Pipeline Logiciel.
Un problème d'ordonnancement cyclique avec contraintes de ressources.

- Rappels d'architecture, exemple.
- Modèle de programmes et formulation du problème de pipeline logiciel.
- Problème cyclique de base.
- Avec contraintes de ressources: la technique du modulo scheduling.
- Compaction de code: performances, limitations.
- L'algorithme de Leiserson et Saxe: optimisation du chemin critique.

Optimisations, potentialités des nouvelles architectures

Un peu de vocabulaire:

- Unités pipelinées, unités parallèles.
- Architectures super-scalaires, VLIW, EPIC.
- Déplacement d'instructions à l'exécution.
- Mécanisme de renommage de registres, registres rotatifs.
- Prédiction de branchements.
- Instructions avec prédicats.
- Spéculation sur le contrôle.
- Spéculation sur les données.
- Registres dédiés, etc...

Un exemple: le processeur LANai 3.0 de Myricom

Une unité non parallèle, pipelinée, à 4 étages (mais le "délai" d'une instruction est 1 ou 2).

Trois types principaux d'instructions.

- Accès à la mémoire (load et store): une ombre (shadow) pour le load.
- Branche: une ombre (et l'instruction dans l'ombre EST exécutée).
- Opérations sur les registres: pas d'ombre.

Une seule possibilité de "control hazard":

```
r1 = load(toto)
r1 = r2 + 1
```

C'est l'opération sur les registres qui a priorité.

Un exemple de code généré pour le LANai 3.0

Code initial

```
L400:
   ld[r26], r27
      nop
   add r27, 6740, r26
   ld 0x1A54[r27], r27
      nop
   sub.f r27, r25, r0
   bne L400
      nop
L399:
```

Temps $8 + 8n$.

Code "compacté"

```
L400:
   ld[r26], r27
      nop
   ld 0x1A54[r27], r27
   add r27, 6740, r26
   sub.f r27, r25, r0
   bne L400
      nop
L399:
```

Temps $7 + 7n$.

Code "software pipeliné"

```
   ld[r26], r27
      nop
   add r27, 6740, r26
L400:
   ld 0x1A54[r27], r27
   ld[r26], r27 /* spéculatif */
   sub.f r27, r25, r0
   bne L400
   add r27, 6740, r26 /* spéculatif */
L399:
```

Temps $8 + 5n$.

Modèle (suite)

- Unités parfois pipelinées, parfois non-pipelinées, parfois plus complexes.
- Unités spécifiques ou non.
- Durée d d'une opération: deux significations.
  - durée d'utilisation de la ressource: $d(u)$ pour une opération u.
  - délai entre deux opérations dépendantes: $d(e)$ pour un arc e.
- Distance de dépendances: $w(e)$ pour un arc e.
- Contraintes de registres.

☛ Modèle plus général = graphes avec délais sur sommets et arcs + tables d'utilisation des ressources (+ contraintes de registres).

Ordonnancement cyclique (pipeline logiciel)

Trouver un ordonnancement t tel que:

- $\forall\, e = (u, v)$, $\forall k \in \mathbb{N}$, $t(v, k) \geq t(u, k - w(e)) + d(u)$ (dépendances).
- Les contraintes de ressources sont satisfaites.
- $\lambda = \liminf_{k \to \infty} \dfrac{\max\{t(v, i) + d(v) \mid v \in V,\, i \leq k\}}{k}$ est minimal.

λ est le **temps de cycle moyen** (ou l'"initiation interval"). On cherchera des ordonnancements cycliques: $t(u, k) = \lambda k + \rho_u$. Alors période = temps de cycle moyen.

Bornes inférieures

Soit $T = \max\{t(v,i) + d(v) \mid v \in V,\, i \leq k\}$.

- Si on dispose de p ressources: $pT \geq k \sum_{v \in V} d(v)$ ☛ $\boxed{\lambda \geq \frac{\sum_v d(v)}{p}}$
- Soit C un circuit. $t(v,k) \geq t(v, k - w(C)) + d(C)$, puis $t(v,k) \geq t(v, k - \lfloor \frac{k}{w(C)} \rfloor w(C)) + \lfloor \frac{k}{w(C)} \rfloor d(C)$. Donc $T \geq \lfloor \frac{k}{w(C)} \rfloor d(C)$. On en déduit:

$$\frac{T}{k} \geq (\frac{k}{w(C)} - 1)\frac{d(C)}{k} = (1 - \frac{w(C)}{k})\frac{d(C)}{w(C)} \quad \text{☛} \quad \boxed{\lambda \geq \frac{d(C)}{w(C)}}$$

Sans contrainte de ressources: ☛ $\boxed{\lambda_\infty \geq \max\{\lceil \frac{d(C)}{w(C)} \rceil \mid C \text{ circuit}\}}$

Ordonnancements cycliques

Ordonnancement cyclique de la forme $\sigma(u,k) = \lambda k + \rho_u$, $\lambda \in \mathbb{N}$, $\rho_u \in \mathbb{N}$.

