Parallélisation automatique
TD – Enoncé

Plan

1. Prise en main de Tiny/Petit
2. Analyse de dépendances
3. Elimination de dépendances

1 Prise en main de Tiny/Petit

Logiciel TINY
– TINY (Tiny Loop Restructuring Tool)
– Michael Wolfe, Oregon State University
– http ://www.cse.ogi.edu/Sparse/tiny.html
– Tiny computes data dependence relations and interactively performs many restructuring transformations, such as loop interchanging, distribution, skewing, ...

Logiciel PETIT
– PETIT (Pugh's Extension of Tiny Is not Tiny)
– The Omega project group at Univ. of Maryland
– http ://www.cs.umd.edu/projects/omega/index.html

Documentation
– TINY User Guide
– User Interface for PETIT
– Se familiariser avec les menus
– Utilisation du premier exemple ci-dessous, fichier base.t

2 Analyse de dépendances

Dépendances de base

File base.t

$$\begin{array}{ll} S_1 & \mathsf{A = B + C} \\ S_2 & \mathsf{D = E + F} \\ S_3 & \mathsf{C = A + C} \\ S_4 & \mathsf{E = C + D} \\ S_5 & \mathsf{E = 1} \end{array}$$

1. Effectuer l'analyse de dépendance sur le code
2. Vérifier avec Tiny.

Dépendances de boucles

File boucle.t

```
for i = 1 to N do
for j = 1 to N do
```

$S_1 : a(i+1, j-1) = b(i, j+4) + c(i-2, j-3) + 1$
$S_2 : b(i-1, j) = a(i, j) - 1$
$S_3 : c(i, j) = a(i, j-2) + b(i, j)$

1. Donner le GDR, en identifiant les dépendances de flot et anti
2. Vérifier avec Tiny.

Parallélisation de boucles

File para.t

```
for i = 1 to N do
for j = 1 to N do
```

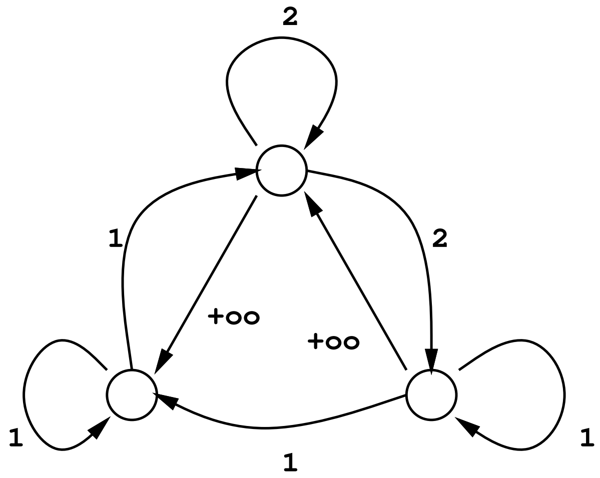
$S_1 : a(i+1, j+1) = a(i+1, j) + b(i, j+2)$

$S_2 : b(i+1, j) = a(i+1, j-1) + b(i, j-1)$

$S_3 : a(i, j+2) = b(i+1, j+1) - 1$

1. Donner le GDR avec le type des dépendances (flot, anti, sortie) leur expression (vecteur de dépendance) et leur niveau
2. Vérifier avec Tiny
3. Appliquer l'algorithme de Kennedy et Allen pour restructurer le nid de boucles
4. Vérifier avec Tiny la nature des boucles obtenues.

Algorithme d'Allen et Kennedy



1. Donner un nid de boucles parfait où toutes les dépendances sont uniformes, et dont le graphe de dépendances avec niveaux de dépendances est exactement le graphe représenté ci-dessus
2. Vérifier avec Tiny
3. Exécuter l'algorithme d'Allen et Kennedy sur ce nid de boucles
4. Vérifier avec Tiny.

Factorisation de Cholesky

File cholesky.t

```
for k = 1 to N do
   S1 : a(k,k) := sqrt(a(k,k))
   for i = k+1 to N do
      S2 : a(i,k) := a(i,k)/a(k,k)
      for j = k+1 to i do
         S3 : a(i,j) := a(i,j) - a(i,k) × a(j,k)
```

1. Construire à la main, en étudiant l'algorithme, le graphe de tâches étendu (dont les sommets sont les instances des instructions)
2. Montrer qu'il n'y a que des dépendances de flow (i.e. toutes les autres correspondent à des arcs de transitivité)
3. Calculer combien de paires de références devrait considérer un analyseur de dépendances
4. Faire l'analyse de dépendance sur une output dépendance et montrer comment elle est recouverte
5. Vérifier avec Tiny.

