Pipeline logiciel

Plan

1. Formulation du problème
2. Rappels sur l'ordonnancement de graphes de tâches
3. Résolution avec infinité de ressources

1 Formulation du problème

Software pipelining
– Ordonnancer un ensemble de tâches génériques
– Exécution un grand nombre de fois (coeur d'une boucle externe)
– Ré-ordonner pour utiliser au mieux les ressources matérielles.

Exemple de référence

```
DO k=0, N-1
    (A) : a(k) = c(k-1)
    (B) : b(k) = a(k-2) * d(k-1)
    (C) : c(k) = b(k) + 1
    (D) : d(k) = f(k-1)/3
    (E) : e(k) = sin(f(k-2))
    (F) : f(k) = log(b(k) + e(k))
ENDDO
```

– 6 instructions A, B, C, D, E, et $F \to$ tâches génériques
– N itérations, N paramètre de valeur inconnue
– (v, k) instance de la tâche générique v à l'itération k, délai $d(v)$.

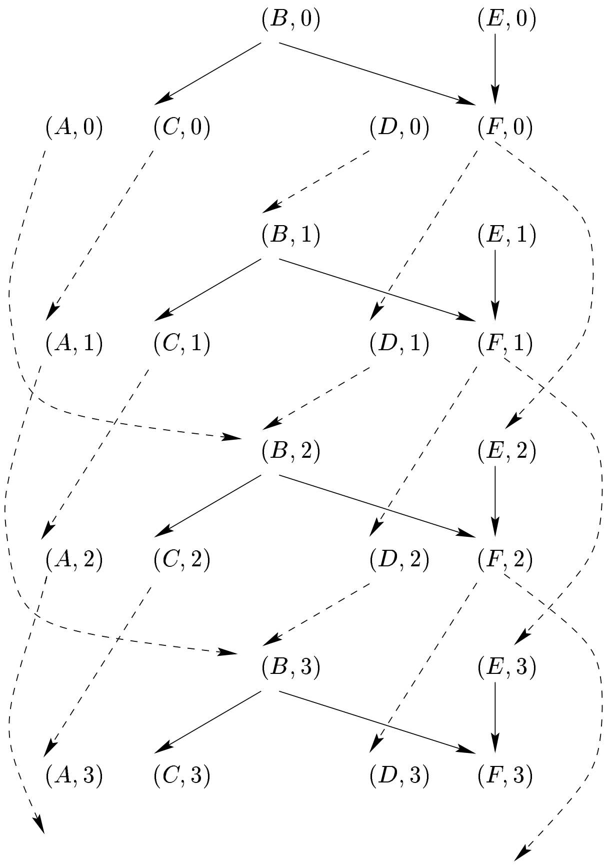
Contraintes de précédence

– GDE de sommets $\{(v,k) \mid v \in V,\ 0 \le k < N\}$

– $V = \{A, B, C, D, E, F\}$

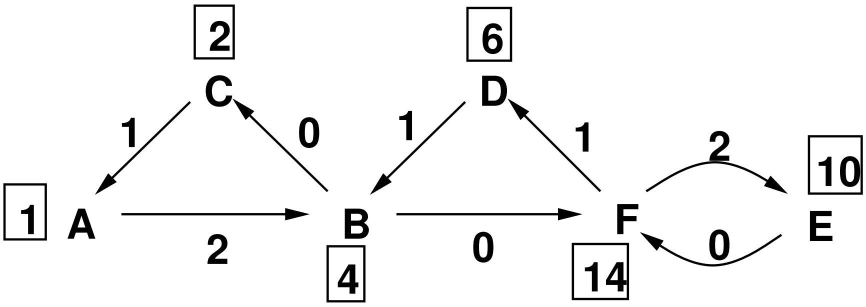
– Structure régulière des dépendances

1. de niveau ∞ (trait plein)
2. de niveau 1 (pointillé).



