Outline

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

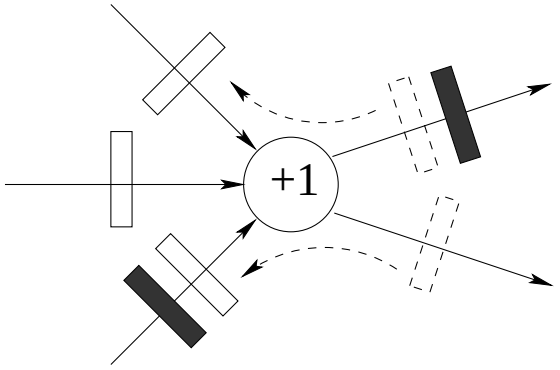
Correspondances Décalage = retiming. Graphe = circuit.

$w(e)$ = 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 $O(VE \log(V))$.

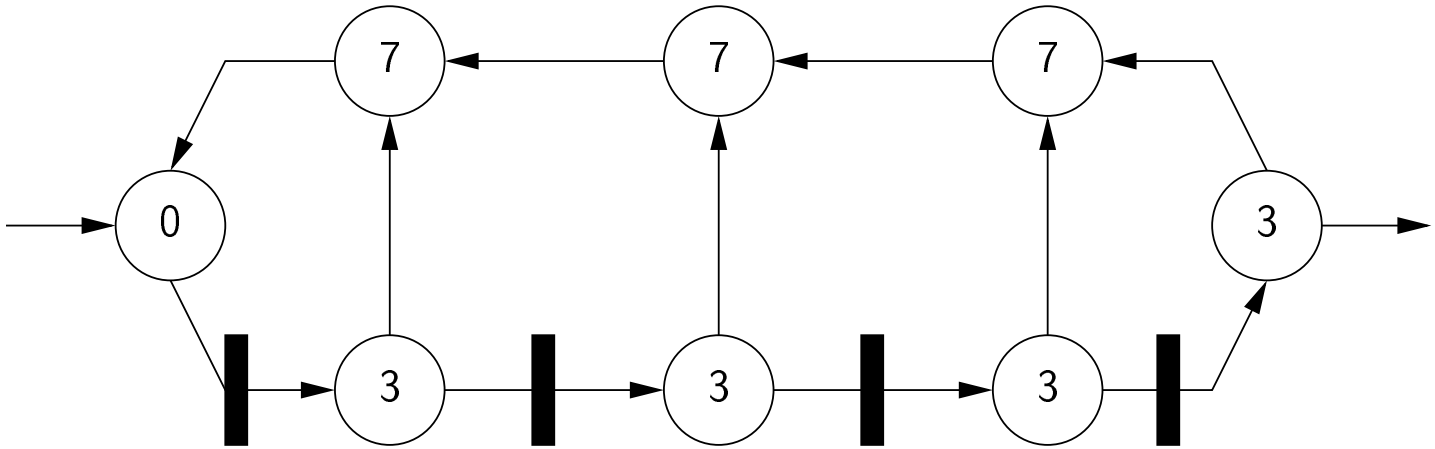
Variante optimisée

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.

Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

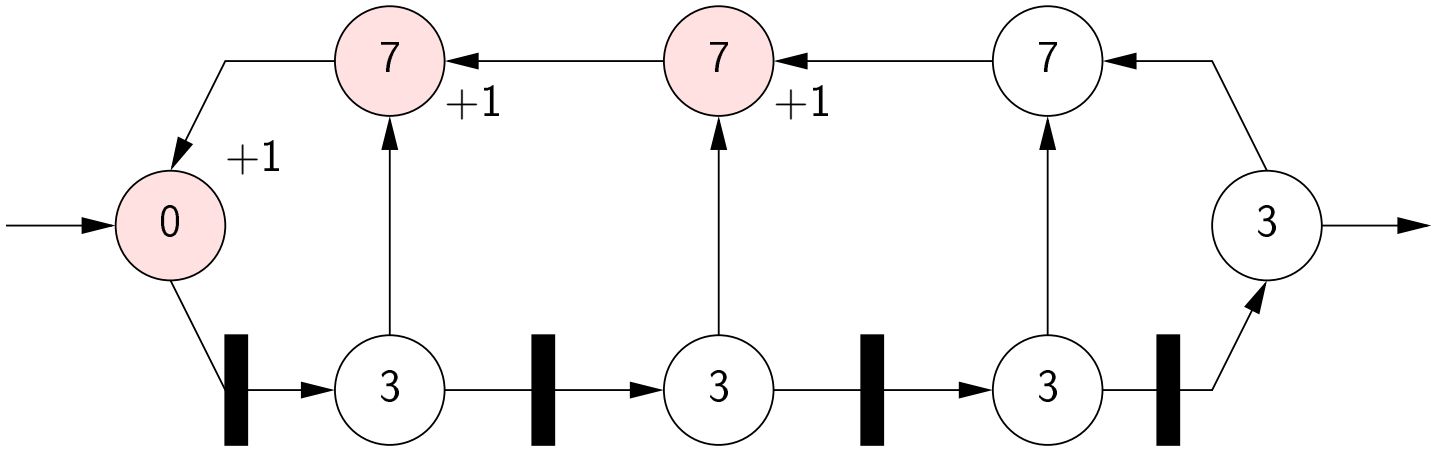
Initial circuit : $\phi(G) = 24$



Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

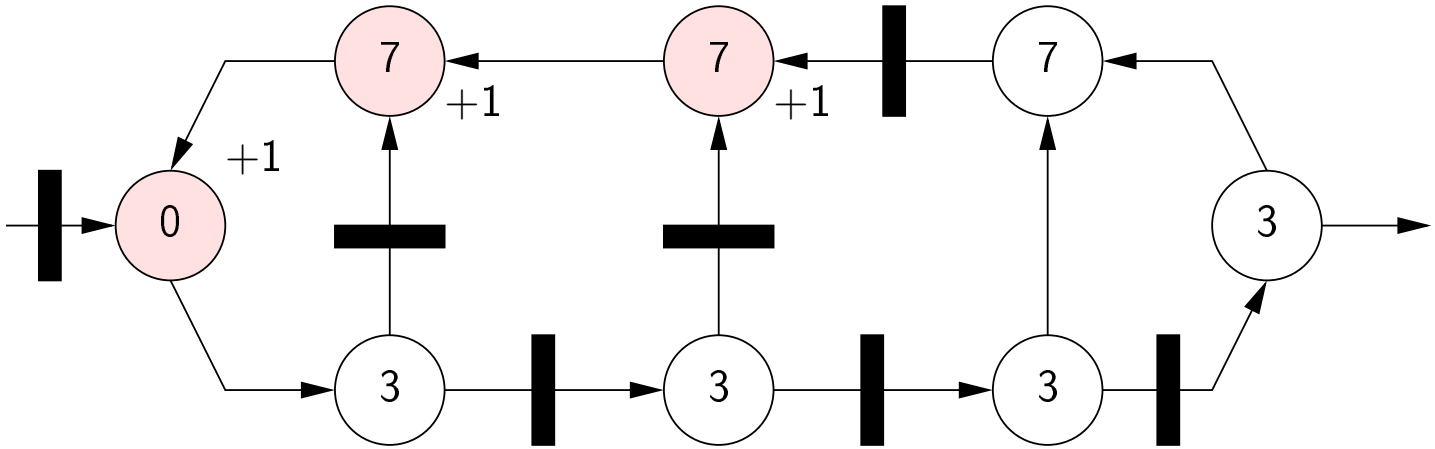
First retiming to apply targeting $\phi(G) = 13$



Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

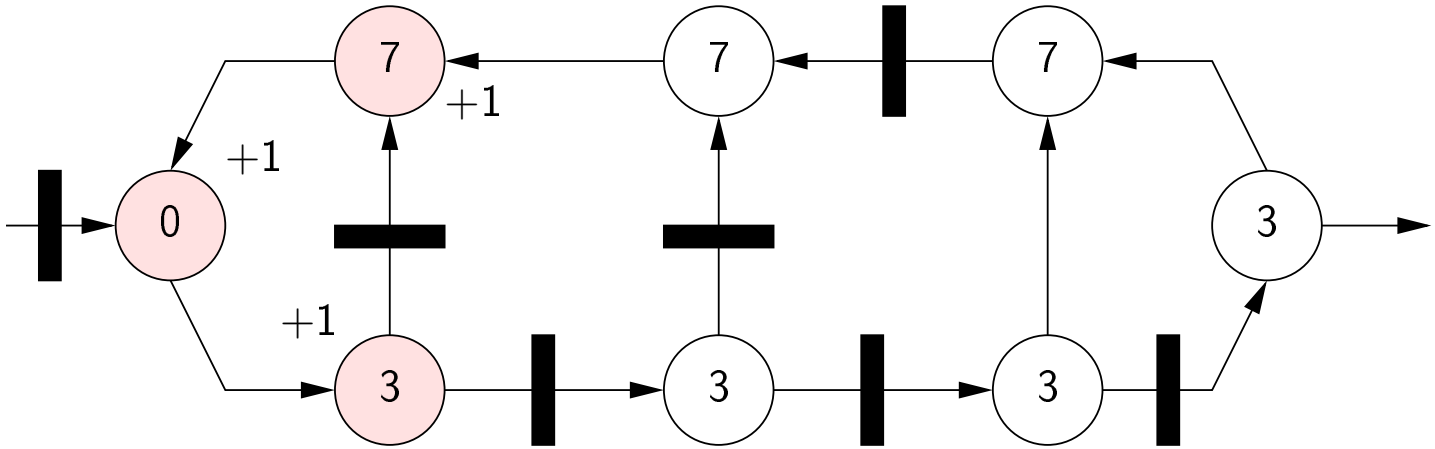
First retiming applied : $\phi(G) = 17$



Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

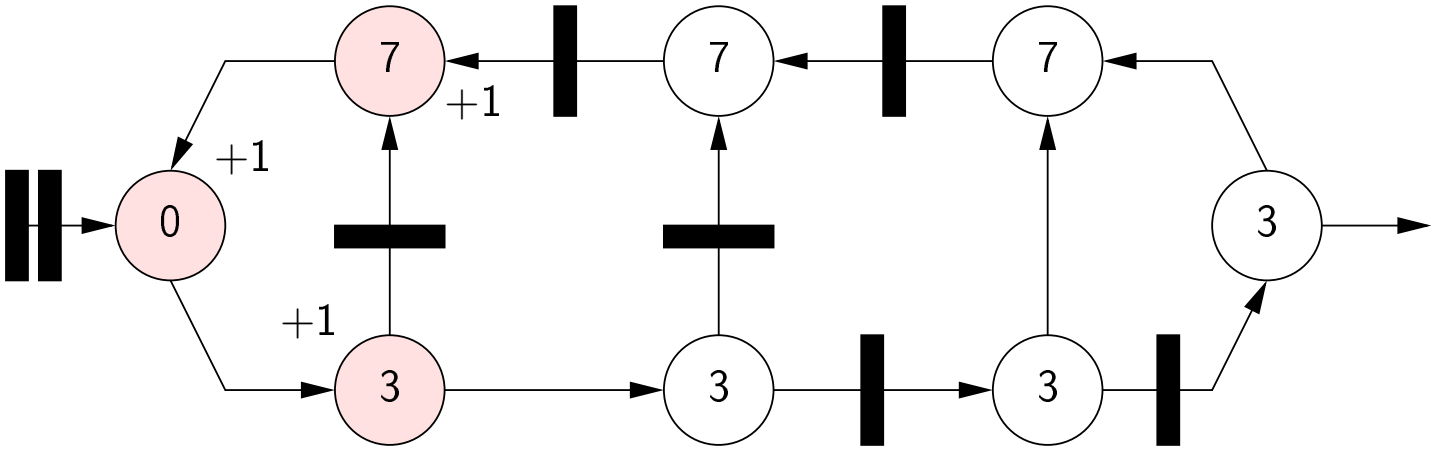
Second retiming to apply targeting $\phi(G) = 13$



Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

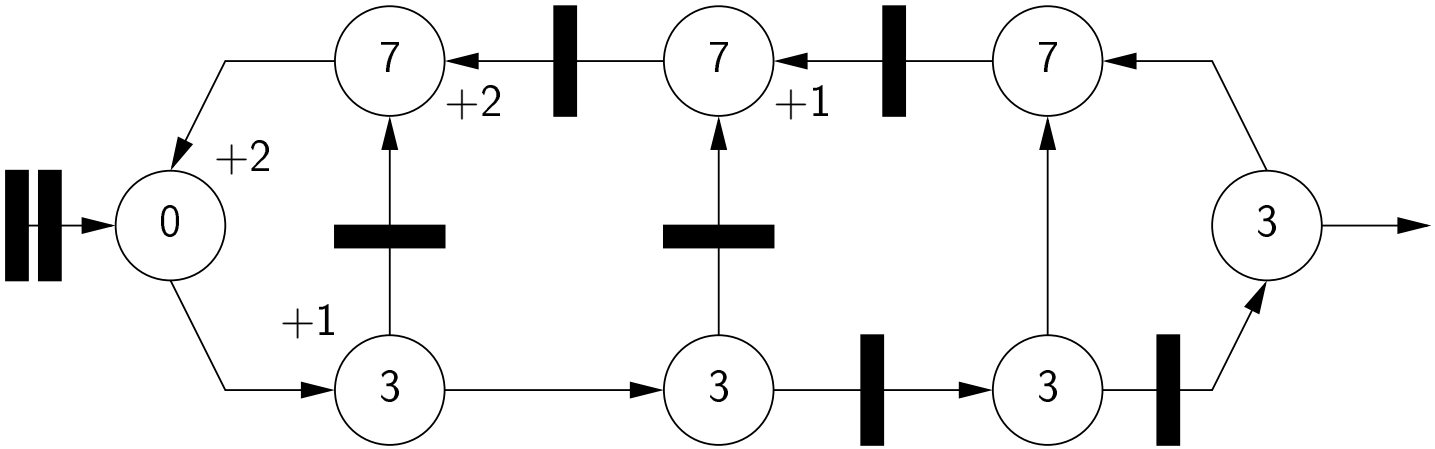
Second retiming applied : $\phi(G) = 17$



Leiserson-Saxe circuit retiming (1991)

Delay operators : move registers from out-going to in-going edges.
Retiming changes number of registers & clock period of the circuit.

