Ecole Polytechnique

Cours : "Parallélisme"
Travaux dirigés
E. Goubault, S. Putot & O. Bouissou & S. Zennou & S. Krishnan

TD 4 - corrigé

14 février 2007

1 Parallélisation, avec `petit`

On considère le code (en format `petit`) suivant:

```
real X(101), Y(102)

for I=1, 100 do
  X(I) = Y(I+2)+1
  Y(I) = X(I+1)+1+X(I)
endfor
```

- Calculez les dépendances de flot, de sortie et anti, avec `petit`. Pour ce faire, se reporter à l'annexe de ce sujet. Décrivez le GDRN correspondant.
- Appliquez l'algorithme d'Allen et Kennedy sur ce code.
- Montrez qu'en introduisant un tableau auxiliaire (sans l'aide de `petit`), il est possible de transformer le code de cette boucle de sorte que la distribution de boucle produise des boucles parallèles.
- Refaire cette distribution de boucle sous `petit` (autre solution possible qu'à la question précédente - cf. annexe)

Corrigé:

- On obtient sous `petit` le graphe de dépendances de la figure 1.

  On a donc essentiellement le GDRN de la figure 2.
- L'algorithme d'Allen et Kennedy constate qu'il n'y a qu'une composante fortement connexe, comprenant les deux noeuds (4 et 5): donc génère une boucle séquentielle externe. En conclusion, on n'arrive pas à paralléliser le code.
- On écrit maintenant:

  ```
  real X(101), Y(102), W(101)

  for I=1, 100 do
    X(I) = Y(I+2)+1
    W(I) = X(I+1)
    Y(I) = W(I)+1+X(I)
  endfor
  ```

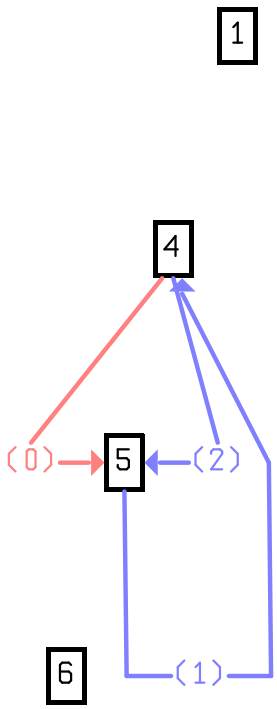


Figure 1:

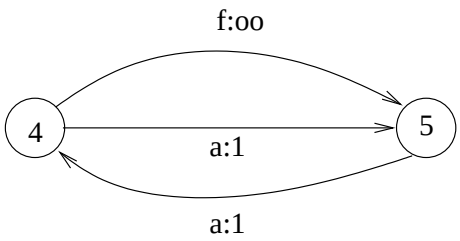


Figure 2:

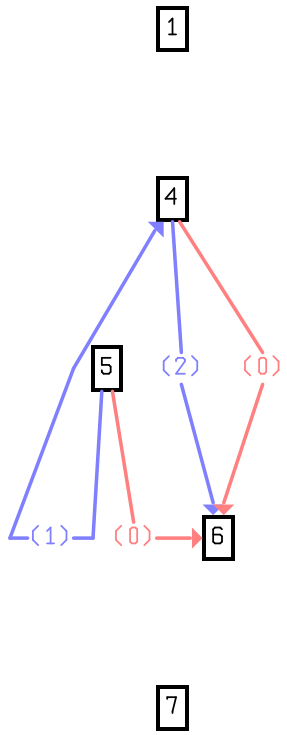


Figure 3:

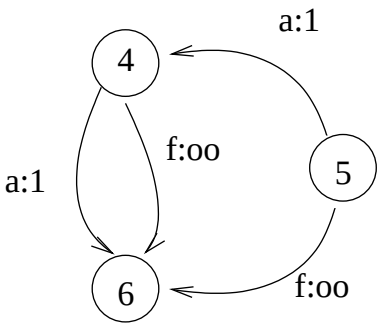


Figure 4:

On obtient alors sous `petit` le GDR de la figure 3.

C'est à dire qu'on obtient le GDRN de la figure 4.

