UNIVERSITAT POLITÈCNICA DE CATALUNYA

LOOP PIPELINING WITH RESOURCE AND TIMING CONSTRAINTS

Autor: Fermín Sánchez

October, 1995

4 ANALYSIS OF DATA DEPENDENCES

4.1 INTRODUCTION

Analysis of dependences is usually statically performed [Ban89]. This chapter studies data dependences from a new point of view.

A scheduling algorithm schedules the instructions one by one. After scheduling each instruction, some dependences cease to constrain the scheduling process, whilst others change the way in which they impose constraints. In a given instance of the scheduling process, some instructions have already been scheduled, while others have not. In this instance, the constraints imposed by the dependences are, in general, quite different from the ones imposed before starting the scheduling.

A combination between *dependence retiming* and scheduling of π-graphs allows a loop to be pipelined. Let us give an example. Figure 4.1(a) depicts a π-graph after retiming dependence (v, x). *A possible pipelined schedule for the loop is shown in Figure 4.1(b).* All the iterations are executed in the same fashion, and therefore scheduling the π-graph is equivalent to scheduling the entire loop. If instruction u from iteration i is scheduled at cycle c, instruction u from k iteration will be scheduled later at cycle $c + k \cdot II$, II being the length (initiation interval) of the schedule of an iteration.

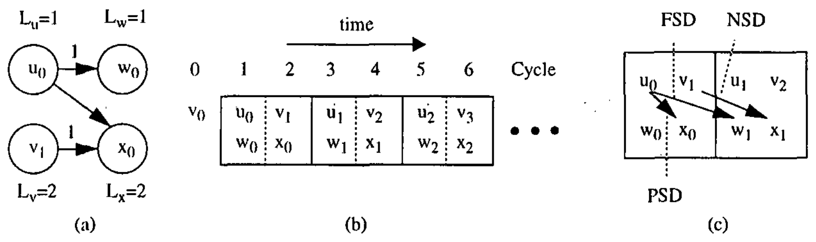


Figure 4.1 Types of dependences in a schedule
(a) Example of retimed π-graph
(b) Schedule of the loop
(c) Types of dependences in a schedule

Which kind of algorithms can be used to schedule the π-graph? Scheduling algorithms for basic blocks have not been devised for loops. They only consider data dependences which are equivalent to ILDs, ignoring the effect of LCDs. However, in a loop with LCDs, an iteration cannot be scheduled without taking into account how the previous iterations have been scheduled. In the example from Figure 4.1, x cannot be scheduled at the first cycle of a given iteration if v has been scheduled at the last cycle of the previous iteration. Otherwise, the LCD (v, x) would not be honored. We conclude that scheduling algorithms for basic blocks cannot be used to schedule π-graphs.

Some scheduling algorithms attempt to guess the initiation interval of the schedule before scheduling the instructions [RG81]. Therefore, they can foresee whether a LCD will be honored or not. These algorithms can be used to schedule a π-graph. ILDs always constrain the scheduling process. In the example from Figure 4.1(c), x is scheduled after u on each iteration due to ILD $u \xrightarrow{0} x$. For this reason, we call such dependences *positive scheduling dependences* (PSDs). However, not all LCDs constrain the scheduling process. In the example, LCD $u \xrightarrow{1} w$ does not constrain the scheduling, regardless of the cycle in which u is scheduled. Therefore, it may be ignored by the scheduling algorithm. We call these kind of dependences *free scheduling dependences* (FSDs). On the other hand, LCD $v \xrightarrow{1} x$ forces x to be scheduled from the same cycle as v (because $II = 2$). Therefore, the cycle in which x is scheduled depends on the cycle on which v has been scheduled. We call such LCDs *negative scheduling dependences* (NSDs). Figure 4.1(c) illustrates the three types of dependences.

4.2 SCHEDULE OF A π-GRAPH

The scope of this chapter is limited to π-graph schedules in which the (expected) *initiation interval* is known in advance. Henceforth, we will assume that the cycles of a schedule are numbered from 0 to $II - 1$, II being the *initiation interval* of the schedule. Let us consider a π-graph $\pi = G(V, E, \lambda, \delta)$. Let S(u) be the cycle at which instruction u has been scheduled.

