Programmation parallèle (PPAR)
Cours 4 : parallélisation automatique, application au parallélisme de controle

P. Fortin
pierre.fortin @ upmc.fr

D'après le cours de J.-L. Lamotte

Master 2 Informatique - UPMC

Comment paralléliser un programme séquentiel ?

- Dans le cadre du parallélisme de tâches, la parallélisation d'un algorithme séquentiel requiert en premier lieu un partitionnnement de cet algorithme en tâches si possibles indépendantes.
- Le partitionnement peut être fait :
  - par le programme (compilateur)
  - par l'utilisateur.
- L'exécution des tâches est alors soumise à des contraintes de temps. Il est donc nécessaire d'élaborer pour l'algorithme un graphe de tâches permettant de repérer les tâches effectivement indépendantes.
- Il faut ensuite affecter les tâches aux processeurs en respectant l'architecture du système informatique.

Hypothèses sur l'architecture

- Le matériel consiste en un ensemble de processeurs identiques possédant chacun une mémoire locale ou partageant une mémoire commune (machine à mémoire distribuée ou partagée).
- Dans le cas d'une machine distribuée, les transferts de données sont faits par un réseau d'interconnexion. Les temps de tranfert des données ne sont pas pris en compte.
- La lecture simultanée ou l'envoi simultané d'une même donnée à plusieurs processeurs est possible.
- L'écriture simultanée par des processeurs différents dans une même cellule de mémoire est impossible (de même que la réception simultanée d'une même donnée depuis plusieurs processeurs).

Notion de tâche

- Une tâche est une unité indivisible de traitement caractérisée par son comportement extérieur. Elle est représentée par un quadruplet.

$$T = (E, S, f, t)$$

avec :

- E, l'ensemble des entrées de la tâche T
- S, l'ensemble des sorties de la tâche T
- f, un opérateur de E dans S, *i.e.* une suite d'opérations élémentaires ordonnées de E dans S
- t : est le temps d'exécution de la tâche T

Notion de tâche ...

- *Remarque 1 :* Le temps t d'exécution d'une tâche est en général la somme du temps de calcul correspondant aux opérations élémentaires de la tâche et du temps de transfert (ou de communication) des données entre processeurs et mémoire (ou entre processeurs).
- *Remarque 2 :* Dès qu'une tâche est affectée à un processeur celui-ci ne peut s'interrompre avant la fin de la tâche. La tâche est donc indivisible.

Quelques définitions

Définition (Algorithme séquentiel)

Un **algorithme séquentiel** *est un ensemble de tâches muni d'un ordre total noté* $\longrightarrow$

$$A : (T_1, T_2, ..., T_n, \longrightarrow) \qquad T_1 \longrightarrow T_2 \longrightarrow ... \longrightarrow T_n$$

A peut aussi être considéré comme une tâche représentée par

$$A : (E_A, S_A, f_A, t_A)$$

et on a :

$$E_A \subseteq \bigcup_{i=1}^{n} E_{T_i} \quad S_A \subseteq \bigcup_{i=1}^{n} S_{T_i} \quad f_A \subseteq \bigotimes_{i=1}^{n} f_{T_i} \quad t_A \leq \sum_{i=1}^{n} t_{T_i}$$

Remarque : Si un algorithme comporte un branchement conditionnel celui-ci est supposé être interne à une tâche et le temps de la tâche est le maximum des divers temps possibles.

Définition (Granularité)

La décomposition d'un algorithme en tâches peut être plus ou moins fine. La granularité est une mesure grossière de cette décomposition correspondant au rapport

$$r = \frac{\text{temps moyen d'exécution d'une tâche}}{\text{temps total d'exécution de l'algorithme}}$$

Remarque : Le choix de la granularité est difficile et dépend de l'algorithme et de l'architecture du système.

