## Ordonnancement (fonction de temps)

$t : \mathbf{V} \times \mathcal{D} \rightarrow \mathbf{N}$ telle que

$$(V, x) \leftarrow (U, y) \Rightarrow t(V, x) > t(U, y).$$

Un ordonnancement $t$ est dit *au plus tôt*, si :

$$t(V,x) = \begin{cases} 1 \text{ si } (\nexists (U,y), y \in \mathcal{D}) \text{ t.q. } (V,x) \stackrel{1}{\leftarrow} (U,y)) \\ 1 + \max\{t(U,y) \,|\, (V,x) \stackrel{1}{\leftarrow} (U,y) \text{ et } y \in \mathcal{D}\}, \text{ sinon.} \end{cases}$$



**Propriété :** Si un système admet un ordonnancement, alors, il a un ordonnancement au plus tôt unique, noté $T_f$.

Par extension, on pose $T_f(V, x) = +\infty$ lorsque l'ordonnancement n'est pas défini en $(V, x)$.

$T_f(V, x)$ est la longueur du plus long chemin arrivant en $(V, x)$ dans $\Gamma$.



## Calculabilité

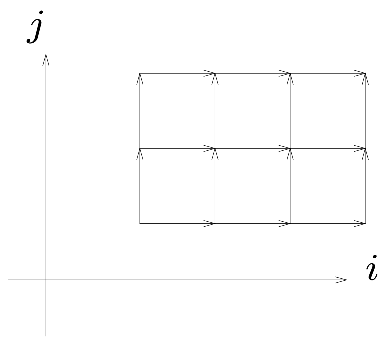
**Définition :** une variable $V$ est dite *calculable* – ou encore *définie explicitement* – ssi

$$x \in \mathcal{D} \Rightarrow T_f(V, x) < +\infty.$$

**Lemme.** $T_f(V, x) = +\infty \Leftrightarrow \Gamma$ *contient un chemin infini arrivant en* $(V, x)$. Cela signifie, soit qu'il y a un cycle dans le graphe, soit qu'il existe un chemin non cyclique, mais infini...

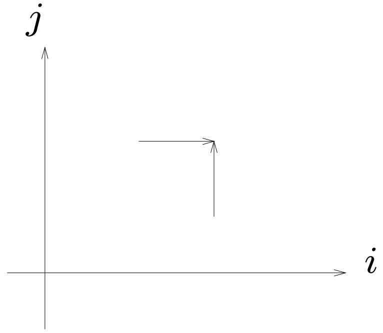


## Exemples



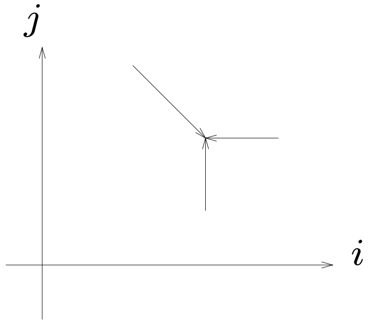

$$i, j \geq 0 \rightarrow V(i, j) = V(i-1, j) + V(i, j-1)$$

# Exemples



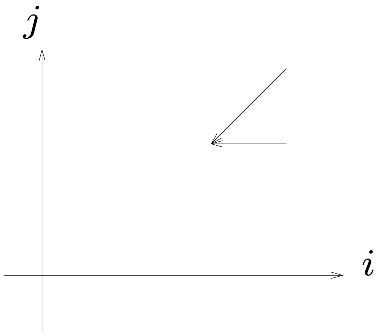

$$i, j \geq 0 \rightarrow V(i,j) = V(i-1,j) + V(i,j-1)$$





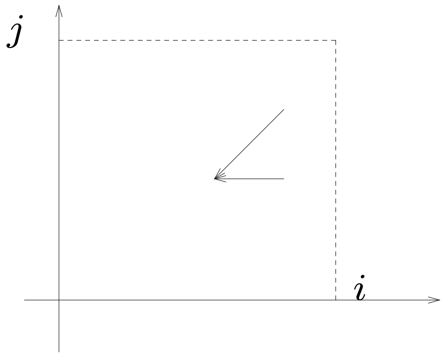

$$i, j \geq 0 \rightarrow V(i,j) = V(i,j-1) + V(i+1,j) + V(i-1,j+1)$$






$$i, j \geq 0 \rightarrow V(i,j) = V(i+1,j+1) + V(i+1,j)$$






$$i, j \geq 0; i, j \leq N \rightarrow V(i,j) = V(i+1,j+1) + V(i+1,j)$$

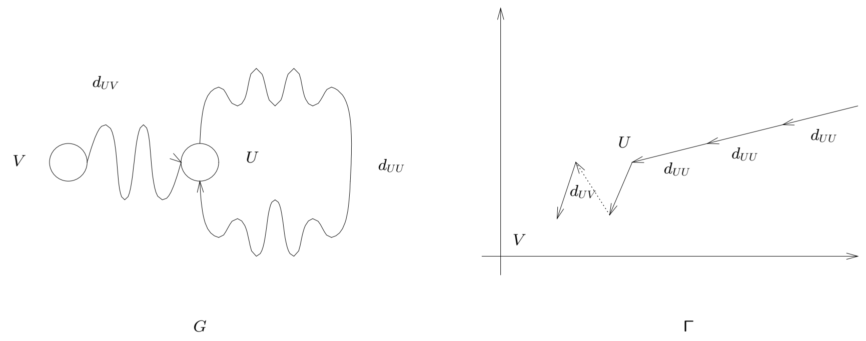
## Calculabilité d'un système uniforme

$d(\pi)$ poids d'un chemin $\pi$ de $G$, c'est-à-dire somme des vecteurs de dépendances de ses arcs.

**Théorème 1 (Karp, Miller et Winograd, 1967) :** Soit $G$ un graphe de dépendance défini sur le domaine $\mathcal{D} = \mathbf{N}^n$. La variable $V$ est calculable ssi $G$ ne contient pas un chemin arrivant en $V$ et dont l'origine $U$ est contenue dans un cycle de poids négatif ou nul.



## Illustration du théorème 1






## Cas d'une seule équation

**Théorème 2:** Étant donnée une équation récurrente uniforme

$$V(x) = f[V(x-d_1), ..., V(x-d_s)],$$

définie sur le domaine $\mathcal{D} = \mathbf{N}^n$, les conditions suivantes sont équivalentes :

1. $V(x)$ est calculable;



## Cas d'une seule équation

2. il n'existe pas de vecteur semi positif (i.e., non négatif et non nul) $(u_1, ..., u_s)$ tel que
$$\sum_{j=1}^{s} -u_j d_j \geq 0;$$

3. soit $\tau = (\tau_1, ..., \tau_n)$ un vecteur de $\mathbf{Z}^n$. Le système d'inéquations $d_j.\tau \geq 1$, $1 \leq j \leq s$, et $\tau_i \geq 0$, $1 \leq i \leq n$ a une solution ;

## Cas d'une seule équation

4. pour tout $x \in \mathbf{N}^n$, les deux problèmes linéaires suivants ont une valeur optimale commune. $m(x)$:

$$I \begin{cases} x - \sum_{j=1}^{s} u_j d_j \geq 0, \\ u_j \geq 0, 1 \leq j \leq s, \\ \max \sum_{i=1}^{s} u_j \end{cases} \qquad II \begin{cases} \tau_i \geq 0, 1 \leq i \leq n \\ d_j.\tau \geq 1, 1 \leq j \leq s, \\ \min x.\tau \;\; . \end{cases}$$

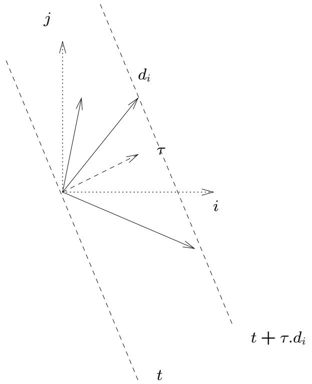


## Remarques

- interprétation géométrique de la condition *3*: tous les vecteurs de dépendance sont strictement contenus dans un demi espace.