Theorem 4.1 *In order to honor any dependence $(u, v) \in E$, the following equation must be fulfilled:*

$$S(v) \geq \max_{(u,v)\in E} (S(u) + L_u - II \cdot \delta(e)) \tag{4.1}$$

Proof:

A dependence $u \xrightarrow{\delta(e)} v$ indicates that the data produced by instruction u at the ith iteration is consumed by operation v at iteration $i + \delta(e)$. Instruction $v_{i+\delta(e)}$ must be scheduled at least L_u cycles after the scheduling of u_i. Thus:

$$S(v_{i+\delta(e)}) \geq S(u_i) + L_u$$

Given that v_i is scheduled $II \cdot \delta(e)$ cycles before $v_{i+\delta(e)}$, by substituting the equation $S(v_{i+\delta(e)}) = S(v_i) + II \cdot \delta(e)$ with the former equation we obtain:

$$S(v_i) + II \cdot \delta(e) \geq S(u_i) + L_u$$

$$S(v_i) \geq S(u_i) + L_u - II \cdot \delta(e)$$

□

Definition 4.1 $S(\pi)$ **: Schedule of a π-graph**

For a given II, a schedule *of $\pi = G(V, E, \lambda, \delta)$, is an integer labelling $S : V \mapsto \mathbb{N}$ which fulfills the following conditions:*

- $\forall u \in V, 0 \leq S(u) \leq II - 1$.
- $\forall (u, v) \in E, S(v) \geq S(u) + L_u - II \cdot \delta(u, v)$, *i.e. all dependences must be honored.*
- *There are sufficient resources to execute the instructions scheduled at each cycle.*

Let us consider a loop schedule S with an initiation interval II.

Definition 4.2 $Th(S)$ **: Execution throughput of S**

The execution throughput *of S, Th(S), is the average number of iterations executed per cycle in S.*

$$Th(S) = \frac{1}{II} \tag{4.2}$$

4.3 SCHEDULING DEPENDENCES

Definition 4.3 PSD: Positive Scheduling Dependence

A data dependence $e = (u, v) \in E$ is a positive scheduling dependence when the scheduling of v must follow the scheduling of u at each iteration of π ($S(v) > S(u)$).

ILDs are always PSDs. From Theorem 4.1 we have:

$$S(v) \geq \max_{(u,v)\in E}(S(u) + L_u - II \cdot \delta(e))$$

We conclude that PSDs honor:

$$L_u - II \cdot \delta(e) > 0 \tag{4.3}$$

Definition 4.4 FSD: Free Scheduling Dependence

Data dependences $e = (u, v) \in E$ so that the scheduling of u does not impose constraints on the scheduling of v are called free scheduling dependences.

If $e = (u, v) \in E$ is an FSD, then $S(v) \geq 0$. By using Theorem 4.1, we conclude a dependence $e = (u, v) \in E$ is an FSD when $S(u) + L_u - II \cdot \delta(e) \leq 0$. Since $S(u) \in [0, II - 1]$, the more restrictive value of $S(u)$ for scheduling v is $S(u) = II - 1$. Therefore, we conclude that FSDs fulfill:

$$L_u - II \cdot \delta(e) \leq 1 - II$$

Definition 4.5 Negative Scheduling Dependences

A data dependence $e = (u, v) \in E$ is a negative scheduling dependence *when v might not be freely scheduled at any one of the II cycles of the schedule but e is not a PSD.*

In this case, the datum produced by the instruction u from the current iteration is consumed by instruction v from a following iteration, but v may be scheduled before u. Moreover, NSDs fulfill $S(v) > 0$ (otherwise they would be FSDs). By using the former expressions and Theorem 4.1, we conclude that NSDs fulfill:

$$1 - II < L_u - II \cdot \delta(e) \leq 0$$

By using the previous definitions, scheduling dependences can be classified according to their distance as follows:

- PSD: $L_u - II \cdot \delta(u, v) > 0 \Longrightarrow \delta(u, v) < \frac{L_u}{II}$
- NSD: $1 - II < L_u - II \cdot \delta(u, v) \leq 0 \Longrightarrow \frac{L_u + II - 1}{II} > \delta(u, v) \geq \frac{L_u}{II}$

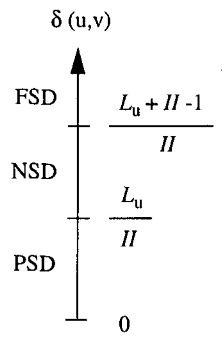


Figure 4.2 Types of scheduling dependences according to the value of $\delta(u,v)$

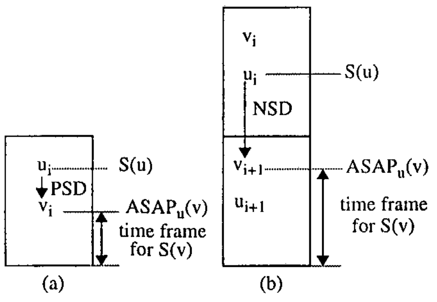


Figure 4.3 Time frame for $S(v)$
(a) When e is a PSD
(b) When e is an NSD

- FSD: $L_u - II \cdot \delta(e) \leq 1 - II \Longrightarrow \delta(u,v) \geq \frac{L_u+II-1}{II}$

Figure 4.2 illustrates this classification.

Let us consider a dependence $e = (u,v)$ in a π-graph. The relationship between the scheduling of u and v depends on whether dependence is a PSD or an NSD. Figure 4.3 shows an example. Note that v must be scheduled after u when e is a PSD, whereas it might be scheduled before u when e is an NSD.

4.4 POSITIVE DEPTH AND HEIGHT

4.4.1 Positive path

Definition 4.6 Path

A path in $\pi = G(V, E, \lambda, \delta)$ is defined as a set of edges $e_i \in E$, $\{e_1 = (u_1, u_2), e_2 = (u_2, u_3), \ldots, e_n = (u_n, u_{n+1})\}$, with $e_i \neq e_j$ $\forall i \neq j$.

The edges e_1 and e_n are called respectively the head edge and the tail edge of the path. In the same way, u_1 and u_{n+1} are called respectively the head node and the tail node of the path. A node u is included in (or belongs to) a path p if $\exists(u, v) \in p$ or $\exists(v, u) \in p$. With this assumption, we say that $u \in p$.

Definition 4.7 Positive Path

A path $p = \{e_1, \ldots, e_n\}$ is called a positive path *if, $\forall e \in p$, e is a PSD.*

A positive path p determines a partial execution order of the instructions represented by the nodes belonging to p. A cycle (recurrence) composed only of PSDs is called a *positive recurrence*.

Theorem 4.2 *A π-graph cannot contain positive recurrences.*

Proof: (by contradiction)

Let us assume that a π-graph has a positive recurrence R composed of n nodes $(u_1, u_2, \cdots, u_n)$ and n edges $(e_1, e_2, \cdots, e_n)$.

Let $S(u_i)$ be the cycle at which instruction u_i starts their execution. Given that $\delta(e_i)$ is a PSD $\forall e_i \in E$, we have that $S(u_1) < S(u_2)$, $S(u_2) < S(u_3)$ and, in general $S(u_i) < S(u_{i+1})$. Moreover, $S(u_n) < S(u_1)$. Therefore, $S(u_1) < S(u_1)$, which is false. □