Définition (Système de tâches)

Un **système de tâches** *est un ensemble de tâches muni d'un ordre partiel :*

$$S = (T_1, ..., T_n, \prec)$$

L'ordre partiel est noté $\prec$

Définition (Indépendance)

Deux tâches T_i *et* T_j *sont* **indépendantes** *si et seulement si elles vérifient les conditions de Bernstein :*

$$S_{T_i} \bigcap S_{T_j} = S_{T_i} \bigcap E_{T_j} = E_{T_i} \bigcap S_{T_j} = \emptyset$$

Définition (Système de précédence)

Un système de tâches est un **système de précédence** *si et seulement si* $\forall i$ et $j \in [1,...,n]$ *une seule des trois conditions est vérifiée :*

1. $T_i \prec T_j$
2. $Tj \prec Ti$
3. T_i et T_j sont indépendantes.

Définition (Consécutivité)

Deux tâches T_i *et* T_j *sont* **consécutives** *si et seulement si :*

$T_i \prec Tj$ et $\nexists\ T_k$ tel que $T_i \prec T_k \prec T_j$

Définition (Graphe de précédence)

Le **graphe de précédence** *d'un ensemble de tâches associé à un système de précédence* $S = (T_1,...,T_n,\prec)$ *est tel que :*

- l'ensemble des sommets est l'ensemble des tâches,
- deux sommets T_i et T_j sont reliées par un arc (de T_i vers T_j) si et seulement si les tâches T_i et T_j sont consécutives.

Construction du système de précédence

Lemme

Soit $A = (T_1,...,T_n,\longrightarrow)$ *un algorithme séquentiel. Il existe un système de précédence unique :*

$$S = (T_1,...,T_n,\prec)$$

tel que $\prec$ *soit un sous ordre de* $\{\longrightarrow\}$*, la relation partielle* $\prec$ *étant définie par*

$$T_i \prec T_j \Leftrightarrow T_i \longrightarrow T_j \text{ et } T_i, T_j \text{ non indépendantes (dépendantes).}$$

Démonstration du lemme

- Le fait que $\{\prec\}$ soit un sous ordre de $\{\longrightarrow\}$ est évident par définition.
- Vérifions alors que S est un système de précédence :
  - Soient T_i et T_j deux tâches non ordonnées par $\prec$,
  - comme A est séquentiel, T_i et T_j sont ordonnées par $\{\longrightarrow\}$ soit $T_i \longrightarrow T_j$.
  - En conséquence, d'après la définition de l'ordre partiel $\prec$, T_i et T_j sont indépendantes.
- L'unicité se vérifie de la même façon.

Algorithme

Construction du système de précédence

1. Choisir T_1
2. Considérer T_2
   Si T_1 et T_2 ne sont pas indépendantes
   $\Longleftrightarrow (S_{T_1} \bigcap S_{T_2}) \bigcup (S_{T_1} \bigcap E_{T_2}) \bigcup (E_{T_1} \bigcap S_{T_2}) \neq \emptyset$
   Alors poser $T_1 \prec T_2$.
3. Pour k de 3 à n :
   Pour i de 1 à $k-1$:
   Si T_i et T_k non indépendantes Alors poser $T_i \prec T_k$

- Cet algorithme crée un système de précédence auquel est associé un graphe orienté sans cycle.
- Ce graphe n'est pas encore le graphe de précédence car dans ce graphe il existe un arc entre deux sommets correspondants à deux tâches dont l'une est avant l'autre mais non nécessairement de façon consécutive.