## Remarques






## Remarques

- condition *4*:

  . $m(x)$ représente la longueur du plus long chemin arrivant au point dans le graphe $\Gamma$.

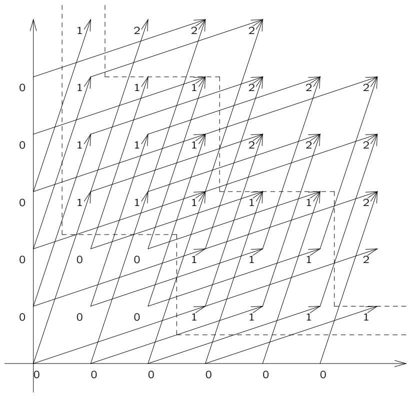
  . $m(x)$ est un ordonnancement, que l'on peut calculer point par point par programmation linéaire.

- $m(x)$ est "aussi rapide" que l'ordonnancement au plus tôt, $T_f$, sauf au "bords du domaine"

- le cas d'un système est plus compliqué à traiter

# Exemple

$$i, j \geq 0 \rightarrow V(i,j) = V(i-1, j-3) + V(i-3, j-1)$$





# Ordonnancements linéaires

**Corollaire :** Toute solution $\tau$ du système d'inéquations de la condition 3 du théorème I donne un ordonnancement $t(x) = \tau.x$ linéaire en $x$.



# Exemple

$$i, j \geq 0 \rightarrow V(i,j) = V(i-1, j-3) + V(i-3, j-1)$$

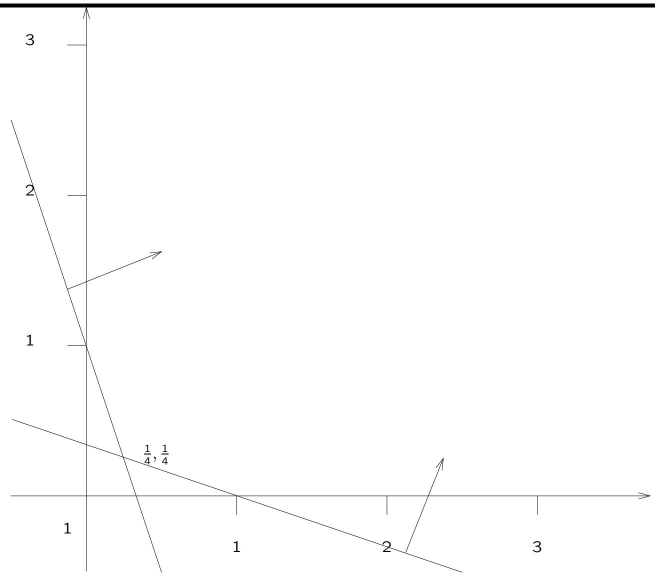
Dépendances : $(1,3)$ et $(3,1)$

$$\text{II} \begin{cases} \tau_1, \tau_2 \geq 0 \\ \tau_1 + 3\tau_2 \geq 1 \\ 3\tau_1 + \tau_2 \geq 1 \end{cases} .$$

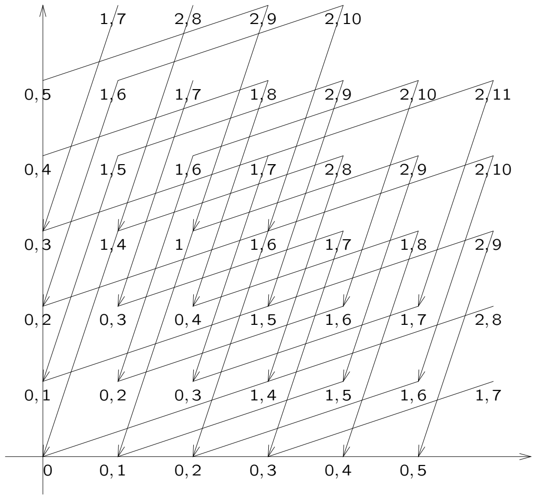
Solution rationnelle : $(1/4, 1/4)$.
Solutions entières : $(1,0)$ $(T_v(i,j) = i)$ ou $(1,1)$ $(T_v(i,j) = i + j)$.



# Exemple



# Exemple






# Exemple

$$
\begin{array}{l}
1 \leq i,j,k \leq 3 \rightarrow \\
\quad C(i,j,k) = C(i,j,k-1) + A(i,j,k) \times B(i,j,k) \\
1 \leq i,j,k \leq 3 \rightarrow A(i,j,k) = A(i,j-1,k) \\
1 \leq i,j,k \leq 3 \rightarrow B(i,j,k) = B(i-1,j,k)
\end{array}
$$

Dépendances : $(0,0,1)^t, (0,1,0)^t, (1,0,0)^t$ et $(0,0,0^t$



Programme linéaire $II$ :

$$
\begin{array}{rcll}
(0,0,1).\tau & \geq & 1 & \qquad \tau_3 \geq 1 \\
(0,1,0).\tau & \geq & 1 & \text{ i.e. } \tau_2 \geq 1 \\
(1,0,0).\tau & \geq & 1, & \qquad \tau_1 \geq 1,
\end{array}
$$

qui admet la solution $\tau = (1,1,1)^t$.
$\Rightarrow t(i,j,k) = i + j + k$ est un ordonnancement linéaire.



# Plan

I) Equations rcurrentes

II) Ordonnancement

II.1 Ordonnancement linéaire

II.2 Ordonnancement optimal

## Ordonnancement linéaire de latence minimale

On suppose que l'équation a pour domaine le polytope

$$\mathcal{D} = \{z | Az \leq b\} \ .$$

On choisit une fonction de temps linéaire $T_l(z) = \lfloor \tau z + \alpha \rfloor$ ($\tau$ peut être rationnel).



Le temps total de calcul est :

$$\max_{z_1, z_2 \in \mathcal{D}} T_l(z_1) - T_l(z_2) = \max_{(Az_1 \leq b, Az_2 \leq b)} (\lfloor \tau z_1 \rfloor - \lfloor \tau z_2 \rfloor)$$

Problème : comment déterminer la solution optimale :

$$\min_{\tau D \geq 1} \max_{z_1, z_2 \in \mathcal{D}} (\lfloor \tau z_1 \rfloor - \lfloor \tau z_2 \rfloor)$$



## Exemple

$$\left\{ \begin{array}{l} i \geq 0, j \geq 0, \\ N \leq i + j \leq 2N \end{array} \right\} \rightarrow V(i,j) =$$

$$f[V(i-1, j+3), V(i-2, j-4), V(i-2, j)]$$

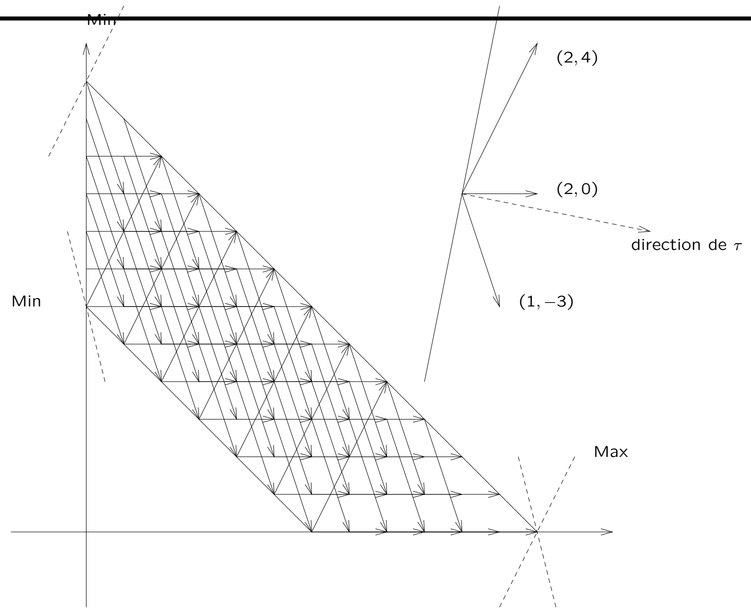
$$\begin{pmatrix} 1 \\ -3 \end{pmatrix} \qquad \begin{pmatrix} 2 \\ 4 \end{pmatrix} \qquad \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$



$$A = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ -1 & -1 \end{pmatrix} \ b = \begin{pmatrix} 0 \\ 0 \\ 2N \\ -N \end{pmatrix}$$

Vecteurs de dépendance :

$$\begin{pmatrix} 1 \\ -3 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$






## Résolution par programmation linéaire

Principe : se ramener à une seule variable.