Méthode aux différences finies

File diff.t

```
for i = 1 to N do
for j = 1 to N do
  a(i,j) = 1/4[a(i-1,j) + a(i+1,j) + a(i,j-1) + a(i,j+1)]
```

1. Donner le GDR avec le type des dépendances (flot, anti, sortie) et vérifier avec Tiny
2. Donner le GDE pour $N = 5$
3. Imaginer une torsion de l'espace d'itération qui permette d'extraire une boucle parallèle
4. Vérifier avec Tiny.

3 Elimination de dépendances

Exemple simple

File ex-avant.t

$$\begin{array}{l} \text{For } i = 1 \text{ to } N \text{ do} \\ \quad S_1 : \mathrm{a}(i) = \mathrm{b}(i) + \mathrm{c}(i) \\ \quad S_2 : \mathrm{a}(i+1) = \mathrm{a}(i) + 2 \times \mathrm{d}(i) \end{array}$$

(a) Code original

File ex-apres.t

$$\begin{array}{l} \text{For } i = 1 \text{ to } N \text{ do} \\ \quad S_1' : \mathrm{temp}(i) = \mathrm{b}(i) + \mathrm{c}(i) \\ \quad S_1 : \mathrm{a}(i) = \mathrm{temp}(i) \\ \quad S_2 : \mathrm{a}(i+1) = \mathrm{temp}(i) + 2 \times \mathrm{d}(i) \end{array}$$

(b) Code après découpe de sommet

1. Donner le GDR du code original (avec Tiny)
2. Donner le GDR après découpe de sommet (avec Tiny)
3. Conclusion ?

Méthode
Dans le cas général, la découpe de sommet s'applique ainsi :

$$
\begin{array}{l}
\textsf{For } i = 1 \textsf{ to } N \textsf{ do} \\
\quad \ldots \\
\quad S : \mathrm{lhs}(f(i)) = \mathrm{rhs}(\ldots) \\
\quad \ldots \\
\quad \ldots \qquad\qquad\quad = \mathrm{lhs}(g(i))
\end{array}
$$

(a) Code original

$$
\begin{array}{l}
\textsf{For } i = 1 \textsf{ to } N \textsf{ do} \\
\quad \ldots \\
\quad S' : \mathrm{temp}(f(i)) = \mathrm{rhs}(\ldots) \\
\quad S : \mathrm{lhs}(f(i)) = \mathrm{temp}(f(i)) \\
\quad \ldots \\
\quad \ldots \qquad\qquad\quad = \mathrm{temp}(g(i))
\end{array}
$$

(b) Code après découpe de sommet

Montrer ce que deviennent respectivement les six types de dépendance possibles (dépendances en entrée sur S de type flot, anti, output et dépendances en sortie de S de type flot, anti, output) sur le nouveau graphe avec S et S'.

Application

File cycle.t

$$
\begin{array}{l}
\textsf{for } i := 4 \textsf{ to } N \textsf{ do} \\
\quad S_1 : a(i+5) := c(i-3) + b(2i+2) \\
\quad S_2 : b(2i) := a(i-1) + 1 \\
\quad S_3 : a(i) := c(i+5) - 1 \\
\quad S_4 : c(i) := b(2i-4)
\end{array}
$$

1. Montrer que ce programme contient un cycle de dépendance (avec Tiny)
2. Existe-t-il un cycle de dépendance qui ne comprend que des dépendances de flot ?
3. Quel est le nouveau graphe de dépendance quand on découpe les sommets S_2 et S_3 ?

Un peu de théorie

1. Soit G le GDR d'un nid de boucles L, et G' le graphe obtenu en transformant toutes les instructions par découpe de sommet
2. Montrer qu'un cycle de G' est uniquement composé de dépendances de flot ou est uniquement composé de dépendances de sortie

On élimine ainsi presque tous les cycles. Le problème est de choisir un nombre réduit d'instructions à transformer pour arriver au même résultat (car chaque transformation nécessite d'allouer un tableau temporaire en mémoire).

Déterminer le nombre minimum d'instructions à transformer est NP-difficile, même si G ne comporte que des anti-dépendances : montrer que le problème de décision TRANS-GRAPHE-ANTI est NP-complet :

TRANS-GRAPHE Soit $G = (V, E, \ell)$ un GDR, où l'étiquette $\ell(e)$ de chaque arête donne la nature de a dépendance : $\ell(e) \in \{f, a, s\}$. Soit K un entier positif. Existe-t-il K sommets dont la transformation conduit à un graphe sans cycle ?

TRANS-GRAPHE-ANTI Soit $G = (V, E, \ell)$ un GDR, avec $\ell(e) = a \ \forall e \in E$. Soit K un entier positif. Existe-t-il K sommets dont la transformation conduit à un graphe sans cycle ?