Université d'Orléans — DEA d'Informatique

Module "compilation avancée"

Sujet d'examen proposé par C. Eisenbeis et F. Thomasset : corrigé

Mars 2001

Exercice 1

1. Chacune des instructions `S` et `T` écrit dans un tableau utilisé par l'autre.
   - tableau `X`, lecture de `X(I+1)` : soit (i_w) un vecteur d'itération de `S`, et (i_r) un vecteur d'itération de `T` ; outre les bornes de boucles, il faut exprimer l'accès aux mêmes éléments de tableaux :

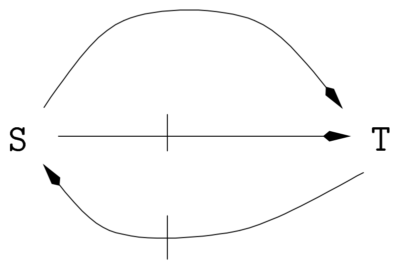
     $$i_w = i_r + 1$$

     Cette équation admet des solutions compatibles avec les bornes de boucles. Comme $i_w > i_r$, l'écriture par $\langle \mathtt{S}, \vec{i_w} \rangle$ a lieu *après* la lecture par $\langle \mathtt{T}, \vec{i_r} \rangle$. Nous avons donc une *anti-dépendance* de `T` vers `S` (de distance 1).
   - tableau `Y` : on pose de même le système de dépendance, en notant cette fois respectivement par (i_w, j_w) et (i_r, j_r) des vecteurs d'itération de `T` et `S` :

     $$i_w = i_r + 2$$

     La lecture par $\langle \mathtt{S}, \vec{i_r} \rangle$ a lieu *avant* l'écriture par $\langle \mathtt{T}, \vec{i_w} \rangle$: il s'agit donc d'une anti-dépendance de `S` vers `T`.
   - tableau `X`, lecture de `X(I)` : dépendance de type 'flow" de `S` vers `T`.

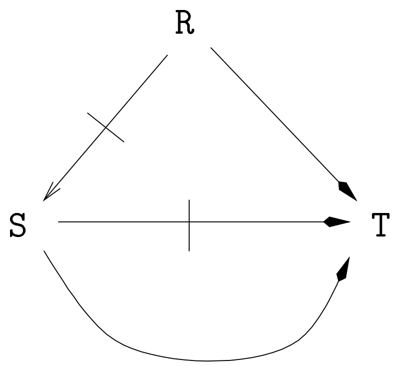
   Le graphe de dépendances est donc cyclique :



2. Le graphe de dépendances contient une seule composante fortement connexe à deux nœuds ; l'algorithme de distribution reconstruit une boucle par composante, c'est-à-dire la même boucle. Une boucle à 2 nœuds n'est pas parallèle.
3. L'obstacle au parallélisme vient de la présence d'un cycle dans le graphe ; en déplaçant l'une des anti-dépendances, il est possible de "casser" ce cycle ; ceci est possible au prix de l'introduction d'un tableau auxilliaire `W`, qui recevra le résultat des calculs du second membre de `T` :

```
    DO I = 1, 100
S:     X(I) = Y(I+2) + 1
R:     W(I) = X(I+1)
T:     Y(I) = W(I) + 1 + X(I)
    END DO
```

Cette transformation permet d'obtenir un graphe acyclique :



L'algorithme d'Allen-Kennedy produit alors 3 boucles parallèles.