On pose

$$A' = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix},\ c = (\tau, -\tau),\ b' = \begin{pmatrix} b \\ b \end{pmatrix},\ z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}.$$



Le problème :

$$\begin{cases} Az_1 \leq b \\ Az_2 \leq b \\ \max \tau(z_1 - z_2) \end{cases}$$

devient

$$\begin{cases} A'z \leq b' \\ \max c.z \ . \end{cases}$$



Par application de la dualité, c'est la même chose que :

$$\begin{cases} y \geq 0 \\ yA' = c \\ \min yb' \ . \end{cases}$$

## Problème final à résoudre

En posant $y = (\tau_1, \tau_2)$, et en ajoutant les contraintes de dépendance $\tau D \geq 1$ :

$$\begin{cases} \tau D \geq 1 \\ \tau_1 A = \tau \\ \tau_2 A = -\tau \\ \tau_1 \geq 0 \\ \tau_2 \geq 0 \\ \min\,(\tau_1 + \tau_2)b \end{cases} .$$



ou encore :

$$\begin{cases} (\tau_1 + \tau_2)A = 0 \\ \tau_1 AD \geq 1 \\ \tau_1 \geq 0 \\ \tau_2 \geq 0 \\ \min\,(\tau_1 + \tau_2)b \end{cases}$$



## Exemple

Contrainte $(\tau_1 + \tau_2)A = 0$ :

$$-(\tau_{11} + \tau_{21}) + (\tau_{13} + \tau_{23}) - (\tau_{14} + \tau_{24}) = 0$$
$$-(\tau_{12} + \tau_{22}) + (\tau_{13} + \tau_{23}) - (\tau_{14} + \tau_{24}) = 0$$



Contraintes $\tau_1 AD \geq 1$ :

$$-2\tau_{11} - 4\tau_{12} + 6\tau_{13} - 6\tau_{14} \geq 1$$
$$2(-\tau_{11} + \tau_{13} - \tau_{14}) \geq 1$$
$$-\tau_{11} + 3\tau_{12} - 2\tau_{13} + 2\tau_{14} \geq 1$$

A optimiser :

$$Min - \tau_{14} + 2(\tau_{13} + \tau_{23}) - \tau_{24}$$

$$\tau = (-\tau_{11} + \tau_{13} - \tau_{14}, -\tau_{12} + \tau_{13} - \tau_{14})^t$$

Solution : $\tau = (\frac{7}{10}, \frac{-1}{10})$



## Plan





## Cas des récurrences affines

- Les équations sont de la forme $\forall z \in \mathcal{D}$ : $V(z) = f_V[U(I(z)), ...]$.
- $I(z) = Az + B$.
- Arc $e$ du graphe de dépendance valué avec $(V, U, \mathcal{D}, I)$.



- Forme des fonctions de temps : $t_V(z) = \tau_V^1 z_1 + ... + \tau_V^n z_n + \alpha_V$.
- Le domaine peut être infini.

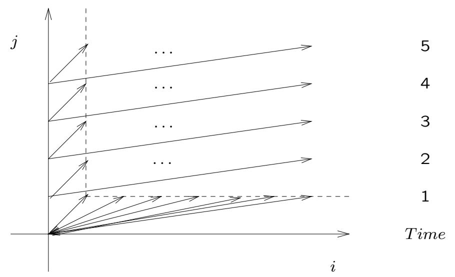
## Exemple

$$1 \leq i,j \leq n, n > 0 \;\rightarrow\; U(i,j,n) = U(0,j-1,n) + V(i,j,n)$$

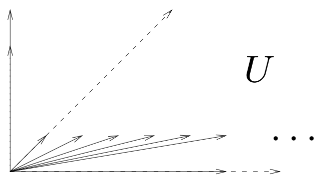
$$1 \leq i,j \leq n, n > 0 \;\rightarrow\; V(i,j,n) = V(i,j-1,n)$$



## Graphe de dépendance étendu de $U$






Les vecteurs de dépendance de $U$ sont dans un domaine convexe.



En général :

- les variables peuvent ne pas avoir le même nombre d'indices

$\Rightarrow$ la notion de vecteur de dépendance peut ne pas avoir de sens

- même si les équations sont homogènes, un ordonnancement commun peut ne pas exister

$\Rightarrow$ il faut étendre le modèle.





Problème: Comment respecter une dépendance $U(z) \leftarrow V(I(z))$ uniquement sur un domaine convexe.

Problème générique: imposer une contrainte linéaire pour tous les points contenus dans un polyèdre convexe?

- méthode des sommets
- méthode de Farkas



## Méthode dite “des sommets”

**Théorème :** Les nombres entiers $\tau_V$, $\alpha_V, V \in \mathbf{V}$ définissent un ordonnancement affine ssi :

- pour tout arc $(V, U, \mathcal{D}, z \rightarrow Az + B)$ du graphe de dépendance,

*(i)* pour tout sommet $\sigma$ de $\mathcal{D}$, $\tau_V.\sigma - \tau_U.(A\sigma + B) + \alpha_V - \alpha_U > 0$,

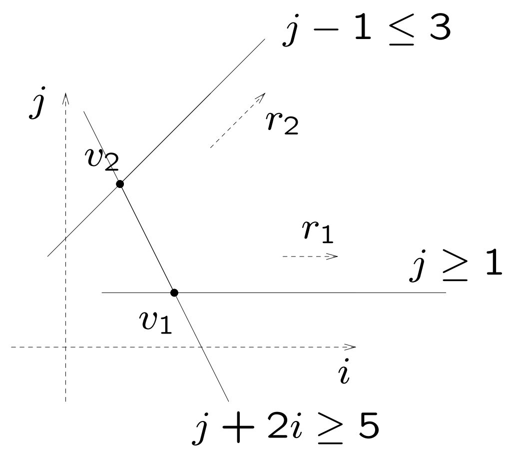
*(ii)* pour tout rayon $\rho$ de $\mathcal{D}$, $\tau_V.\rho - \tau_U.A.\rho \geq 0$ ;



- pour toute variable $V$:

*(iii)* pour tout sommet $\sigma$ du domaine $\mathcal{D}_V$ de $V$, $\tau_V.\sigma + \alpha_V \geq 0$,

*(iv)* pour tout rayon $\rho$ de $\mathcal{D}_V$, $\tau_V.\rho \geq 0$.





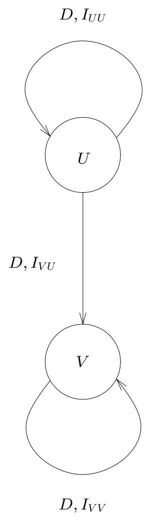

## Forme simplifiée

Les conditions (i) et (ii) sont appelées *conditions de causalité*, et (iii) et (iv), *conditions de positivité*.

Lorsque de domaine d'une équation est un polytope (pas de rayon, on peut ne conserver que la condition (i) : les conditions (ii) et (iv) portent sur les rayons, et de plus, on peut facilement ajuster les constantes $\alpha_V$ pour que (iii) soit vérifiée.