En appliquant l'algorithme d'Allen et Kennedy on obtient alors le code:

```
real X(101), Y(102), W(101)

doall I=1, 100 do
  W(I) = X(I+1)
doall I=1, 100 do
  X(I) = Y(I+2)+1
doall I=1, 100 do
  Y(I) = W(I)+1+X(I)
endfor
```

En effet: les noeuds `4`, `5` et `6` forment chacun une composante fortement connexe dans le graphe 4. Le tri topologique donne, par exemple, `5` avant `4` avant `6`.

- Par `petit`, il faut se placer (par `Browse`, `DD`) sur une des deux antidépendances à casser dans le graphe 1. On a alors deux cas:
  - Soit on se place sur la dépendance:

    ```
    anti     5: X(I+1)          -->   4: X(I)            (1)              [ Mo]
    ```

    En appuyant alors sur `Expd`, on obtient:

    ```
      1: Entry
      1: inout real X(1:101)
      1: auto real X_r(1:101)
      1: inout real Y(1:102)
      3: for I = 1,100 do
      4:   X_r(I) = Y(I+2)+1
      5:   Y(I) = X(I+1)+1+X_r(I)
      3: endfor
      6: Exit
    ```

    On obtient le graphe de dépendance de la figure 5, et on peut bien évidemment paralléliser, après distribution de boucles:

    ```
    inout real X(1:101)
    auto real X_r(1:101)
    inout real Y(1:102)
    doall I = 1,100 do
      X_r(I) = Y(I+2)+1
    doall I = 1,100 do
      Y(I) = X(I+1)+1+X_r(I)
    endfor
    ```

  - Soit sur:

    ```
    anti     4: Y(I+2)          -->   5: Y(I)            (2)              [ Mo]
    ```

    En appuyant alors sur `Expd`, on obtient:

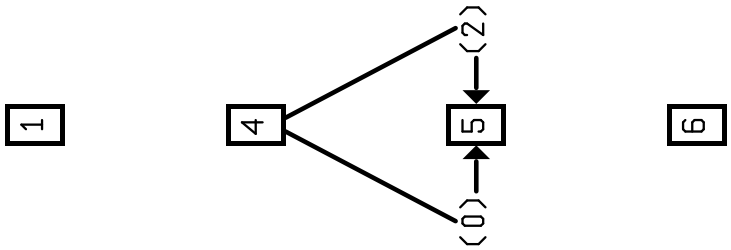


Figure 5:

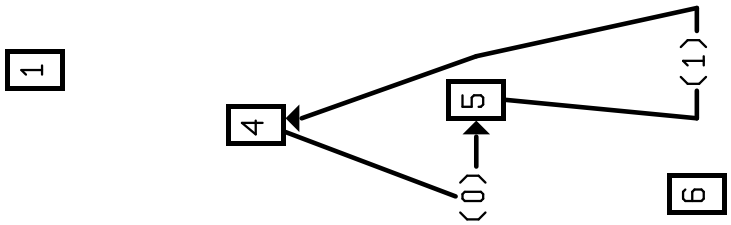


Figure 6:

```
1: Entry
1: inout real X(1:101), Y(1:102)
1: auto real Y_r(1:102)
3: for I = 1,100 do
4:   X(I) = Y(I+2)+1
5:   Y_r(I) = X(I+1)+1+X(I)
3: endfor
6: Exit
```

On obtient le graphe de dépendance de la figure 6, et on ne peut pas paralléliser directement!

2 Parallélisations simples

On souhaite calculer:

(1) Les entrées $B(i,j)$ $(1 \leq i \leq N,\ 1 \leq j \leq N+1)$ du triangle de Pascal. En supposant que l'on initialise le tableau B à 0 sauf $B(1,1) = 1$, cela revient à calculer:

$$B(i,j) = B(i-1,j) + B(i-1,j-1)$$

pour i de 1 à N et j de 1 à i

Question a Quel est le graphe de dépendance de (1)?

Question b Comment paralléliser (1)?