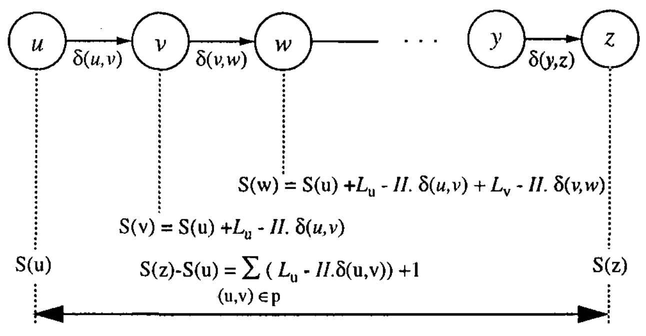
Figure 4.4 shows a positive path p in a π-graph. Instruction v must be scheduled at least $L_u - II \cdot \delta(u, v)$ cycles after the scheduling of u. In the same way, w must be scheduled $L_v - II \cdot \delta(v, w)$ cycles after v, i.e., $(L_u - II \cdot \delta(u, v)) + (L_v - II \cdot \delta(v, w))$ cycles after u, and so on.

When all instructions in p are scheduled *as soon as possible*, the total number of elapsed cycles between the scheduling of u and the scheduling of z is:

$$S(z) - S(u) = (L_u - II \cdot \delta(u, v)) + (L_v - II \cdot \delta(v, w)) + \ldots + (L_y - II \cdot \delta(y, z))$$

$$S(z) - S(u) = \sum_{(u,v) \in p} (L_u - II \cdot \delta(u, v)) \tag{4.4}$$

Since instruction z must start its execution inside the scheduling of the current iteration, the minimum length of the schedule of p is $S(z) - S(u) + 1$.



u
v
w
y
z
δ(u,v)
δ(v,w)
δ(y,z)
S(w) = S(u) +L_u - II. δ(u,v) + L_v - II. δ(v,w)
S(v) = S(u) +L_u - II. δ(u,v)
S(u)
S(z)-S(u) = Σ_(u,v)∈p (L_u - II.δ(u,v)) +1
S(z)

Figure 4.4 Length of a positive path

Definition 4.8 *Length(p)*: **Length of a positive path** p

The length *of a positive path p, Length(p), is defined as the minimum number of cycles elapsed between the starting of the instructions represented by the* head node *and the* tail node *of p. The length of a positive path p is:*

$$Length(p) = 1 + \sum_{(u,v)\in p} (L_u - II \cdot \delta(u,v))$$

4.4.2 Maximal positi^v^e path, positi^v^e depth and height

Definition 4.9 $MPP(\pi)$: **Maximal Positive Path of** π

A maximal positive path (MPP) *of* $\pi = G(V, E, \lambda, \delta)$, *is a positive path p such that, for any other positive path* $p' \subseteq E$, $Length(p) \geq Length(p')$.

An *MPP* will also be called a *critical path.* Henceforth, MPP will denote the length of the critical path. For any schedule S of a loop, $II \geq MPP$.

Definition 4.10 E^+ : **Set of PSDs of a** π**-graph**

E^+ *is the set of the PSDs of* $\pi = G(V, E, \lambda, \delta)$ *for a given II.*

$$E^+ = \{(u,v) \in E \mid L_u - II \cdot \delta(u,v) > 0\ \}$$

Definition 4.11 $D^+(u)$: **Positive Depth of node** u **in a** π**-graph**

The positive depth of a node u in $\pi = G(V, E, \lambda, \delta)$, $D^+(u)$, *is:*

$$D^+(u) = \begin{cases} 1 & \text{if } \nexists(u,v) \in E^+ \\ \max_{(u,v)\in E^+} (D^+(v) + L_u - II \cdot \delta(u,v)) & \text{otherwise} \end{cases}$$

Definition 4.12 $H(v)$ **: Height of node** v

For a given π*-graph* $\pi = G(V, E, \lambda, \delta)$ *the* height *of a node* v*,* $H(v)$*, is:*

$$H(v) = \begin{cases} 0 & \textit{if } \nexists (u,v) \in E^+ \\ \max\limits_{(u,v)\in E^+} (H(u) + L_u - II \cdot \delta(u,v)) & \textit{otherwise} \end{cases}$$

As in the calculation of positive depth, only *PSDs* are taken into account to compute the *height* of a node. The *positive depth* of a node indicates the *urgency* to schedule the node [Gof76]. The *height* of a node is the first cycle at which the node can be scheduled without violating precedence constraints.