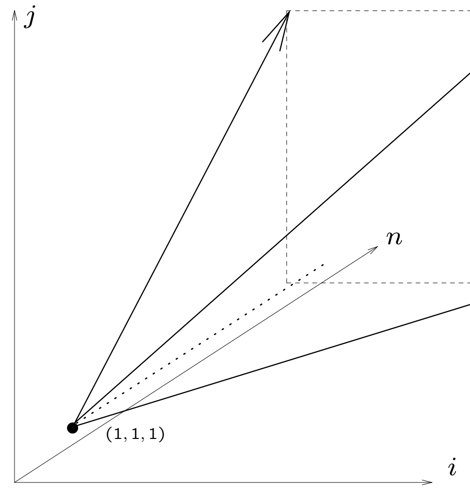
## Exemple




$$D = \{1 \leq i, j \leq n, n > 0\}$$
$$I_{UU} = (0, j-1, n)$$
$$I_{VU} = (i, j, n)$$
$$I_{VV} = (i, j-1, n)$$



## Exemple



- Ordonnancement de $V$ noté

$$\tau_V^1 i + \tau_V^2 j + \tau_V^3 n + \alpha_V$$

- Ordonnancement de $U$ noté

$$\tau_U^1 i + \tau_U^2 j + \tau_U^3 n + \alpha_U$$



- Sommets de $D$ : (1,1,1)

- Rayons de $D$ : (0,0,1), (0,1,1), (1,0,1), (1,1,1)

- $I_{UU} = A_{UU}\begin{pmatrix} i \\ j \\ n \end{pmatrix} + B_{UU}$ avec $A_{UU} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
  et $B_{UU} = (0, -1, 0)^T$



- $I_{VU} =$ identité

- $I_{VV} = A_{VV}\begin{pmatrix} i \\ j \\ n \end{pmatrix} + B_{VV}$ avec $A_{VV} =$ identité
  et $B_{VV} = (0, -1, 0)^T$



Conditions *(i)* et *(ii)* du théorème 3

Arc $(U, U)$:

$$\begin{aligned}
\tau_U.(1,1,1)^T - \tau_U.(A_{UU}.(1,1,1)^T + B) &> 0 \\
\tau_U.(0,0,1)^T - \tau_U.A_{UU}.(0,0,1)^T &\geq 0 \\
\tau_U.(0,1,1)^T - \tau_U.A_{UU}.(0,1,1)^T &\geq 0 \\
\tau_U.(1,0,1)^T - \tau_U.A_{UU}.(1,0,1)^T &\geq 0 \\
\tau_U.(1,1,1)^T - \tau_U.A_{UU}.(1,1,1)^T &\geq 0
\end{aligned}$$

Arc $(U,V)$ (si $\tau_U = \tau_V$):

$$\alpha_U - \alpha_V \;>\; 0$$

Arc $(V,V)$:

$$\tau_V - \tau_V.(A_{VV}.(1,1,1)^T + B_{VV}) \;>\; 0$$





$$\begin{aligned} \tau_V^1 &\geq 0 \\ \tau_U^1 + \tau_U^2 &> 0 \\ \alpha_U - \alpha_V &> 0 \\ \tau_V^2 &> 0 \end{aligned}$$

qui a la solution

$$t_U(i,j,n) = j \text{ et } t_V(i,j,n) = j - 1$$



## Plan





## $2^{ème}$ méthode : méthode de Farkas

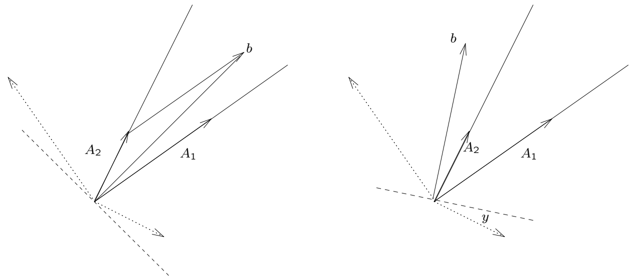
Lemme de Farkas: étant donnée un matrice A et un vecteur b

$$(\exists x \geq 0 \text{ tel que } Ax = b) \Leftrightarrow (\forall y,\; yA \geq 0 \Rightarrow yb \geq 0)$$

Utilisation: respecter les conditions de causalité (dépendances) et de positivité.

## Lemme de Farkas

$$(\exists x \geq 0 \text{ tel que } Ax = b) \Leftrightarrow (\forall y,\ yA \geq 0 \Rightarrow yb \geq 0)$$






## Lemme de Farkas : forme affine

La forme utilisée est la forme affine:

soit $D = \{x \ |Ax \leq b\}$ et $\Phi(x) = \phi x + \phi_0$ une fonction affine.

$\Phi(x)$ est positive sur $D$ si et seulement si il existe $\lambda, \lambda_0 \geq 0$ tels que $(\phi \ \phi_0) = \lambda(-A \ b) + (0 \ \lambda_0)$.



## Exemple

Soit l'équation:

$\{i, j, N | 0 \leq i \leq j \leq N\}: \ X[i, j] = f(Y[i]);$

La dépendance est assurée si l'on impose:

$T_X(i, j, N) - T_Y(i, N) - 1 \geq 0$

sur : $\{i, j, N | 0 \leq i \leq j \leq N\}$.

On applique donc le lemme avec :

$\Phi(i, j, N) = (\tau_{Xi} - \tau_{Yi})i + \tau_{Xj}j + (\tau_{XN} - \tau_{YN})N + \alpha_X - \alpha_Y - 1$.



## Exemple

On doit imposer $\Phi(i, j, N) \geq 0$

sur $D = \{i, j, N \ | \ Ax \leq b\}$ avec:

$$A = \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix} \text{ et } b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Après application du lemme:

$$\begin{array}{rcl} \tau_{Xi} - \tau_{Yi} & = & \lambda_1 - \lambda_2 - \lambda_5 \\ \tau_{Xj} & = & \lambda_3 - \lambda_4 + \lambda_5 \\ \tau_{XN} - \tau_{YN} & = & \lambda_2 + \lambda_4 \\ \alpha_X - \alpha_Y - 1 & = & \lambda_0 \\ \lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5 & \geq & 0 \end{array}$$

$$\Rightarrow \alpha_X - \alpha_Y - 1 \geq 0$$

$$\tau_{Xi} = \tau_{Yi} \quad \tau_{Xj} = 0 \quad \tau_{XN} = \tau_{YN}$$





On traite ainsi :

- Chaque dépendance ;
- chaque variable (fonction de temps positive, et inférieure à la fonction objective).



## À retenir

On sait trouver un ordonnancement **linéaire** pour un système d'équations récurrentes affines.

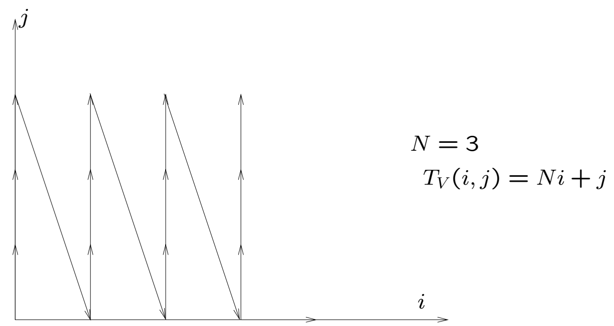
Deux méthodes: la méthodes des sommets, la méthode de Farkas.

Il existe des système qui n'admettent pas d'ordonnanc
lineaire



## Exemple

$$\begin{array}{ll} \{i,j \mid 1 \leq i,j \leq N\} & \rightarrow V(i,j) = V(i,j-1) \\ \{i,j \mid 1 \leq i \leq N; j = 0\} & \rightarrow V(i,j) = V(i-1,N) \end{array}$$



## Plan

I) Introduction

II) Ordonnancement

III) Transformation spatio-temporelles

IV) Contraintes de ressources

IV-1 Partitionnement ;



## Contraintes de ressources

Problème:

Comment allouer les taches sur les ressources pour minimiser le temps total d'exécution.

Dans le cas non cyclique, le problème d'allocation de $n$ taches sur $m$ machines est difficile ($NP$-complet dès que $m > 2$)



## Cas cyclique:

Dans le cas cyclique, deux types de contraintes de ressources :

- Pas assez de processeurs ;
- Pas assez de ressources dans chaque processeur.