Complete optimal retiming : $\phi(G) = 13$



➽ Nice $O(VE \log(V))$ algorithm for clock period minimization.
Similar concepts used in software pipelining and loop shifting.

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 $\boxed{\Phi_{\text{opt}}(G) \leq \lceil \lambda_\infty \rceil + \max\{d(u) - 1 \mid u \in V\}}$.

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

Programmation linéaire

$$\min \left\{ \begin{array}{l} 11t + 10u \mid \\ \quad 2t + 3u \geq 5 \\ \quad 3t + 2u \geq 4 \\ \quad 5t + u \geq 12 \\ \quad t >= 0,\ u >= 0 \end{array} \right. \quad = \quad \max \left\{ \begin{array}{l} 5x + 4y + 12z \mid \\ \quad 2x + 3y + 5z \leq 11 \\ \quad 3x + 2y + z \leq 10 \\ \quad x \geq 0,\ y \geq 0,\ z \geq 0 \end{array} \right.$$

Problème 1 du dopé acheter la proportion la moins chère de dopants t et u (de prix respectifs 11 et 10) pour un apport suffisant en 3 éléments de base (resp. 5, 4 et 12) connaissant l'apport dans chaque dopant.

Problème 2 du trafiquant vendre les trois éléments de base au prix le plus cher, tout en étant moins cher que chacun des dopants.

Programmation linéaire

Théorème de dualité

$$\min\{c.x \mid Ax \geq b,\ x \geq 0\} = \max\{y.b \mid yA \leq c,\ y \geq 0\}$$

Complexité

- Solution rationnelle : temps polynomial (L. Khachiyan)
- Solution entière : NP-complet et inégalité seulement.

Algorithmes Algorithmes du simplexe, algorithmes de réduction de base en petites dimensions (Lenstra), programmation linéaire paramétrée (second membre).

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:

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

Approche primale-duale

Retiming Fonction $r : V \to \mathbb{Z}$ telle que $\forall\, e = (u, v)$, $r(v) - r(u) + w(e)$.

☛ "On fait passer un même nombre de registres de chacun des arcs entrants vers chacun des arcs sortants".

Flot positif (= union de circuits)

Fonction $f : E \to \mathbb{Z}$ telle que $\forall\, v \in V, \sum_{.\xrightarrow{e} v} f(e) = \sum_{v \xrightarrow{e} .} f(e)$.

☛ "On entre dans un sommet autant de fois qu'on en sort".

Principe Donner des coûts $\Gamma(r)$ et $\Sigma(f)$ aux retimings et flots pour que:

- $\Gamma(r) \geq \Sigma(f)$ quels que soient r et f,
- Il existe r et f tels que $\Gamma(r) = \Sigma(f)$.

Exemple: minimisation du nombre d'arcs de poids nul

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

Retiming On pose $v_r(e) = 0$ si $w_r(e) > 0$ et $v_r(e) = 1$ si $w_r(e) = 0$.

On cherche à minimiser $\boxed{\Gamma(r) = \sum_{e \in E} v_r(e)}$.

Flot positif On pose $z_f(e) = 0$ si $f(e) = 0$ et $z_f(e) = 1 - f(e)w_r(e)$ sinon.

On cherche à maximiser $\boxed{\Sigma(f) = \sum_{e \in E} z_f(e)}$.

Propriété 1 $\displaystyle\sum_{e \in E} f(e)w(e) = \sum_{e \in E} f(e)w_r(e)$ ☛ $\Sigma(f)$ ne dépend pas de r.

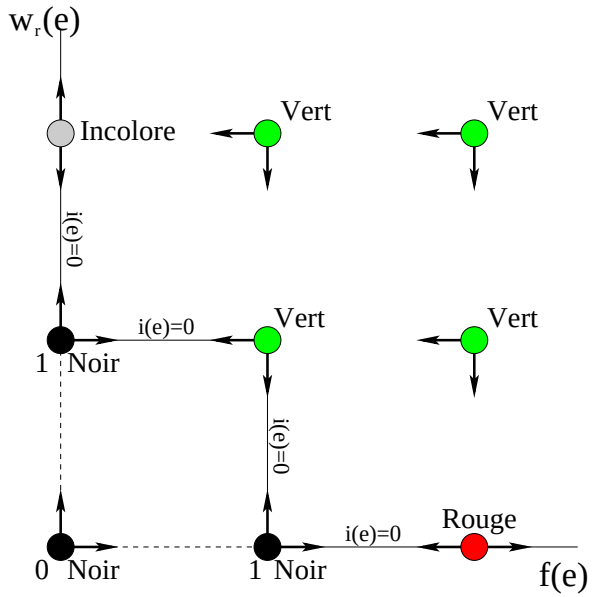
Propriété 2 $v_r(e) \geq z_f(e)$ ☛ indice de conformité $ic(e) = v_r(e) - z_f(e) \geq 0$.

Propriété 3 $\forall\, r$ et f, $\Gamma(r) \geq \Sigma(f)$ et optimal ssi $\forall\, e \in E, ic(e) = 0$.

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. Une des 2 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;

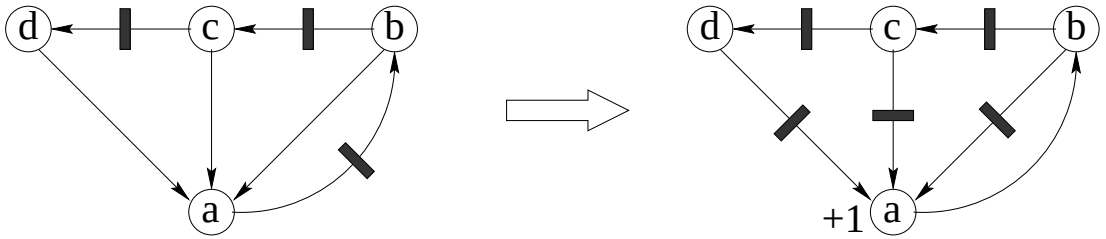
   ce qui permet de rendre e_0 conforme sans créer d'arc non conforme.

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

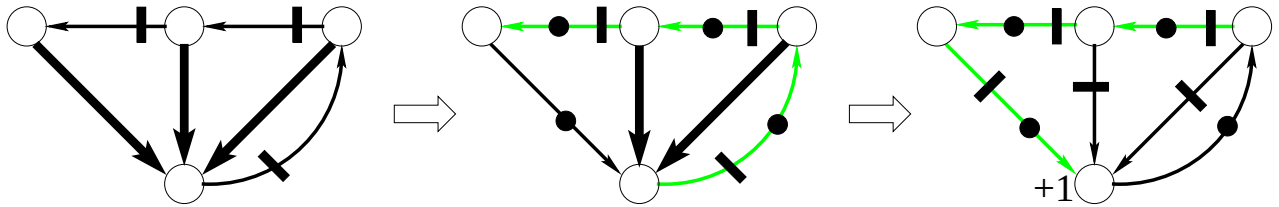
☛ Complexité: $O(|E|(|E| + |V|))$. Possibilité de rajouter des arcs d'horloge.