Graphe de dépendance réduit
- $G = (V, E, d, w)$ GDR
- $d : V \to \mathbb{N}^*$: délai de chaque tâche générique
- $w : E \to \mathbb{N}$: distance de dépendance de chaque arête
- Ordonnancement : $\sigma : V \times \mathbb{N} \to \mathbb{N}$
- Contraintes de dépendance :

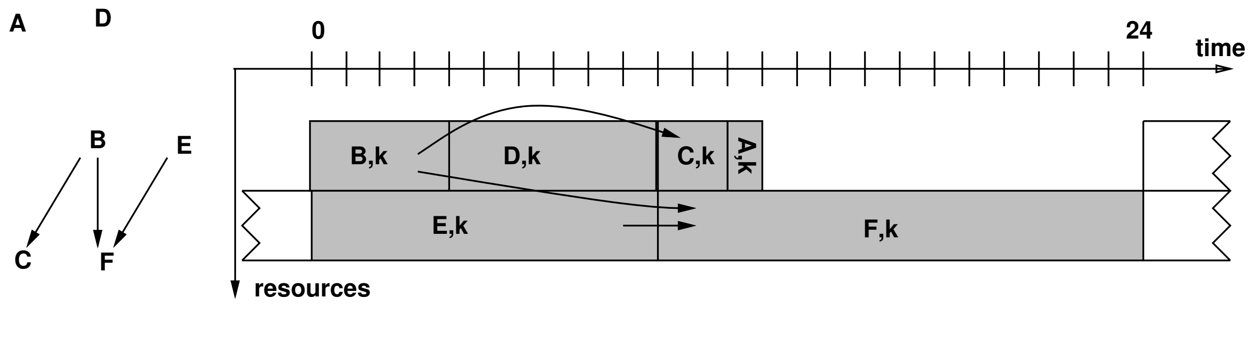
$$\forall e = (u, v) \in E,\ \forall k \geq 0,\ \sigma(v, k + w(e)) \geq \sigma(u, k) + d(u)$$



Ordonnancements cycliques
- $\sigma(v, k) = c_v + \lambda k$ avec $c_v \in \mathbb{N}$ et $\lambda \in \mathbb{N}$
- σ ordonnancement périodique, de période λ
- σ ordonnance des “tranches” de longueur λ
- A l’intérieur de chaque tranche, une et une seule instance de chaque tâche générique est initiée.

Retour à l'exemple de référence

– $A(G)$ sous-graphe du GDR avec seulement les arêtes de poids nul
– $A(G)$ caractérise toutes les dépendances de niveau ∞
– $A(G)$ est acyclique
– Ordonnancement de liste pour $A(G)$
– Chemin critique σ_a avec 2 processeurs



(a) (b)

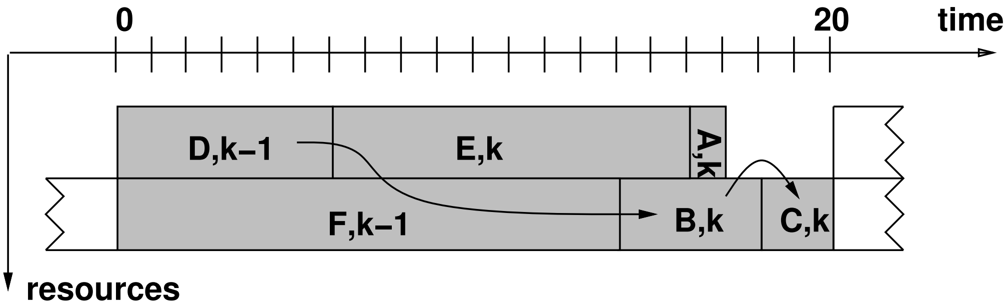
– $\lambda = 24$: latence de l'ordonnancement de liste σ_a
– $\sigma(v, k) = \sigma_a(v) + \lambda k$
– Contraintes de dépendance satisfaites :

