Laboratoire de l'Informatique du Parallélisme

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Stage de DEA

Ordonnancement multidimensionnel pour les équations récurrentes uniformes et affines

Frédéric Vivien

sous la direction conjointe de
Alain Darte
et
Yves Robert

Juin 1994

Résumé

Dans ce rapport nous présentons un algorithme de construction d'un ordonnancement multidimensionnel pour les systèmes d'équations récurrentes uniformes. Cet algorithme est optimal en ordre de grandeur: il extrait tout le parallélisme existant du système traité. Cet algorithme est étendu au cas des systèmes d'équations récurrentes affines dont les dépendances sont décrites dans l'algèbre $(\mathbb{Z}, +, -, *)$. Une généralisation de cet algorithme au cas des nids de boucles non-parfait conclut l'exposé.

"On a tort de consacrer à la spéculation pure un temps que l'on pourrait occuper à l'examen des réalités tangibles, audibles, en un mot accessibles à nos organes de perception."

Les bâtisseurs d'Empire ou le Schmürz, Boris Vian.

Table des matières

Chapitre 1

Qu'est-ce que la parallélisation automatique?

1.1 Objectif

L'objectif de la parallélisation automatique est de prendre en entrée un programme écrit dans un langage impératif séquentiel et de le transformer en un programme à parallélisme explicite équivalent.

Le but envisagé étant par trop ambitieux, certaines restrictions sont apparues naturellement. La classe des programmes sources séquentiels étant mal définie et trop générale, la parallélisation automatique a focalisé son attention sur les programmes uniquement constitués de boucles imbriquées (ou *nids de boucles*) définies par des fonctions affines. Cette classe de programme est assez générale pour couvrir une grande part des programmes à caractère scientifique et gourmands en puissance de calcul. Ces programmes se caractérisent généralement par la simplicité du code, la régularité des données (présence de tableaux et abscence de structures irrégulières).

1.2 Motivations

La motivation profonde de la parallélisation automatique est de faciliter l'utilisation des machines parallèles. S'il utilise un compilateur-paralléliseur, un programmeur n'a plus besoin d'expliciter les messages liés aux communications, il n'a plus besoin de rechercher le parallélisme plus ou moins implicite des algorithmes à implanter, etc.

La parallélisation automatique est donc facteur de simplicité, de portabilité et d'ouverture du parallélisme vers de simples utilisateurs non-spécialistes.

1.3 Moyens

La paralléllisation d'un programme repose sur trois manipulations fondamentales:

- La répartition des calculs.
- La répartition des données.
- Les restructurations de code.

Toutes ces manipulations ne peuvent être mises en œuvre qu'après la détection du parallélisme. Avant toute chose il convient de déterminer le degré de parallélisme du programme étudié, de décider si une parallélisation induira ou non un gain de temps substantiel. Un programme de multiplication de matrices carrées de taille N coûte N^3 opérations qui peuvent s'exécuter en parallèle en un temps en N^2. Il y a donc possibilité de gain, le nombre de synchronisations pouvant être absorbé par le parallélisme. Mais ce n'est pas forcément le cas. La détection du parallélisme est donc une première étape essentielle à la parallélisation (ou non...) Une fois la détection faite il conviendra de répartir les calculs dans l'espace et le temps, de trouver la bonne granularité, d'optimiser les communications (éviter au maximum les communications point à point), puis réécrire le code... toutes ces étapes reposant sur la détection initiale du parallélisme.

1.4 Techniques usuelles

L'analyse de dépendance est une étape incontournable de la détection du parallélisme.

Cette première phase indispensable est la plupart du temps suivie par une phase de **vérification** du parallélisme. Le problème consiste alors à déterminer si une boucle est ou non parallèle. Les techniques employées sont souvent basées sur des tests et sont plutôt des heuristiques ("que se passe-t-il si je fais telle ou telle inversion de boucles") que des transformations générales. Il n'existe pas d'approche globale mis à part Paul Feautrier et son ordonnancement multidimensionnel, le tandem Darte-Robert et son ordonnancement linéaire ou la méthode de Kennedy-Allen, de loin la plus connue.

Nous ne pratiquerons pas ici de vérification du parallélisme, mais une **détection** à partir d'une généralisation des travaux de Karp *et al.*, de Rao et de Paul Feautrier, tout en gardant l'optique du tandem Darte/Robert: "Peut-on justifier nos manipulations?".

Chapitre 2

Définitions

2.1 Graphes

Commencerons par rappeler quelques définitions de base sur les graphes. Nous poserons par la même occasion quelques unes des notations et conventions qui seront utilisées tout au long de ce rapport. Les définitions proviennent du livre de Xuong [Xuo92]

Définition 1 *Un* ***graphe*** *G est un couple (V, E) formé de deux ensembles finis et disjoints $V = \{v_1, v_2, \ldots, v_n\}$ où $n \geq 1$ et $E = \{e_1, e_2, \ldots, e_m\}$ avec $m \geq 0$, tels que, pour tout i, $0 \leq i \leq m$, e_i est une paire d'éléments de V. V est appelé ensemble des sommets et E ensemble des arêtes de G.*

Définition 2 *Un* ***graphe valué*** *est un graphe pour lequel un vecteur de poids est associé à chaque arête.*

Nous noterons ici $\mathbf{n}$ la dimension des vecteurs de poids des graphes valués, et $\mathbf{w}$ la fonction qui à une arête e associe son poids.

Définition 3 *Un graphe est* ***orienté*** *lorsque, pour toute arête e de E, on a ordonné l'ensemble de ses extrémités.*

Soient un graphe orienté G et une arête e de E. e est orientée de v_i vers v_j. v_i (resp. v_j) est dénommé le sommet initial ou la *queue* (resp. l'extrémité terminale ou la *tête*) de e et son indice dans V est donné par la fonction $tail(e)$ (resp. $head(e)$).

