# Calcul de dépendances dans les boucles

F. Thomasset
INRIA - Rocquencourt
Projet A3

http://www-rocq.inria.fr/~thomasse/dea/

http://www-rocq.inria.fr/~thomasse/dea/articles.html

11 décembre 2001

Analyse de programmes itératifs
(programmation *impérative*)

Objectif : parallélisation, debug

Un calcul itératif s'exprime par une boucle

```
DO I = 1, 100
   X(I) = X(I-1) + 4
END DO
```

ou par des procédures récursives :

```
void Proc (int i) {
  X[i] = ...
  if (...) Proc(i+1)
  if (...) Proc(i-1)
  ... = X[i]
}
```

Une dépendance entre opérations contraint l'ordre d'exécution

Référence à l'ordre d'exécution standard $\prec$ ( *"séquentiel"*)

Les analyses des dépendances posent deux types de questions :

- Deux opérations écrivent-elles dans la même cellule mémoire ?
  *"Memory-based"*

- Une opération de lecture étant donnée, quelle est l'opération d'écriture qui a produit la donnée correspondante ?
  *"Analyse du flot des données par instances"*
  Dépendances "directes"
  *"Value-based dependences"*

Différence entre les deux types de questions pour des variables scalaires

```
S1:    A = ...
R1:    ... = A + 1
S2:    A = ...
R2     ... = A + 1
```

Dépendances *"Memory-based"* :

$$\mathtt{S1} \xrightarrow{\text{pc}} \mathtt{R1} \qquad \mathtt{S2} \xrightarrow{\text{pc}} \mathtt{R2}$$
$$\mathtt{R1} \xrightarrow{\text{cp}} \mathtt{S2} \qquad \mathtt{S1} \xrightarrow{\text{pc}} \mathtt{R2}$$
$$\mathtt{S1} \xrightarrow{\text{pp}} \mathtt{S2}$$

Dépendances "directes" :

$$\mathtt{S1} \xrightarrow{\text{pc}} \mathtt{R1} \qquad \mathtt{S2} \xrightarrow{\text{pc}} \mathtt{R2}$$
$$\mathtt{R1} \xrightarrow{\text{cp}} \mathtt{S2}$$

"pc" : producteur $\longrightarrow$ consommateur ( *"flot"*)
"cp" : consommateur $\longrightarrow$ producteur ( *"anti"*)
"pp" : producteur $\longrightarrow$ producteur ( *"output"*)

# “Memory-based”

Étant donné 2 opérations, utilisent-elles la même case mémoire en écriture ?

Si la réponse est non, on peut envisager de les exécuter en parallèle

```
   DO I = 1, 100
S:    X(I) = Y(I) + 4
   END DO
```

Quelques cas plus compliqués :

```
   DO I = 1, 100
A:    Z(I) = X(I) * 2
B:    X(I+1) = Y(I)
C:    W(I) = Z(I+1) +1.
   END DO
```

```
   DO I = 2, 100
      DO J = 2, 100
S:       X(I,J) = (X(I+1,J)
                 + X(I-1,J)
                 + X(I,J+1)
                 + X(I,J-1))*.25
      END DO
   END DO
```

# Analyse exacte du flot des données

La question posée :
Une opération de lecture étant donnée,
quelle est l'opération d'écriture qui a produit la donnée correspondante ?
S'il y a plusieurs écritures candidates ?

```
   DO I = 1, 100
S:     X(I) = X(I-1) + 4
   END DO
```

Pour `X(2)`, (lue par $\langle$S,3$\rangle$)
la donnée a été produite par $\langle$S,2$\rangle$ :
l'instruction "source"

Plus compliqué avec des tests :

```
   DO I = 1, 100
     IF (...) THEN
S1:     X(I) = X(I-1) + 4
     ELSE
S2:     X(I) = X(I-1) + 8
     END IF
   END DO
R: ... = X(100)
```

Qui a produit la donnée utilisée par $\langle$`S1`,i$\rangle$ ?
$\langle$`S1`,i-1$\rangle$ ou $\langle$`S2`,i-1$\rangle$

Qui a produit la donnée utilisée par `R` ?
$\langle$`S1`,100$\rangle$ ou $\langle$`S2`,100$\rangle$

Pourquoi chercher la *source* d'une donnée?