1. de niveau ∞ : imposées par σ_a ;
2. de niveau 1 : choix de λ

Parenthèse : meilleure solution

– $\sigma(v,k) = c_v + \lambda k$, où $\lambda = 20$

Tâche générique v	*A*	*B*	*C*	*D*	*E*	*F*
Constante c_v	16	14	18	20	6	20



– Validité de cet ordonnancement :
  – Vérification directe pour chacune des 8 arêtes de dépendance dans le GDR
  – Lire le poly :-)

Contraintes de ressources

– Chaque ressource ne peut exécuter qu'une opération à la fois
– Opérations atomiques (une fois initiée, ne peut être interrompue)
– Avec p ressources identiques :

$$\forall t \in \mathbb{N},\ |\{(v,k) \mid t - d(v) < \sigma(v,k) \leq t\}| \leq p$$

– Cas général : ressources de différents types (additionneurs et multiplieurs), ou pipelinées (une opération peut être initiée à chaque top).

Bornes inférieures pour λ

– Indépendamment du nombre de ressources, $M \geq w(\mathcal{P})$ pour tout chemin $\mathcal{P}$ dans le graphe
– Si on dispose de p ressources, $M \geq \frac{\text{Seq}}{p}$, où Seq est le temps séquentiel, i.e. la somme des durées de toutes les tâches
– Pour un cycle $\mathcal{C}$, $\rho(\mathcal{C})$ est le *rapport* durée sur distance $\frac{d(\mathcal{C})}{w(\mathcal{C})}$.
– Pour tout ordonnancement σ de temps de cycle λ_σ :

$$\text{pour tout cycle } \mathcal{C},\ \lambda_\sigma \geq \rho(\mathcal{C}) \tag{1}$$

$$\text{si } p \text{ ressources sont disponibles, } p\lambda_\sigma \geq \sum_{v \in V} d(v) \tag{2}$$

2 Rappels sur l'ordonnancement de graphes de tâches

Graphes de tâches

- Ordonnancement d'un système de tâches $G = (V, E, w)$: fonction $\sigma : V \longrightarrow \mathbb{N}^*$ avec $\sigma(u) + w(u) \leq \sigma(v)$ pour toute arête $e = (u, v) \in E$.
- Soit $G = (V, E, w)$ un système de tâches. Il existe un ordonnancement ssi G est sans cycle
- Avec p processeurs $\rightarrow$ contraintes de ressource : fonction d'allocation alloc : $V \longrightarrow \mathcal{P}$, où $\mathcal{P} = \{1, \ldots, p\}$ avec

$$\mathrm{alloc}(T) = \mathrm{alloc}(T') \Rightarrow \left\{ \begin{array}{rl} & \sigma(T) + w(T) \leq \sigma(T') \\ ou & \sigma(T') + w(T') \leq \sigma(T) \end{array} \right.$$

- Latence $\mathrm{MS}(\sigma, p)$: temps d'exécution total

$$\mathrm{MS}(\sigma, p) = \max_{v \in V}\{\sigma(v) + w(v)\} - \min_{v \in V}\{\sigma(v)\}$$

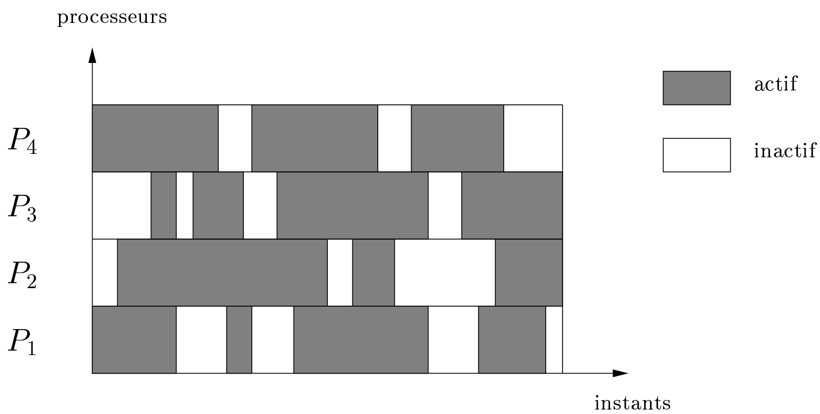
- $\mathrm{Pb}(\infty)$: déterminer un ordonnancement de latence minimale avec ressources illimitées
- $\mathrm{Pb}(p)$: déterminer un ordonnancement de latence minimale avec p processeurs.

