3.2 Periodische Ablaufplanung

Zur Vorlesung

Embedded Systems

WS 14/15

Reiner Kolla

3.2.1 Problemstellung

Bei periodischen Ablaufplanungsproblemen liegt eine iterative Definition der Berechnung vor, d.h. es werden immer wieder die gleichen Operationen in Abhängigkeit von einem Index *n*, der einer Iteration entspricht, ausgeführt.

Man könnte solche Planungsprobleme auch durch einen riesigen azyklischen Problemgraphen beschreiben, und mit den uns bekannten Methoden lösen. Wir wollen hier untersuchen, ob man solche Probleme auch auf der Basis einer kompakten Definition der Iteration selbst planen kann:

Definition

Ein **iterativer Algorithmus** besteht aus einer Menge linear indizierter Gleichungen $S_1[n]$, $S_2[n]$, ..., $S_{|V|}[n]$, wobei $S_i[n]$ eine Gleichung der Form

$$x_i[n] = f_i(...,x_j[n-s_{j,i}],...)$$

ist. Der Index $n \in N_0$ gibt die Iteration des Algorithmus an. Die Variablen $x_i[n]$ sind skalar indiziert und f_i sind beliebige Funktionen. Zwischen der Berechnung der Variablen $x_i[n]$ und $x_j[n]$ können konstante Indexverschiebungen $s_{j,i} \in N_0$ bestehen.

Beispiel eines iterativen Algorithmus

(Schreibweise aus der Definition)

$x_1[n] = f_1(x_4[n-1]);$

$x_2[n] = f_2(x_1[n]);$

$x_3[n] = f_3(x_1[n]);$

$x_4[n] = f_4(x_2[n],x_3[n-2]);$

```
-- als VHDL-Code
LOOP
  x1(n) := f1(x4(n-1));
  x2(n) := f2(x1(n));
  x3(n) := f3(x1(n));
  x4(n) := f4(x2(n),x3(n-2));
     n  := n + 1;
END LOOP;
```

Definition

Eine **Iteration** bezeichnet die Berechnung aller Variablen $x_i[n]$ eines gegebenen iterativen Algorithmus für ein festes *n.*

Iterativer Problemgraph

Definition

Ein **iterativer Problemgraph** *G(V,E,s)* ist ein Netzwerk mit einer Kantengewichtsfunktion $s:E \rightarrow N_0$. Die Funktion *s* ordnet jeder Kante $(v_i,v_j) \in E$ die Indexverschiebung $s_{i,j} := s(v_i,v_j)$ zu.

Bemerkung

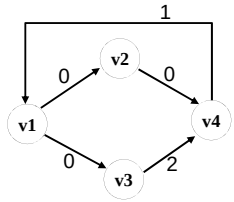
Ein iterativer Problemgraph kann Zyklen besitzen!

$x_1[n] = f_1(x_4[n-1]);$

$x_2[n] = f_2(x_1[n]);$

$x_3[n] = f_3(x_1[n]);$

$x_4[n] = f_4(x_2[n],x_3[n-2]);$



Periodischer Ablaufplan

Definition

Ein **periodischer Ablaufplan** (mit Periode *P*) eines iterativen Problemgraphen $G(V,E,s)$ ist eine Funktion $t: V \rightarrow N_0$, die jedem Knoten $v_i \in V$ die Startzeitpunkte

$$\tau(v_i,n) = t(v_i)+n \cdot P \qquad \forall n \in N_0$$

zuordnet, so dass ferner für alle Kanten $(v_i,v_j) \in E$

$$t(v_j)-t(v_i) \geq w_i - s_{i,j} \cdot P$$

gilt. Dabei gelte $\tau(v_i,n) = 0$ für alle $n < 0$. $\tau(v_i,n)$ stellt den Startzeitpunkt der n-ten Iteration von Knoten v_i dar.

Iterationsintervall

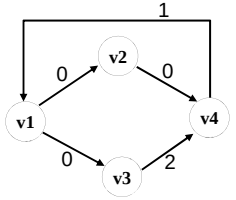
Definition

Als **Iterationsintervall** bezeichnet man die Menge der Zeitschritte zwischen dem Beginn und Ende der Abarbeitung aller Aufgaben einer Iteration.

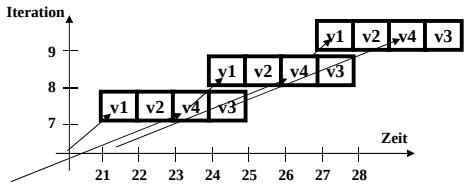
$IT(t,n) := [\ min\{t(v_i) \mid v_i \in V\} + nP,\ max\{t(v_i)+w(v_i) \mid v_i \in V\} + nP\ ]$

$L(t) := max\ \{\ t(v)+w(v)-t(u) \mid u,v \in V\ \}$

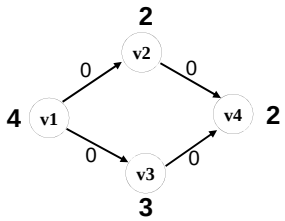
heißt **Iterationsintervalllänge**.