Une application : expansion des scalaires (ou tableaux)

```
DO I = 1, 100
   Z = calcul(I)
   X(I) = Z + 2
   Y(I) = Z * 2
END DO
```

Réutilisation de la case mémoire `Z` :
inhibe la parallélisation

Observation : quand on lit `Z` à l'itération $i$, la valeur a été produite par l'exécution de la même itération.

Allouer un tableau pour ranger les valeurs
venant d'itérations différentes
dans des cases différentes.

```
DO I = 1, 100
   Z1(I) = calcul(I)
   X(I) = Z1(I) + 2
   Y(I) = Z1(I) * 2
END DO
```

Attention : si `Z` n'est pas morte en fin de boucle, restaurer sa valeur

Sur multiprocesseur : “privatiser” `Z`

Extensions : expansion de tableau

# Les outils

– Analyse ("classique") de flots de données
(Analyse statique, interprétation abstraite)
découverte de propriétés de variables scalaires :
propagation des constantes,
vivacité des variables, . . .
Préalable indispensable à l'analyse de dépendances.

– Programmation linéaire
en nombres entiers :
objet de ce cours

– Théorie des automates
Analyse de programmes récursifs :
`www-rocq.inria.fr/`
`~acohen/publications/thesis.ps.gz`

# Un problème plus simple : “Memory-based dependence”

La parallélisation automatique pose habituellement une question plus simple que la recherche des sources des données : *“Memory-based dependence”* :

Étant donné 2 opérations, utilisent-elles la même case mémoire en écriture ?

Si la réponse est non, on peut envisager de les exécuter en parallèle

```
   DO I = 1, 100
S:     X(I) = Y(I) + 4
   END DO
```

Tests de dépendances "memory based"

– PGCD

– Banerjee

Tests de dépendances "directes"

– Fourier-Motzkin
  Test "Omega"

– Programmation linéaire
  PIP

Que faire des non linéarités?

# Programmes à contrôle "statique"

Le modèle de programme :

- Boucles "DO" : nombre d'itérations connu au début de l'exécution
- Compteur de boucle non modifié dans le corps de boucle
- Bornes de boucles : expressions quasi-affines des compteurs des boucles englobantes et de constantes de boucle
- Accès aux tableaux : expressions quasi-affines
- Expressions booléennes dans if..then..else : expressions quasi-affines
- *"Paramètres structurels"* : constantes de boucle

Un exemple de programme à contrôle “statique” :

```
DO I = 1, M
   DO J = I+1, 2*I+N
      IF ( (J >= I+5) AND (J+I <= M+N) )
      THEN
         X(I,J+I) = Y(I+1,J-1) * 5
      ELSE
         X(I,J+I) = Y(I-1,J+1) * 5
      ENDIF
   END DO
   X(I,I) = Z(M-I+1)
END DO
```

Expressions quasi-affines :
contiennent des divisions entières
ou des modulos entiers

```
M   +   I ÷ 2   -   3 * (K-J+M) ÷ 2
```

```
ExpQuasiAffine =
      ConstanteEntière
    | Var
    | ExpQuasiAffine + ExpQuasiAffine
    | ExpQuasiAffine - ExpQuasiAffine
    | ConstanteEntière * ExpQuasiAffine
    | ExpQuasiAffine ÷ ConstanteEntière
                          (division entière)
    | ExpQuasiAffine % ConstanteEntière
```

# Niveaux de dépendances

Niveau d'une boucle :
1 + nombre de boucles qui la contiennent

```
   DO I = 1, N                          ← niveau 1
     DO J = 1, N                        ← niveau 2
S:     A(I,J) = A(I+1,J) + A(I,J-1)
     END DO
   END DO
```

Niveau d'une dépendance :
niveau de la boucle qui "porte" la dépendance

$\langle \texttt{S},i,j\text{-}1\rangle \xrightarrow{\text{pc}} \langle \texttt{S},i,j\rangle$
2 itérations de la même boucle `DO J` :
dépendance de niveau 2