Propriétés

– Soit $G = (V, E, w)$ un DAG et σ a ordonnancement pour G avec p processeurs. Alors $\mathrm{MS}(\sigma, p) \geq w(\Phi)$, pour tout chemin Φ de G.

– Efficacité $e(\sigma, p) == \frac{\mathrm{Seq}}{p \times \mathrm{MS}(\sigma,p)}$, où $\mathrm{Seq} = \sum_{v \in V} \; w(v)$
Alors

$$0 \leq e(\sigma, p) \leq 1$$



Complexité de Pb(∞)

– Pour $v \in V$, PRED(v) prédécesseurs immédiats de v
– Pour $v \in V$, SUCC(v) successeurs immédiats de v
– $v \in V$ est un sommet d'entrée si PRED(v) $= \emptyset$.
– $v \in V$ est un sommet de sortie si SUCC(v) $= \emptyset$.
– Niveau supérieur $tl(v)$ (top level) : plus grand poids d'un chemin menant d'un sommet d'entrée à v, en excluant le poids de v.
– Niveau inférieur $bl(v)$ (bottom level) : plus grand poids d'un chemin menant de v à un sommet de sortie, incluant le poids de v

– Calcul en traversant le graphe :

$$tl(v) = 0 \text{ si } v \text{ sommet d'entrée}$$

$$tl(v) = \max\{tl(u) + w(u); u \in \mathrm{PRED}(v)\} \text{ sinon}$$

– De même, $bl(v) = \max\{bl(u); u \in \mathrm{SUCC}(v)\} + w(v)$ (et $bl(v) = w(v)$ pour un sommet de sortie v).

Proposition Soit $G = (V, E, w)$ un DAG. L'ordonnancement libre σ_{free} défini par

$$\sigma_{free}(v) = tl(v), \forall v \in V$$

est un ordonnancement optimal de G
$\rightarrow$ Pb(∞) peut être résolu en temps $O(|V| + |E|)$.

Résolution de Pb(p)

– NP-complétude de Pb(p)
– Heuristiques de liste : à chaque instant, on alloue autant de tâches que possible aux processeurs disponibles
– Toute stratégie *ne laissant pas délibérément un processeur inactif* est à un facteur deux de l'optimal.

Garantie de performance

– $G = (V, E, w)$ DAG, σ un ordonnancement pour G
– $v \in FREE(\sigma, t)$ (libre au temps t) si son exécution n'a pas encore commencé ($\sigma(v) \geq t$), mais tous ses prédécesseurs ont été exécutés au temps t ($\forall\, u \in \mathrm{PRED}(v),\ \sigma(u) + w(u) \leq t$)
– Ordonnancement de liste : à tout instant t,

  si $|FREE(\sigma, t)| = r \geq 1$,
  si q processeurs sont disponibles,
  alors on débute l'exécution de $\min(r, q)$ tâches libres.