Exercice 2

1. On examine toutes les paires (écriture, lecture), et on pose le système de dépendances[1]. L'écriture est toujours `X(I+1,J)`, puisque l'exemple ne comporte qu'une seule instruction. On note à chaque fois (i_w, j_w) et (i_r, j_r) des vecteurs d'itération correspondant respectivement à l'écriture et à la lecture.
   - lecture de `X(I,J+2)` :
   $$\begin{aligned} i_w + 1 &= i_r \\ j_w &= j_r + 2 \end{aligned}$$
   Lecture après écriture : dépendance de type flot.
   Vecteur de dépendance : $\begin{bmatrix} i_r \\ j_r \end{bmatrix} - \begin{bmatrix} i_w \\ j_w \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$
   - lecture de `X(I,J)` :
   $$\begin{aligned} i_w + 1 &= i_r \\ j_w &= j_r \end{aligned}$$
   Lecture après écriture : dépendance de type flot.
   Vecteur de dépendance : $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
   - lecture de `X(I+1,J)` : la lecture de l'élément de tableau écrit par la même instance de l'instruction d'impose aucune contrainte à l'ordre d'exécution ; cette lecture n'engendre aucune dépendance.
   - lecture de `X(I+2,J-1)` :
   $$\begin{aligned} i_w + 1 &= i_r + 2 \\ j_w &= j_r - 1 \end{aligned}$$
   Écriture après lecture : anti-dépendance.
   Vecteur de dépendance : $\begin{bmatrix} i_w \\ j_w \end{bmatrix} - \begin{bmatrix} i_r \\ j_r \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
   - lecture de `X(I+1,J+1)` :
   $$\begin{aligned} i_w + 1 &= i_r + 1 \\ j_w &= j_r + 1 \end{aligned}$$

1. Je n'écris pas les contraintes liées aux bornes de boucles, qui cependant font partie du système.

Écriture après lecture : anti-dépendance.

Vecteur de dépendance : $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Les vecteurs de dépendances sont donc constants, les dépendances sont donc uniformes (indépendantes des itérations).

2. Rangés dans l'ordre lexicographique, les vecteurs de dépendances sont :
$\vec{d^1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\vec{d^2} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $\vec{d^3} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $\vec{d^4} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
On cherche d'abord le vecteur ligne $\vec{\pi} = \begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix}$ de la première ligne de la transformation unimodulaire ; ce vecteur doit satisfaire $\vec{\pi} \cdot \vec{d} > 0$ pour chacun des vecteurs de dépendance.
$\vec{d^1} : \pi_2 = 1$
$\vec{d^2} : \pi_1 - 2\pi_2 > 0$
$\vec{d^3} : \pi_1 - \pi_2 > 0$
$\vec{d^4} : \pi_1 > 0$
On obtient le vecteur $\vec{\pi} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, qui constitue la première ligne de la matrice de transformation T.

3. On complète pour obtenir une matrice unimodulaire, par exemple par : $T = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}$
Soient `K1, K2` les indices du nouvel espace d'itération : $\begin{bmatrix} \mathtt{K1} \\ \mathtt{K2} \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathtt{I} \\ \mathtt{J} \end{bmatrix}$
D'où en inversant T : $\begin{cases} \mathtt{I} = \mathtt{K2} \\ \mathtt{J} = \mathtt{K1} - 3\mathtt{K2} \end{cases}$

```
DO K1 = ? , ?
   DO K2 = ? , ?
      X(K2+1, K1 - 3 * K2) = X(K2, K1 - 3 * K2 + 2)
               + X(K2, K1 - 3 * K2) + X(K2+1, K1 - 3 * K2)
               + X(K2+2, K1 - 3 * K2 - 1) + X(K2+1, K1 - 3 * K2 + 1)
   END DO
END DO
```

Comme toutes les dépendances sont portées par la boucle externe (`K1`), la boucle interne est parallèle.