Since a π-graph does not contain positive recurrences (see theorem 4.2), a positive path cannot form a recurrence. Therefore, a positive depth and a height can be assigned to each node in the π-graph in $O(V + E)$ time by using a topological sort algorithm [CLR90].

4.4.3 Example of computing positive depth

Figure 4.5 shows an example of computing the MPP and assigning positive depth to nodes in a π-graph. The expected initiation interval of the schedule is $II = 1$ (the *MII*). The π-graph shown in Figure 4.5(a) shows the PSDs as bold arrows. The result latency of instructions is $L_z = L_x = L_q = 1$, $L_w = 2$ and $L_u = L_v = 6$. The π-graph shown in Figure 4.5(b) represents the set of dependences $E^+ = \{(z,u),(u,w),(v,w),(w,q)\}$. Bold arrows represent the critical path ($v \rightarrow w \rightarrow q$), with *Length* $= 7$. Thus, the minimum number of cycles for any schedule of this π-graph is $II = 7$ (and therefore no schedule exists with the previous assumptions). The depth of the nodes is computed as follows:

$D^+(x) = 1$
$D^+(q) = 1$
$D^+(w) = D^+(q) + L_w - II \cdot \delta(w,q) = 1 + 2 - 0 = 3$
$D^+(v) = D^+(w) + L_v - II \cdot \delta(v,w) = 3 + 6 - 2 = 7$
$D^+(u) = D^+(w) + L_u - II \cdot \delta(u,w) = 3 + 6 - 4 = 5$
$D^+(z) = D^+(u) + L_z - II \cdot \delta(z,u) = 5 + 1 - 0 = 6$

Note that, despite the number of edges in the path ($z \rightarrow u \rightarrow w \rightarrow q$) being greater than the number of edges in the *MPP*, the length is shorter (*Length* $= 6$).

4.5 ASAP AND ALAP TIME

The ASAP and the ALAP time of an instruction are well-known concepts. They indicate respectively the first and the last cycle at which the instruction may be scheduled. The ASAP and the ALAP time of a given instruction may change during the scheduling process, but their initial values can be statically computed. $ASAP(v), ALAP(v) \in [0, II - 1]$.

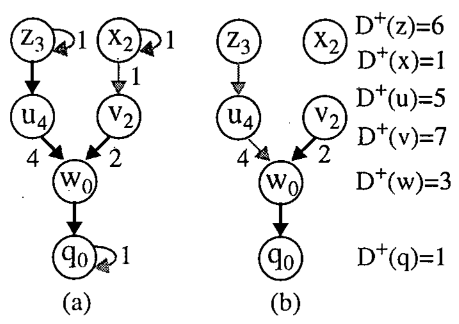


Figure 4.5 Positive Depth of the nodes in a π-graph when $II = 1$
(a) π-graph example
(b) PSDs of the π-graph and positive depth of nodes

Let us consider a π-graph $\pi = G(V, E, \lambda, \delta)$ and a schedule of π with an expected initiation interval II.

Definition 4.13 $IASAP(u)$ **: Initial ASAP time of instruction** u

The initial ASAP time of instruction $u \in V$, $IASAP(u)$, is the first cycle at which u can be scheduled without violating any dependence (computed before starting the scheduling process).

$$IASAP(u) = H(u) \tag{4.5}$$

Definition 4.14 $IALAP(u)$ **: Initial ALAP time of instruction** u

The initial ALAP time of u, $IALAP(u)$, is the last cycle at which u may be scheduled in a correct schedule (computed before starting the scheduling process).