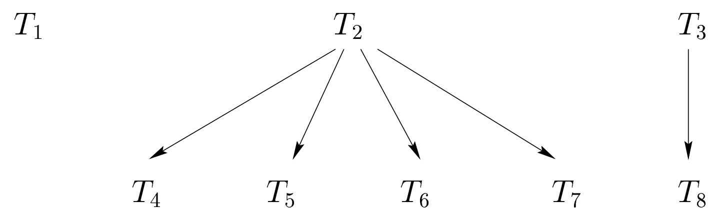
Théorème Soit $G = (V, E, w)$ un DAG avec p processeurs. Soit σ un ordonnancement de liste quelconque pour G. Soit $\mathrm{MS}_{opt}(p)$ la latence d'ordonnancement optimal pour G. Alors

$$\mathrm{MS}(\sigma, p) \leq (2 - \frac{1}{p})\mathrm{MS}_{opt}(p)$$

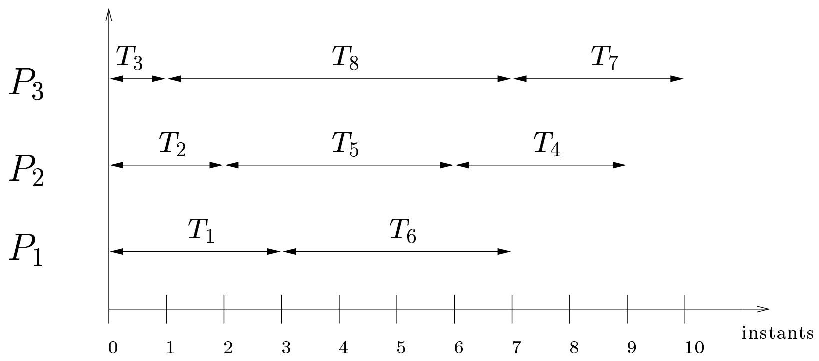
Implémentation d'un ordonnancement de liste

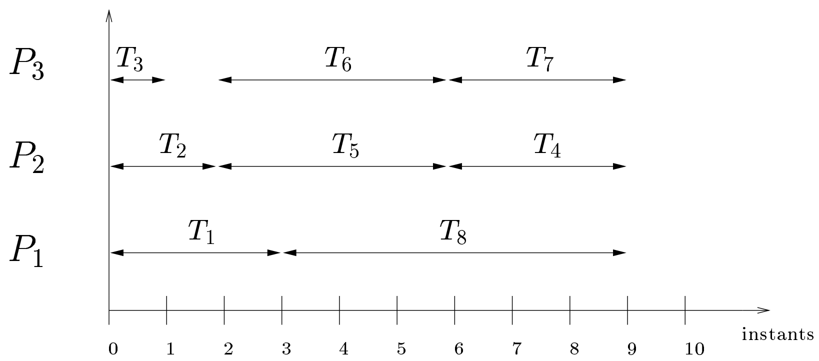
1. *Initialisation :*
   (a) Calculer le niveau de priorité de toutes les tâches.
   (b) Constituer la file de priorité comme étant la liste des tâches libres (tâches sans prédécesseurs) triées en ordre de priorité décroissante.
   (c) Soit t l'instant courant : $t = 0$.
2. *Tant qu'il reste des tâches à exécuter* :
   (a) Ajouter les nouvelles tâches libres, s'il y en a, à la file de priorité. Si l'exécution d'une tâche se termine à l'instant t, supprimer cette tâche de la liste des prédécesseurs de chacun de ses successeurs. Ajouter les tâches dont la liste des prédécesseurs est devenue vide à la file.
   (b) S'il y a q processeurs disponibles et r tâches dans la file, supprimer les $\min(q, r)$ premières tâches de la file et les exécuter ; si T est l'une de ces tâches, poser $\sigma_{cp}(T) = t$.
   (c) Incrémenter t.

Chemin critique
– *Chemin critique* d'une tâche : son niveau inférieur
– Utilisé pour déterminer les niveaux de priorité entre tâches



Tâches	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
Poids	3	2	1	3	4	4	3	6
Chemin critique	3	6	7	3	4	4	3	6





Ordonnancement optimal

3 Résolution avec infinité de ressources

Outils
- Graphes à potentiel
- Algorithme de Bellman-Ford
- Lire le poly !