4. En reportant dans les bornes de boucles sur `I, J` leurs expressions en fonction de `K1, K2`, on obtient les contraintes suivantes :

$$\begin{cases} 1 \leq k_2 \leq 50 \\ k_2 \leq k_1 - 3k_2 \leq 200 - k_2 \end{cases}$$

Les deux premières inégalités ne peuvent certainement pas suffire à borner `K2`, puisqu'il faut satisfaire les 2 autres qui dépendent du compteur de la boucle externe. On doit donc commencer par trouver des bornes pour le compteur de la boucle externe `K1`, indépendantes de `K2`, ce que l'on effectue en éliminant `K2` par Fourier-Motzkin.

$$\begin{matrix} 1 \\ \frac{k_1}{2} - 100 \end{matrix} \left\{ \leq k_2 \leq \right\} \begin{matrix} 50 \\ \frac{k_1}{4} \end{matrix}$$

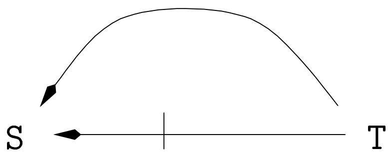
L'élimination de k_2 donne les bornes de k_1 : $4 \leq k_1 \leq 400$
On en déduit les bornes sur k_2 en fonction de k_1 : $\max(1, -100 + \frac{k_1}{2}) \leq k_2 \leq \min(50, \frac{k_1}{4})$

La correspondance entre les points réels k_1, k_2 satisfaisant ces inégalités, et les points réels i, j de l'espace de départ est biunivoque, par les propriétés de Fourier-Motzkin ; il faut se restreindre aux points entiers dans l'espace k_1, k_2 en prenant les parties entières :
$\max(1, -100 + \lceil \frac{k_1}{2} \rceil) \leq k_2 \leq \min(50, \lfloor \frac{k_1}{4} \rfloor)$

```
DO K1 = 4, 400
   DO K2 = MAX(1, -100+CEILING(K1/2)), MIN(50, FLOOR(K1/4))
      X(K2+1, K1 - 3 * K2) = X(K2, K1 - 3 * K2 + 2)
               + X(K2, K1 - 3 * K2) + X(K2+1, K1 - 3 * K2)
               + X(K2+2, K1 - 3 * K2 - 1) + X(K2+1, K1 - 3 * K2 + 1)
   END DO
END DO
```

Exercice 3

1. Nous avons deux dépendances de `T` vers `S` : une anti-dépendance de distance 1, et une dépendance “flot” de distance 2.



2. On peut donner l'ordonnancement suivant pour une itération, en nottant respectivement `MEM1`, `MEM2` et `ADD1`, `ADD2` les unités d'accès à la mémoire et d'addition :

	MEM1	MEM2	ADD1	ADD2
0	ld y(i),r1	ld x(i+1),r2		
1				
2				
3			r3 = r1 + 1	r4 = r2 + 1
4				
5	st r3,x(i)	st r4,y(i+2)		
6				
7				

La durée d'une itération est donc 8 tops d'horloge.

3. Boucle déroulée :

```
    DO I = 1, 97, 3
S:     X(I) = Y(I) + 1
T:     Y(I+2) = X(I+1) + 1
S:     X(I+1) = Y(I+1) + 1
T:     Y(I+3) = X(I+2) + 1
S:     X(I+2) = Y(I+2) + 1
T:     Y(I+4) = X(I+3) + 1
    END DO
    X(100) = Y(100) + 1
    Y(102) = X(101) + 1
```

Comme les opérations sont pipelinées, on peut lancer successivement deux accès à la mémoire sur le même canal.

On note que le chargement de `Y(i+2)` (pour l'itération `i+2` de `S`) doit attendre la fin de l'écriture par $\langle \mathtt{T}, \vec{i} \rangle$, à moins de transmettre la valeur dans le registre de sortie de `T`, comme ci-dessous :

	MEM1	MEM2	ADD1	ADD2
0	`ld y(i),r1`	`ld x(i+1),r2`		
1	`ld y(i+1),r1'`	`ld x(i+2),r2'`		
2		`ld x(i+2),r2''`		
3			`r3 = r1 + 1`	`r4 = r2 + 1`
4			`r3' = r1' + 1`	`r4' = r2' + 1`
5	`st r3,x(i)`	`st r4,y(i+2)`	`r4'' = r4 + 1`	`r4'' = r2'' + 1`
6	`st r3',x(i+1)`	`st r4',y(i+3)`		
7	`st r3'',x(i+2)`	`st r4'',y(i+4)`		
8				
9				

On obtient ainsi 10 tops pour 3 itérations, soit $\frac{10}{3} \sim 3.3$ tops par itération.