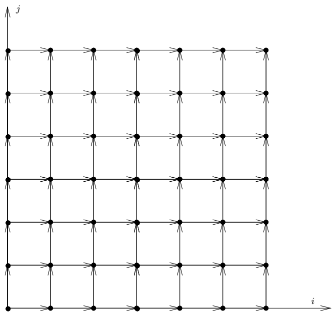


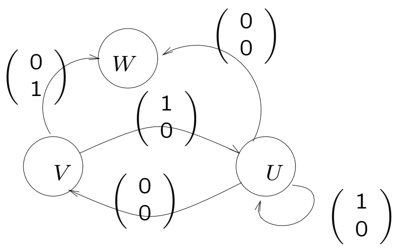
## Solution:

- Pas assez de processeurs ;
  $\Rightarrow$ partitionnement de l'algorithme.
- Pas assez de ressource dans chaque processeur.
  $\Rightarrow$ Ordonnancement cyclique sous contraintes de ressources.

# Exemple

$$0 \le i,j \le N \;\rightarrow\; U(i,j) = U(i-1,j) + V(i-1,j)$$
$$0 \le i,j \le N \;\rightarrow\; V(i,j) = 2 * U(i,j)$$
$$0 \le i,j \le N \;\rightarrow\; W(i,j) = V(i,j-1) * U(i,j)$$

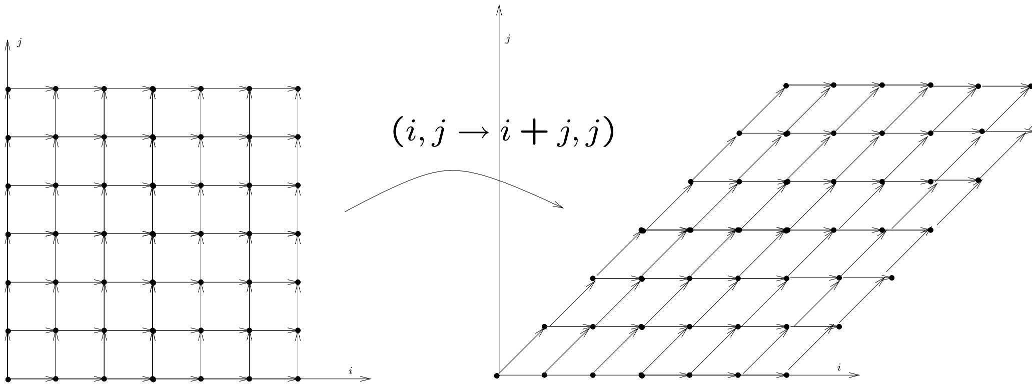







# Changement de base

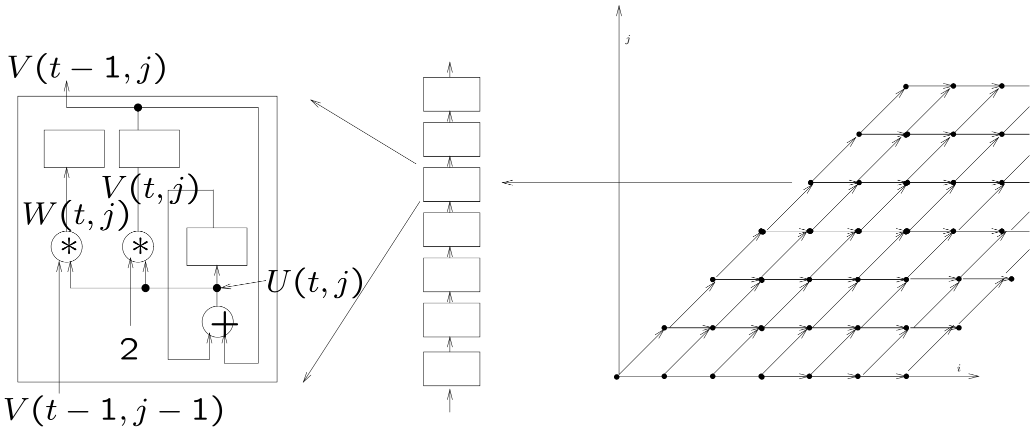
$(i,j)$ calculé au top $i+j$ sur le processeur $j$






# Processeurs virtuels

8 processeurs virtuels

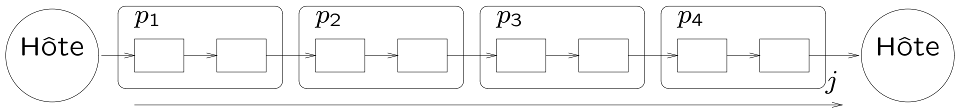





# Processeurs réels

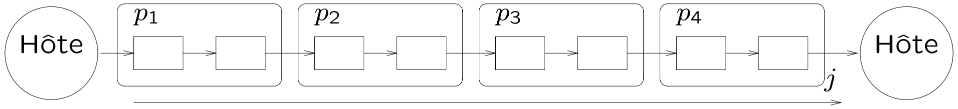
Si l'on ne dispose que de 4 processeurs réels:

1$^{ere}$ solution: émuler 2 processeurs virtuels sur un processeur réel.




Partitionnement LSGP (Localement Séquentiel, Globalement Parallèle).

## Partitionnement LSGP



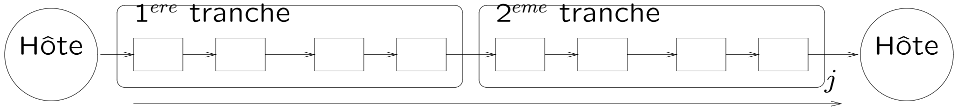

Inconvénients:
Séquentialisation dans chaque processeur.
Mémoire augmentée dans chaque processeur.



## Processeurs réels

$2^{eme}$ solution:

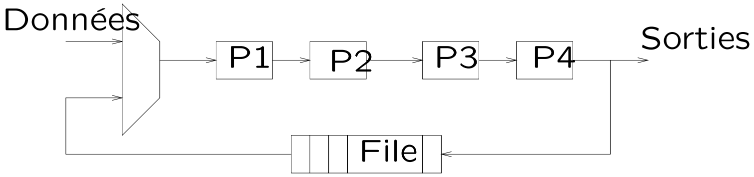
"couper" le réseau virtuel en "tranches" de 4 processeurs et exécuter les "tranches" les unes après les autres




Partitionnement LPGS (Localement Parallèle, Globalement Séquentiel).



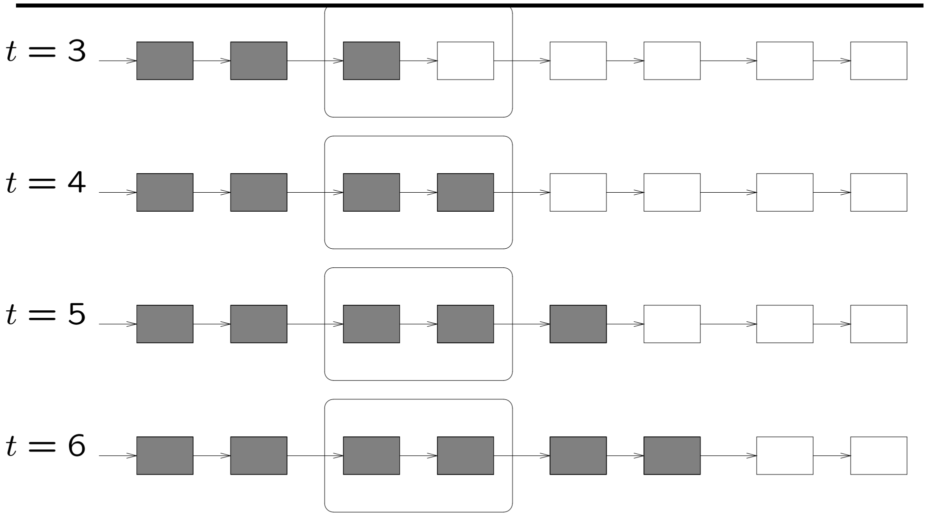
## Partitionnement LPGS




Inconvénients:
Contrôle de l'hôte plus complexe.
Impossible si le réseau est bi-directionnel.