Petit exemple

Résultat sur un exemple:



après deux étapes:



Flot unitaire

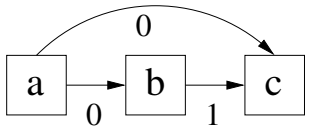
Arc non conforme (noir)

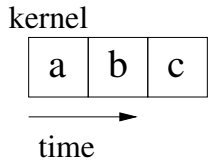
Outline

Pipeline logiciel et durées de vie. Ex : Maxlive

Maxlive = maximum number of simultaneous live values.

Dependence graph
+ fixed allocation.

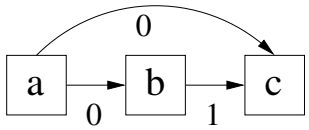


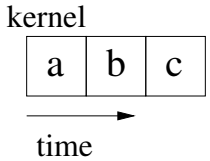


Pipeline logiciel et durées de vie. Ex : Maxlive

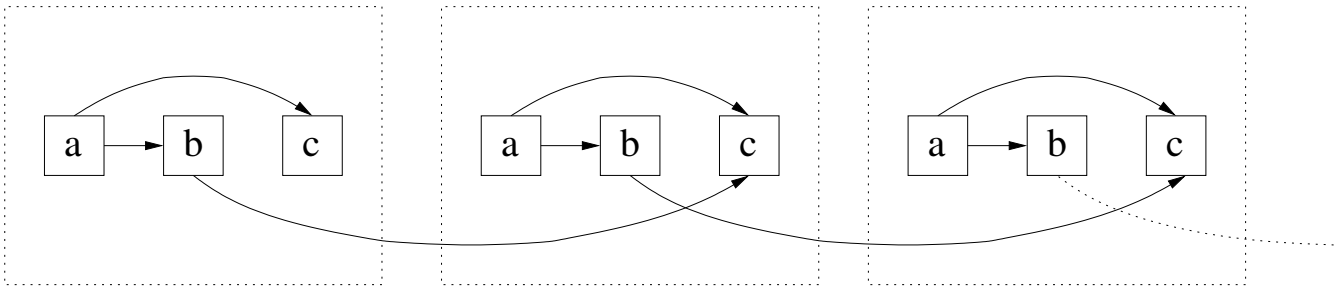
Maxlive = maximum number of simultaneous live values.

Dependence graph
+ fixed allocation.





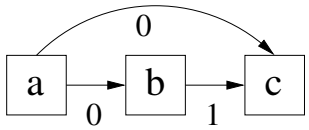
Maxlive = 3 (after b and before c) :

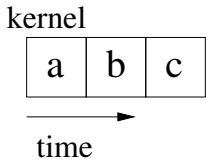


Pipeline logiciel et durées de vie. Ex : Maxlive

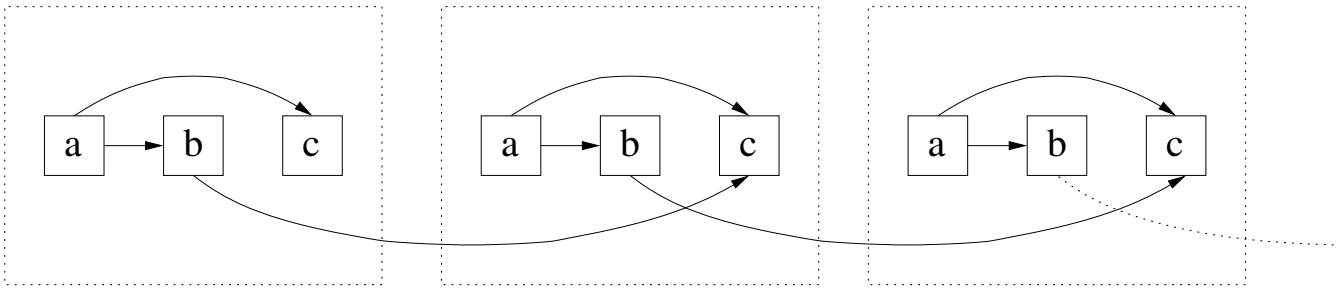
Maxlive = maximum number of simultaneous live values.

Dependence graph
+ fixed allocation.

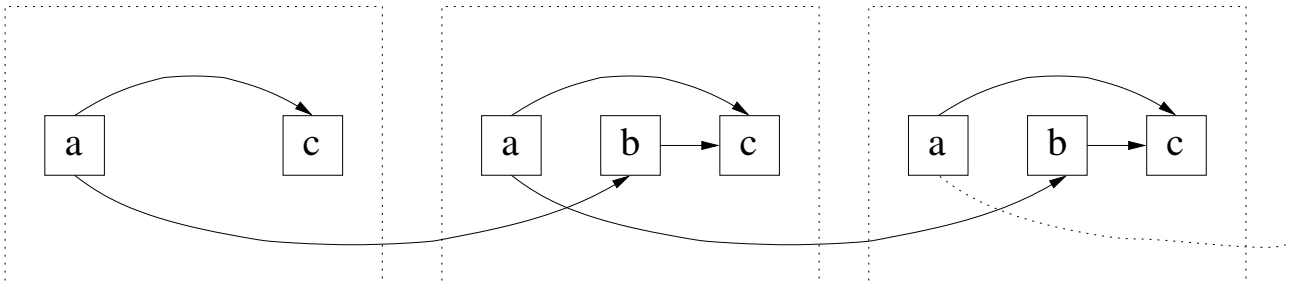




Maxlive = 3 (after *b* and before *c*) :



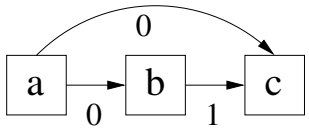
Maxlive = 2, same allocation, *b* delayed by 1 :

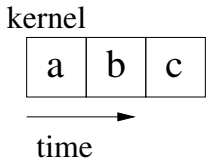


Pipeline logiciel et durées de vie. Ex : Maxlive

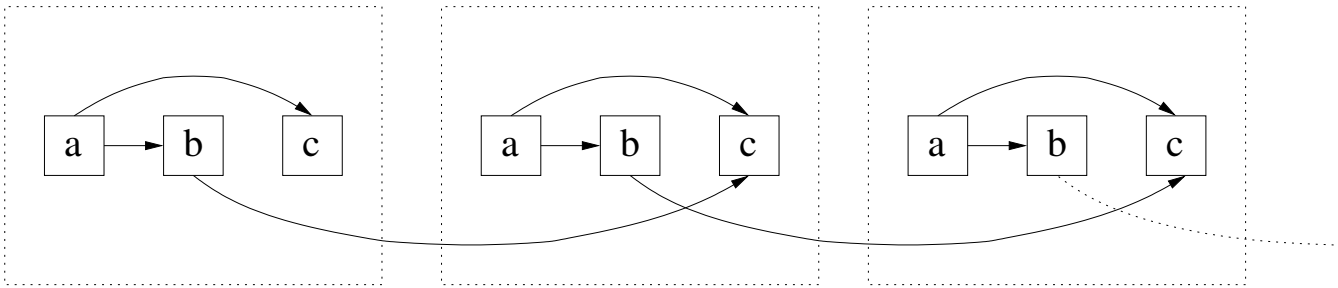
Maxlive = maximum number of simultaneous live values.