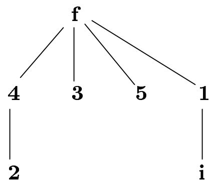
Si l'on ne réutilise pas `r4` pour `Y(i+2)`, le chargement de `Y(i+2)` doit être lancé au cycle 8, l'addition au cycle 11, le store au cycle 13, qui se termine au cycle 16, d'où $\frac{17}{3} \sim 5.7$ cycles par itération !

4. Pour obtenir une meilleure performance, on peut remarquer que `Y(i+2)` pourrait être gardé en registre pour les itérations ultérieures ; il est également possible de dérouler la boucle d'un facteur plus grand.

Exercice 4 1. La post-dominance est une notion "duale" de la dominance.
Soit $G_r = (V_r, E_r)$ le graphe "inverse" de $G = (V, E)$:
   – les nœuds de G_r sont les nœuds de G : $V_r = V$;
   – les arcs de G_r sont obtenus en retournant les arcs de G : $(a,b) \in E_r \iff (b,a) \in E$

   Alors **a** post-domine **b** ssi **b** domine **a** dans G_r.

2. La post-dominance peut se représenter par un arbre de racine **f**, pour la même raison que la dominance se représente par un arbre de racine **i** (par exemple **1** post-domine **i**) :



3. Soit $DF^r(a)$ la frontière de dominance de a dans G_r.
   $b \in DF^r(a) \iff b\ \delta^c\ a$: résulte des définitions.

Commentaires : Informellement, $b\,\delta^c\,a$ si le résultat de b détermine l'exécution pour a : b contient par exemple un test dont le résultat conditionne l'exécution de a. Ainsi, si b est le calcul de la condition d'un `if-then-else`, tout nœud correspondant à l'une des branches `then` ou `else`, est contrôle-dépendant [2] de b.

Les dépendances de "contrôle" sont l'ingrédient principal du *"Program Dependence Graph"*, une structure généralisant le graphe de dépendances[FOW77], et utilisable pour la parallélisation de programmes avec graphes de contrôle généraux ; le PDG contient à la fois les dépendances de données et de contrôle. La relation entre dépendance de contrôle et frontière de dominance est expliquée par Cytron et al.[CFRWZ]

Références

[CF89] Robert Cartwright, Matthias Felleisen. The Semantics of Program Dependence. In *Proceedings of the SIGPLAN '89 Conference on Programming Language Design and Implemen-*

2. C'est une traduction littérale de *"control-dependent"* ; en anglais, *"control"* désigne traditionnellement le point de programme qui "contrôle" à un instant donné l'exécution ; je suis preneur de toute suggestion qui m'éviterait d'écrire du franglais !

tation (PLDI '89), pages 13–27. ACM SIGPLAN Notices, 24(7), June 1989.
`http://www.cs.rice.edu/CS/PLT/Publications/pldi89-cf.ps.gz`

[CFRWZ] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Transactions on Programming Languages and Systems*, 13(4), pp. 451-490, October 1991.

[FOW77] J. Ferrante, K.J. Ottenstein, J.D. Warren. The Program Dependence Graph and its use in Optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, July 1977.
Attention : leur définition de la post-dominance est un peu différente de celle donnée ici ; la post-dominance n'est pas réflexive dans cet article.

[PB95] Keshav Pingali, Gianfranco Bilardi. APT: A Data Structure for Optimal Control Dependence Computation. In *Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI),* pp. 32-46, 18-21 June 1995.
`http://www.cs.cornell.edu/Info/Projects/Bernoulli/`