$$IALAP(u) = II - D^+(u) \tag{4.6}$$

Theorem 4.3

$$S(u) \leq \min_{(u,v)\in E} S(v) - L_u + II \cdot \delta(u,v)$$

Proof: If v has already been scheduled, from equation (4.1) we have:

$$S(v) \geq S(u) + L_u - II \cdot \delta(e)$$

$$S(u) \leq S(v) - L_u + II \cdot \delta(e)$$

□

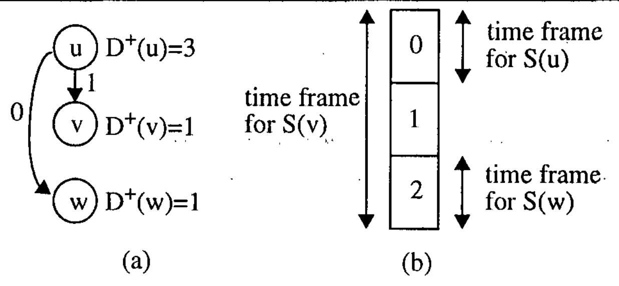


Figure 4.6 NSD (u,v) that does not constrain the scheduling process
(a) π-graph example
(b) Time frame for scheduling instructions

4.6 NEGATIVE DEPTH

4.6.1 Negative restrictive dependences

As with FSDs, some NSDs do not constrain the scheduling process either. Figure 4.6 shows an example in which the result latency for all instructions is 2 and $II = 3$. Positive depth for each instruction is shown on the right of each node. Edge (u, v) is an NSD, since $L_u - II \cdot \delta(u, v) = -1$. This means that v cannot start more than 1 cycle before the starting of u.

The time frame for scheduling an instruction goes from its *ASAP* time to its *ALAP* time. When the π-graph from Figure 4.6(a) is scheduled in 3 cycles, u can be scheduled only at cycle 0, and w can be scheduled only at cycle 2. In this assumption, v may be scheduled from cycle 0 to cycle 2. Figure 4.6(b) illustrates this issue. As can be seen, dependence (u, v) does not constrain the scheduling because v may be scheduled at any cycle.

An NSD $e = (u, v)$ constrains the scheduling depending on the cycle at which u is scheduled.

Theorem 4.4 *A necessary (but not sufficient) condition for an NSD $e = (u, v)$ to constrain the scheduling is:*

$$D^+(u) - II < L_u - II \cdot \delta(u, v)$$

Proof:

Dependence $e = (u, v)$ constrains the schedule if it causes $S(v) > 0$. By substituting $S(v)$ with Equation (4.1), we obtain $S(u) + L_u - II \cdot \delta(u, v) > 0$.

The maximum constraint imposed by an NSD $e = (u, v)$ is the case $S(u) = ALAP(u)$. Therefore, $ALAP(u) + L_u - II \cdot \delta(u, v) > 0$.

Since the maximum value for $ALAP(u)$ is $IALAP(u)$, by substituting $ALAP(u)$ with Equation (4.6) we obtain $II - D^+(u) + L_u - II \cdot \delta(u, v) > 0$.

Thus, we conclude that $L_u - II \cdot \delta(u, v) > D^+(u) - II$. □

In the example from Figure 4.6, we have $L_u - II \cdot \delta(u,v) = -1$ and $D^+(u) - II = 0$. Thus, we conclude that (u,v) does not constrain the scheduling process.

Definition 4.15 *INRD* **: Initial Negative Restrictive Dependence**

A dependence $e = (u,v)$ is called an initial negative restrictive dependence*, INRD, if:*

$$D^+(u) - II < L_u - II \cdot \delta(u,v)$$

Before starting the scheduling process, an INRD constrains the scheduling. However, it might cease to constrain the scheduling during the scheduling process. If u has already been scheduled, an NSD $e = (u,v)$ constrains the scheduling process if $S(u) + L_u - II \cdot \delta(e) > 0$ (a necessary condition for $ASAP(v) > 0$). On the other hand, if u has not yet been scheduled, we will use Theorem 4.4 to decide whether e constrains the scheduling or not.