Dépendances $\sigma(v, k + w(e)) \geq \sigma(u,k) + d(u)$, c'est-à-dire
$\rho_v + \lambda w(e) \geq \rho_u + d(u)$. En sommant sur un circuit, on retrouve
$\lambda w(C) \geq d(C)$.

Astuce à la Bellman-Ford Dans le graphe $G'_\lambda = (V, E, w')$ avec
$w'(e) = d(u) - \lambda w(e)$ pour $e = (u,v)$, tous les circuits sont de poids
négatif ou nul si et seulement si $\lambda w(C) \geq d(C)$.

Calcul de λ_∞: complexité $O(|V|(|V|+|E|)\log(|V|d_{\max})$

Pour λ fixé on construit le graphe $G'_\lambda = (V, E, w')$ avec $w'(e) = d(u) - \lambda w(e)$ si $e = (u, v) \in E$.

Algorithme de Bellman-Ford Si G' a un circuit de poids strictement positif alors λ est trop petit, sinon λ est trop grand.

Recherche dichotomique dans l'intervalle $[0; \sum d(v)]$.

Pour λ_∞ On note ρ_v le plus long chemin arrivant en v dans G'_λ. On a alors $\rho_v \geq \rho_u + w'(e)$, c'est-à-dire:

$$\rho_v + \lambda w(e) \geq \rho_u + d(u)$$

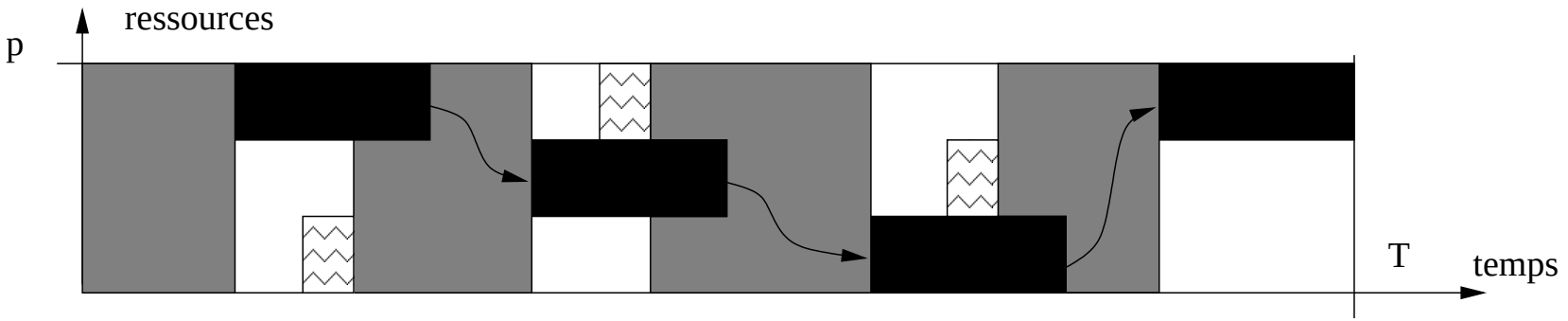
☞ l'ordonnancement cyclique $\sigma(u, k) = \rho_u + \lambda k$ est optimal!

Note calculer $\frac{d(C)}{w(C)}$ au lieu de $\lceil \frac{d(C)}{w(C)} \rceil$ est possible mais plus compliqué.

DAGs, "list scheduling" et borne de Graham

$T_l(p)$: temps d'exécution d'un ordonnancement de liste pour p ressources.

- Pour tout chemin $\mathcal{P}$, $T_l(p) \geq d(\mathcal{P})$ (durée du chemin).
- "Surface" de calcul: $pT_l(p) =$ Idle time $+$ Travail $(= \sum_{v \in V} d(v))$.
- Il existe un chemin $\mathcal{P}_0$ tel que Idle time $\leq (p-1)d(\mathcal{P}_0)$.