Ordonnancement optimal à ressources illimitées
- Graphe $G = (V, E, d, w)$
- d définit les poids des sommets, w ceux des arêtes
- Ordonnancement $\sigma(v, k) = \lambda k + c(v)$
- On cherche λ minimal

- **Théorème**

$$\lambda_\infty = \max\{\rho(\mathcal{C}) \mid \mathcal{C} \text{ cycle élémentaire de } G\}$$

  (et $\lambda_\infty = 0$ si G est acyclique)
- Constante $c(v)$: plus long chemin de s à v dans le graphe $G'_\lambda = (V', E', w'_\lambda)$:
  1. Sommets de G'_λ : ajouter à V un nouveau sommet s :

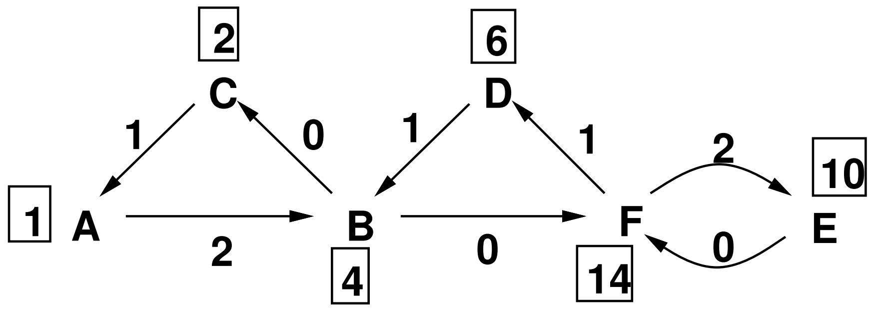
$$V' = V \cup \{s\}.$$

  2. Arêtes de G'_λ : pour chaque sommet $v \in V$ ajouter à E une arête de s à v : $E' = E \cup (\{s\} \times V)$.
  3. Poids des arêtes de G'_λ :

$$w'_\lambda(e) = 0 \text{ si } e = (u, v) \in E' \setminus E$$

$$w'_\lambda(e) = d(u) - \lambda w(e) \text{ sinon.}$$

Retour à l'exemple



– Trois cycles élémentaires

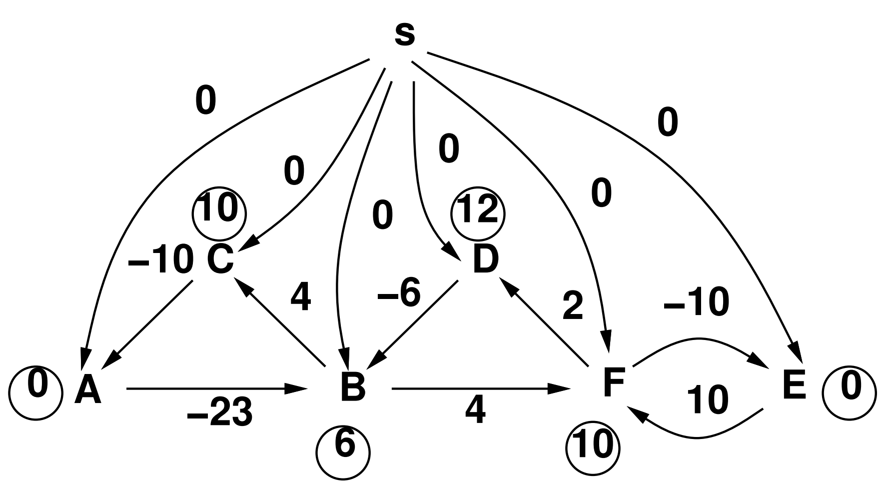
– $\rho(C_1) = \frac{7}{3}$ pour $C_1 : A \to B \to C \to A$