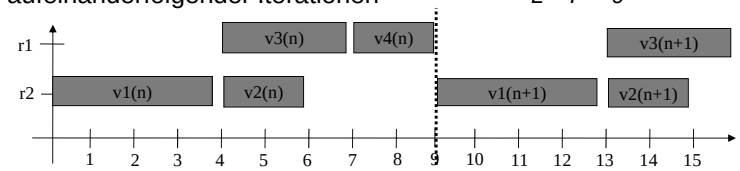
hier: $P = 3$, $L = 4$



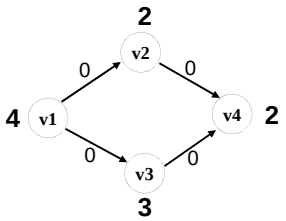
Parallelismus bei iterativen Algorithmen (1)



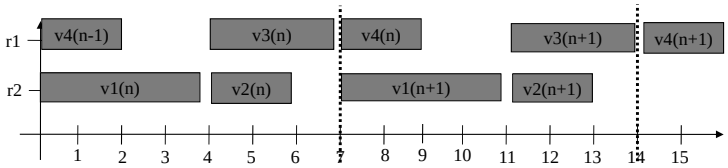
- Parallelisierung eines Intervalls bei sequentieller Abarbeitung aufeinanderfolgender Iterationen $L = P = 9$



Parallelismus bei iterativen Algorithmen (2)

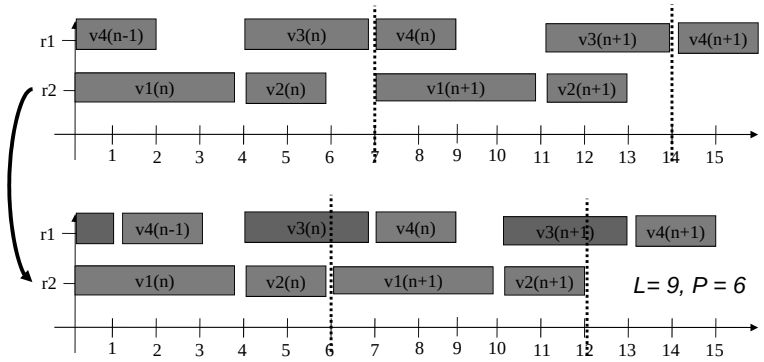


- Nichtüberlappende, parallele Abarbeitung aufeinander folgender Iterationen $L = 9$ $P = 7$



Parallelismus bei iterativen Algorithmen (3)

- Überlappende parallele Abarbeitung aufeinander folgender Iterationen



Anmerkung zur Parallelisierung

Die Art der Parallelisierung hat offenbar starken Einfluss auf die Periode eines möglichen Ablaufplans und damit auf die **Datenrate** (= Daten/Periode) mit der die Aufgaben bearbeitet werden können.

Sequentielle Abarbeitung der Iterationen ist dann erforderlich, wenn sich alle Daten einer Iteration mit einem Takt synchronisieren müssen. (Beispiel: synchrones, sequentielles Schaltwerk).

Nichtüberlappende Abarbeitung ist erforderlich, wenn sich alle Prozesse nach einer Iteration synchronisieren müssen. Es fällt auf, dass nichtüberlappende Abarbeitung im Grunde eine sequentielle Abarbeitung ist, wenn man vorab auf den Aufgaben eine geeignete Indextransformation (Retiming) durchführt.

Überlappende Abarbeitung ist schließlich das mächtigste Parallelisierungsmodell für periodische Planung. Die Periode kann, bei geeigneter Planung, sehr viel kürzer sein, als die Iterations-intervalllänge (Fließbandverarbeitung).

Eine weitere Klassifikation erhält man über die Bindung:

Vollstatische versus zyklostatische Bindung

Bei vollstatischer Bindung sind der Ressourcentyp und die Instanz, an die ein Knoten v_i gebunden wird, für alle Iterationen gleich.

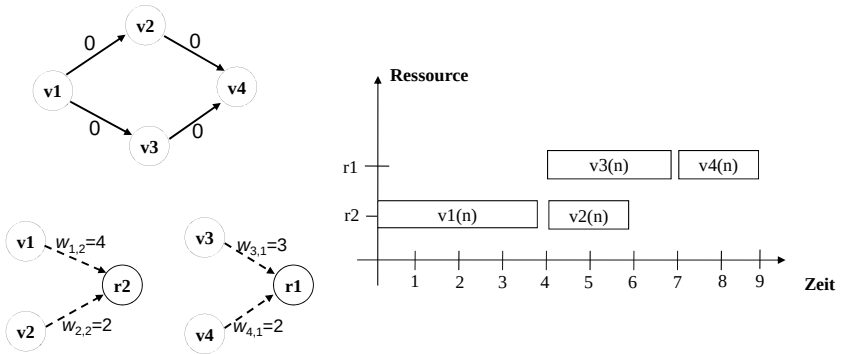
Definition

Gegeben sei eine Spezifikation $(G(V,E,s),G_R(V_R,E_R))$ bestehend aus einem iterativen Problemgraphen G und einem Ressourcegraph G_R. Eine vollstatische Bindung lässt sich sich durch ein Tupel (b,g) von Funktionen mit

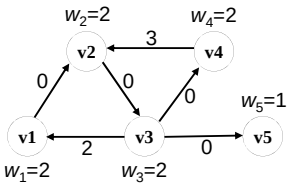
- $b: V_S \rightarrow V_T$ mit $\beta(v_i,n)=b(v_i)$ $\quad \forall n \in N_0$ und $(v_i,b(v_i)) \in E_R$
- $g: V_S \rightarrow N$ mit $\gamma(v_i,n)=g(v_i)$ $\quad \forall n \in N_0$ und $g(v_i) \leq \alpha(b(v_i))$

beschreiben. Dabei bedeuten $\beta(v_i,n)$ bzw. $\gamma(v_i,n)$ den Ressourcetyp bzw. die Instanz, auf der die n-te Berechnung von Knoten v_i erfolgt.

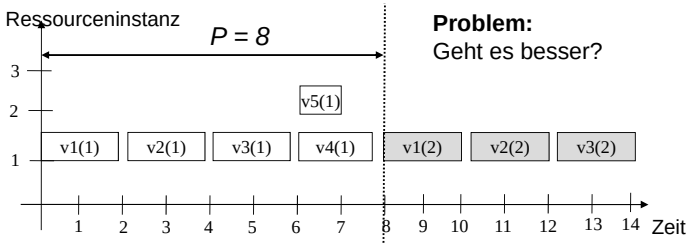
Vollstatische Bindung: Beispiel



Vollstatische Bindung: Beispiel



Annahme:
Allen Knoten ist der gleiche Ressourcentyp zugeordnet



Problem:
Geht es besser?