Dependence graph
+ fixed allocation.

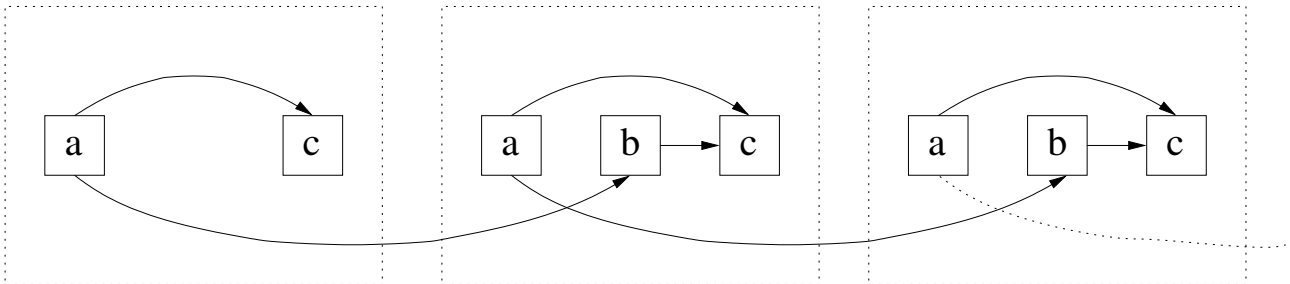




Maxlive = 3 (after b and before c) :



Maxlive = 2, same allocation, b delayed by 1 :



➽ Retiming changes both Maxlive and FIFO sizes (if used).

FIFO sizes : general constraints

More accurate use of memory

For each $e = (u, v) \in E$, define $t_{\text{out}}(u)$ and $t_{\text{in}}(e)$ such that :

- if $t \geq \sigma(u, i) + t_{\text{out}}(u)$, the result of (u, i) is available.
- if $t \geq \sigma(v, i) + t_{\text{in}}(e)$, the value was already read by (v, i).

By definition of delays d, $d(e) > t_{\text{out}}(u) - t_{\text{in}}(e)$.

FIFO sizes : general constraints

More accurate use of memory

For each $e = (u, v) \in E$, define $t_{\text{out}}(u)$ and $t_{\text{in}}(e)$ such that :

- if $t \geq \sigma(u, i) + t_{\text{out}}(u)$, the result of (u, i) is available.
- if $t \geq \sigma(v, i) + t_{\text{in}}(e)$, the value was already read by (v, i).

By definition of delays d, $d(e) > t_{\text{out}}(u) - t_{\text{in}}(e)$.

Reminder

for a modulo schedule $\sigma(u, i) = \lambda.i + \rho_u$, the dependence constraint is expressed as :

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

FIFO sizes : general constraints

More accurate use of memory

For each $e = (u, v) \in E$, define $t_{\text{out}}(u)$ and $t_{\text{in}}(e)$ such that :

- if $t \geq \sigma(u, i) + t_{\text{out}}(u)$, the result of (u, i) is available.
- if $t \geq \sigma(v, i) + t_{\text{in}}(e)$, the value was already read by (v, i).

By definition of delays d, $d(e) > t_{\text{out}}(u) - t_{\text{in}}(e)$.

Reminder

for a modulo schedule $\sigma(u, i) = \lambda.i + \rho_u$, the dependence constraint is expressed as :

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

FIFO at least $k + 1$ locations if :

$\sigma(u, i + k) + t_{\text{out}}(u) \leq \sigma(v, i + w(e)) + t_{\text{in}}(e)$, i.e.,
$\lambda.k + \rho_u + t_{\text{in}}(e) \leq \lambda.w(e) + \rho_v + t_{\text{out}}(u)$

➽ FIFO of size at least $1 + w(e) + \lfloor \frac{\rho_v - \rho_u + t_{\text{out}}(u) - t_{\text{in}}(e)}{\lambda} \rfloor$.

Linear system

Constraints With retiming $r(u)$:
$\rho_u = \lambda.r(u) + s_u$, with $0 \leq s_u < \lambda$ (s_u is fixed).

- $r(v) - r(u) \geq \lceil \frac{d(e)+s_u-s_v}{\lambda} \rceil - w(e) = d_1(e)$.
- $1 + w(e) + \lfloor \frac{\rho_v-\rho_u+t_{\text{out}}(u)-t_{\text{in}}(e)}{\lambda} \rfloor = r(v) - r(u) + d_2(e)$.

Linear system

Constraints With retiming $r(u)$:
$\rho_u = \lambda.r(u) + s_u$, with $0 \leq s_u < \lambda$ (s_u is fixed).

- $r(v) - r(u) \geq \lceil \frac{d(e)+s_u-s_v}{\lambda} \rceil - w(e) = d_1(e)$.
- $1 + w(e) + \lfloor \frac{\rho_v-\rho_u+t_{out}(u)-t_{in}(e)}{\lambda} \rfloor = r(v) - r(u) + d_2(e)$.

Linear program

$$\min\{\sum_{u \in V} M(u) \mid r(v) - r(u) \geq d_1(e),\ M(u) \geq r(v) - r(u) + d_2(e)\}$$

Linear system

Constraints With retiming $r(u)$:
$\rho_u = \lambda.r(u) + s_u$, with $0 \leq s_u < \lambda$ (s_u is fixed).

- $r(v) - r(u) \geq \lceil \frac{d(e)+s_u-s_v}{\lambda} \rceil - w(e) = d_1(e)$.
- $1 + w(e) + \lfloor \frac{\rho_v - \rho_u + t_{out}(u) - t_{in}(e)}{\lambda} \rfloor = r(v) - r(u) + d_2(e)$.

Linear program

$$\min\{\sum_{u \in V} M(u) \mid r(v) - r(u) \geq d_1(e),\ M(u) \geq r(v) - r(u) + d_2(e)\}$$

Variant Add new vertex u' for each u and $r(u') = M(u) + r(u)$:

$$\min\{\sum_{u \in V} r(u') - r(u) \mid r(v) - r(u) \geq d_1(e),\ r(u') - r(v) \geq d_2(e)\}$$

➽ Connection matrix = totally unimodular = polynomial problem

With shared storage, e.g., register bank : Maxlive

Need to consider each time t modulo λ, actually all $\sigma(v,i)+t_{\text{in}}(e)$.
Produced at time t : $\text{prod}(e,t) = |\{i \in \mathbb{N} \mid \sigma(u,i)+t_{\text{out}}(u) \leq t\}|$.
Consumed at t : $\text{cons}(e,t) = |\{i \in \mathbb{N} \mid \sigma(v,i+w(e))+t_{\text{in}}(e) \leq t\}|$.
Live along e at t : $\text{live}(e,t) = \text{prod}(e,t) - \text{cons}(e,t)$.
Live at t $\text{live}(u,t) = \text{prod}(e,t) - \min_{e=(u,v)} \text{cons}(e,t)$.