Definition 4.16 *NRD* **: Negative restrictive dependence**

An NSD $e = (u,v)$ is an NRD if and only if:

$$\begin{cases} L_u - II \cdot \delta(e) > \; -S(u) & \textit{If } u \textit{ has already been scheduled} \\ L_u - II \cdot \delta(e) > D^+(u) - II & \textit{If } u \textit{ has not yet been scheduled} \end{cases} \tag{4.7}$$

4.6.2 Assigning negati‑e depth to nodes

In the same way that a positive depth can be assigned to each node in the π-graph by only taking PSDs into account, a negative depth can be assigned by only considering NRDs. Initially, the negative depth is assigned by considering INRDs. However, whilst the positive depth of a node never changes, the negative depth might change because some INRDs may cease to be NRDs during the scheduling process. This chapter shows how to calculate the initial negative depth for each node. Changes on negative depth produced during the scheduling process will be studied in Chapter 5.

Definition 4.17 E^-**: Set of the INRDs of a π-graph**

E^- is the set of the INRDs of $\pi = G(V,E,\lambda,\delta)$ for a given II.

$$E^- = \{ e = (u,v) \in E \mid D^+(u) - II < L_u - II \cdot \delta(u,v) \}$$

Definition 4.18 $D^-(v)$ **: Negative Depth of node v in a π-graph**

The negative depth of node v in $\pi = G(V,E,\lambda,\delta)$, $D^-(v)$, is defined as:

$$D^-(v) = \begin{cases} 0 & \textit{if } \nexists (u,v) \in E^- \\ \max\limits_{(u,v)\in E^-} (D^-(u) + L_u - II \cdot \delta(u,v)) & \textit{otherwise} \end{cases}$$

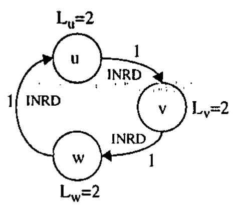


Figure 4.7 Negative recurrence in a π-graph

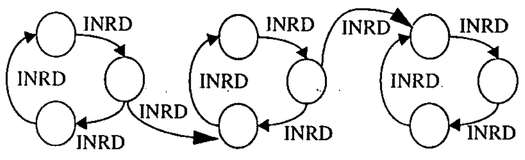


Figure 4.8 Π-graph with negative recurrences chained

Definition 4.19 Negative Path

A path $p = \{e_1, \ldots, e_n\}$ is called negative path *of $\pi = G(V, E, \lambda, \delta)$ if, $\forall e \in p$, e is an INRD.*

An NRD $e = (u, v)$ imposes a maximum time constraint between u and v, since v cannot start more than $L_u - II \cdot \delta(e)$ cycles before u. Therefore, a chain of NRDs also impose a maximum time constraint [KD92]. The negative depth of a node gives an idea of the maximum time constraints imposed by the node on the scheduling process.

A recurrence only composed of NRDs will be called a negative recurrence. Unlike PSDs, NRDs may produce negative recurrences, as shown in Figure 4.7. Moreover, a set of negative recurrences may be chained by INRDs, as shown in Figure 4.8. In order to assign a negative depth to the nodes from a negative recurrence, and given that all nodes impose maximum time constraints among them, we will assign the same negative depth to all nodes belonging to the same negative recurrence.

Therefore, a way of computing the negative depth of the nodes of a π-graph π is by substituting all negative recurrences R with single nodes x, creating a new π-graph π'. A dependence $(u, v) \in E$ so that $u \in R$ and $v \notin R$ is replaced by a dependence (x, v) in π'. In the same way, a dependence $(u, v) \in E$ so that $v \in R$ and $u \notin R$ is replaced by a dependence (u, x) in π'.

The algorithm to compute the negative depth performs the following steps:

1. Substitute each negative cycle R with a single node x. Retain only the dependence with the longest distance for each node x with several incoming (outgoing) edges from the same source (target).

2. Compute the negative depth for any node of the new π-graph. Once negative recurrences have been eliminated from the π-graph, the algorithm to compute the negative depth is similar to that used to compute the positive depth.

3. For each node x replacing a negative recurrence R, substitute x with the original nodes of R (and the original edges). Assign the negative depth of x to all nodes in R.