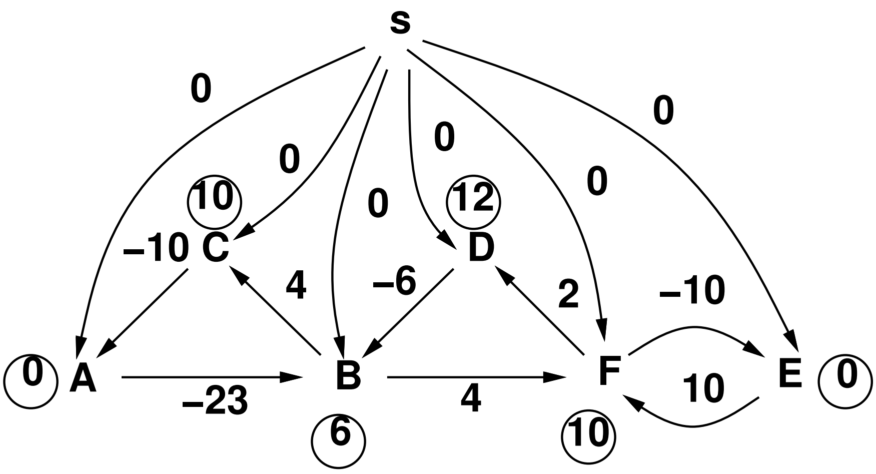
– $\rho(C_2) = \frac{24}{2}$ pour $C_2 : B \to F \to D \to B$

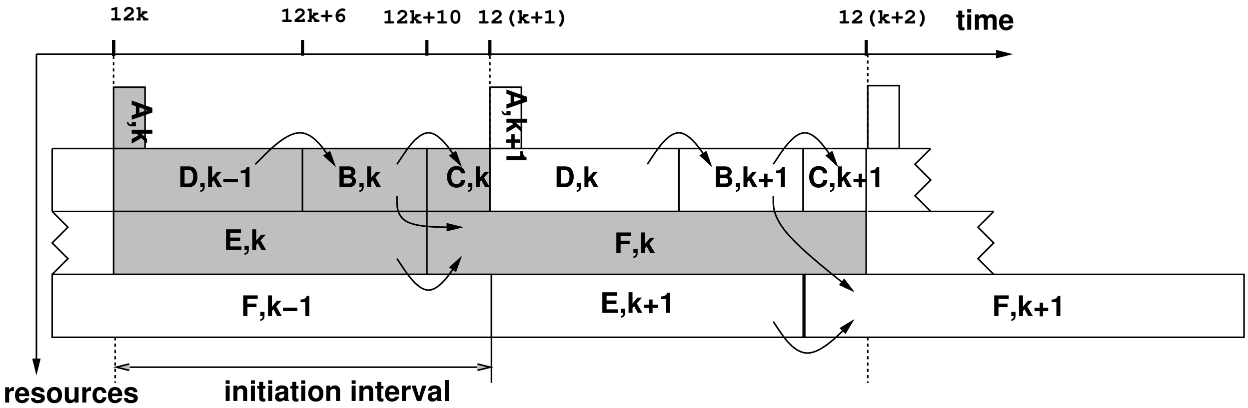
– $\rho(C_3) = \frac{24}{2}$ pour $C_3 : E \to F \to E$

– Donc $\lambda_\infty = 12$

Calcul des plus longs chemins dans le graphe G'_{12} :







Quatre processeurs sont nécessaires :

Instant de calcul	λk	$\lambda k{+}6$	$\lambda k{+}10$	$\lambda k{+}12$	$\lambda k{+}18$	$\lambda k{+}22$
Processeur P_1	(A,k)			$(A,k+1)$		
Processeur P_2	$(D,k{-}1)$	(B,k)	(C,k)	(D,k)	$(B,k{+}1)$	$(C,k{+}1)$
Processeur P_3	(E,k)		(F,k)			
Processeur P_4				$(E,k{+}1)$		$(F,k{+}1)$