Note : a) $\text{cons}(e,t)$ is minimal when $\sigma(v,i+w(e))+t_{\text{in}}(e)$ is maximal (last reader) ; b) v forced to be last reader with similar constraints : $\sigma(v,i+w(e))+t_{\text{in}}(e) \geq \sigma(v',i+w(e'))+t_{\text{in}}(e')$.

Inequalities with retiming

Constraints

- $\sigma(u,i) + t_{\text{out}}(u) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - r(u)$.
- $\sigma(v, i + w(e)) + t_{\text{in}}(e) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor - w(e) - r(v)$.
- $\text{live}(e,t) = w_r(e) + s(e,t)$ with $w_r(e) = w(e) + r(v) - r(u)$ and $s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor$.

Inequalities with retiming

Constraints

- $\sigma(u,i) + t_{\text{out}}(u) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - r(u)$.
- $\sigma(v, i + w(e)) + t_{\text{in}}(e) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor - w(e) - r(v)$.
- $\text{live}(e,t) = w_r(e) + s(e,t)$ with $w_r(e) = w(e) + r(v) - r(u)$ and $s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor$.

Linear program

$\text{Opt}_1 = \text{ninimize } \max_{t \in [0..\lambda[} \sum_{u \in V} \max_{e=(u,v)} (w_r(e) + s(e,t))$

$$= \min\{M \quad | \quad r(v) - r(u) \geq d_1(e), \forall e \in E$$
$$M(u,t) \geq r(v) - r(u) + w(e) + s(e,t), \forall u \in V, \forall t$$
$$M \geq \sum_{u \in V} M(u,t), \forall t\}$$

➽ Strongly NP-complete.

Inequalities with retiming

Constraints

- $\sigma(u,i) + t_{\text{out}}(u) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - r(u)$.
- $\sigma(v, i + w(e)) + t_{\text{in}}(e) \leq t$ iff $i \leq \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor - w(e) - r(v)$.
- $\text{live}(e,t) = w_r(e) + s(e,t)$ with $w_r(e) = w(e) + r(v) - r(u)$ and $s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor$.

Linear program

$\text{Opt}_1 = \text{ninimize } \max_{t \in [0..\lambda[} \sum_{u \in V} \max_{e=(u,v)} (w_r(e) + s(e,t))$

$$= \min\{M \quad | \quad r(v) - r(u) \geq d_1(e), \forall e \in E$$
$$M(u,t) \geq r(v) - r(u) + w(e) + s(e,t), \forall u \in V, \forall t$$
$$M \geq \sum_{u \in V} M(u,t), \forall t\}$$ ➽ Strongly NP-complete.

If only one last reader (along e_u) for each u : polynomial because $\max_t \sum_u (w_r(e_u) + s(e_u,t)) = \sum_u w_r(e_u) + \max_t \sum_u s(e_u,t)$.

Approximation up to $|V|$

$\text{live}(e,t) = w_r(e) + s(e,t),\ s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor.$

- $t_{\text{out}}(u) + s_u = \lambda.\alpha_u + \beta_u$ with $0 \leq \beta_u < \lambda$.
- $t_{\text{in}}(e) + s_v = \lambda.\delta_e + \gamma_e$ with $0 \leq \gamma_e < \lambda$.

➽ $s(e,t) = \alpha_u + \delta_e + s'(e,t)$ with $s'(e,t) = \lfloor \frac{t - \beta_u}{\lambda} \rfloor - \lfloor \frac{t - \gamma_e}{\lambda} \rfloor.$

Approximation up to $|V|$

$\text{live}(e,t) = w_r(e) + s(e,t)$, $s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor$.

- $t_{\text{out}}(u) + s_u = \lambda.\alpha_u + \beta_u$ with $0 \leq \beta_u < \lambda$.
- $t_{\text{in}}(e) + s_v = \lambda.\delta_e + \gamma_e$ with $0 \leq \gamma_e < \lambda$.

➽ $s(e,t) = \alpha_u + \delta_e + s'(e,t)$ with $s'(e,t) = \lfloor \frac{t - \beta_u}{\lambda} \rfloor - \lfloor \frac{t - \gamma_e}{\lambda} \rfloor$.

- if $\beta_u = \gamma_e$, $s'(e,t) = 0$. Let $s_{\min}(e) = 0$.
- if $\beta_u < \gamma_e$, $s'(e,t) = 0$ or 1. Let $s_{\min}(e) = 0$.
- if $\beta_u > \gamma_e$, $s'(e,t) = -1$ or 0. Let $s_{\min}(e) = -1$.

Approximation up to $|V|$

$\text{live}(e,t) = w_r(e) + s(e,t)$, $s(e,t) = \lfloor \frac{t - t_{\text{out}}(u) - s_u}{\lambda} \rfloor - \lfloor \frac{t - t_{\text{in}}(e) - s_v}{\lambda} \rfloor$.

- $t_{\text{out}}(u) + s_u = \lambda.\alpha_u + \beta_u$ with $0 \leq \beta_u < \lambda$.
- $t_{\text{in}}(e) + s_v = \lambda.\delta_e + \gamma_e$ with $0 \leq \gamma_e < \lambda$.

➽ $s(e,t) = \alpha_u + \delta_e + s'(e,t)$ with $s'(e,t) = \lfloor \frac{t - \beta_u}{\lambda} \rfloor - \lfloor \frac{t - \gamma_e}{\lambda} \rfloor$.

- if $\beta_u = \gamma_e$, $s'(e,t) = 0$. Let $s_{\min}(e) = 0$.
- if $\beta_u < \gamma_e$, $s'(e,t) = 0$ or 1. Let $s_{\min}(e) = 0$.
- if $\beta_u > \gamma_e$, $s'(e,t) = -1$ or 0. Let $s_{\min}(e) = -1$.

This leads to the polynomially-solvable linear program :

$$\text{Opt}_2 = \min\{\textstyle\sum_{u \in V} M(u) \mid r(v) - r(u) \geq d_1(e), \forall e \in E,$$
$$M(u) \geq r(v) - r(u) + w(e) + \alpha_u + \delta_e + s_{\min}(e), \forall u \in V, \forall e = (u,v)\}$$

➽ $\text{Opt}_2 \leq \text{Opt}_1 \leq \text{Opt}_2 + |V|$

Outline

Loop fusion : interest and problems

Why ?

- Increase size of basic blocks.
- Useful for both spatial and temporal locality.
- Useful for array contraction.
- Be careful of "over"-fusion.

How ?

- Polynomial algorithms ?
- NP-completeness ?
- Heuristics ?

When ?

- Always difficult to find a cost model...
- But complementary tool for more general transformations.

A few references for the basics

- Kennedy and McKinley. Typed Fusion with Applications to Parallel and Sequential Code Generation.
- McKinley and Kennedy. Maximizing Loop Parallelism and Improving Data Locality via Loop Fusion and Distribution.
- Roth and Kennedy. Loop Fusion in High Performance Fortran.
- Alain Darte. On the Complexity of Loop Fusion.