Conséquence pour l'ordonnancement acyclique:

$pT_l(p) \leq (p-1)d(\mathcal{P}_0) + \sum_{v \in V} d(v)$ ☛ $\boxed{T_l(p) \leq (2 - \frac{1}{p})T_{\text{opt}}(p)}$

Borne de Graham pour l'ordonnancement de liste

$T_l(p)$: temps d'exécution d'un ordonnancement de liste pour p ressources.

- Pour tout chemin $\mathcal{P}$, $T_l(p) \geq d(\mathcal{P})$ (durée du chemin).
- "Surface" de calcul: $pT_l(p) =$ Idle time + Travail ($= \sum_{v \in V} d(v)$).
- Il existe un chemin $\mathcal{P}_0$ tel que Idle time $\leq (p-1)d(\mathcal{P}_0)$.

Conséquence pour l'ordonnancement acyclique

$pT_l(p) \leq (p-1)d(\mathcal{P}_0) + \sum_{v \in V} d(v)$ ☛ $\boxed{T_l(p) \leq (2 - \frac{1}{p})T_{\text{opt}}(p)}$

Conséquence sur la compaction de boucle

$$\lambda = T_l(p) \leq \tfrac{p-1}{p} d(\mathcal{P}_0) + \underbrace{\frac{\sum_{v \in V} d(v)}{p}}_{\leq \lambda_p}$$

☛ $\boxed{\lambda \leq (1 - \frac{1}{p})d(\mathcal{P}_0) + \lambda_p}$

Choix du décalage

Idée double:

- Minimiser le chemin critique pour la compaction.

  ☛ Équivalent à minimiser la durée maximale d'un chemin n'ayant que des arcs de poids nul (**période d'horloge**). Algorithme de Leiserson et Saxe, complexité $O(VE\log(V))$.

- Minimiser le nombre de contraintes pour la compaction.

  ☛ Équivalent à miminiser le **nombre d'arcs de poids nul**. Variante de l'algorithme out-of-kilter, complexité $O(E^2)$ (version de base) ou $O(V^2E)$ sous contrainte de période d'horloge.

Retiming et circuits

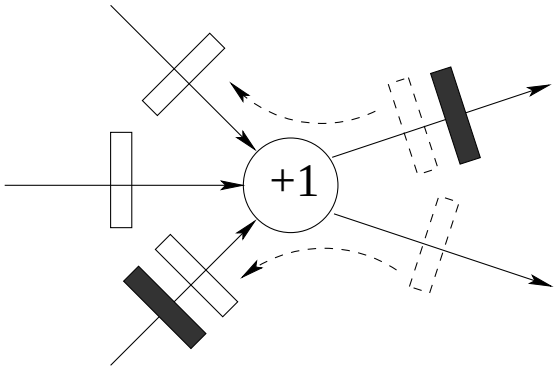
Correspondances Décalage = retiming. Graphe = circuit.

$w(e)$ = nombre de registres sur l'arc e. $d(u)$ = durée de l'opérateur u.

Période d'horloge $\Phi(G)$ = durée maximale d'un chemin sans registres.

Retiming Fonction $r : V \to \mathbb{Z}$ telle que, pour tout arc $e = (u, v)$,

$w_r(e) = w(e) + r(v) - r(u) \geq 0$ (nombre positif de registres).



But Trouver r tel que $\Phi(G_r)$ soit minimal.

L'algorithme de Leiserson et Saxe (1/2)

On calcule:

- $W(u,v) = \min\{w(\mathcal{P}) \mid u \xrightarrow{\mathcal{P}} v\}$.
- $D(u,v) = \max\{d(\mathcal{P}) \mid u \xrightarrow{\mathcal{P}} v \text{ et } w(\mathcal{P}) = W(u,v)\}$.

☛ Floyd-Warshall (all-pairs shortest-paths): $O(V^3)$.

Propriété 1 $\Phi(G) = D(u,v)$ pour certains u et v.

Propriété 2 $\Phi(G) \leq c \Leftrightarrow (D(u,v) > c \Rightarrow W(u,v) \geq 1)$.

Propriété 3 $W_r(u,v) = W(u,v) + r(v) - r(u)$ et $D_r(u,v) = D(u,v)$.

L'algorithme de Leiserson et Saxe (2/2)

Conclusion

Il existe un retiming r tel que $\phi(G_r) \leq c$ ssi $r(v) - r(u) + W(u,v) \geq 1$ lorsque $D(u,v) > c$.

