

Nids de boucles - suite

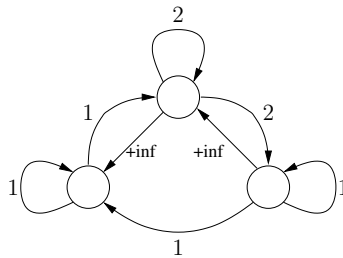
1 Algorithme d'Allen et Kennedy

▷ **Question 1** On considère le code suivant :

Pour $i = 1$ à N **Pour** $j = 1$ à N
 $S_1 : a(i + 1, j + 1) \leftarrow a(i + 1, j) + b(i, j + 2)$
 $S_2 : b(i + 1, j) \leftarrow a(i + 1, j - 1) + b(i, j - 1)$
 $S_3 : a(i, j + 2) \leftarrow b(i + 1, j + 1) - 1$

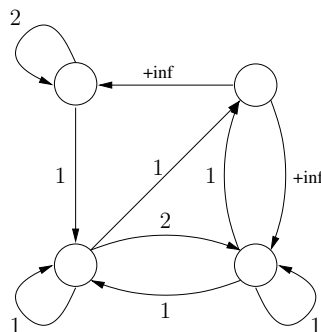
Donner le graphe de dépendances réduit, avec pour chaque dépendance son type (flot, anti, sortie) et son niveau. Appliquer l'algorithme d'Allen et Kennedy pour restructurer le nid de boucles et vérifier la nature des boucles obtenues.

▷ **Question 2** Donner un nid de boucles 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.

▷ **Question 3** Donner un nid de boucles 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.

2 Permutation de boucles

On considère un nid de boucle parfait de profondeur n , qui comprend m dépendances uniformes. Soit D la matrice de taille $n \times m$, obtenue à partir du signe des composantes des vecteurs de dépendance. Si d_i est le i -ème vecteur de dépendance, on stocke le signe $+$, 0 ou $-$ de ses composantes dans la i -ème colonne de D . Par exemple si $n = 3$ et $d_1 = (2, -2, 0)^t$, on stockera $(+, -, 0)^t$ dans la première colonne de D .

▷ **Question 4** Donner la matrice D_1 pour le nid suivant :

```

for i = 1 to n do
  for j = 1 to n do
    for k = 1 to n do
      S1 : a(i + 1, j, k) = a(i, j, k) + 2
      S2 : b(i, j, k + 1) = b(i, j, k) - 1
      S3 : c(i + 1, j + 1, k + 1) = c(i, j, k) + 1

```

▷ **Question 5** Donner la matrice D_2 pour le nid suivant :

```

for i = 1 to n do
  for j = 1 to n do
    for k = 1 to n do
      S1 : a(i, j, k + 1) = a(i, j - 1, k) + a(i - 1, j, k + 2) + 2

```

▷ **Question 6** Donner un nid dont la matrice D_3 soit

$$D_3 = \begin{pmatrix} + & + & + & 0 & 0 & 0 \\ + & 0 & 0 & + & 0 & 0 \\ 0 & + & 0 & 0 & + & 0 \\ 0 & 0 & + & 0 & 0 & + \end{pmatrix}$$

L'algorithme de parallélisation fonctionne ainsi :

- Si une ligne de la matrice D (disons la i -ème) ne contient que des 0, alors on place cette boucle à l'extérieur du nid et on l'exécute en parallèle : on permute les boucles, la i -ème boucle devient la première boucle, qui est exécutée en parallèle.
- Si aucune ligne de la matrice D ne contient que des 0, on sélectionne la ligne qui contient le plus de +, on place la boucle correspondante à l'extérieur et on l'exécute en séquentiel.

On exécute alors récursivement l'algorithme sur les boucles et les dépendances restantes, i.e. qui n'ont pas été satisfaites par la séquentialisation de la boucle.

Par exemple pour D_1 , on choisit de séquentialiser soit la première soit la troisième boucle. Si on choisit la première, pas besoin de permuter, les dépendances 1 et 3 sont satisfaites. Il reste deux boucles à l'intérieur, avec la matrice réduite $\overline{D}_1 = (0, +)^t$. On en déduit le code parallèle, où `forseq` représente une boucle exécutée en séquentielle, et `forpar` représente une boucle exécutée en parallèle :

```

forseq i
  forpar j
    forseq k
    ...