$\langle \texttt{S},i\text{-}1,j\rangle \xrightarrow{\text{cp}} \langle \texttt{S},i,j\rangle$
2 itérations de la boucle `DO I` :
dépendance de niveau 1

# Définition de la dépendance

S et R deux opérations accédant au même tableau

```
S:    ... A(f(x)) ...
R:    ... A(g(y)) ...
```

`x, y` vecteurs d'itération
`f(x)` et `g(y)` fonctions quasi-affines de `x, y`
(et des paramètres structurels)

Condition d'existence d'une dépendance $pc$
de $\langle \mathtt{S},x\rangle$ vers $\langle \mathtt{R},y\rangle$

1. $\langle \mathtt{S},x\rangle$ et $\langle \mathtt{R},y\rangle$ accèdent au même élément de tableau (dont au moins une écriture) : `f(x) = g(y)`

2. $\langle \mathtt{S},x\rangle \prec \langle \mathtt{R},y\rangle$
   ($\langle \mathtt{S},x\rangle$ est éxécutée *avant* $\langle \mathtt{R},y\rangle$
   dans l'ordre standard)

3. Pour calculer une dépendance directe :
   aucune écriture dans le même élément de tableau ne s'intercale entre $\langle \mathtt{S},x\rangle$ et $\langle \mathtt{R},y\rangle$

1. $\langle$`S`,x$\rangle$ et $\langle$`R`,y$\rangle$ accèdent au même élément de tableau
   `f(x) = g(y)`

2. $\langle$`S`,x$\rangle \prec \langle$`R`,y$\rangle$
   `x` $\ll$ `y` $\vee$ ( `x = y` $\wedge$ S $\lhd$ R )

$\ll$ ordre lexicographique
$\lhd$ ordre textuel

`x` $= [\ x[1], ..., x[p]\ ]$, `y` $= [\ y[1], ..., y[p]\ ]$

$$
\begin{aligned}
\texttt{x} \ll \texttt{y} \Longleftrightarrow\ & x[1] < y[1] \\
& \vee\ (\ x[1] = y[1] \wedge x[2] < y[2]\ ) \\
& \vee\ (\ x[1] = y[1] \wedge x[2] = y[2] \wedge x[3] < y[3]\ ) \\
& \vee\ ... \\
& \vee\ (\ x[1] = y[1] \wedge ... \wedge x[p-1] = y[p-1] \\
& \qquad\qquad\qquad \wedge\ x[p] < y[p]\ )
\end{aligned}
$$

$p = N_{S,R} =$ nombre de boucles communes à `S` et `R`

Condition pour une dépendance directe
de $\langle \mathtt{S},x\rangle$ vers $\langle \mathtt{R},y\rangle$ :

```
T: A(h(z)) = ...
```

$$\left.\begin{array}{l}\mathtt{h(z)\ =\ g(y)}\\ \langle \mathtt{T},z\rangle \prec \langle \mathtt{R},y\rangle\end{array}\right\} \Longrightarrow \langle \mathtt{T},z\rangle \prec \langle \mathtt{S},x\rangle$$

```
   DO I = ...
      DO J = ...
S:       ......
      END DO
R:       ......
      DO J = ...
         DO K = ...
T:          ......
U:          ......
         END DO
      END DO
   END DO
```

$\mathtt{S} \lhd \mathtt{R} \lhd \mathtt{T} \lhd \mathtt{U}$

$$\langle \mathtt{S},1,j\rangle \prec \langle \mathtt{R},2\rangle \prec \langle \mathtt{S},2,j'\rangle$$
$$(\vec{1}) \ll (\vec{2})$$

$$\langle \mathtt{T},3,2,1\rangle \prec \langle \mathtt{U},3,2,1\rangle \prec \langle \mathtt{T},3,3,1\rangle \prec \langle \mathtt{S},4\rangle$$
$$(3,\vec{2}) \ll (3,\vec{3})$$
$$(\vec{3}) \ll (\vec{4})$$