Algorithme

- Nouveaux arcs (dits d'horloge): de u vers v de poids $W(u,v) - 1$ lorsque $D(u,v) > c$ (en plus des arcs initiaux).
- Existence d'un retiming ssi pas de circuit de poids strictement négatif. ☛ Bellman-Ford: $O(VE) = O(V^3)$.
- Emploi des deux premières étapes, par recherche dichotomique sur les quantités $D(u,v)$ pré-calculées.

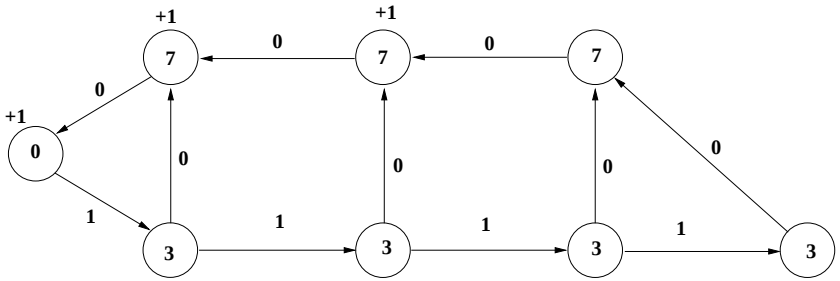
☛ Complexité totale $O(V^3 \log(V))$. Amélioration possible en $O(VE \log(V))$.

Variante optimisée

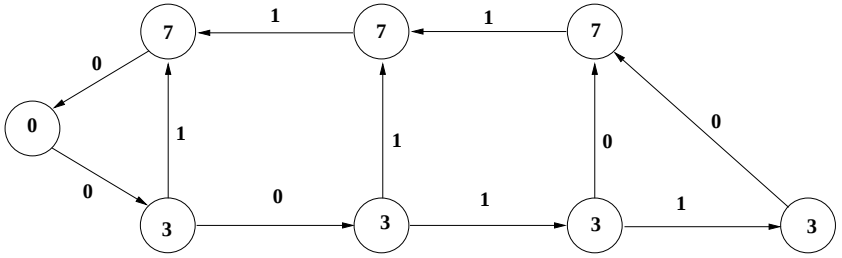
1. Pour tout sommet $v \in V$, $r(v) = 0$.
2. Répéter $V - 1$ fois:
   (a) Calculer $\Delta(v)$ la durée maximale d'un chemin sans registres dans G_r d'extrémité v.
   (b) Pour tout v tel que $\Delta(v) > c$, $r(v) = r(v) + 1$.
3. Si $\Phi(G_r) > c$, impossible d'atteindre c, sinon r est le retiming désiré.

Algorithme de Leiserson et Saxe, exemple

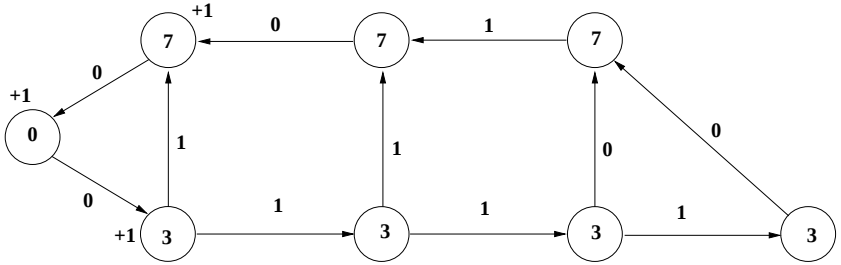
Période 24 et 1er décalage:



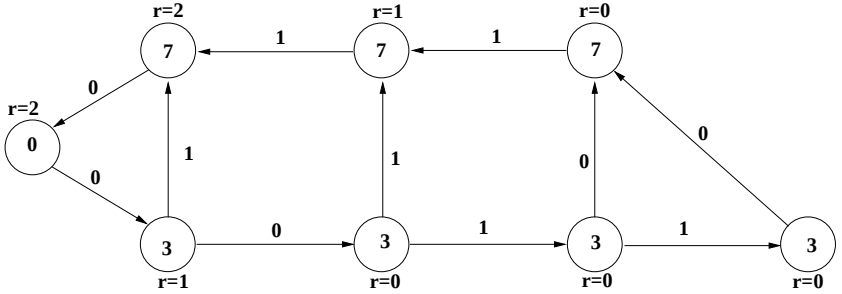
Période finale 13:



Période 17 et 2nd décalage:



Décalage total:



Retour sur le pipeline logiciel

Décalage à la Leiserson-Saxe + compaction par liste:

(i) $\lambda \leq (1 - \frac{1}{p})d(\mathcal{P}_0) + \lambda_p$ donc $\boxed{\lambda \leq (1 - \frac{1}{p})\Phi_{\text{opt}}(G) + \lambda_p}$.