```

Bien sûr, on aurait pu choisir d'abord la boucle sur k et obtenir :

```

forseq k
  forpar j
    forseq i
    ...

```

▷ **Question 7** Faire tourner l'algorithme sur l'exemple avec D_2 .

▷ **Question 8** Faire tourner l'algorithme sur l'exemple avec D_3 .

▷ **Question 9** Prouver la correction de l'algorithme. Pour cela, prouver qu'on a le droit de déplacer à l'extérieur et paralléliser une boucle correspondant à une ligne nulle, et préciser quelles sont les dépendances qui sont satisfaites après le choix d'une boucle, son déplacement à l'extérieur et sa séquentialisation.

▷ **Question 10** Pensez-vous que cet algorithme conduise à un nombre maximal de boucles parallèles (utiliser l'exemple avec D_3) ?

▷ **Question 11** Montrer que trouver le nombre maximal de boucles parallèles est NP-complet, par réduction à partir de SET-COVER : étant donné un ensemble X à n éléments, K sous-ensembles de X et une borne B , peut-on trouver B sous-ensembles parmi les K tels que tout élément de X appartienne à au moins l'un de ces B sous-ensembles ?

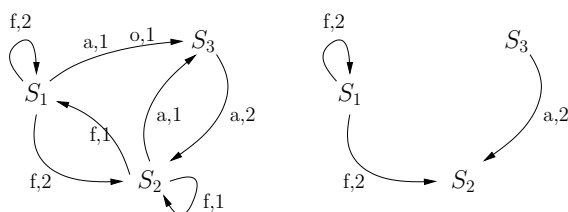
3 Réponses aux exercices

▷ Question 1, page 1

On obtient les dépendances suivantes :

- Flot $S_1 \rightarrow S_1$: variable a , distance $(0, 1)$
- Flot $S_1 \rightarrow S_2$: variable a , distance $(0, 2)$
- Flot $S_2 \rightarrow S_1$: variable b , distance $(1, -2)$
- Flot $S_2 \rightarrow S_2$: variable b , distance $(1, 1)$
- Anti $S_1 \rightarrow S_3$: variable a , distance $(1, -2)$
- Anti $S_2 \rightarrow S_3$: variable a , distance $(1, -3)$
- Anti $S_3 \rightarrow S_2$: variable b , distance $(0, 1)$
- Sortie $S_1 \rightarrow S_3$: variable a , distance $(1, -1)$

Le GDRN initial est dessiné sur la gauche :



Le GDRN est fortement connexe et il y a des dépendances de niveau 1. La boucle sur i sera donc séquentielle. En passant à la deuxième étape, on obtient le GDRN de droite sur la figure. L’algorithme d’Allen et Kennedy conduit alors au nid restructuré suivant :

Pour $i = 1$ to N en séquentiel :

Pour $j = 1$ to N en séquentiel :

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

Pour $j = 1$ to N en parallèle :

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

Pour $j = 1$ to N en parallèle :

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

On notera qu’il est parfaitement possible d’intervertir la boucle portant S_3 avec celle portant S_1 .

▷ Question 2, page 1

Voici le nid de boucles :

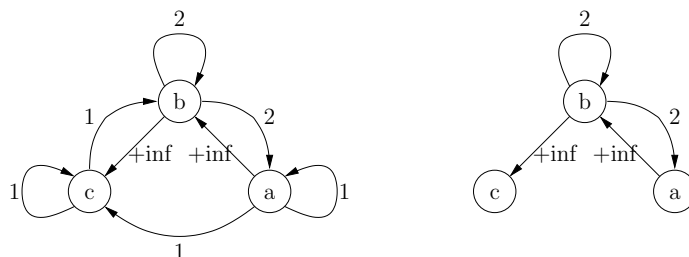
Pour $i = 1$ to N :

Pour $j = 1$ to N :

$$S_1 : a(i, j) \leftarrow a(i - 1, j) + b(i, j - 1)$$

$$S_2 : b(i, j) \leftarrow a(i, j) + b(i, j - 1) + c(i - 1, j)$$

$$S_3 : c(i, j) \leftarrow a(i - 1, j) + b(i, j) + c(i - 1, j)$$



L’algorithme d’Allen et Kennedy conduit au nid restructuré suivant :