Construction du graphe de précédence

- À partir du système de précédence S, il suffit de déterminer quelles sont les tâches consécutives.
- Notations :
  - Le graphe de précédence sera noté G.
  - Pour un nœud j de G, A_j est l'ensemble des nœuds i tels que $T_i \prec T_j$ (antécédents de j) et C_j est l'ensemble des nœuds i tels que T_i et T_j sont consécutives (prédécesseurs directs de j).
  - Les paires (i,j) telles que $i \in C_j$ sont les arcs de G

Algorithme

Construction du graphe de précédence

1. $A(1) = \varnothing$, $C(1) = \varnothing$;
2. SI T_1 et T_2 non indépendantes
   ALORS $A(2) = C(2) = \{1\}$;
   SINON $A(2) = C(2) = \varnothing$;
3. POUR k variant de 3 à n FAIRE
   $A(k) = \varnothing$; $C(k) = \varnothing$;
   POUR i variant de $k-1$ à 1 par pas de -1 FAIRE
   SI (T_i et T_k non indépendantes et $i \notin A(k)$) ALORS
   $C(k) = C(k) \bigcup \{i\}$;
   $A(k) = A(k) \bigcup A(i) \bigcup \{i\}$;
   FIN POUR
   FIN POUR

Exercice

Exemple

T_1 :lire D	$E_1 = \{\varnothing\}$	$S_1 = \{D\}$
T_2 :V=D*D	$E_2 = \{D\}$	$S_2 = \{V\}$
T_3 :W=D+V	$E_3 = \{D, V\}$	$S_3 = \{W\}$
T_4 :X=D*V	$E_4 = \{D, V\}$	$S_4 = \{X\}$
T_5 :Y=W+X	$E_5 = \{W, X\}$	$S_5 = \{Y\}$
T_6 :Z=Y-V	$E_6 = \{Y, V\}$	$S_6 = \{Z\}$

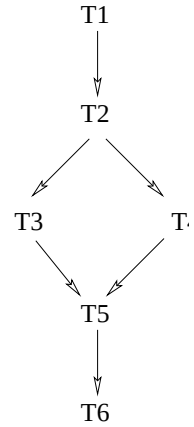
Résultat : $Z = D + D^3$

Système de précédence :

$$\begin{array}{lll} T_1 \prec T_2 & T_1 \prec T_3 & T_1 \prec T_4 \\ T_2 \prec T_3 & T_2 \prec T_4 & T_2 \prec T_6 \\ T_3 \prec T_5 & & \\ T_4 \prec T_5 & & \\ T_5 \prec T_6 & & \end{array}$$

Graphe de précédence :

étape	A	C
1	$\varnothing$	$\varnothing$
2	$\{1\}$	$\{1\}$
3	$\{2, 1\}$	$\{2\}$
4	$\{2, 1\}$	$\{2\}$
5	$\{4, 3, 2, 1\}$	$\{3, 4\}$
6	$\{5, 4, 3, 2, 1\}$	$\{5\}$



L'ordonnancement des tâches

- **Ordonnancement (et placement)** : attribuer une date de début d'exécution à chaque tâche et affecter les tâches à des processeurs disponibles, en respectant les contraintes du graphe de précédence des tâches.

Théorème

L'ordonnancement optimal d'un système quelconque de tâches sur un nombre fixé de processeurs est un problème NP complet.

- Voir la démonstration dans le livre Operating system theory, E.G Coffmann et P.J. Denning, Prentice Hall, 1972.
- Il est cependant possible de résoudre l'ordonnancement pour des systèmes de précédence particuliers.

Définition (Ordonnancement)

Soit p un nombre fini de processeurs. Un ordonnancement compatible avec un système de précédence $S = (T_1, ..., T_n, \prec)$ est une application notée ord telle que :

$$(T_1, ..., T_n) \xrightarrow{ord} (1, ..., p) \times \mathbb{N}$$
$$ord(T_k) = (proc(T_k), tps(T_k))$$

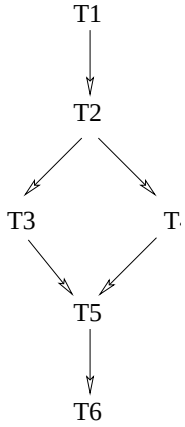
$$\begin{array}{lcll} & & \overset{et}{\Longrightarrow} & \\ (1) & T_i \prec T_k & \Longrightarrow & tps(T_i) + t_i \leq tps(T_k) \\ (2) & proc(T_i) = proc(T_k) & \Longrightarrow & tps(T_i) + t_i \leq tps(T_k) \\ & & ou & tps(T_k) + t_k \leq tps(T_i) \end{array}$$