Zyklostatische Bindung

Bei zyklostatischer Bindung wird jede *K*-te Iterationen einer Aufgabe an dieselbe Ressourceninstanz gebunden.

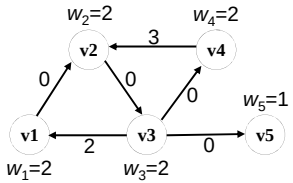
Definition

Gegeben sei eine Spezifikation $(G(V,E,s),G_R(V_R,E_R))$ bestehend aus einem iterativen Problemgraphen G und einem Ressourcegraph G_R, sowie eine Zahl $K \in \mathbb{N}$. Eine **zyklostatische Bindung** mit **Periodizität *K*** lässt sich sich durch ein Tupel (b,g) von Funktionen mit

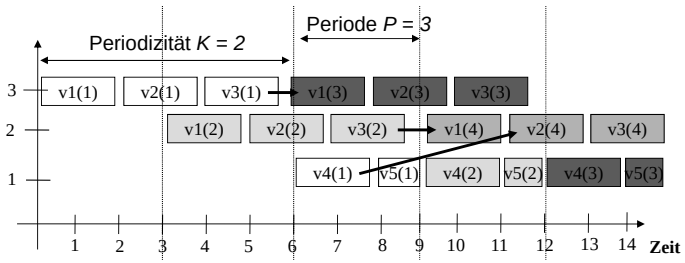
- $b: V \times \{0,1,...,K-1\} \rightarrow V_T$ mit $\beta(v_i,n)=b(v_i,n \bmod K)$ $(v_i,b(v_i,k)) \in E_R$ $\forall n \in \mathbb{N}_0$ und $\forall k$
- $g: V \times \{0,1,...,K-1\} \rightarrow \mathbb{N}$ mit $\gamma(v_i,n)=g(v_i,n \bmod K)$ $g(v_i,k) \le \alpha(b(v_i,k))$ $\forall n \in \mathbb{N}_0$ und $\forall k$

beschreiben. Dabei bedeuten $\beta(v_i,n)$ bzw. $\gamma(v_i,n)$ den Ressourcetyp bzw. die Instanz, auf der die *n*-te Berechnung von Knoten v_i erfolgt.

Zyklostatische Bindung: Beispiel



Annahme:
Allen Knoten ist der gleiche Ressourcentyp zugeordnet



3.2.2 Sequentielle periodische Ablaufplanung

Wir wollen zunächst das einfachste Problem, die Erstellung eines sequentiellen, periodischen Ablaufplans, betrachten. Dazu

Definition

Gegeben sei ein iterativer Problemgraph $G=(V,E,s)$ mit den Knotenausführungsdauern $w(v)$. Sei $p = e_1,\ldots,e_k$ ein **Pfad** von $q(p):=q(e_1)$ nach $z(p):=z(e_k)$. Dann sei

$$s(p) := \sum_{i=1}^{k} s(e_i) \quad \text{sowie} \quad w(p) := \sum_{i=1}^{k} w(q(e_i)) + w(z(p))$$

Wir betrachten nun zu jedem Knotenpaar (u,v) folgende Größen:

- $S(u,v) := min\{\, s(p) \mid p$ ist Pfad von u nach $v \,\}$
- $W(u,v) := max\{\, w(p) \mid p$ ist unter s kürzester Pfad von u nach v d.h. $s(p) = S(u,v) \,\}$

Anmerkung: *S(.)* und *W(.)* lassen sich durch ein all pair shortest path Verfahren in Zeit O(|V||E|log|V|) berechnen (vgl. RGL WS07/08)

Ablaufplan in Iterationsintervalllänge

Satz

Es gibt einen gültigen, sequentiellen, periodischen Ablaufplan ohne Ressourcenbeschränkung mit Iterationsintervalllänge L für einen Problemgraphen $G=(V,E,s)$, genau dann, wenn

$$\forall u,v \in V: W(u,v) > L \Rightarrow S(u,v) \geq 1$$

Beweis:

$\Rightarrow$: Gäbe es ein Paar (u,v), mit $W(u,v) > L$ und $S(u,v) = 0$, dann gäbe es einen Pfad p von u nach v mit $s(p) = 0$ und $w(p) = W(u,v)$.

Damit wäre aber $t(v) + w(v) \geq t(u) + W(u,v)$ in jedem gültigen Ablaufplan, die Iterationsintervalllänge also $> L$. Widerspruch.

Ablaufplan in Iterationsintervalllänge -- ff

Beweis -- ff:

$\Leftarrow$: Wir konstruieren einen gültigen Ablaufplan mit Iterationsintervalllänge L:

Betrachte den Teilgraphen $G'=(V,E')$ von G mit $E' = E \setminus \{ e \mid s(e) > 0\}$

G' ist offenbar azyklisch (keine zyklischen Datenabhängigkeiten ohne Indexverschiebung), und kann mit ASAP oder ALAP unter Latenz

$$max \{ W(u,v) \mid S(u,v) = 0 \} \leq L$$

geplant werden. Sei also $t(v) := \tau^{ASAP}_{G'}(v)$

Nun gilt für jede Kante $e \in E$ mit $s(e) = 0$ per Konstruktion schon :

$$t(q(e)) \leq t(z(e)) - w(q(e))$$

Für alle Knoten gilt ferner: $0 \leq t(v) \leq L - w(v)$

Also gilt für alle anderen Kanten e, $s(e) \geq 1$:

$$t(q(e)) \leq L - w(q(e)) \leq t(z(e)) + L - w(q(e)) \leq t(z(e)) + s(e) \cdot L - w(q(e))$$

Ablaufplan in Iterationsintervalllänge -- ff

Man kann also aus der Konstruktion im Beweis folgendes ablesen

- Einen gültigen Ablaufplan mit minimaler Iterationsintervalllänge L lässt sich durch Berechnen von $S(.)$ und $W(.)$ vermöge

  $$L_{min} := max \{ W(u,v) \mid S(u,v) = 0 \}$$

  durch geeignete Graphalgorithmen sofort finden.
- Man kann auf der Basis von L_{min} mit dem reduzierten Graphen G' nun auch versuchen, durch die Techniken im letzten Abschnitt Ressourcenminimierung unter Latenzschranke L_{min} oder Latenzminimierung unter einer Ressourcenschranke zu betreiben.
- **Problem:** Kann man durch geeignete Indextransformation die Iterationsintervalllänge L_{min} weiter verbessern?