Pour $i = 1$ to N en séquentiel :

Pour $j = 1$ to N en séquentiel :

$$S_1 : a(i, j) \leftarrow a(i - 1, j) + b(i, j - 1)$$

$$S_2 : b(i, j) \leftarrow a(i, j) + b(i, j - 1) + c(i - 1, j)$$

Pour $j = 1$ to N en parallèle :

$$S_3 : c(i, j) \leftarrow a(i - 1, j) + b(i, j) + c(i - 1, j)$$

▷ **Question 3, page 1**

On peut construire le code suivant :

Pour $i = 1$ **to** N :

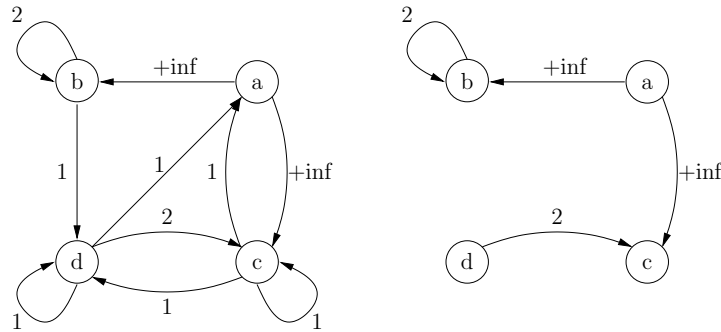
Pour $j = 1$ **to** N :

$$S_1 : a(i, j) \leftarrow d(i - 1, j) + c(i - 1, j)$$

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

$$S_3 : c(i, j) \leftarrow a(i, j) + c(i - 1, j) + d(i, j - 1)$$

$$S_4 : d(i, j) \leftarrow c(i - 1, j) + d(i - 1, j) + b(i - 1, j)$$



L'algorithme d'Allen et Kennedy conduit au nid restructuré suivant :

Pour $i = 1$ **to** N **en séquentiel** :

Pour $j = 1$ **to** N **en parallèle** :

$$S_4 : d(i, j) \leftarrow c(i - 1, j) + d(i - 1, j) + b(i - 1, j)$$

Pour $j = 1$ **to** N **en parallèle** :

$$S_1 : a(i, j) \leftarrow d(i - 1, j) + c(i - 1, j)$$

Pour $j = 1$ **to** N **en parallèle** :

$$S_3 : c(i, j) \leftarrow a(i, j) + c(i - 1, j) + d(i, j - 1)$$

Pour $j = 1$ **to** N **en séquentiel** :

$$S_4 : d(i, j) \leftarrow c(i - 1, j) + d(i - 1, j) + b(i - 1, j)$$

▷ **Question 4, page 2**

$$D_1 = \begin{pmatrix} + & 0 & + \\ 0 & 0 & + \\ 0 & + & + \end{pmatrix}$$

▷ **Question 5, page 2**

$$D_2 = \begin{pmatrix} 0 & + \\ + & 0 \\ + & - \end{pmatrix}$$

▷ **Question 6, page 2**

La boucle suivante donne la matrice souhaitée :

```

for  $i = 1$  to  $n$ 
  for  $j = 1$  to  $n$ 
    for  $k = 1$  to  $n$ 
      for  $l = 1$  to  $n$ 
         $S_1 : a(i, j, k, l) = a(i - 1, j - 1, k, l)$ 
           $+ a(i - 1, j, k - 1, l)$ 
           $+ a(i - 1, j, k, l - 1)$ 
           $+ a(i, j - 1, k, l)$ 
           $+ a(i, j, k - 1, l)$ 
           $+ a(i, j, k, l - 1)$ 
    
```

▷ **Question 7, page 2**

En appliquant la méthode sur D_2 , en choisissant par exemple de séquentialiser d'abord la boucle sur i , on obtient plus qu'une seule dépendance : $\overline{D_2} = \begin{pmatrix} 0 \\ + \\ + \end{pmatrix}$. On séquentialise ensuite la boucle sur j , ce qui enlève la dernière dépendance, on peut donc paralléliser la boucle sur k . On obtient :

```

forseq i
  forseq j
    forpar k
      S1 : a(i, j, k + 1) = a(i, j - 1, k) + a(i - 1, j, k + 2) + 2