Notations :

$Proc(T_i)$: processeur executant la tâche T_i
$tps(T_i)$: temps total (« date ») au début de l'execution de T_i
t_i : temps d'execution de T_i

Exemple

Exemple sur le graphe de précédence et 2 processeurs ($p = 2$), avec $\forall i, t_i = 1$:



$ord(T_1) = (1,0)$
$ord(T_2) = (1,1)$
$ord(T_3) = (1,2)$
$ord(T_4) = (2,2)$
$ord(T_5) = (1,3)$
$ord(T_6) = (1,4)$

Définition (Algorithme parallèle)

Un **algorithme parallèle** *est un couple* $P_a = (S, ord)$ *tel que ;*

- *S est système de précédence associé à un algorithme séquentiel A,*
- *ord est ordonnancement compatible avec S.*

- Le temps d'exécution d'un algorithme parallèle est :

$$T_{Pa} = \max_{k=\{1\cdots n\}} (tps(T_k) + t_k)$$

- L'algorithme parallèle est optimal si T_{Pa} est minimal.

Décomposition d'un graphe de précédence par prédécesseurs : D_p

Niveau 1 : toutes les tâches n'ayant pas de prédécesseurs

Niveau k : tâches dont tous les prédécesseurs sont à un niveau inférieur et dont au moins un prédécesseur est au niveau $k-1$.

- Itérer le processus de $k = 2, \ldots, H(G)$ où $H(G)$ est la hauteur du graphe (définie comme le nombre de sommets du plus long chemin dans le graphe)
- Cet ordonnancement conduit à un temps de début d'exécution minimum pour chacune des tâches.

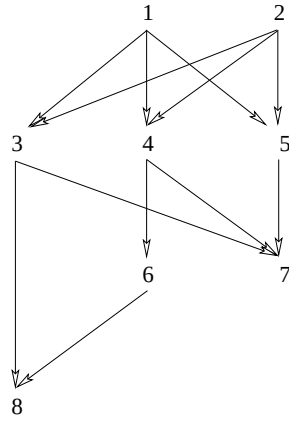
Décomposition d'un graphe de précédence par successeurs : D_s

Niveau H(G) (hauteur du graphe) : toutes les tâches n'ayant pas de successeurs.

Niveau k : toutes les tâches dont tous les successeurs sont à des niveau superieurs et dont au moins un successeur est au niveau $k+1$.

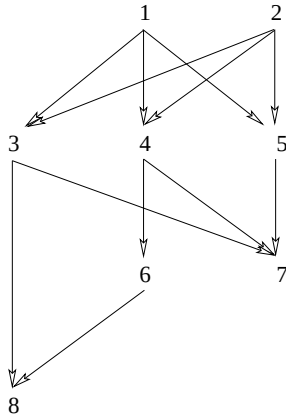
- Il faut itérer le processus de $k = H(G), ..., 1$.
- Cet ordonnancement conduit à un temps de début d'exécution le plus tard possible pour chacune des tâches.

Exemple



niveau	D_p	D_s
1	(1, 2)	(1, 2)
2	(3, 4, 5)	(4)
3	(6, 7)	(6, 3, 5)
4	(8)	(7, 8)

D'autres décomposition en niveaux sont possibles



niveau	D_1	D_2
1	(1, 2)	(1, 2)
2	(3, 4)	(4, 5)
3	(5, 6)	(3, 6)
4	(7, 8)	(7, 8)

Propriétés

- Si toutes les tâches ont le même temps d'exécution et si le nombre de processeurs est illimité, le temps d'un algorithme parallèle optimal est la hauteur du graphe :

$$t_{opt} = H(G)$$

- La ≪ largeur ≫ de la décomposition détermine le nombre minimal de processeurs nécessaires.