3.2.3 Retiming

Wir wollen nun versuchen, durch Transformation der Indizes die Iterationsintervalllänge zu minimieren. Dies ist im Grunde gleichzusetzen mit dem Problem der nichtüberlappenden periodischen Ablaufplanung mit minimaler Periode

Definition

Gegeben sei ein iterativer Problemgraph $G=(V,E,s)$ mit den Knotenausführungsdauern $w(v)$. Ein **Retiming** von G ist gegeben durch eine Abbildung $r: V \rightarrow \boldsymbol{Z}$ für die eine nichtnegative Neuzuordnung der Indexverschiebungen s' existiert, mit

$$s'(e) := s(e) - r(q(e)) + r(z(e)) \geq 0$$

Anmerkung: Ein Retiming ist im Grunde ja eine Neufestlegung des Iterationsindexes der Aufgabe (u,n) auf $(u,n + r(u))$.

Eine Abhängigkeit zwischen (u,n) und $(v,n+s(u,v))$ übersetzt sich damit in eine Abhängigkeit zwischen $(u,n+r(u))$ und $(v,n+s(u,v)+r(v))$,

also zwischen (u,n) und $(v,n + \underbrace{s(u,v) - r(u) + r(v)}_{s'(u,v)})$

Einfluss eines Retimings auf die Intervalllänge

Wir wollen nun untersuchen, wie sich ein Retiming auf die Iterationsintervalllänge auswirkt

Beobachtung

Gegeben sei ein iterativer Problemgraph *G=(V,E,s)* mit den Knotenausführungsdauern *w(v)* und ein Retiming *r: V* $\rightarrow$ ***Z***

Dann gilt für jeden Pfad $p = e_1,\ldots,e_k$ in G: *s'(p) = s(p) - r(q(p))+r(z(p))*

Beweis:

$$s'(p) := \sum_{i=1}^{k} s'(e_i)$$

$$= \sum_{i=1}^{k} \left[s(e_i) - r(q(e_i)) + r(z(e_i)) \right]$$

$z(e_i) = q(e_{i+1}) \Rightarrow$ Auslöschung!

$$= -r(q(p)) + \left(\sum_{i=1}^{k} s(e_i) \right) + r(z(p))$$

$$= s(p) - r(q(p)) + r(z(p))$$

Einfluss eines Retimings ff

Folgerung

Gegeben sei ein iterativer Problemgraph *G=(V,E,s)* mit den Knotenausführungsdauern *w(v)* und ein Retiming *r: V* $\rightarrow$ ***Z***

Dann gilt

S'(u,v) = S(u,v) - r(u) + r(v)

W'(u,v) = W(u,v)

Beweis:

Da die Änderung der Pfadlängen nur von Quelle und Ziel des Pfades *p* abhängen, bleiben kürzeste Pfade auch nach dem Retiming kürzeste Pfade. Demnach ändert sich die Größe *W(.)* nicht und *S(.)* ändert sich nur in Abhängigkeit von *u* und *v*.

Folgerung

Gegeben sei ein iterativer Problemgraph *G=(V,E,s)* mit den Knotenausführungsdauern *w(v)* und ein Retiming *r: V* $\rightarrow$ ***Z***

Dann ist unter *r* die neue Iterationsintervalllänge

$L'_{min} = max\{ W(u,v) \mid S(u,v) - r(u) + r(v) = 0\}$

Berechnung eines optimalen Retimings

Beobachtung

Aus der letzten Folgerung wird klar:

- Es kann nur Intervalllängen der Form *W(u,v)* für ein Paar *(u,v)* geben, da wir stets das Maximum über solche Paare bilden.
- Das Problem ist, zu entscheiden, ob es für ein festes Paar *(u,v)* ein gültiges Retiming *r* gibt, so dass $L'_{min} = W(u,v)$ ist.

Idee:

Wie schon gezeigt, ändern sich unter einem Retiming *r* die Größe *W(.)* nicht und *S(.)* nur in Abhängigkeit von *r(u)* und *r(v)*.

Wir erhalten also für eine feste Schranke *L* folgende Bedingungen an ein gültiges Retiming:

- $\forall e \in E: s'(e) \geq 0 \Leftrightarrow \boxed{\forall e \in E: r(z(e)) \geq r(q(e)) - s(e)}$
- $\forall (u,v)$ mit $W(u,v) > L: S'(u,v) \geq 1$

  $\Leftrightarrow \quad \boxed{\forall (u,v) \text{ mit } W(u,v) > L: r(v) \geq r(u) - (S(u,v)-1)}$

Berechnung eines optimalen Retimings ff

Beobachtung

Ein Retiming *r*, das die beiden letzten Bedingungen erfüllt, ist Lösung eines single source longest path Problems mit Quelle *s* in folgendem Graphen $G_{r,L} = (V \cup \{s\}, E_r, \lambda)$ mit

- zu jedem $v \in V$ gebe es genau eine Kante *(s,v)* mit Länge $\lambda(s,v) = 0$ in E_r ,
- zu jedem $e \in E$ gebe es genau eine Kante *e* der Länge $\lambda(e) = -s(e)$ in E_r ,
- zu jedem Paar *(u,v)* mit $W(u,v) > L$ gebe es genau eine Kante *(u,v)* der Länge $\lambda(u,v) = -S(u,v)+1$ in E_r .