# Profondeur d'une dépendance

$$\langle \mathtt{S,x} \rangle \prec \langle \mathtt{R,y} \rangle \iff \bigvee_{p=0}^{p=N_{S,R}} \langle \mathtt{S,x} \rangle \prec_p \langle \mathtt{R,y} \rangle$$

$\prec_p$ : Dépendance à la profondeur $p$ :

– $0 \leq p < N_{S,R}$ :
$$\langle \mathtt{S,x} \rangle \prec_p \langle \mathtt{R,y} \rangle \iff x[1..p] = y[1..p] \wedge x[p+1] < y[p+1]$$

– $p = N_{S,R}$ :
$$\langle \mathtt{S,x} \rangle \prec_p \langle \mathtt{R,y} \rangle \iff x[1..p] = y[1..p] \wedge \mathtt{S} \lhd \mathtt{R}$$

Système de dépendance : exemple

```
  DO I = 1, N
     DO J = 1, N
S:      A(I,J) = A(J,I) + A(I,J-1)
     END DO
  END DO
```

dépendance de $\langle$S,i,j$\rangle$ vers $\langle$S,i',j'$\rangle$ ?

Lecture de a(j,i) :

– Égalité des éléments accédés : $i = j',\ j = i'$

– $\langle$S,i,j$\rangle$ $\prec$ $\langle$S,i',j'$\rangle$ :
$i < i' \quad \vee ( \ i = i' \wedge j < j' \ )$

Solution : $\langle$S,i,j$\rangle$ $\xrightarrow{\text{pc}}$ $\langle$S,i',j'$\rangle$ ?
si $i < j$ alors $(i', j') = (j, i)$
sinon pas de dépendance

dépendance de niveau 1
$(i' - i, j' - j) = (j - i, i - j)$
$i' - i = j - i \in [1, N - 1],\ j' - j \in [-N + 1, N - 1]$

anti-dépendance $\xrightarrow{\text{cp}}$?

i.e. <u>lecture de `a(j,i)` avant écriture de `a(i',j')`?</u>

– Égalité des éléments accédés :
  $j = i',\ i = j'$

– $\langle$S,i,j$\rangle \prec \langle$S,i',j'$\rangle$ :
  $i < i' \quad \vee (\ i = i' \wedge j < j'\ )$

Solution : $\langle$S,i,j$\rangle \xrightarrow{\text{cp}} \langle$S,i',j'$\rangle$?
si $i < j$ alors $(i', j') = (j, i)$
sinon pas de dépendance

anti-dépendance de niveau 1

dépendance pour la <u>lecture de `a(i,j-1)`</u> ?

– Égalité des éléments accédés :
  $i = i',\ j = j' - 1$

– $\langle$S,i,j$\rangle \prec \langle$S,i',j'$\rangle$ :
  $i < i' \quad \vee (\ i = i' \wedge j < j'\ )$

Solution :
$\langle$S,i,j$\rangle \xrightarrow{\text{cp}} \langle$S,i',j'$\rangle$
avec $(i', j') = (i, j + 1)$

dépendance de niveau 2

Résumé :
Le système de dépendances :

- Accès aux éléments de tableau

- séquencement

- appartenance au domaine d'exécution

- contexte : contraintes sur les paramètres
  exemple :

```
IF (M > 0) THEN
   DO I = 1, N
      X(I+M) = ...
   END DO
```

- solutions entières !

# Quelle représentation de dépendances?

Représentation synthétique nécessaire.
Plusieurs représentations selon le type de transformation visé : approximations plus ou moins précises :

- Relations entre itérations en dépendance

- Polyèdres de dépendances

- Distances de dépendance :
  différences entre itérations en dépendance :
  $\{y - x\}$ tels que $\langle \mathtt{S,x} \rangle \xrightarrow{\text{pc}} \langle \mathtt{R,y} \rangle$
  (si $y - x$ n'est pas constant,
  approximer par un intervalle, par exemple $[1, \infty]$)

- Vecteurs de dépendances : conserve seulement les signes des différences entre itérations en dépendance (échange de boucle)

- Niveau des dépendances : distribution de boucle (Allen-Kennedy)