time units for non-pipelined design. \square

From the above lemma, we can derive the minimum iteration period under a maximum unfolding factor F for non-pipelined design in the following lemma. For pipelined design, the minimum rate-optimal unfolding factor is already 1, so we need not consider the case of smaller F .

Theorem 5.10 *Let F be the maximum unfolding factor feasible under resource requirements. The non-pipelined schedule with the minimum iteration period using an unfolding factor not exceeding F has minimum unfolding factor \hat{f} and cycle period \hat{c} .*

(a) *If $\max_v t(v) \geq F \cdot \mathcal{B}(G)$, we have $\hat{f} = F$ and $\hat{c} = \max_v t(v)$.*

(b) If $\max_v t(v) < F \cdot \mathcal{B}(G)$, \hat{f} equals the minimum rate-optimal unfolding factor $\left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil$ and $\hat{c} = \hat{f} \cdot \mathcal{B}(G)$.

5.6 Conclusion

This chapter described a unified static scheduling technique on two models and two implementation styles. All the minimum rate-optimal unfolding factors under these four combination were obtained. Given a DFG with the iteration bound $\mathcal{B}(G) = \sigma/\rho$ in its irreducible form, we summarize the minimum rate-optimal unfolding factors in the following table.

Min Rate-optimal Unfolding Factor	Timing Models	
	Fractional	Integral
Pipelined Design	1	ρ
Nonpipelined Design	$\left\lceil \frac{\max_v t(v)}{\mathcal{B}(G)} \right\rceil$	$\left\lceil \frac{\max_v t(v)}{\sigma} \right\rceil \cdot \rho$

For a given maximum unfolding factor, we can compute the corresponding best legal static schedule. An efficient polynomial-time scheduling algorithm to compute a static schedule with unfolding factor f was presented which works on the original DFG instead of the larger unfolded DFG. We should point out that this algorithm has performed the effect of retiming implicitly; in other words, explicit retiming cannot get better results. Therefore, the results in this chapter still hold even if explicit retiming is performed. This chapter lays a theoretical foundation for static scheduling, and has potential to be generalized to other scheduling models.

The static schedule generated for the software pipelining problem in Section 4.4 is actually under the integral timing model and pipelined design, since only the issue times of operations are considered. For this reason, the theorems in Chapter 4 appear again in Section 5.4. For a cycle period c , the algorithm in Section 4 generates the repeating pattern of length c directly, while the algorithm in this chapter generates

a schedule to be overlapped with initiation interval c . These two results actually correspond to the same schedule.

In order to obtain non-pipelined implementation under the model of Chapter 4, we can add a self cycle of f delay for every node if we are seeking a schedule with unfolding factor f . Then, the algorithm in Chapter 4 can be applied to find schedules under non-pipelined implementation.