Folgerung

Hat der oben konstruierte Graph $G_{r,L}$ einen positiven Zyklus, dann gibt es kein zulässiges Retiming mit Intervalllänge *L*.

Minimierung der Intervalllänge durch Retiming

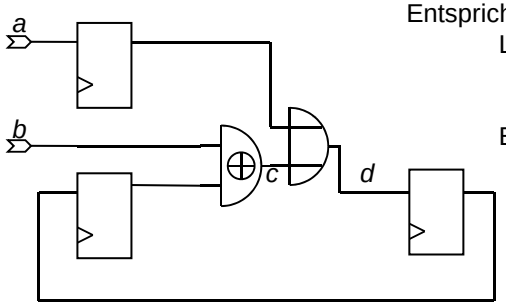
Wir können nun ein Verfahren angeben, das durch Retiming die Intervalllänge minimiert

Algorithmus -- optimales Retiming

- Berechne zu einem Problemgraphen die Größen *S(.)* und *W(.)* durch Lösung eines all pair shortest path Problems O(|V||E|log|V|)
- Sortiere die Menge *{ W(u,v) | u,v ∈ V }* $O(|V|^2 \log|V|)$
- Bestimme durch Binärsuche das kleinste *L=W(u,v)* aus obiger Menge, für das single source longest path Problem in $G_{r,L}$ eine Lösung *r* hat. Wähle *r* als Retiming. $O(|V|^3 \log|V|)$

Beispiel

Ein typisches Beispiel (für das dieses Verfahren auch ursprünglich entwickelt wurde), ist die Minimierung der Zykluszeit eines gegebenen, synchronen Schaltkreises:

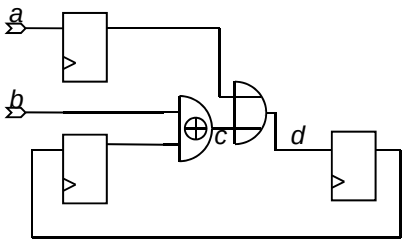


Entspricht dem iterativen Algorithmus

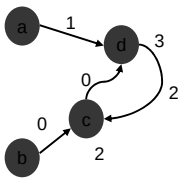
```
LOOP
   c[n] := b[n] exor d[n-2];
   d[n] := a[n-1] or c[n];
END LOOP
```

Die Minimierung der Iterationsintervalllänge entspricht gerade der Minimierung der Zykluszeit!

Beispiel -- ff



Problemgraph (*w(or) = 3, w(exor) = 2*)

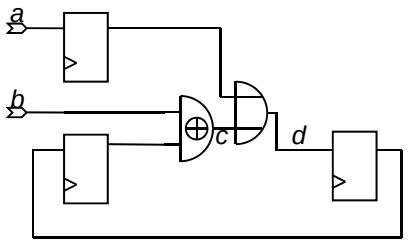


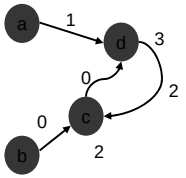
S	*a*	*b*	*c*	*d*
a	*0*		*3*	*1*
b		*0*	*0*	*0*
c			*0*	*0*
d			*2*	*0*

W	*a*	*b*	*c*	*d*
a	*0*		*5*	*3*
b		*0*	*2*	*5*
c			*2*	*5*
d			*5*	*3*

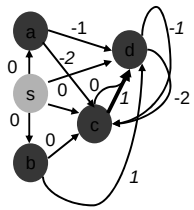
$L_{min}=5$

Beispiel -- ff





Retiming $G_{r,3}$

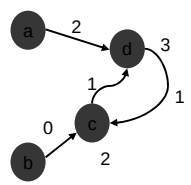


	r
a	*0*
b	*0*
c	*0*
d	*1*

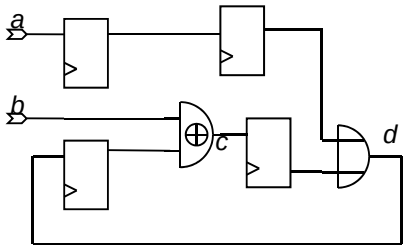
W	*a*	*b*	*c*	*d*
a	*0*		*5*	*3*
b		*0*	*2*	*5*
c			*2*	*5*
d			*5*	*3*

S	*a*	*b*	*c*	*d*
a	*0*		*3*	*1*
b		*0*	*0*	*0*
c			*0*	*0*
d			*2*	*0*

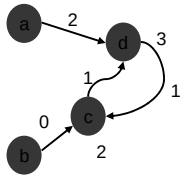
Problemgraph nach Retiming *r*



Beispiel -- ff



Problemgraph nach Retiming *r*



Schaltwerk nach retiming

Eine weitere Verkürzung ist nicht möglich, da ja stets $L \geq max\{ w(v) \}$ sein muss.

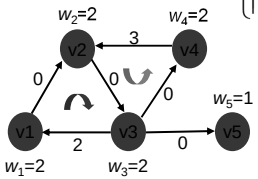
3.2.4 Überlappende, periodische Ablaufplanung

Bis zu welcher Periode kann man höchstens "zusammenschieben"?

Satz -- Maximales Zyklengewicht

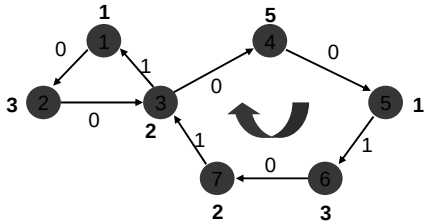
Gegeben sei ein iterativer Problemgraph $G(V,E,s)$. Für jede Kante e sei $w(q(e))$ die Berechnungszeit ihres Quellknotens. Dann ist die Periode nach unten beschränkt durch die **Periodenschranke** P_{min}, definiert durch

$$P_{min} := max\left\{ \left\lceil \frac{\sum_{e \in Z} w(q(e))}{\sum_{e \in Z} s(e)} \right\rceil \;\middle|\; Z \text{ ist Zyklus in G} \right\}$$



$P_{min} = max\{ 6/2 , 6/3 \} = 3$

Beweisidee am Beispiel:



Betrachte den Knoten v_7 und den Zyklus auf dem er liegt. Es gilt für jeden gültigen Ablaufplan:

$\tau(v_7,k+1)$

$\geq \tau(v_6,k+1)+3 \quad \geq \tau(v_5,k)+3+1 \quad \geq \tau(v_4,k)+4+5 \quad \geq \tau(v_3,k)+9+2$

$\geq \tau(v_7,k-1)+13$

Die Periode eines gültigen Ablaufplans muss demnach größer gleich $\lceil 13/2 \rceil$ sein.