Array contraction

Goal : reduce the size of a local array, possibly into a scalar.

```
do i=2,N
   a(i) = d(i) + 1
   b(i) = a(i)/2
   c(i) = b(i) + b(i-1)
enddo
```

```
do i=2,N
   a = d(i) + 1
   b(i) = a/2
   c(i) = b(i) + b(i-1)
enddo
```

Array assignments and forall loops

```
A = B + C
B = D + 1
A[1:n] = A[0:n-1] + A[2:n+1]
```

equivalent to :

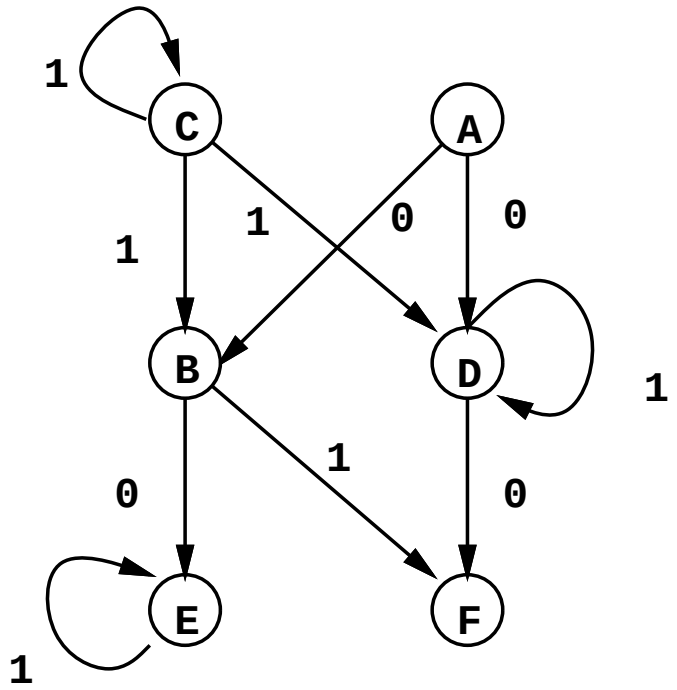
```
forall(i=0,n+1)
   A(i) = B(i) + C(i)
endforall
forall(i=0,n+1)
   B(i) = D(i) + 1
endforall
forall(i=1,n)
   A(i) = A(i-1) + A(i+1)
endforall
```

and, then, to :

```
doall(i=0,n+1)
   A(i) = B(i) + C(i)
enddoall
doall(i=0,n+1)
   B(i) = D(i) + 1
enddoall
doall(i=0,n+1)
   A'(i) = A(i)
enddoall
doall(i=1,n)
   A(i) = A'(i-1) + A'(i+1)
enddoall
```

Partial loop distribution

```
DO i=1,N
   A(i) = 2*A(i) + 1
   B(i) = C(i-1) + A(i)
   C(i) = C(i-1) + G(i)
   D(i) = D(i-1) + A(i) + C(i-1)
   E(i) = E(i-1) + B(i)
   F(i) = D(i) + B(i-1)
ENDDO
```



```
DOPAR i=1,N
  A(i) = 2*A(i) + 1
ENDDOPAR
DOSEQ i=1,N
  C(i) = C(i-1) + G(i)
ENDDOSEQ
DOPAR i=1,N
  B(i) = C(i-1) + A(i)
ENDDOPAR
DOSEQ i=1,N
  E(i) = E(i-1) + B(i)
ENDDOSEQ
DOSEQ i=1,N
  D(i) = D(i-1) + A(i) + C(i-1)
ENDDOSEQ
DOPAR i=1,N
  F(i) = D(i) + B(i-1)
ENDDOPAR
```

How to get the following ?

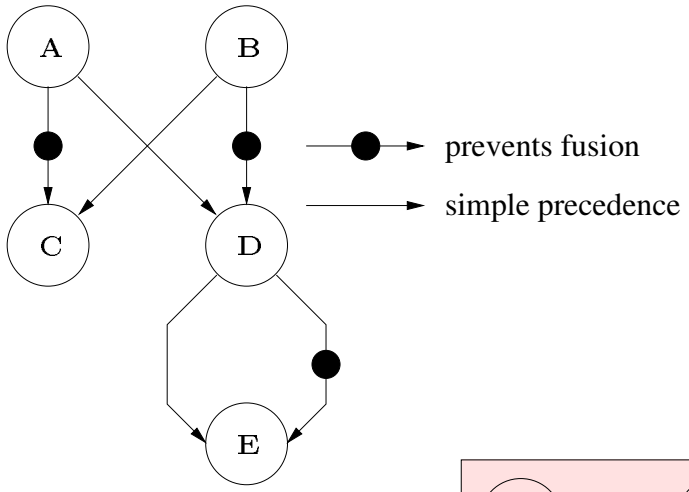
```
DOSEQ i=1,N
  C(i) = C(i-1) + G(i)
ENDDOSEQ
DOPAR i=1,N
  A(i) = 2*A(i) + 1
  B(i) = C(i-1) + A(i)
ENDDOPAR
DOSEQ i=1,N
  D(i) = D(i-1) + A(i) + C(i-1)
  E(i) = E(i-1) + B(i)
ENDDOSEQ
DOPAR i=1,N
  F(i) = D(i) + B(i-1)
ENDDOPAR
```

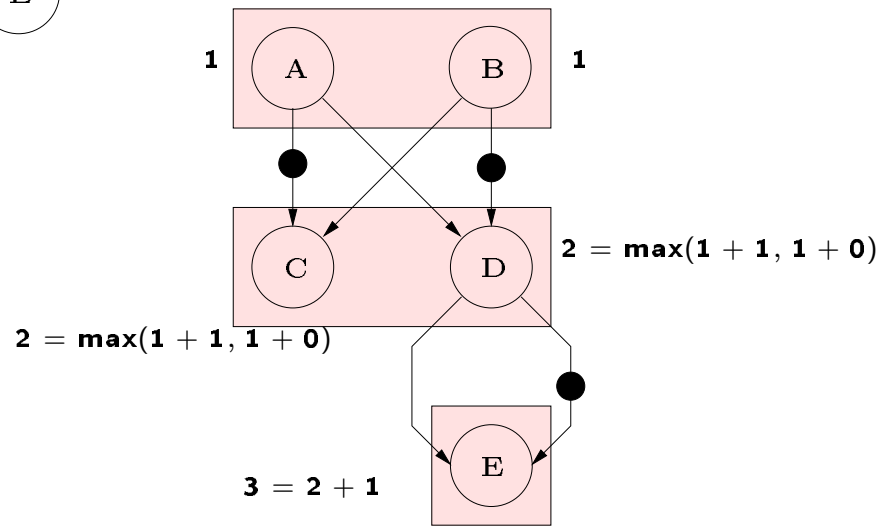
Outline

Simple loop fusion/distribution

```
DO i=2, n
   a(i) = f(i)
   b(i) = g(i)
   c(i) = a(i-1) + b(i)
   d(i) = a(i) + b(i-1)
   e(i) = d(i-1) + d(i)
ENDDO
```

```
DOPAR i=2, n
   a(i) = f(i)
   b(i) = g(i)
ENDDOPAR
DOPAR i=2, n
   c(i) = a(i-1) + b(i)
   d(i) = a(i) + b(i-1)
ENDDOPAR
DOPAR i=2, n
   e(i) = d(i-1) + d(i)
ENDDO
```