(ii) Avec $s(v)$ durée max. d'un chemin $\rightsquigarrow v$, sans registres dans G_r,
$\sigma(v,k) = s(v) + (r(v) + k)\Phi(G_r)$ ordonnancement: $\boxed{\lambda_\infty \leq \Phi_{\text{opt}}(G)}$.

(iii) Pour ordo. optimal $\sigma_\infty(v,k) = s(v) + \lambda_\infty(r(v) + k)$ avec
$0 \leq s(v) < \lambda_\infty$, tout chemin $\mathcal{P}$ de u_1 à u_n, sans registres dans G_r, vérifie
$d(\mathcal{P}) \leq s(u_n) + d(u_n) - s(u_1) < \lambda_\infty + \max\{d(u) \mid u \in V\}$.
Donc $\boxed{\Phi_{\text{opt}}(G) \leq \lceil \lambda_\infty \rceil + \max\{d(u) - 1 \mid u \in V\}}$.

☞ $\boxed{\lambda \leq (2 - \frac{1}{p})\lambda_p + \max\{d(u) - 1 \mid u \in V\}}$

Note sur les registres (au sens circuits)

$S(G_r)$: nombre de registres après décalage par r.

$$\begin{aligned} S(G_r) &= \sum_{e \in E} w_r(e) = \sum_{u \xrightarrow{e} v} w(e) + r(v) - r(u) \\ &= S(G) + \sum_{v \in V} r(v)(\text{indegree}(v) - \text{outdegree}(v)) \end{aligned}$$

☛ Programme linéaire en nombres entiers:

$$\min\{\sum_{v \in V} r(v)c(v) \mid \forall e = (u,v) \in E,\ w(e) + r(v) - r(u) \geq 0\}$$

dual d'un algorithme de flot (matrice de contrainte = matrice d'incidence de graphe) et soluble en temps polynomial:

$$\max\{\sum_{e \in E} f(e)w(e) \mid \forall v \in V, \sum_{v \xrightarrow{e}} f(e) - \sum_{\xrightarrow{e} v} f(e) = c(v), \forall e \in E,\ f(e) \geq 0\}$$

Autre ex., minimisation du nombre d'arcs de poids nul

(Note: maximisation = NP-complet au sens fort)

Coût d'un arc pour un décalage r: $c_r(e) = \begin{cases} 1 \text{ si } w_r(e) = 0 \\ 0 \text{ sinon} \end{cases}$

Étant donné un décalage r et un flot f: $z_{f,r}(e) = \begin{cases} 0 \text{ si } f(e) = 0 \\ 1 - f(e)w_r(e) \text{ sinon} \end{cases}$

Indice de conformité: $i_{f,r}(e) = c_r(e) - z_{f,r}(e) \geq 0$.

(i) Le coût d'un flot est invariant par décalage.

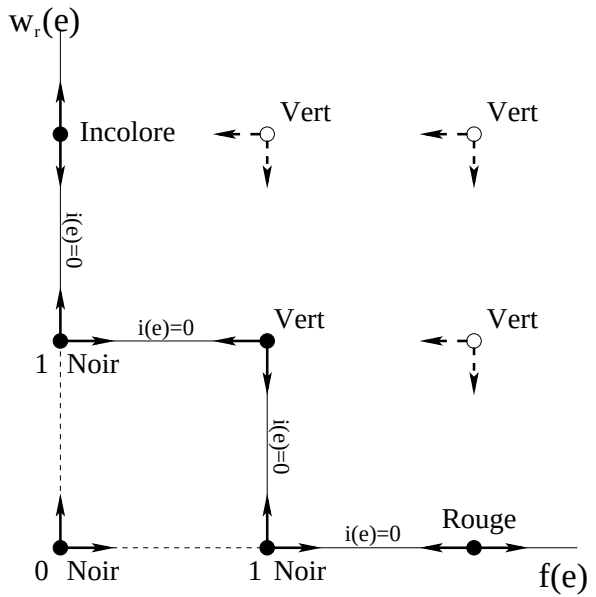
(ii) Le coût d'un décalage est toujours supérieur au coût d'un flot.