Beweis der Periodenschranke

Betrachte einen bel. gerichteten Zyklus $Z=(v_1,...,v_q)$ im iterativen Problemgraph.

Sei

W die Summe der Laufzeiten der Knoten aus Z und

S die Summe der Indexverschiebungen auf den Kanten von Z.

Dann gilt $\forall n \in \mathbf{N}$, $n \geq S$,

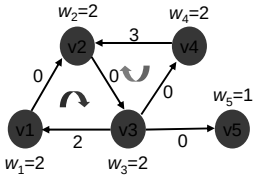
(v_1,n) ist datenabhängig von $(v_1,n-S)$, und somit

$$\tau(v_1,n)) \geq \tau(v_1,n-S)) + W$$

$$t(v_1) + n \cdot P \geq t(v_1) + (n-S) \cdot P + W$$

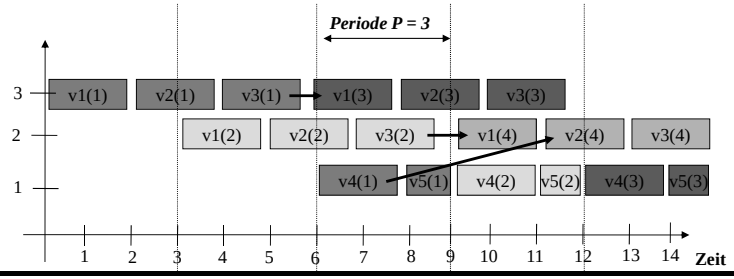
Demnach muss für die Periode $P \cdot S \geq W$ bei allen Zyklen Z gelten.

Periodenschranke: Beispiel



$P \geq max\{ 6/2 , 6/3 \} = 3$

Es geht sogar mit $P = 3$, d.h. $P = 3$ ist optimal :



Berechnung der Periodenschranke

Lemma

Die Periodenschranke eines iterativen Problemgraphen mit n Knoten kann in Zeit $O(|V|\cdot|E|\cdot log(\sum_{v\in V} w(v)))$ berechnet werden.

Beweis:

Betrachte zum Problemgraphen G den kantenbewerteten Graphen $G(P)$, mit $V_{G(P)} = V \cup \{ s \}$, $E_{G(P)} = E \cup \{ (s,v) \mid v \in V \}$ und der Kantenbewertung

$\lambda(e) = 0$, falls $e \in \{ (s,v) \mid v \in V \}$

$\lambda(e) = w(q(e)) - P\cdot s(e)$ sonst.

BF: Löse ein single source longest path Problem mit Quelle s mit dem Bellman Ford Algorithmus in $O(|V|\cdot|E|)$

Keine Lösung: Dann gibt es einen Zyklus Z mit positiver Länge, d.h. $0 < \sum_{e \in Z} \lambda(e) = \sum_{e \in Z} (w(q(e)) - P\cdot s(e))$ also ist $P < P_{min}$

Lösung existiert: Dann ist $P \geq P_{min}$

Bestimme durch Binärsuche in $\leq \log \sum_{v\in V} w(v)$ BF - Schritten P_{min}.

Periodische Ablaufplanung ohne Ressourcenbeschränkung

Erinnerung: periodischer Ablaufplan (t,P) mit $t:V\rightarrow N_0$ und P als Periode.

(!) $\tau(v,k) = t(v)+k\cdot P$

Beobachtung:

Eine Lösung $llp(s,v)$ des längste Wegeproblems aus dem Beweis des letzten Lemmas liefert stets einen gültigen periodischen Ablaufplan vermöge $t(v) = llp(s,v)$, denn für jede Kante e des Problemgraphen gilt

$\forall k:\quad \tau(z(e),k) \geq \tau(q(e),k-s(e)) + w(q(e))$

$\Leftrightarrow \forall k:\quad t(z(e)) + k\cdot P \geq t(q(e)) + k\cdot P - s(e)\cdot P + w(q(e))$

$\Leftrightarrow \quad t(z(e)) \geq t(q(e)) + w(q(e)) - s(e)\cdot P$

$\Leftrightarrow \quad llp(s,z(e)) \geq llp(s,q(e)) + \lambda(e)$

Man konstruiert mit diesem Algorithmus also auch optimale Ablaufpläne.

Allerdings fehlt uns noch eine **Bindung**.

Graphen mit perfekter Rate

Das angegebene Verfahren liefert noch keine Bindung. Wir werden (nicht!) sehen, dass stets eine zyklostatische Bindung zu einem solchen Ablaufplan existiert.

Definition

Ein Problemgraph hat eine **perfekte Rate**, wenn für jeden einfachen Zyklus Z die Summe $\sum_{e\in Z} s(e) = 1$ ist.

Beobachtung:

Hat der Graph G eine perfekte Rate, dann gibt es zu dem Ablaufplan minimaler Periode auch eine statische Bindung.

Beweis:

Binde jede Aufgabe v an eine eigens für v vorgesehene Ressource.

Nach Ablauf einer Iteration ist wegen

$P_{min} \geq max\{ \sum_{e\in Z} w(e) \mid Z \text{ einfacher Zyklus } \} \geq max\{ w(e) \mid e \in E \}$

die Resource zu jeder Aufgabe v auch wieder frei.