Temps d'exécution quelconque pour les tâches

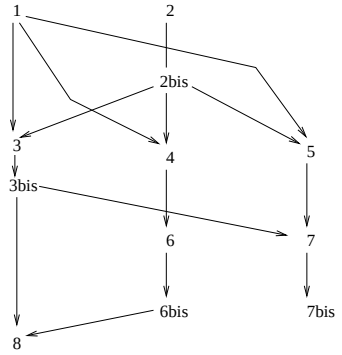
Si les tâches ont des temps d'exécution quelconques on se ramène au cas précédent en découpant ces tâches en sous-tâches dont les temps d'exécution sont multiples d'un même nombre.

Exemple

reprenons le graphe de l'exemple 2 avec :

$$t_1 = t_4 = t_5 = t_8 = 1$$
$$t_2 = t_3 = t_6 = t_7 = 2$$

niveau	D_1
1	(1, 2)
2	(2bis)
3	(3, 4, 5)
4	(3bis, 6)
5	(6bis, 7)
6	(8, 7bis)



D'où l'ordonnancement associé à G sur 3 processeurs :

temps	1	2	3	4	5	6
Proc 1	T_1		T_3			T_8
Proc 2	T_2		T_4	T_6		
Proc 3			T_5		T_7	

Bornes pour le temps d'exécution optimal avec p processeurs

Théorème (Brent, 1974)

Si un algorithme peut être effectué en parallèle en un temps t avec une infinité de processeurs et demande q tâches, ce même algorithme peut-être effectué avec p processeurs en un temps

$$t_p \leq t + \frac{(q-t)}{p}$$

Démonstration du théorème

- Hypothèse : toutes les tâches ont même temps d'exécution choisi comme unité de temps.
- Sur une infinité de processeurs :

à l'instant 1	on a exécuté	q_1	tâches
à l'instant 2	on a exécuté	q_2	tâches supplémentaires
à l'instant t	on a exécuté	q_t	tâches supplémentaires

- Le nombre total de tâches est : $q = \sum_{i=1}^{t} q_i$ exécutées en un temps t.
- Sur un nombre p de processeurs :

L'exécution de q_1	tâches nécessite	$t_1' = \lceil \frac{q_1}{p} \rceil$	unités de temps
L'exécution de q_2	tâches nécessite	$t_2' = \lceil \frac{q_2}{p} \rceil$	unités de temps
L'exécution de q_t	tâches nécessite	$t_t' = \lceil \frac{q_t}{p} \rceil$	unités de temps

d'où $t_p = \sum_{i=1}^{t} \lceil \frac{q_i}{p} \rceil$

Démonstration du théorème (suite)

Calculons une majoration de t_p :

$$\text{nous avons} \quad \frac{q_i}{p} \leq \lceil \frac{q_i}{p} \rceil \leq \frac{q_i}{p} + 1 - \frac{1}{p}$$

$$\text{exemple :} \quad \frac{3}{2} \leq \lceil \frac{3}{2} \rceil \leq \frac{3}{2} + 1 - \frac{1}{2}$$

$$\begin{array}{lllllll} \text{d'où} & t_p & = & \sum_{i=1}^{t} \lceil \frac{q_i}{p} \rceil & \leq & \sum_{i=1}^{t} (\frac{q_i}{p} + 1 - \frac{1}{p}) & = & \frac{1}{p} \sum_{i=1}^{t} (q_i + p - 1) \\ & & & & \leq & \frac{1}{p}(q + tp - t) & = & t + \frac{q-t}{p} \end{array}$$

D'où $t_p \leq t + \frac{q-t}{p}$, ou encore : $t_p \leq H(G) + \frac{t_{seq} - H(G)}{p}$