High Performance Computing

Théorèmes sur les dépendances d'instructions

Rémi Garde

Février 2018

Parallélisation d'une boucle

Théorème. *Une boucle de programme est parallélisable si et seulement si deux instructions associées à des itérations différentes ne présentent aucune dépendances de données ou de sorties.*

Preuve. Soit B une boucle de programme . Par commodité, à chaque itération $i \in [\![1, n]\!]$ de B nous associons une instruction unique I_i, combinaison de toutes les instructions de cette itération. Montrons que B est parallélisable si et seulement si I_i et I_j n'ont aucunes dépendance de données ou de sorties pour $i \neq j$.
$\Rightarrow$ Supposons B parallélisable, i.e. son output ne dépend pas de l'ordre dans lequel l'itération est effectuée. Supposons qu'il existe une dépendance de données: $\exists (i,j), In(I_i) \cap Out(I_j) \neq \emptyset$. Dans ce cas,$Out(I_i)$ va dépendre de $In(I_i)$, donc de $Out(I_j)$. La sortie de I_i dépend de l'exécution ou non de I_j : c'est impossible. Supposons qu'il existe une dépendance de sorties. $\exists (i,j), Out(I_i) \cap Out(I_j) \neq \emptyset$. Dans ce cas, la sortie de I_i sera (au moins en partie) écrasée par l'exécution de I_j, et réciproquement. On ne eut intervertir leur ordre sans changer la sortie globale. C'est impossible de même: il ne peut y avoir ni dépendance d'entrées ni de sorties.
$\Leftarrow$ Supposons qu'il n'y ait aucune dépendance dans B. Soit o un output : $o \in \bigcup_i Out(I_i)$. $\exists! k, o \in Out(I_k)$. Par indépendance de sorties, k est unique. L'état de o en sortie de B ne dépend donc que $In(I_k)$. Par indépendance de données, $In(I_k)$ ne dépend d'aucune sortie des autres instructions. Donc o ne dépend pas de l'exécution d'une autre instruction, i.e. o ne dépend pas de l'ordre d'exécution. B est donc parallélisable. □

Théorème. *Les deux boucles imbriquées d'indices i et j sont permutables si et seulement si il n'existe pas de dépendances de données ou de sorties entre des instances $(i + k_i, j - k_j)$ ou $(i - k_i, j + k_j)$ et l'instance (i, j), $(k_i, k_j) \in \mathbb{N}^*$*

Preuve. La permutation des boucles ne va changer que partiellement l'ordre d'exécution. Pour chaque instance (i, j), on note

- $A_1(i, j)$ les instances exécutées après l'instance (i, j) lorsque l'on utilise l'ordre d'exécution (i, j) : ce sont les instances (p, q) avec $p > i$ ou $p = i, q > j$
- $A_2(i, j)$ les instances exécutées après l'instance (i, j) lorsque l'on utilise l'ordre d'exécution (j, i) ce sont les instances (p, q) avec $q > j$ ou $p > i, q = j$
- $B_1(i, j)$ les instances exécutées avant l'instance (i, j) lorsque l'on utilise l'ordre d'exécution (i, j) : ce sont les instances (p, q) avec $p < i$ ou $p = i, q < j$
- $B_2(i, j)$ les instances exécutées avant l'instance (i, j) lorsque l'on utilise l'ordre d'exécution (j, i) ce sont les instances (p, q) avec $q < j$ ou $p < i, q = j$

On peut permuter si et seulement si, pour chaque instance (i, j), il n'y a aucune dépendance de données ou de sorties entre (i, j) et les instances qui changent d'ordre avec (i, j) lors de la permutation. Ces instances sont $A_1(i, j) \cap B_2(i, j)$ et $A_2(i, j) \cap B_1(i, j)$.
Considérons l'instance (p, q), $I_{p,q} \in A_1(i, j) \cap B_2(i, j)$.

$$\left\{ \begin{array}{ll} p > i & \text{ou } p = i, q > j \\ q < j & \text{ou } p < i, q = j \end{array} \right. \text{ soit } \left\{ \begin{array}{l} p > i \\ q < j \end{array} \right. \text{ ou encore } \exists (k_i, k_j) \in \mathbb{N}^*, (p, q) = (i + k_i, j - k_j)$$

De la même façon, $A_2(i, j) \cap B_1(i, j)$ donne $\exists (k_i, k_j) \in \mathbb{N}^*, (p, q) = (i - k_i, j + k_j)$.
Donc les boucles sont permutables si et seulement si, $\forall (i, j), \forall (k_i, k_j) \in \mathbb{N}^*$, il n'y a aucune indépendance de données entre les instances (i, j) et $(i - k_i, j + k_j)$ ou $(i + k_i, j - k_j)$.

□