Entfaltungstransformation

Da eine zyklostatische Bindung eine Periodizität *K* hat, kann man sie im Grunde gleichsetzen mit einer statischen Bindung bei einem *K*-fach abgerollten oder entfalteten Problemgraphen.

Definition

Zu einem Problemgraph $G=(V,E,s)$ sei der *K*-fach entfaltete Graph $G^{(K)}=(V',E',s')$ wie folgt definiert:

- zu jedem Knoten *v* gibt es *K* Instanzen $v^{(0)},\ldots,v^{(K-1)}$
- zu jeder Kante $e=(u,v)$ gibt es *K* Instanzen $e^{(0)},\ldots,e^{(K-1)}$ mit $e^{(i)}=(u^{(i)},v^{((i+s(e)) \bmod K)})$ mit $s'(e^{(i)})=(i+s(e)) \operatorname{div} K$

Beobachtung:

Der entfaltete Graph $G^{(K)}$ beschreibt die gleichen Datenabhängigkeiten wie *G*, wenn man die Zuordnung $(v^{(i)},n)$ auf $(v,K\cdot n+i)$ vornimmt.

Entfaltungstransformation ff

Beweis:

Sei $e=(u,v)$ eine beliebige Kante in *G* mit Indexverschiebung $s(e)$.

Dann steht *e* für eine Datenabhängigkeit zwischen

(u,r) und $(v,r+s(e))$ für alle $r \in \mathbf{N}_0$

mit $n = r \operatorname{div} K$ und $i = r \bmod K$ also für eine Abhängigkeit zwischen

$(u, K\cdot n+i)$ und $(v, K\cdot(n+s'(e))+j')$,

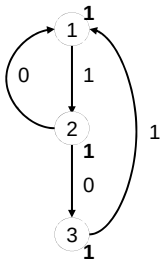
wobei $K\cdot s'(e)+j' = i+s(e)$ und $j' < K$

dies liefern aber gerade die in der Definition benutzten Indizes

$j' = (i+s(e)) \bmod K$ und $s'(e) = (i+s(e)) \operatorname{div} K$ für die Kanten zwischen $u^{(i)}$ und $v^{(j')}$

Beispiel: Entfaltung

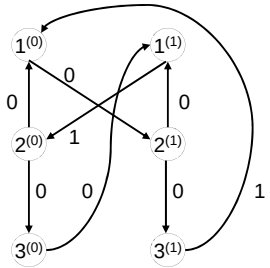
Problemgraph *G*



Periode

$P_{min} = max\{3/2,2/1\}$
$= 2.$

Entfaltung $G^{(2)}$



Periode

$P_{min} = max\{3/1,4/1\}$
$= 4.$

Entfaltung zur perfekten Rate

Beobachtung: -- Parhi und Messerschmitt

Sei $K = kgV\{\sum_{e\in Z} s(e) \mid Z \text{ Zyklus in } G\}$. Dann hat $G^{(K)}$ perfekte Rate.

falsch!

Beweis:

Sei $v_1,\ldots v_n,v_1$ ein beliebiger Zyklus in *G* und sei o.E. $s(v_n,v_1) > 0$.

Sei ferner *S* die Summe der Indexverschiebungen auf diesem Zyklus.

Dann rollt sich dieser Zyklus in $G(K)$ wie folgt ab:

$v_1^{(0)},\ldots,v_n^{(S-s(vn,v1))},v_1^{(S)},\ldots,v_n^{(2\cdot S-s(vn,v1))},v_1^{(2\cdot S)},\ldots$

$\ldots,v_n^{((K/S)\cdot S-s(vn,v1))},v_1^{(0)}$

wobei die einzige Kante mit Verschiebung > *0* die Kante $(v_n^{((K/S)\cdot S-s(vn,v1))}, v_1^{(0)})$ ist. Diese hat Verschiebung *1*.

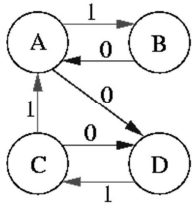
Also hat, da der Zyklus beliebig gewählt war, jeder Zyklus das Gewicht *1*.

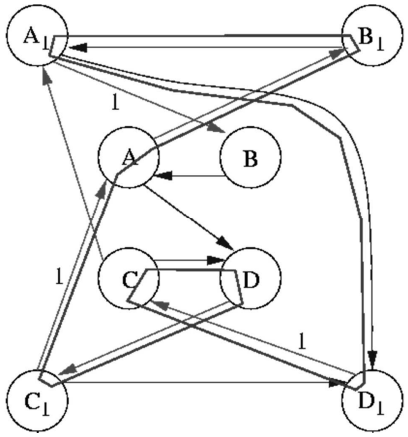
Entfaltung zur perfekten Rate ff

Die Beobachtung ist falsch!

Der Beweis ist unvollständig, weil nur über Zyklen des Ursprungsgraphen argumentiert wird. Es gibt Beispiele, in denen der K-fach abgerollte Graph nicht perfekte Rate hat, weil er Zyklen mit Indexverschiebung *>1* enthält, die nicht zu einfachen Zyklen im Ursprungsgraphen korrespondieren.
Das Ergebnis ist seit 10 Jahren publiziert und in Lehrbüchern zitiert. Scheinbar hat noch niemand zuvor diesen Fehler entdeckt.

Widerlegung durch Wolz





Periodenschranke bei vollstatischer Bindung

Unter der Nebenbedingung einer vollstatischen Bindung kann man die Periodenschranke noch wie folgt verschärfen:

Lemma:

Unter vollstatischer Bindung gilt für die Periode *P* eines gültigen Ablaufplanes zu einem Problemgraphen *G* stets:

$$P \geq P^*_{min} := max\{ P_{min} , max\{ w(v) \mid v \in V \} \}$$

Beweis

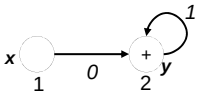
Da jeder Aufgabe *v* in jeder Iteration die gleiche Ressourcen-Instanz zugeordnet wird, muss die Periode größer sein als die Ausführungszeit der Aufgabe selbst.