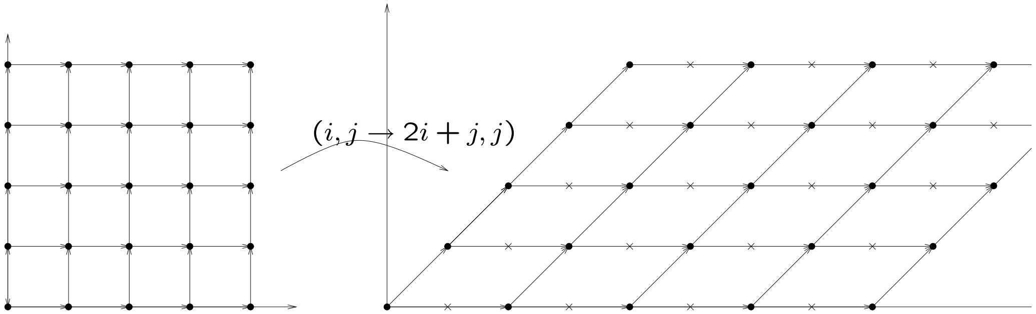


## Remarque: activité du réseau



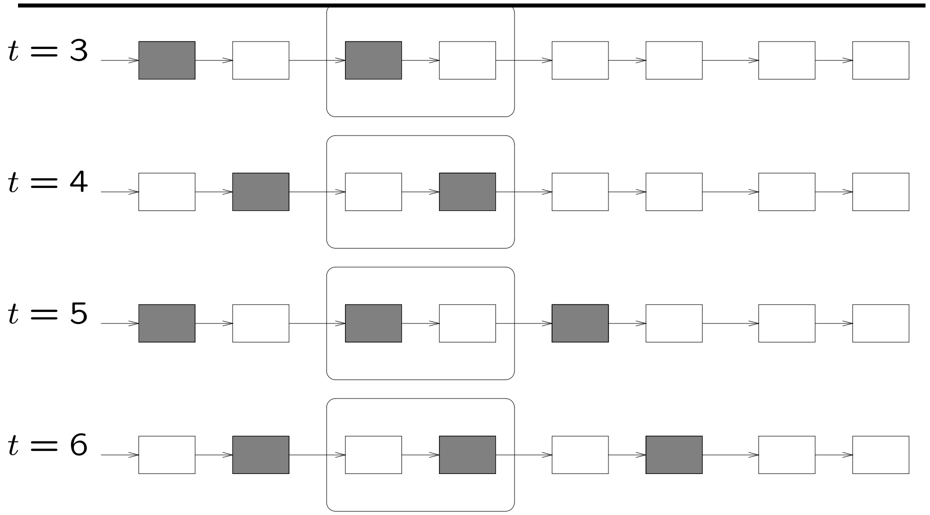
## Remarque: activité du réseau

Si on avait choisi la fonction de temps
$T(i,j) = 2i + j$






## Remarque: activité du réseau






## Remarque: activité du réseau

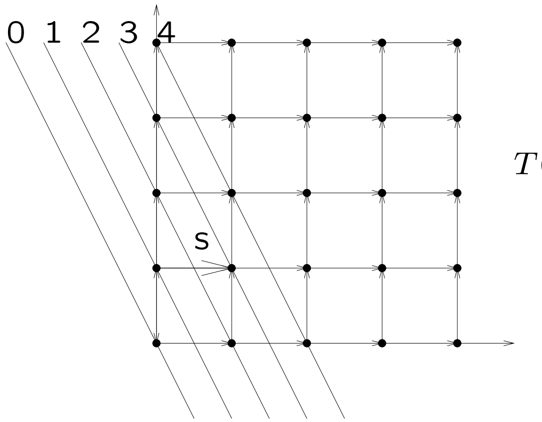
Chaque processeur est actif un top sur deux (activité : 50 %).

Le temps total d'exécution est deux fois plus long.

Explication: $\tau.s = \begin{pmatrix} 2 & 1 \end{pmatrix} . \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 2.$



## Partitionnement LSGP




$T(x) - T(x - s) = \tau.x - \tau(x - s)$

## Partitionnement LSGP

Sur un même processeur, deux calculs successifs sont séparés de $c = \tau.s = 2$ tops.

On peut donc regrouper deux processeurs virtuels dans un processeur réel sans changer le temps d'exécution de l'algorithme.



## Méthodologie de partitionnement LSGP

On a fait l'ordonnancement et l'allocation, et on trouve un réseau virtuel de $M$ processeurs.

On ne dispose que de $P$ processeurs réels.



## Méthodologie de partitionnement LSGP

On modifie l'ordonnancement pour obtenir $c = \tau.s = M/P$

On sait que chaque processeur virtuel est actif 1 top tout les $c$ On regroupe $c$ processeurs virtuels en 1 processeur réel.

On obtient un réseau de $P$ processeurs



## Partitionnement: conclusion

Partitionnement indispensable pour la génération de code ou de dessin de masque.

Partitionnement de réseau réguliers: LPGS ou LSGP.

Un méthode pour générer un réseau facilement partitionnable (LSGP), et pour trouver un partitionnement valide.

# Plan

I) Introduction

II) Ordonnancement

III) Transformation spatio-temporelles

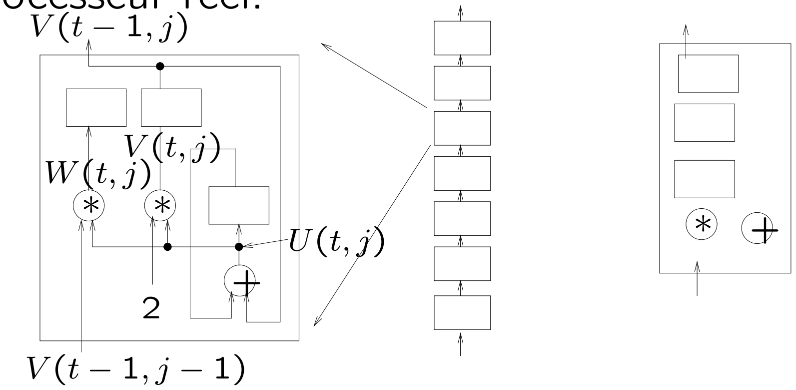
IV) Contraintes de ressources

IV-1 Partitionnement ; IV-2 Ordonnancement cyclique sous contraintes de ressource.



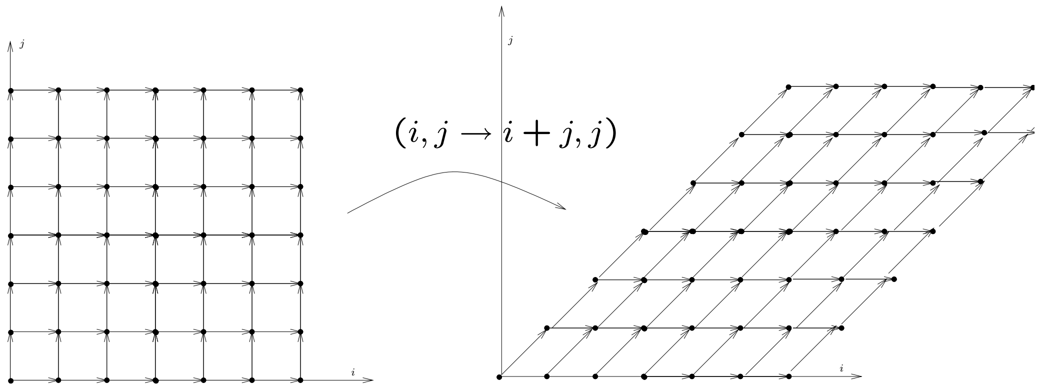
# Retour sur l'exemple initial

Si on ne dispose pas de suffisamment de ressources pour implémenter le processeur virtuel dans le processeur réel.






