High Performance Computing

Dépendances et parallélisation

Plessia Stanislas

Février 2018

Parallélisation d'une boucle

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

Preuve. Soit B une boucle de programme.
Pour chaque itération de B d'indice $i \in [\![1, n]\!]$, nous notons B_i l'ensemble des instructions de cette itération.

Nous cherchons à montrer que $\forall (i, j) \in [\![1, n]\!]^2$, B_i et B_j sont permutables (ie le programme est parallélisable) si et seulement si B_i et B_j n'ont aucune dépendances de données ou de sorties.

$\Rightarrow$ Supposons B parallélisable, i.e. sa sortie ne dépend pas de l'ordre des $B_i \forall i$.

Supposons qu'il existe une dépendance de données: $\exists (i, j), In(B_i) \cap Out(B_j) \neq \emptyset$.

Dans ce cas, commme $Out(B_i) \leftarrow In(B_i)$, $Out(B_i)$ dépend de $Out(B_j)$ La sortie de B_i dépend de l'exécution ou non de B_j : on arrive à une contradiction.

Supposons maintenant qu'il existe une dépendance de sorties. $\exists (i, j), Out(B_i) \cap Out(B_j) \neq \emptyset$. On reviens au cas précédent ou la sortie de B_i va dépendre de l'éxecution ou non de B_j : on a de nouveau une contradiction.

$\Leftarrow$ Supposons qu'il n'y ait aucune dépendance dans B.
Soit s la sortie d'une instruction : $\exists ! k, s \in Out(B_k)$.

Par indépendance de sorties, k est unique.
On sait donc que s ne dépend que de $In(B_k)$.

Par indépendance de données, $\forall i \neq k, In(B_k) \cap Out(B_i) = \emptyset$.
Donc s ne dépend que de B_k, i.e. s ne dépend pas de l'ordre d'exécution.

B est donc parallélisable. $\square$

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:

- $After_1(i,j)$ l'ensemble des instructions après celle d'indice (i,j) lorsque j est à l'interieur : ce sont les instances $(i, j+k_j)$ ou $(i+k_i, j+l_j)$ avec $k_i, k_j \in \mathbb{N}^*$ et $l_j \in \mathbb{Z}$
- $After_2(i,j)$ l'ensemble des instructions après celle d'indice (i,j) lorsque i est à l'interieur : ce sont les instances $(i+l_i, j+k_j)$ ou $(i+k_i, j)$ avec $k_i, k_j \in \mathbb{N}^*$ et $l_i \in \mathbb{Z}$
- $Before_1(i,j)$ l'ensemble des instructions après celle d'indice (i,j) lorsque j est à l'interieur : ce sont les instances $(i, j-k_j)$ ou $(i-k_i, j+l_j)$ avec $k_i, k_j \in \mathbb{N}^*$ et $l_j \in \mathbb{Z}$
- $Before_2(i,j)$ l'ensemble des instructions après celle d'indice (i,j) lorsque i est à l'interieur : ce sont les instances $(i+l_i, j-k_j)$ ou $(i-k_i, j)$ avec $k_i, k_j \in \mathbb{N}^*$ et $l_i \in \mathbb{Z}$

On peut permuter si et seulement si, pour chaque instruction (i,j), il n'y a aucune dépendance de données ou de sorties entre (i,j) et les autres instructions qu'on permute.

Ces instructions sont $After_1(i,j) \cap Before_2(i,j)$ et $After_2(i,j) \cap Before_1(i,j)$.

Soit l'instance $(p,q) \in After_1(i,j) \cap Before_2(i,j)$.

$$\begin{cases} (p,q) & = (i, j+k_j) \text{ ou } (i+k_i, j+l_j), (k_i,k_j) \in \mathbb{N}^* \text{ et } l_j \in \mathbb{Z} \\ (p,q) & = (i+l_i, j-k_j) \text{ ou } (i-k_i, j), (k_i,k_j) \in \mathbb{N}^* \text{ et } l_i \in \mathbb{Z} \end{cases} \text{ soit } \begin{cases} p = i + k_i \\ q = j - k_j \end{cases}$$

De la même façon, $After_2(i,j) \cap Before_1(i,j)$ donne $(p,q) = (i-k_i, j+k_j)$.

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 $(i+k_i, j-k_j)$.

□