Corrigé

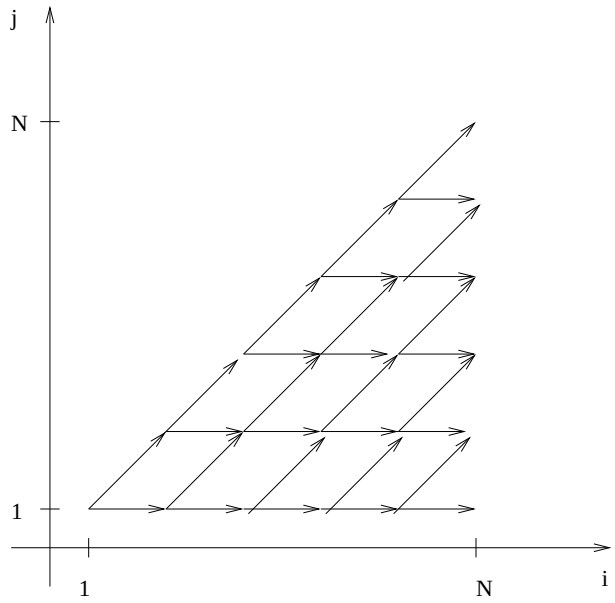


Figure 7: Graphe de dépendance pour (1).

Question a (1)
```
int B[N+1][N+2];

for (i=1;i<=N;i++)
  for (j=1;j<=i;j++)
    B[i][j] = B[i-1][j]+B[i-1][j-1];
```
Le graphe de dépendance est celui de la figure 7. On a deux dépendances de flot, de distance (1,0) et (1,1).

Question b – Pour (1) il n'y a rien à faire: il n'y a pas de dépendance le long de l'axe j, et la direction de dépendance est vers les i croissants. On peut donc paralléliser la boucle sur j (cf. algorithme d'Allen et Kennedy, ou à la main...):

```
B[1][1] = 1;
for (i=1;i<=N;i++)
  forall (j=1;j<=i;j++)
    B[i][j] = B[i-1][j]+B[i-1][j-1];
```

3 Parallélisation de l'algorithme de Cholesky

Soit `A` une matrice symétrique définie positive. L'algorithme de Cholesky permet de résoudre le système linéaire `Ax=b` en décomposant `A` en le produit d'une matrice `M` avec `M` transposée. Ci-dessous, une portion de code implémentant cette décomposition en langage `petit`:

```
real a(100, 100)
real b(100)
integer n
for k=1, n do
  a(k,k)=sqrt(a(k,k))
  for i=k+1, n do
```

```
    a(i,k) = a(i,k)/a(k,k)
    for j=k+1, i do
      a(i,j) = a(i,j)-a(i,k)*a(j,k)
    endfor
  endfor
endfor
```

(1) Déterminer le graphe de dépendance du code donné plus haut. Une des boucles `for` est-elle parallélisable telle quelle?

(2) Essayez de casser les cycles de dépendance impliquant le noeud `5` en appliquant une transformation simple.