# Rappel du graphe de dépendance





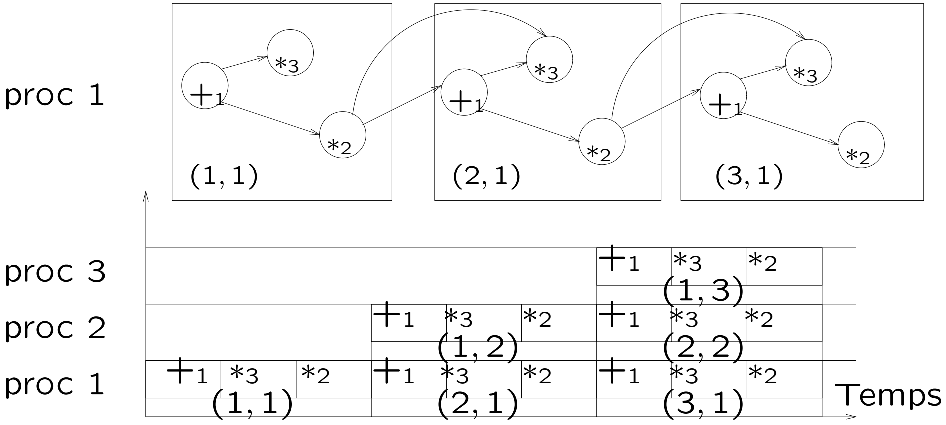

# Ordre des opérations dans un processeur

$$0 \le i,j \le N \;\rightarrow\; U(i,j) = U(i-1,j) +_1 V(i-1,j)$$
$$0 \le i,j \le N \;\rightarrow\; V(i,j) = 2 *_2 U(i,j)$$
$$0 \le i,j \le N \;\rightarrow\; W(i,j) = V(i,j-1) *_3 U(i,j)$$

Si on effectue les opérations dans l'ordre suivant:

$+_1, *_3 *_2$

## Diagramme espace temps




$$T_{total} = 2N(2T_{mul} + T_{add})$$



## Ordre des opérations dans un processeur

$$0 \leq i,j \leq N \;\rightarrow\; U(i,j) = U(i-1,j) +_1 V(i-1,j)$$
$$0 \leq i,j \leq N \;\rightarrow\; V(i,j) = 2 *_2 U(i,j)$$
$$0 \leq i,j \leq N \;\rightarrow\; W(i,j) = V(i,j-1) *_3 U(i,j)$$

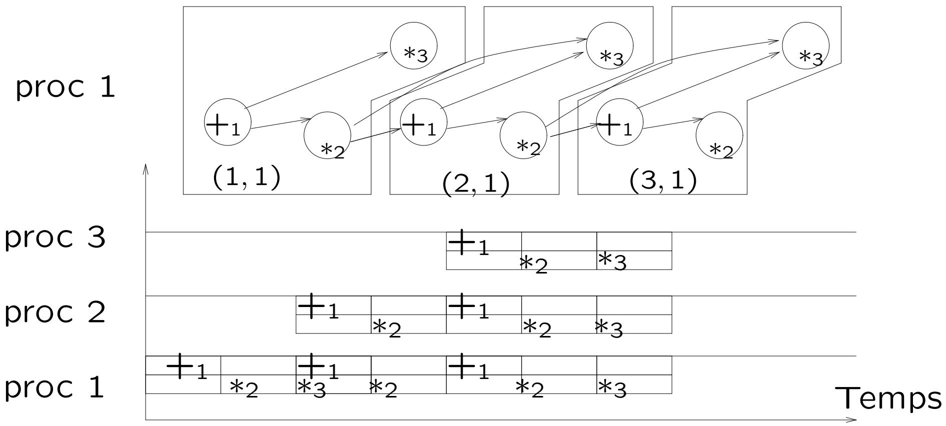
Si l'on effectue les opérations dans l'ordre suivant:

$+_1, *_2, *_3$

On gagne $2NT_{mul}$ tops sur l'exécution totale



## Ordre meilleur




$$T_{total} = 2N(T_{mul} + T_{add})$$



## Problème à résoudre

On veut un comportement cyclique.

Comment optimiser le **pipeline**?

Soit entre le démarrage de deux processeurs successifs.

Soit entre le démarrage de deux itérations successives sur un même processeur.

## Modélisation du problème

Après ordonnancement et Allocation, chaque processeur doit effectuer une série d'itération.

Modèle: on a un processeur qui exécute un ensemble d'opération de manière itérative (boucle sur le temps)

exemple: **for** t:=1 **to** N **do**
$U(t) = U(t-1) +_1 V(t-1)$
$V(t) = 2 *_2 U(t)$
$W(t) = INPUT() *_3 U(t)$



## Modélisation du problème

Chaque itération comporte $n$ opérations (chaque opération dure 1 top).

On dispose de $r$ ressources. L'opération $i$ peut être réalisée par la ressource $R(i)$



## Exemple

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

$n = 3$ opérations $+_1, *_2$ et $*_3$

$r = 2$ un multiplieur et un additionneur disponibles.



## Modélisation du problème

On connaît les dépendances entre les opérations d'une même itération et les dépendances entre itérations successives.

On cherche à minimiser la "période" ($\lambda$) c'est-à-dire le délai entre le début d'une itération et le début de la suivante.

## Exemple

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

Dépendances internes à l'itération:

$*_2 \leftarrow +_1$

$*_3 \leftarrow +_1$

Dépendance entre itérations:
$+_1 \leftarrow +_1$ de l'itération précédente
$+_1 \leftarrow *_2$ de l'itération précédente



## Résolution

On voudrait encore utiliser la programmation linéaire, mais dès qu'on a des contraintes de ressources, on a des contraintes disjonctives:

Exemple: $*_2$ est exécuté strictement avant $*_3$ **ou** (exclusif) $*_3$ est exécuté strictement avant $*_2$



## Utilisation de variables entières

Avec des variables entières, on peut:

- choisir une parmi un nombre fini de valeurs pour une variable ;

- choisir d'activer uniquement $k$ contraintes sur $m$ possibles (disjonction: activer une contrainte sur deux possibles)



## Utilisation en ordonnancement

On utilise souvent pour variables des matrices $X_{ij}$ ayant le rôle de décision: $x_{ij}$ vaut 1 si et seulement si la tâche $i$ est effectuée sur le processeur $j$.

## Résolution

On se fixe une borne supérieure $\Lambda$ pour l'exécution de toutes les opérations.

On va construire un programme linéaire qui nous donnera un ordonnancement des instructions de telle sorte que la période $\lambda$ soit minimale.



## Construction du programme linéaire

On introduit une variable triplement indexée

$$x_{ijk}, \quad 1 \leq i \leq n, \quad 1 \leq j \leq r, \quad 1 \leq k \leq \Lambda$$

$x_{ijk}$ vaut 1 si et seulement si l'opération $i$ est effectuée sur la ressource numéro $j$ au $k^{ieme}$ top de l'itération.



## Contraintes à vérifier

Contraintes de modélisation.

Contraintes de dépendances.

Contraintes de ressources.



## Exemple

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

On pose $\Lambda = 3$

$$x_{ijk}, \quad 1 \leq i \leq 3, \quad 1 \leq j \leq 2, \quad 1 \leq k \leq 3$$

## Contraintes que doit vérifier $x$

$x_{ijk}$ vaut 0 ou 1

$$0 \leq x_{ijk} \leq 1, \quad x \in N$$

une opération est exécutée une et une seule fois, sur une ressource adéquate:

$$\forall i,\ 1 \leq i \leq n \begin{cases} \Sigma_{k,j \in R(i)} x_{ijk} = 1 \\ \Sigma_{k,j \notin R(i)} x_{ijk} = 0 \end{cases}$$



## Exemple: contraintes du modèle

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

X matrice $3 \times 2 \times 3$ de variables booléennes

$$\begin{array}{ll} x_{111} + x_{112} + x_{113} = 1, & x_{121} + x_{122} + x_{123} = 0 \\ x_{221} + x_{222} + x_{223} = 1, & x_{211} + x_{212} + x_{213} = 0 \\ x_{321} + x_{322} + x_{323} = 1, & x_{311} + x_{312} + x_{313} = 0 \end{array}$$