Transformationen zur Minimierung der Periodenschranke (1)

Betrachte den iterativen Algorithmus (IIR Filter)

$y[n]=a \cdot y[n-1]+x[n] \quad \forall n \geq 0$

Der dazugehörige iterative Problemgraph sieht wie folgt aus und hat eine Iterationsintervallschranke $P_{min}=2$

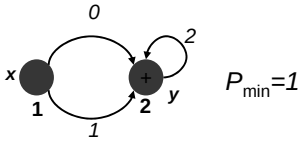


Wie sieht ein optimaler periodischer Ablaufplan dieses Algorithmus‘ aus?

Auswirkung der Vorausberechnungstransformation

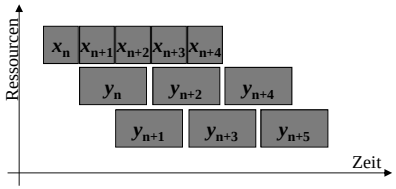
Vorausberechnungstransformation:
Konstruiere einen äquivalenten iterativen Algorithmus durch einmaliges (oder mehrmaliges) rekursives Einsetzen
$\rightarrow y[n]=a^2 \cdot y[n-2]+ a \cdot x[n-1] + x[n] \quad \forall n \geq 0$

Neuer iterativer Problemgraph und die Periodenschranke.



$P_{min}=1$

Wie sieht ein optimaler periodischer Ablaufplan dieses Algorithmus' aus?



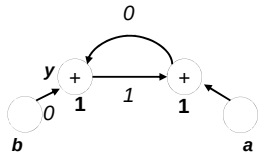
Transformationen zur Minimierung der Periodenschranke (2)

Betrachte den iterativen Algorithmus

$y[n]=y[n-1]+a[n]+b[n] \quad \forall n \geq 0$

Der dazugehörige iterative Problemgraph sieht wie folgt aus

und hat eine Periodenschranke $P_{min}=2$



Versuche einen äquivalenten Problemgraphen mit kleinerer Periodenschranke zu finden!

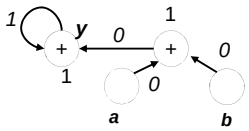
Zyklentransformation
Ausnutzung von Kommutativität, Assoziativität und Distributivität arithmetischer Operatoren

Auswirkung der Zyklentransformationen

Der iterative Algorithmus

$y[n]=y[n-1]+(a[n]+b[n]) \quad \forall n \geq 0$

ist äquivalent zu dem vorherigen Algorithmus



und hat eine Periodenschranke $P_{min}=1$

3.2.5 Periodische Ablaufplanung unter Ressourcenbeschränkung

Periodische Ablaufpläne unter Ressourcenbeschränkung können wieder mit **Integer Linear Programming** (**ILP**), also ganzzahliger linearer Programmierung berechnet werden.

Zunächst wollen wir eine untere Schranke für die Zahl der Ressourcen in Abhängigkeit von der Periode betrachten

Satz -- Processor Bound

Gegeben sei ein iterativer Problemgraph $G = (V,E,s)$ und ein einziger Ressourcetyp (Prozessor) auf dem jeder Knoten v_i in Zeit $w(v_i)$ ausgeführt werden kann. Ferner sei eine Periode P für den Ablaufplan gegeben. Dann gilt für die minimale Zahl von benötigten Ressourcen α_{min}

$$\alpha_{min} = \left\lceil \frac{\sum_i w(v_i)}{P} \right\rceil$$

Das Integer Programm

Sei im folgenden wieder $l_i = \tau^{ASAP}(v_i)$ und $h_i = \tau^{ALAP}(v_i)$, basierend auf einer Ablaufplanung, die nur die mit *0* gewichteten Kanten unter einer Latenzschranke $L > P$ berücksichtigt.

- $x_{i,t} \in \{0,1\}$ $\quad \forall v_i \in V_S \; \forall t$ mit $l_i \le t \le h_i$;
- $\sum_{t=l_i}^{h_i} x_{i,t} = 1$ $\quad \forall v_i \in V_S$
- $\sum_{t=l_i}^{h_i} t \cdot x_{i,t} - \sum_{t=l_j}^{h_j} t \cdot x_{j,t} \ge w_j - s_{j,i} \cdot P$ $\quad \forall (v_j, v_i) \in E_S$
- $\sum_{i \text{ mit } (v_i,k) \in V_R} \; \sum_{p=0}^{w_i - 1} x_{i,(P+t-p) \bmod P} \le \alpha(k)$

$\forall k \in V_T \; \forall t$ mit $1 \le t \le P$

Erläuterung des Integer Programms

- Die binäre Variable $x_{i,t}$ drückt die Ablaufplanung aus. Sie ist genau dann 1, wenn Aufgabe v_i zum Zeitpunkt $t + nP$ gestartet wird, also wenn $\tau(v_i) = t + nP$ gilt.
- Aufgabe v_i darf nur genau einmal pro Periode geplant werden
- Es gilt $\sum_{t=l_i}^{h_i} t \cdot x_{i,t} = \tau(v_i)$ und somit sagt die Bedingung aus, dass Aufgabe v_j frühestens $w_i - s_{i,j} P$ Zeitschritte später als Aufgabe v_i geplant werden darf, wenn es eine Datenabhängigkeit mit Indexverschiebung $s_{i,j}$ zwischen Aufgabe v_i und Aufgabe v_j gibt.
- **Ressourcenbeschränkung:** Man überlege sich, dass die Ressource $\beta(v_i)$ zum Zeitpunkt $t + n P$ durch v_i nur belegt sein kann, wenn $\tau(v_i) \le t \le \tau(v_i) + w_i - 1$ oder
  $\tau(v_i) - P \le t \le \tau(v_i) + w_i - 1 - P$