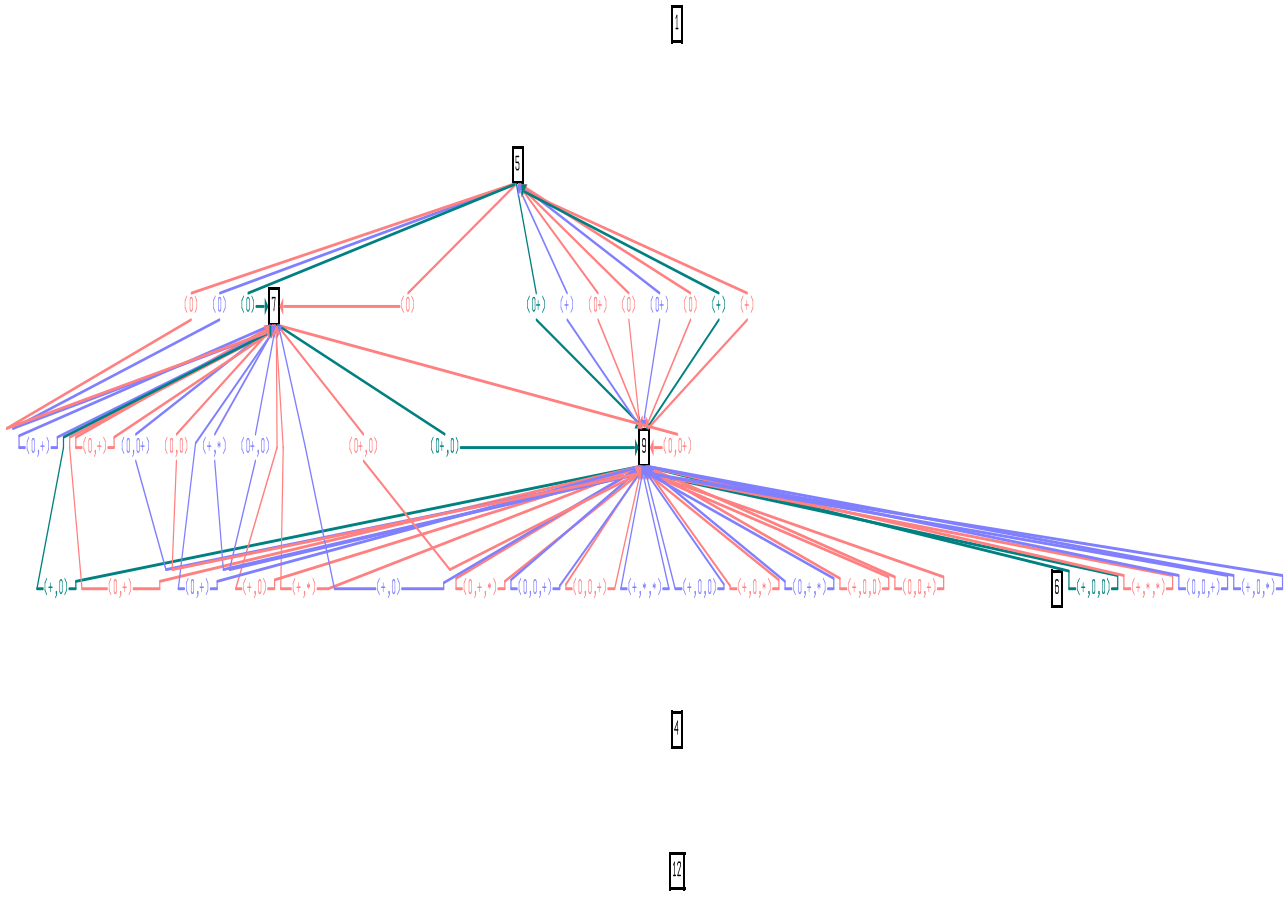
(3) Opérer un échange de boucle entre les noeuds `9` et `10` du programme obtenu à la question précédente (c'est-à-dire les boucle `for i` et `for j` imbriquées). Qu'obtient-on en termes de dépendances?

(4) Inverser l'ordre de parcours de la boucle `for j` dans le programme précédent (noeud `9` a priori). Qu'obtient-on en termes de dépendances?

(5) Quelles sont les boucles parallélisables? Ecrire le code `petit` correspondant.

Corrigé:

1 Les noeuds `5`, `7` et `9` sont inclus dans des cycles de dépendances (voir la sortie de `petit`):



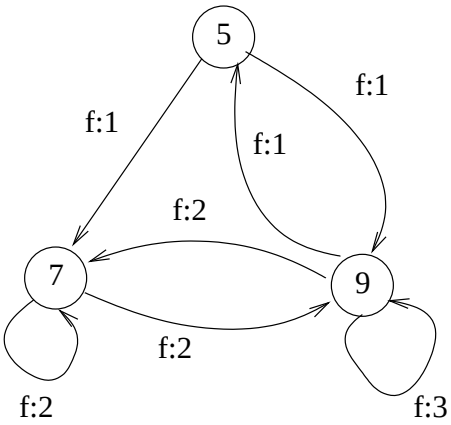


Figure 8:

Les boucles ne sont pas parallélisables sans transformation préliminaire. En utilisant `Browse` et `DD` puis `Cycl`, on peut passer sur toutes les dépendances et ne garder que celles qui sont marquées `MO` (voulant dire qu'elles sont impliquées dans un cycle de dépendance). Ainsi, on obtient le cycle de la figure 8.

2 On fait une distribution de boucle pour obtenir:

```
real a(100, 100)
real b(100)
integer n
for k=1, n do
  a(k,k)=sqrt(a(k,k))
  for i=k+1, n do
    a(i,k) = a(i,k)/a(k,k)
  endfor
  for i=k+1, n do
    for j=k+1, i do
      a(i,j) = a(i,j)-a(i,k)*a(j,k)
    endfor
  endfor
endfor
```

On obtient ainsi le graphe de dépendance:

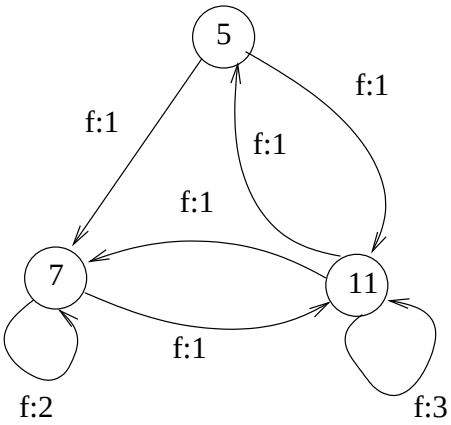
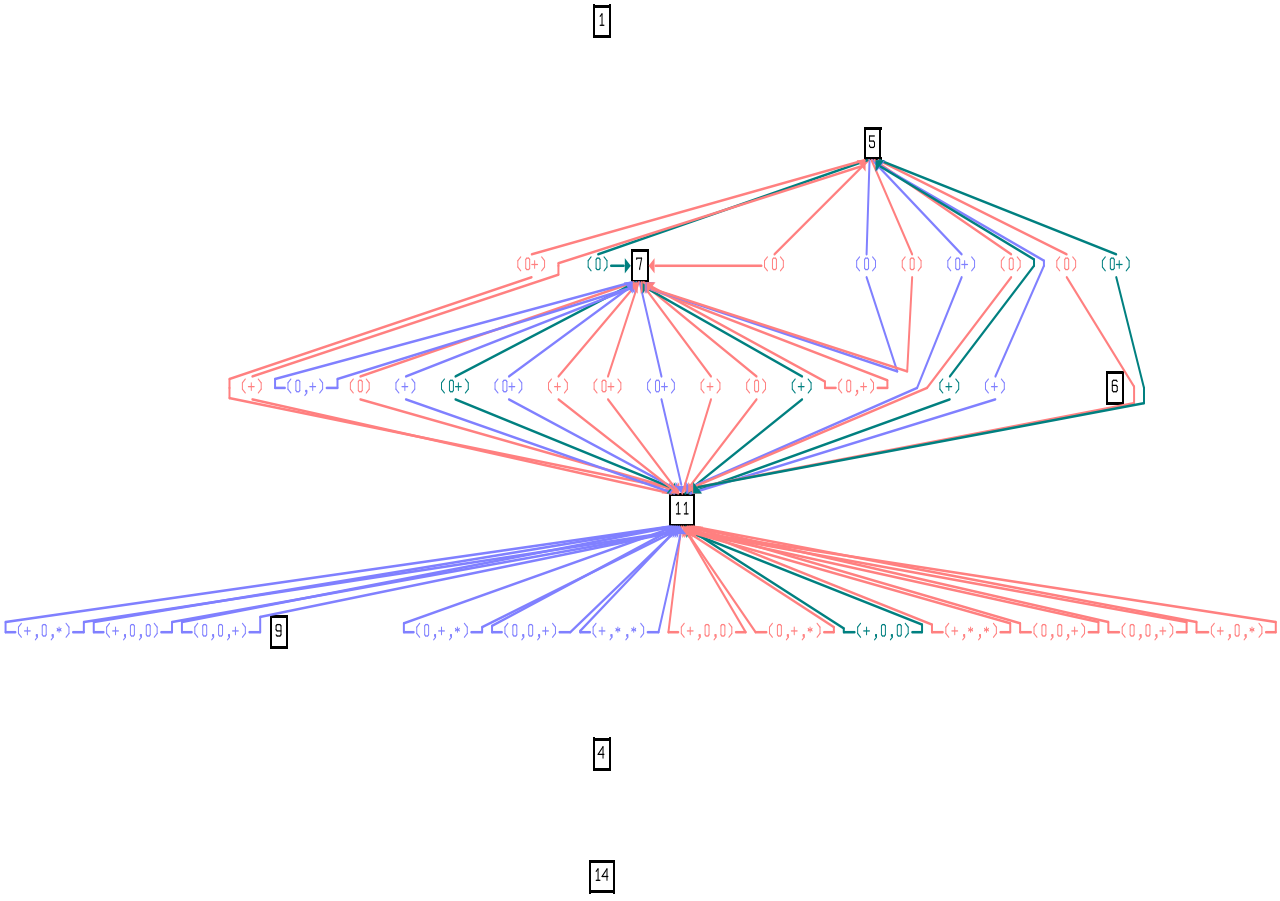


Figure 9:

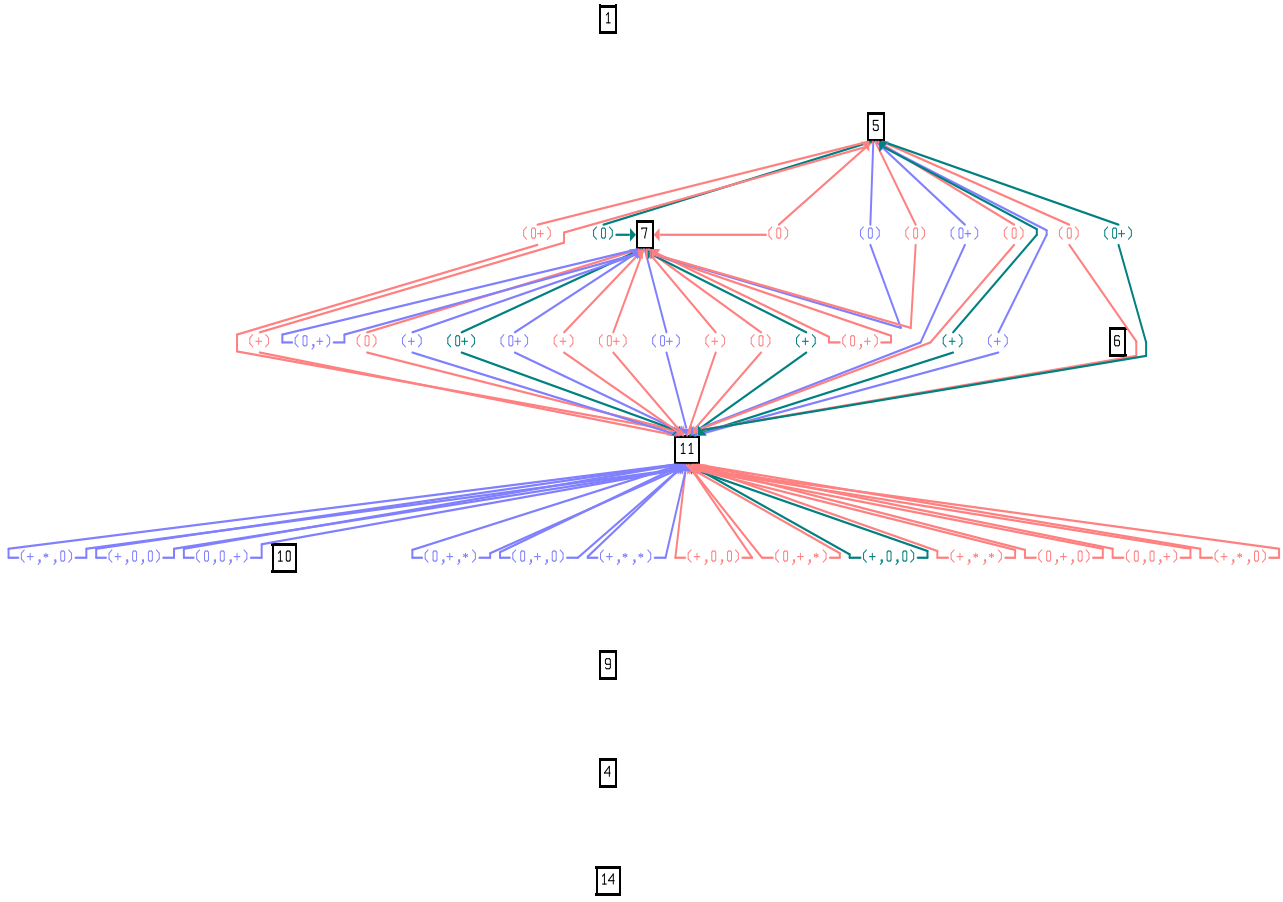


On a toujours un cycle de dépendances, voir la figure 9. On voit que la dépendance entre les noeuds 7 et 11 ont décru, pour être portés par la boucle sur `k` uniquement. On peut en fait paralléliser toutes les boucles présentes sauf la boucle externe sur `k`.