☛ Easy : compute longest dependence paths.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).
- Fixed number of chains : polynomial $O(d \times N^d)$ for d chains of length at most N. Dynamic programming.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).
- Fixed number of chains : polynomial $O(d \times N^d)$ for d chains of length at most N. Dynamic programming.
- Two colors, no fusion-preventing edges : polynomial $O(V+E)$.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).
- Fixed number of chains : polynomial $O(d \times N^d)$ for d chains of length at most N. Dynamic programming.
- Two colors, no fusion-preventing edges : polynomial $O(V+E)$.
- Three colors, no fusion-preventing edges : NP-complete.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

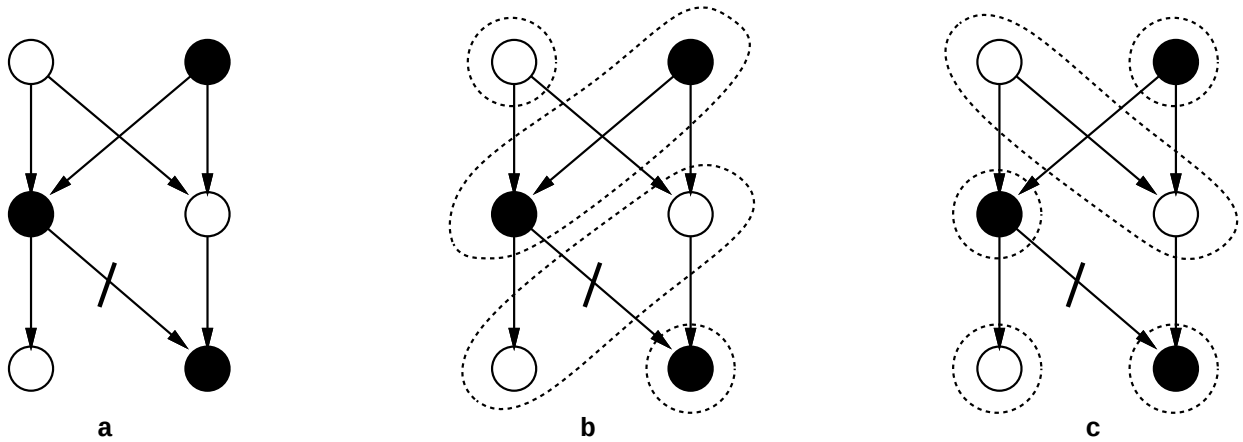
- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).
- Fixed number of chains : polynomial $O(d \times N^d)$ for d chains of length at most N. Dynamic programming.
- Two colors, no fusion-preventing edges : polynomial $O(V+E)$.
- Three colors, no fusion-preventing edges : NP-complete.
- Two colors, fusion-preventing edges : NP-complete.

Maximal unordered **typed** fusion

Constraint : two loops of different types (colors) cannot be fused.

- Arbitrary number of colors : NP-complete (McKinley & Kennedy, 1994), even for chains of length 2.
- Ordered typed fusion : polynomial $O(T(V+E))$ (see after).
- Fixed number of chains : polynomial $O(d \times N^d)$ for d chains of length at most N. Dynamic programming.
- Two colors, no fusion-preventing edges : polynomial $O(V+E)$.
- Three colors, no fusion-preventing edges : NP-complete.
- Two colors, fusion-preventing edges : NP-complete.
- Two colors, fusion-prev. edges for one color : NP-complete.
  Ex : partial loop distribution for parallelism detection.

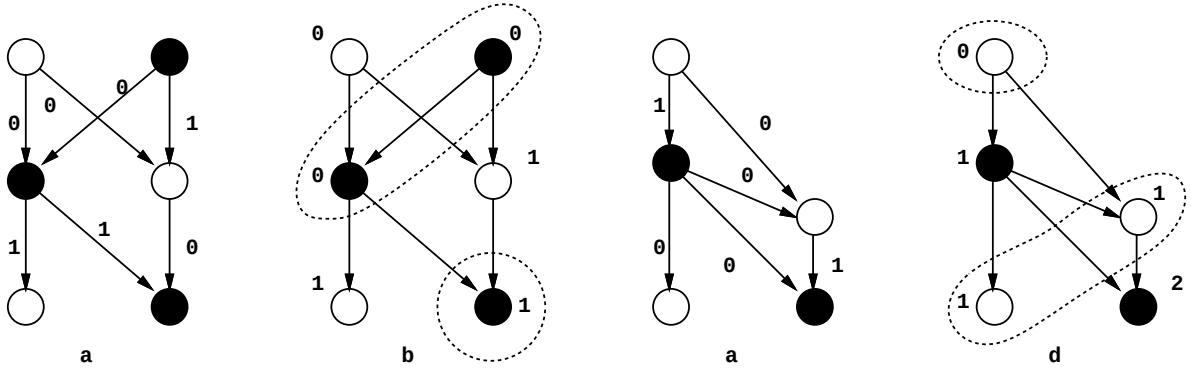
Ordered typed fusion



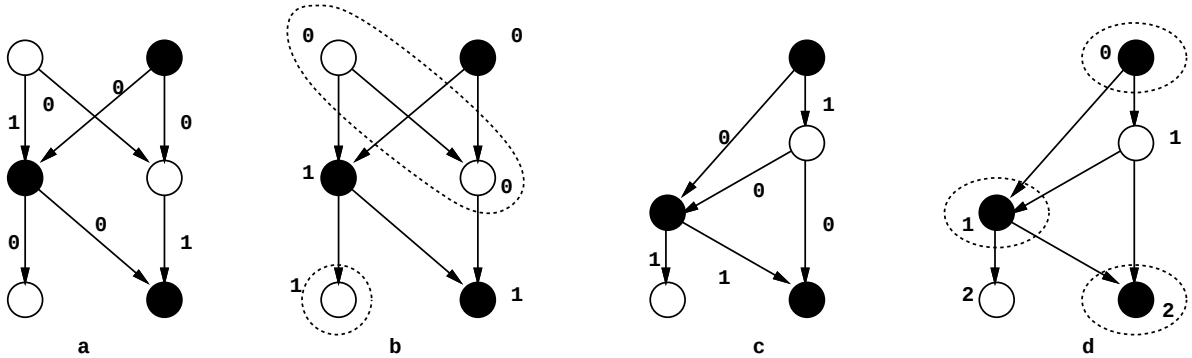
(a) original graph, (b) fuse black then white, (c) fuse white then black.

Optimization of type T :

- W(v) : minimal number of loops of type T that have to be placed before v. $W(v) = 0$ if v has no predecessor.
- For $e = (u, v)$, $w(e) = 1$ if $T(u) = T$ and either $T(v) \neq T$ or e is fusion-preventing. Otherwise $w(e) = 0$.



(a) weighted graph for fusing black, (b) fusion of black, (c) weighted graph for fusing white after black, (d) fusion of white.



(a) weighted graph for fusing white, (b) fusion of white, (c) weighted graph for fusing black after white, (d) fusion of black.