```

▷ **Question 8, page 2**

Pour la matrice D_3 , la méthode ne conduit à aucune parallélisation de boucle, on obtient :

```

forseq i
  forseq j
    forseq k
      forseq l
        S1

```

▷ **Question 9, page 2**

Si la i ème ligne de la matrice est nulle, alors la boucle i ne porte pas de dépendances, et on peut la sortir et l'exécuter en parallèle à condition de laisser les autres boucles dans le même ordre. Lorsqu'on supprime une ligne de la matrice contenant des + et qu'on la séquentialise, on commence par placer cette boucle en tête des autres, ce qui est possible puisque « on ne fait jamais passer un - devant un + » : tout vecteur de dépendance est transformé en vecteur strictement positif dans l'ordre lexicographique, donc la transformation est valide (proposition 9.2 du cours). Cette boucle est séquentialisée, donc tous les vecteurs de dépendance contenant un + pour cette boucle se trouve satisfait : comme on l'avait placé à l'extérieur, c'est le niveau de dépendance le plus externe qu'on supprime, ce qui satisfait la dépendance. Il ne reste que les dépendances qui avaient un 0 sur cette ligne.

▷ **Question 10, page 2**

Cet algorithme ne conduit pas à un nombre maximal de boucles parallèles : sur l'exemple de D_3 , il n'est capable d'exhiber aucune boucle parallélisme, alors que si on choisit d'éliminer les boucles sur j , k et l , on peut ensuite paralléliser celle sur i . On obtient ainsi :

```

forseq j
  forseq k
    forseq l
      forpar i
        S1

```

▷ **Question 11, page 2**

On sent que pour trouver le nombre minimal de boucles séquentielles, il faut trouver un sous-ensemble de lignes de la matrice D qui « couvre » toute la matrice : c'est-à-dire qui permet, si on les supprime (en les séquentialisant), de détruire toutes les dépendances. On pense alors naturellement à SET-COVER.

Avant de prouver la NP-complétude du problème, nous devons définir le problème de décision associé :

Définition 1 (PARA-MATRICE). Étant donné un nid de boucle ayant une matrice D (construite comme précédemment) et une borne B , est-il possible de transformer le nid de boucle pour qu'il contienne au plus B boucles séquentielles ?

Théorème 1. Le problème PARA-MATRICE est NP-complet.

Démonstration. On vérifie que PARA-MATRICE est dans NP : on peut vérifier en temps polynomial que le nid de boucle transformé est équivalent au nid de boucle initial, et qu'il comporte au plus B boucles séquentielles. On montre ensuite qu'il est NP-complet par réduction à SET-COVER. Soit S_1 une instance de SET-COVER, consistant en un ensemble X à n éléments (x_1, \dots, x_n) , k sous-ensembles (C_1, \dots, C_k) et une borne B . On crée une instance S_2 de PARA-MATRICE : on construit la matrice D , à n lignes et k colonnes, telle que $D[i, j] = +$ si $x_i \in C_j$, et $D[i, j] = 0$ sinon. Notez que D ne contient aucun $-$: on pourra donc permuter les boucles sans risque. Montrons que S_1 a une solution si et seulement si S_2 admet une solution.

- $S_1 \Rightarrow S_2$ Il existe alors un sous-ensemble de couvertures C_{a_1}, \dots, C_{a_B} de taille B . On choisit alors de placer en tête et de séquentialiser les boucles correspondant aux lignes a_1, \dots, a_B de la matrice. Pour chaque vecteur de dépendance j , il existe un indice de boucle i parmi celles-ci qui le couvre (c'est-à-dire tel que $D[i, j] = +$). Après la séquentialisation, tous les vecteurs de dépendance ont disparu, les boucles restantes peuvent donc être parallélisées.
- $S_2 \Rightarrow S_1$ Soit a_1, \dots, a_B les indices des boucles séquentialisées. Alors C_{a_1}, \dots, C_{a_B} forme une couverture de X .

■