(3) On obtient le programme:

```
real a(100, 100)
real b(100)
integer n
for k=1, n do
  a(k,k)=sqrt(a(k,k))
  for i=k+1, n do
    a(i,k) = a(i,k)/a(k,k)
  endfor
  for j=k+1, n do
    for i=j, n do
      a(i,j) = a(i,j)-a(i,k)*a(j,k)
    endfor
  endfor
endfor
```

Et le graphe de dépendance suivant:

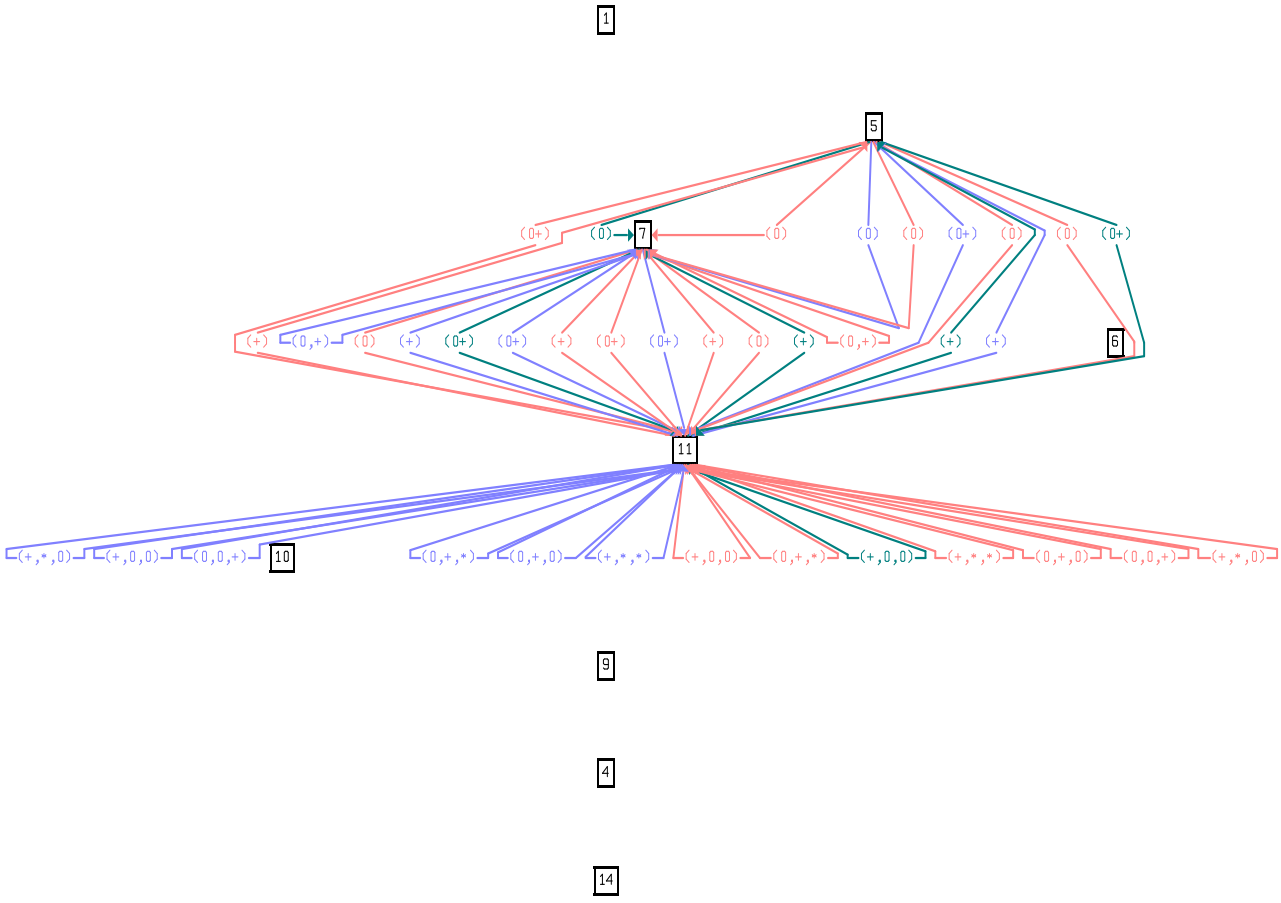


De fait, cela ne permet pas de casser les cycles de dépendances (ou de les faire porter par une boucle plus externe), on obtient toujours le cycle de la figure 9. En fait seules les directions et les distances de dépendances sont interchangées entre les deux boucles ainsi échangées.