(iii) S'il existe un décalage r et un flot f tels que les indices de conformité sont nuls, ils correspondent à une solution optimale.

Coloration des arcs

But: $i_{f,r}(e) = 0$ pour tout arc e. On **colorie** les arcs pour indiquer de quelle manière $i_{f,r}(e)$ peut décroître.

☛ Diagramme de conformité.



On part du flot et du décalage nuls: arcs non conformes initiaux = arcs de poids nul (noirs). On doit:

- faire décroître $i_{f,r}(e)$ sur tout arc non conforme;
- conserver $i_{f,r}(e) = 0$ sur les autres.

Lemme de Minty

Soit $G = (V, E)$ un graphe orienté dont les arcs sont noirs, verts, rouges ou incolores. Soit e_0 un arc noir. Alors une des deux propositions suivantes est vraie:

(i) Il existe un cycle contenant e_0, sans arcs incolores, et dont tous les arcs noirs sont dans le même sens et tous les arcs verts dans le sens contraire.

(ii) Il existe un co-cycle contenant e_0, sans arcs rouges, dont tous les arcs noirs sont dans le même sens et tous les arcs verts dans le sens contraire.

Démonstration constructive par simple marquage en suivant à partir de e_0 les arcs noirs dans le sens de e_0, les arcs verts dans le sens contraire, les arcs rouges dans les deux sens.

Algorithme

Colorer les arcs en fonction du diagramme de conformité pour $r = f = 0$.
Tant qu'il existe des arcs non conformes:

1. Choisir un arc non conforme $e_0 = (u, v)$ et, grâce au lemme de Minty,

   (i) incrémenter (resp. décrémenter) le flot des arcs du cycle qui sont orientés comme e_0 (resp. en sens inverse à e_0);

   (ii) incrémenter le décalage des sommets du sous-ensemble contenant v défini par le co-cycle;

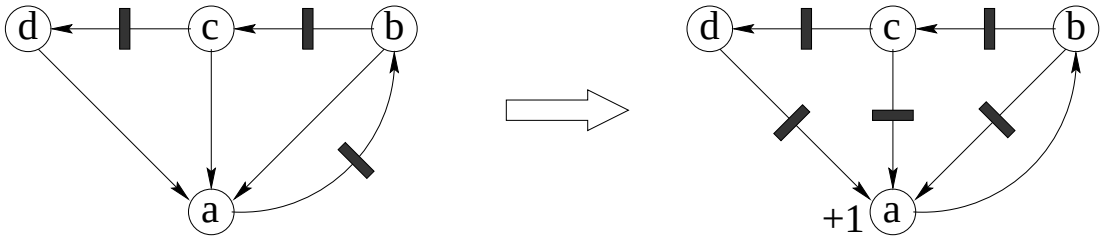
   ce qui permet de rendre e_0 conforme sans créer d'arc non conforme.

2. Recolorer les arcs en fonction du diagramme de conformité et des nouveaux r et f;

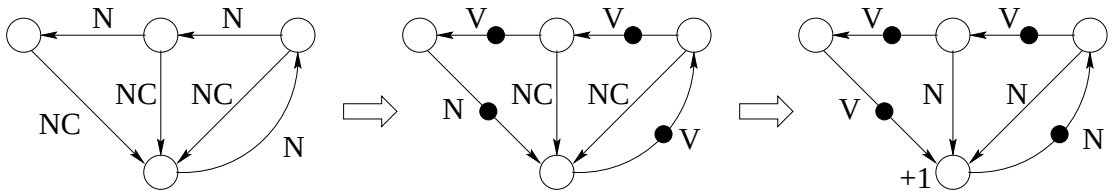
☛ Complexité: $O(|E|(|E| + |V|))$. Possibilité de rajouter des arcs d'horloge.

Petit exemple

Résultat sur un exemple:



après deux étapes:



● Flot unitaire
NC Arc non conforme (noir)
V Arc vert
N Arc noir

Quelques problèmes de décalage voisins

Pour la fusion de boucles parallèles décalage pour obtenir moins de K boucles: NP-complet (mais polynomial pour $K = 1$).

Pour la détection de boucles parallèles décalage en une dimension polynomial (cf $K = 1$ ci-dessus), mais NP-complet pour deux (ou plus) boucles imbriquées.

Pour l'amélioration de la localité NP-complet.

Pour limiter la pression registres NP-complet mais heuristique garantie possible par programmation linéaire.

☛ Mais toujours des approches possibles par programmation linéaire en nombres entiers.