## Contraintes de ressources

à chaque top, chaque ressource n'est utilisée que par une seule opération

$$\forall j, k,\ 1 \leq j \leq r \quad 1 \leq k \leq \Lambda, \quad \Sigma_{i=1}^{n} x_{ijk} \leq 1$$

$$\begin{array}{ll} x_{111} + x_{211} + x_{311} \leq 1, & x_{121} + x_{221} + x_{321} \leq 1, \\ x_{112} + x_{212} + x_{312} \leq 1, & x_{122} + x_{222} + x_{322} \leq 1, \\ x_{113} + x_{213} + x_{313} \leq 1, & x_{123} + x_{223} + x_{323} \leq 1, \end{array}$$



## dates d'exécution

La date effective d'exécution de l'opération $i$ (décalage par rapport au début de l'itération) peut être récupérée par:

$$\forall i, \quad 1 \leq i \leq n, \quad d_i = \Sigma_{k=1}^{\Lambda} k * (\Sigma_{j \in R(i)} x_{ijk})$$

$d_1 = x_{111} + 2x_{112} + 3x_{113}$

$d_2 = x_{211} + 2x_{212} + 3x_{213}$

$d_3 = x_{311} + 2x_{312} + 3x_{313}$

## Contraintes de dépendances

$i$ calculé $d_i$ tops après le début de l'itération

Les dépendances internes à une itération peuvent alors être imposées: si l'opération $i$ dépend directement de l'opération $j$:

$$d_i - d_j \geq 1$$



## Exemple: dépendance intra itération

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

$$d_2 - d_1 \geq 1$$

$$d_3 - d_1 \geq 1$$



## Dépendances inter itérations

Les dépendances entre deux itérations successives: si l'opération $i$ dépend directement de l'opération $j$ de l'itération précédente. La date effective de l'opération $j$ de l'itération précédente est décalée de $-\lambda$ par rapport à la date de l'opération $j$ de l'itération courante. Donc:

$$d_i - d_j + \lambda \geq 1$$



## Exemple: dépendance inter itérations

**for** t:=1 **to** N **do**

$U(t) = U(t-1) +_1 V(t-1)$

$V(t) = 2 *_2 U(t)$

$W(t) = INPUT() *_3 U(t)$

$$d_1 - d_2 + \lambda \geq 1$$

$$\lambda \geq 1$$

## Exemple: Résolution du programme linéaire

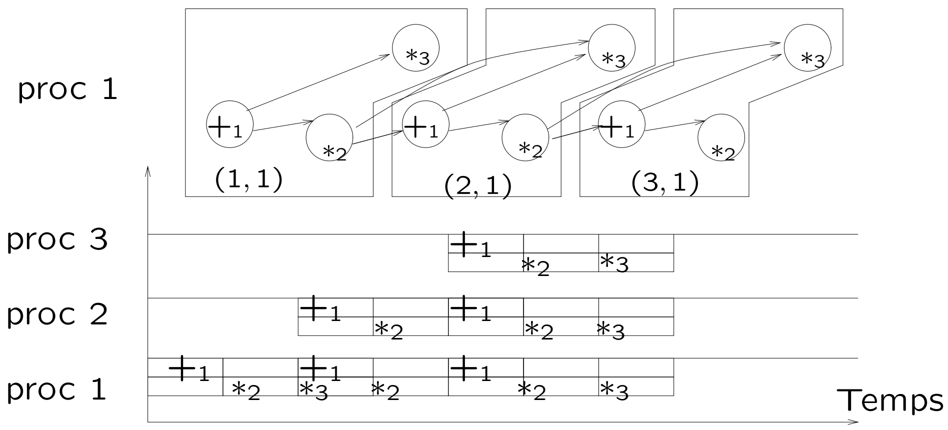
On cherche à minimiser $\lambda$ sous les contraintes

On résout avec un solveur d'inéquations linéaires en nombre entiers.

$x_{111} = 1$, $d_1 = 1$, $x_{222} = 1$, $d_2 = 2$, $x_{333} = 1$, $d_3 = 3$ $\lambda = 2$



## résultat




$$T_{total} = 2N(T_{mul} + T_{add})$$



## En fait ....

Nous n'avons pas pris en compte le fait que deux itérations peuvent se chevaucher ($\Lambda > \lambda$) (contraintes de cyclicité).

Si on veut tout résoudre, on est obligé de fixer $\lambda$ et d'essayer: existe-t-il une solution pour ce $\lambda$?



## contraintes de cyclicité

(Remplace les contraintes de ressourse)
à chaque top, chaque ressource n'est utilisée que par une seule opération (de l'itération courante ou des autres itérations).

$\forall j,k,\ \ 1 \le j \le r \quad 1 \le k \le \Lambda$

$\Sigma_{i=1}^{n} \Sigma_{\alpha \ | \ 1 \le k + \alpha\lambda \le \Lambda} \left( x_{ijk+\alpha\lambda} \right) \le 1$

Si $\lambda = 2$
$x_{111} + x_{211} + x_{311} + x_{113} + x_{213} + x_{313} \le 1$,
$x_{121} + x_{221} + x_{321} + x_{123} + x_{223} + x_{323} \le 1$,
$x_{112} + x_{212} + x_{312} \le 1$,
$x_{122} + x_{222} + x_{322} \le 1$, $x_{113} + x_{213} + x_{313} \le 1$,
$x_{123} + x_{223} + x_{323} \le 1$,

## Généralisation

Si les opérations ont une durée non unitaire:

La ressource $j$ est prise $l_j$ tops après le début de l'opération. La contrainte de ressources s'exprime alors:

"Sur chaque ressource $j$, les départs de deux tâches succéssives sont espacé de $l_j$ tops au moins"



## Opération de durée $l_J$

Pour $l_j$ tops successifs, on peut faire la somme des variables indiquant le départ d'une tâche sur la ressource $j$.

Remarque: Il faut connaître exactement $\lambda$ (la notion de $l_j$ *tops successifs* en dépend).

$\forall k,\ 1 \leq k \leq \Lambda, \forall j,\ 1 \leq j \leq r,$
$\Sigma_{i=1}^{n}(\Sigma_{l=1}^{l_j} x_{ij(k-l)mod\lambda}) \leq 1$



## Généralisation

Pipeline (Période $\lambda_2$) entre deux processeurs successifs

On peut minimiser une fonction de $\lambda$ et $\lambda_2$. Par exemple $\lambda + \lambda_2$ (puisque $T_{total} = 2N(\lambda + \lambda_2)$)

On traite les dépendances entre processeurs comme les dépendances inter-itération



## Pipeline entre deux processeurs successifs

**for** t:=1 **to** N **do**
$U(t,j) = U(t-1,j) +_1 V(t-1,j)$
$V(t,j) = 2 *_2 U(t,j)$
$W(t,j) = \mathbf{V(t-1,j-1)} *_3 U(t,j)$

$d_3 - d_2 + \lambda_2 \geq 1$

## Généralisation

Contraintes de registres.

Les registres sont souvent inclus dans les contraintes de ressources car ils coûtent cher.

Les contraintes de registres sont plus difficiles à modéliser car elles sont du type:

“Cette ressource est prise entre telle opération et telle opération”



## Contraintes de registres

En général, on ne peut pas écrire le programme linéaire si on ne connaît pas exactement la valeur de la période $\lambda$.

$\Rightarrow$ on écrit un programme linéaire pour **vérifier** qu’une certaine période $\lambda$ peut convenir et trouver les allocations correspondantes.



## Application: Software pipelining

Cette technique est implémentée dans les compilateurs pour optimiser l’exécution de boucles sur un processeur en optimisant l’utilisation des registres



## Contraintes de ressources: conclusion

On sait toujours comment poser un problème linéaire qui vérifie qu’une période donnée est valide.

L’ordonnancement cyclique sous contraintes de ressources est un problème difficile (10 opérations, 5 opérateurs, $\Lambda = 25$ tops $\Rightarrow >$ 1250 variables: presque impossible à résoudre).

En pratique on utilise des heuristiques ($\Rightarrow$ expertise nécessaire).