TD 6 Analyse de dépendances - Parallélisation automatique.

6.1 Parallélisation par tri topologique

L'algorithme du *tri topologique* (fig. 1) permet de compiler une graphe de dépendances vers un programme parallèle utilisant la construction parallèle *parbegin*. On peut montrer que le programme parallèle obtenu est optimal en temps d'exécution si toutes les instructions ont la même durée d'exécution et si un nombre non borné de processeurs est disponible.

On considère le fragment de programme suivant :

```
s = y/100 + 1 ;
g = 3*s/4 -12 ;
c = (8*s + 5)/25 - 5 - g ;
o = y mod 19 + 1 ;
u = (11*o + 20 + c) mod 30 ;
a = u + o -11 ;
e = 5*y/14 - g ;
f = e/g ;
v = 44 - a ;
date = v + 7 - (v+f) mod 7 ;
mois = 3 + date/11 ;
jour = (date -1) mod 31 +1 ;
```

1. Donner le graphe de dépendances du programme puis paralléliser.
2. Toutes les instructions sont de durée 1, sauf f qui est de durée 2. Donner un ordonnancement optimal. L'algorithme du tri topologique fournit-il cet ordonnancement ? Ecrire le programme correspondant.

```
pour tout sommet s
   compteur(s) = nombre d'arcs entrants dans s
S1 = liste des sommets sans predecesseur
tant que S1 non vide faire
   S2 = ∅
   engendrer ("parbegin S1 parend")
   pour tout s dans S1
      pour tout t tels que (s,t) est un arc
         compteur(t) = compteur(t) - 1
         si compteur(t) == 0 alors S2 += t
   S1 = S2
```

FIG. 1 – Algorithme du tri topologique

6.2 Graphe de dépendance réduit

On considère les trois programmes suivants.

```
! P1
do i = 1, n
     A(i+1) = B(i) + c
     D(i) = A(i) + e
end do

! P2
do i = 1, n
     A(i) = B(i) + c
     B(i+1) = A(i) + e
end do

! P3
do i = 1, M
     X(i) = Y(i) + k
     do j = 1, M
          B(j) = A(j,n)
          do k = 1, M
               A(j+1,k) = B(j) + C(j,k)
          end do
          Y(i+j) = A(j+1,n)
     end do
end do
```

Pour chaque programme

1. Calculer les prédicats de séquencement.
2. Construire le graphe de dépendances résumé.

6.3 Parallélisation et vectorisation de boucles

1. Dans les programmes P1, P2 et P3, quelles sont les boucles parallèles ?
2. Vectoriser P3 par l'algorithme de Kennedy-Allen.

6.4 Echange de boucles

1. Montrer que dans le programme suivant, il est légitime d'intervertir la boucle `do j` et les instructions d'échange, mais que cette transformation ne préserve par le graphe de dépendances développé.

```
do i = 1, n
     do j = 1, n
          a(i,j) = a(i,j) + 8
     end do
     temp = a(i,1)
     a(i,1)= a(i,2)
     a(i, 2)= temp
end do
```

2. Peut-on échanger les boucles i et j dans le programme P4 ? dans le programme P5 ? Quel est l'intérêt de cet échange ?

```
! P4
do i = 1, n
     do j = 1, n
          A(i+1,j) = A(i,j) + B(i,j)
     end do
end do
```

```
!P5
do i = 1, n
     do j = 1, n
          A(i+1,j+1) = A(i,j) + B(i,j)
     end do
end do
```

6.5 Torsion de boucles

La version séquentielle de l'algorithme de *relaxation de Gauss-Seidel* (N.B. sans rapport avec l'élimination de Gauss) a la structure suivante

```
 ! Relaxation de Gauss-Seidel
do i = 1, n
   do j = 1, n
r     a(i,j) = a(i-1,j) + a(i,j-1) + b(i+1,j) + b(i,j+1)
   end do
end do
```

1. Etudier les dépendances de ce programme. L'une des boucles est-elle parallèle ?
2. On veut définir un ordonnancement affine : (r, i, j) s'exécute à l'instant $ai + bj$, pour une durée 1. Quelles sont les contraintes sur a et b ?
3. Modifier le code pour faire apparaître le parallélisme.

Appendice : Prédicat de séquencement

Définition . *Soient s et t deux instructions incluses dans n boucles* `DO` *ou* `DOALL`. *Soit Z_k à VRAI si et seulement si la boucle de niveau k est une boucle* `DO`.
Le prédicat de séquencement $<<$ est défini par :

$$(s, i_1, \ldots, i_n) << (t, i'_1, \ldots, i'_n) = LC \vee LI$$

avec

$$\begin{aligned}
LI &= (i_1 = i'_1) \wedge (i_2 = i'_2) \wedge \ldots \wedge (i_{n-1} = i'_{n-1}) \wedge (i_n = i'_n) \wedge (s <_{syntax} t) \\
LC &= (i_1 < i'_1) \vee \{(i_1 = i'_1) \wedge (i_2 < i'_2)\} \vee \{(i_1 = i'_1) \wedge (i_2 = i'_2) \wedge \ldots \wedge (i_{n-1} = i'_{n-1}) \wedge (i_n < i'_n)\} \\
&\quad \textit{si toutes les boucles sont des } \texttt{DO} \\
LC &= \{Z_1 \wedge (i_1 < i'_1)\} \vee \{Z_2 \wedge (i_1 = i'_1) \wedge (i_2 < i'_2)\} \vee \\
&\quad \{Z_n \wedge (i_1 = i'_1) \wedge (i_2 = i'_2) \wedge \ldots \wedge (i_{n-1} = i'_{n-1}) \wedge (i_n < i'_n)\} \\
&\quad \textit{dans le cas général}
\end{aligned}$$

Proposition . *Une boucle* `DO` *de profondeur k est équivalente à une boucle* `DOALL` *s'il n'existe pas de dépendance de niveau k.*

Les ensembles Read et Write ne sont pas modifiés par la transformation du `DO` en `DOALL`. Pour simplifier, considérons le cas d'un indice (i, j, k) où il n'existe pas de dépendance de profondeur 2 (boucle j). Les prédicats de séquencement sont les suivants

Niveau	Programme initial	Programme parallèle
1	$(i < i')$	$(i < i')$
2	$(i = i') \wedge (j < j')$	FAUX
3	$(i = i') \wedge (j = j') \wedge (k < k')$	$(i = i') \wedge (j = j') \wedge (k < k')$

S'il n'existe pas de dépendance de niveau 2, il n'existe pas de collision pour des opérations $s(i, j, k)$ et $t(i, j', k')$ avec $j < j'$, donc la différence des prédicats à ce niveau ne modifie pas les dépendances.