(4) On obtient:

```
real a(100, 100)
real b(100)
integer n
for k=1, n do
  a(k,k)=sqrt(a(k,k))
  for i=k+1, n do
    a(i,k) = a(i,k)/a(k,k)
  endfor
  for j=-n, -(k+1) do
    for i=-j, n do
      a(i,-j) = a(i,-j)-a(i,k)*a(-j,k)
    endfor
  endfor
endfor
```

Et le graphe de dépendance suivant:



Rien n'est changé non plus (on change simplement les vecteurs de dépendances).

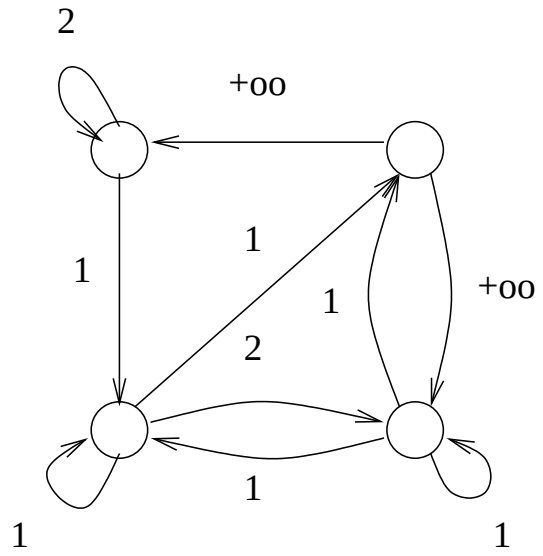
(5) On peut paralléliser le code ainsi:

```
real a(100, 100)
```

```
real b(100)
integer n
for k=1, n do
  a(k,k)=sqrt(a(k,k))
  doall i=k+1, n do
    a(i,k) = a(i,k)/a(k,k)
  endfor
  doall j=-n, -(k+1) do
    doall i=-j, n do
      a(i,-j) = a(i,-j)-a(i,k)*a(-j,k)
    endfor
  endfor
endfor
```

4 Dépendances de données

Donner un nid de boucle parfait où toutes les dépendances sont uniformes, et dont le GDRN est exactement le graphe suivant:



Exécuter l'algorithme d'Allen et Kennedy sur ce nid de boucles.

A Utilisation de `petit`

- Pour le calcul de dépendances "simples":

  lancer `petit` sur le code `ex1.t` pris sur la page web du cours (pour le premier exercice):

  ```
  petit ex1.t &
  ```

  (après avoir paramétré correctement son `PATH` comme expliqué en début de séance). Faites ensuite `System`, `DDalg` puis `eps` pour sélectionner un calcul de dépendances simple. Pressez `Xcape` pour revenir au menu du niveau supérieur (mais pas `Quit` cela quitte l'application). Faites ensuite, au menu de niveau supérieur `CalcDD` pour calculer les dépendances, puis `Graph` pour faire afficher le graphe.

Les arcs de couleur bleue sont des dépendances anti ou de sortie, les arcs de couleur rouge sont des dépendances de flot. Les annotations entre parenthèses sont les mêmes que celles que l'on trouve dans le cours pour les GDRV.

- Pour avoir plus d'informations sur les dépendances:

  Changer le mode de dépendances ou faisant `System`, `DDalg` puis `Omega` (puis deux fois `Xcape`). Refaire `CalcDD`. Faire ensuite, dans le menu de plus haut niveau: `Browse` puis `DD` pour parcourir textuellement les dépendances. Pour les voir apparaitre dans une fenêtre séparée, faire `Togg` sous le même menu. Vous pouvez sous ce même menu parcourir l'ensemble des dépendances ainsi trouvées par `Cycl`. Vous les verrez ainsi défiler dans la fenêtre correspondante, avec les informations sur les vecteurs d'itération concernés. L'attribut `[O]` (éventuellement combiné à d'autres) indique que l'arc de dépendances sur lequel vous êtes est dans un cycle de dépendances.

- Pour faire introduire un tableau intermédiaire:

  Vous devez être en mode de calcul de dépendances `eps` (voir plus haut). Ensuite, vous devez être dans le menu de parcours de dépendances `Browse`, `DD`. Une fois que vous êtes sur l'une des dépendances anti ou de sortie que vous voulez casser, vous pouvez essayer de faire en sorte que `petit` introduise un tableau intermédiaire pour ce faire, par `Expd`.