Computational complexity

The running time of the algorithm to assign the negative depth to the nodes in a π-graph is as follows:

1. Build the π-graph only composed of INRDs has a running time $O(V+E)$. All negative cycles in the π-graph can be found by using Johnson's algorithm to solve the *all-pairs shortest paths problem* [Joh77]. The running time of Johnson's algorithm is $O(V^2 \lg V + VE)$. Substituting the negative recurrences with single nodes has a running time $O(V+E)$. Therefore, the running time of this step is:
$$O(V^2 \lg V + VE)$$

2. The second step has the same running time as the algorithm to compute positive depth: $O(V+E)$.

3. Finally, replacing nodes with negative recurrences (restoring initial dependences) has a running time $O(V+E)$.

Therefore, the final running time of the algorithm to compute the negative depth is:

$$O(V^2 \lg V + VE)$$

4.6.3 Example

Figure 4.9 shows an example of computing the negative depth when the π-graph has negative cycles. Figure 4.9(a) shows an example of π-graph in which the result latency of all instructions is two. The expected initiation interval for the schedule is $II = 3$. Bold nodes D, F and G identify nodes belonging to a negative recurrence R, marked by grey edges in Figures 4.9(a) and 4.9(c). Figure 4.9(b) shows the π-graph (only with the INRDs) after substituting the negative recurrence with a single node u. The negative depth of each node is stated within the node. Figure 4.9(c) shows the final negative depth assigned to the nodes in the initial π-graph, as well as the INRDs.

4.7 SUMMARY AND CONCLUSIONS

In this chapter we present a new analysis of the data dependences in a loop. Data dependences are studied from the point of view of how they constrain the scheduling process. Data dependences

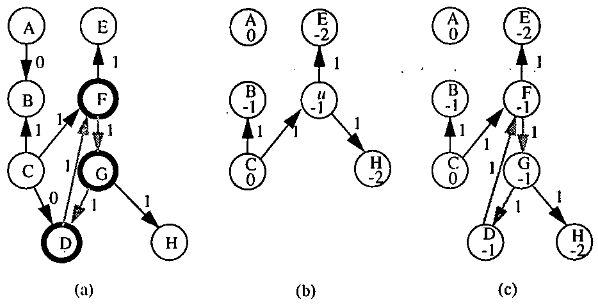


Figure 4.9 Compute of negative depth
(a) π-graph example
(b) NSDs after substituting nodes D, G and F with a single node u
(c) Negative depth assigned to the nodes of the initial π-graph

are classified into three categories: *positive scheduling dependences (PSDs)*, *negative scheduling dependences (NSDs)* and *free scheduling dependences (FSDs)*. The ASAP and the ALAP time for each node are computed by taking only PSDs and NSDs into account, since FSDs do not constrain the scheduling process.

This chapter presents two algorithms which execute in polynomial time:

- An algorithm to assign a positive depth and a height to each node by only considering PSDs. Positive depth influences the length of the schedule because it expresses the minimum time constraints imposed by a node to the scheduling process. Moreover, the initial ALAP time and the initial ASAP time for each node are computed by taking positive depth and height into account.
- An algorithm to compute the negative depth by only using some NSDs. Negative depth expresses the maximum time constraints imposed by a node to the scheduling process.

This chapter presents the following main contributions:

1. Data dependences are studied from a dynamic point of view. For a given dependence $e = (u, v)$, e is classified according to where v may be initially scheduled. If v can be scheduled anywhere into the schedule, e is called an FSD. Otherwise, e is a PSD or a NSD according to whether v may be scheduled after or before u respectively.
2. Positive depth and height have been largely studied in the literature. However, negative depth is a new concept in scheduling. Negative depth of a node is directly related to the maximum time constraints imposed by the node to the scheduling process. As we will show in the following chapter, negative depth is also an important criterion for deciding which node must be scheduled at each moment.
3. PSDs and NSDs initially impose constraints on the scheduling process. However, while PSDs always constrain the scheduling process, not all NSDs impose constraints. Moreover, some NSDs which initially impose constraints may cease to be restrictive during the scheduling process.