Définition 4 *Un* ***chemin*** *d'un graphe orienté $G = (V, E)$ est une séquence alternée de sommets et d'arcs $\Gamma = v_0\ e_1\ v_1\ \ldots\ v_{k-1}\ e_k\ v_k$ telle que, pour tout $i \in [1..k]$, e_i admet v_{i-1} comme sommet initial et v_i comme extrémité terminale.*

Définition 5 *Un graphe orienté $G = (V, E)$ est un* ***graphe fortement connexe*** *si pour toute paire $\{v_1, v_2\}$ de sommets de V il existe un chemin de v_1 à v_2 et un chemin de v_2 à v_1.*

Définition 6 *On appellera ici* ***cycle*** *un chemin fermé fortement connexe, c'est-à-dire un chemin pour lequel $v_0 = v_k$.*

Le poids d'un cycle est la somme des poids de chacune de ses arêtes constitutives, chaque arête étant pondérée par son nombre d'utilisations dans le cycle.

Définition 7 *On appelle* ***multicycle*** *d'un graphe orienté la réunion d'un ensemble fini de cycles de ce graphe.*

Le poids d'un multi-cycle est la somme des poids de chacun de ses multi-cycles constitutifs, chaque cycle apparaissant autant de fois dans la somme que dans le multi-cycle.

Définition 8 *On appelle* ***profondeur*** *d'un arbre, la longueur de sa plus grande branche, c'est-à-dire le nombre d'arêtes de sa plus grande branche.*

Cette définition de la profondeur est d'une unité inférieure à celle choisie par Karp, Miller et Winograd [KMW67] et par Sailesh Rao [Rao85] pour qui la profondeur est le nombre de sommets sur le plus long chemin.

2.2 Programmation linéaire

Voici quelques "rappels" de programmation linéaire. Pour de plus amples renseignements se référer au livre d'Alexander Schrijver [Sch86].

Les résultats suivants sont vrais dans tout corps commutatifs totalement ordonnés, et nous nous placerons dans la suite dans $\mathbb{Q}$.

Théorème 1 *(dit* ***Lemme de Farkas*** *ou lemme de Farkas-Minkowski)*
Soient une matrice A et un vecteur b. Il existe un vecteur $x \geq 0$ tel que $A.x = b$ si, et seulement si, $y.A \geq 0 \Rightarrow yb \geq 0$.

Théorème 2 *(**Théorème de dualité**) Soient une matrice A et deux vecteurs b et c. Alors:*

$$max\{cx \mid A.x \leq 0\} = min\{yb \mid y \geq 0; y.A = c\}$$

dans le cas où les deux polyèdres sont non vides.

Le résultat du théorème de dualité reste valide si un seul des deux ensembles est non vide.

Corollaire 1 *(**Lemme de Farkas affine**) Soient une matrice A et deux vecteurs b et c. Supposons qu'il existe un vecteur x tel que $A.x \leq b$ et ($A.x \leq b \Rightarrow cx \leq \delta$). Il existe alors $y \geq 0$ tel que $y.A = c$ et $yb = \delta' \leq \delta$.*

Théorème 3 *(Complementary slackness)*
Soient A une matrice, b un vecteur colonne et c un vecteur ligne. Considérons l'équation de dualité entre programmes linéaires suivante:

$$max\{cx \mid \ Ax \leq b\} = min\{yb \mid \ y \geq 0, yA = c\}$$

Supposons que les deux optima soient finis, alors pour chaque inégalité $a_i x \leq b_i$ dans $Ax \leq b$, exactement une des deux propriétés suivantes est vérifiée:

- *Le problème de maximisation admet une solution optimale x_0 avec $a_i x_0 < b_i$*
- *Le problème de minimisation admet une solution optimale y_0 de $i^{ème}$ composante strictement positive.*

2.3 Equations Récurrentes et Nids de boucles

Dans la suite, nous nous intéressons à deux structures de programmes, les systèmes d'équations récurrentes et les nids de boucles. La première, plus générale, sera l'objet de l'étude théorique du chapitre 4. Les principaux résultats de cette étude et les algorithmes qui y sont développés seront ensuite appliqués au cas des nids de boucles affines au chapitre 5.

2.3.1 Equations récurrentes

Définition 9 *Un système d'**équations récurrentes uniformes** est un ensemble d'équations de la forme:*

$$V_i(z) = f_i(V_{i_1}(z - d_{i_1,i}), \dots, V_{i_{m_i}}(z - d_{i_{m_i},i}))$$

où V_i désigne une variable définie sur les points entiers d'un polyèdre D_i inclus dans Z^n (z et $d_{i,j}$ sont des vecteurs de Z^n) et f_i est une fonction stricte à m_i arguments.

On dit que le calcul de la variable V_i au point z **dépend** des valeurs des variables V_{i_1}, ..., $V_{i_{m_i}}$ aux points $(z - d_{i_1,i})$, ..., $(z - d_{i_{m_i},i})$. D'autre part, puisque cette dépendance se traduit par un accès à "distance" constante, on dit qu'on a affaire à une dépendance **uniforme**. Si au contraire les vecteurs $d_{i,j}$ ne sont pas des constantes mais des fonctions affines en z ($d_{i,j}(z)$), la dépendance est dite **affine** et le système est un système d'**équations récurrentes affines**.

Les vecteurs $d_{i,j}$ sont appelés **vecteurs de dépendance**. En fait, dans la suite, les fonction f_i importent peu mais c'est le schéma d'utilisation des données induit par ces équations qui est important. On représente alors les calculs associés au système d'équations par un graphe, appelé graphe réduit, cyclique dans le cas général, défini comme suit:

Définition 10 *Le **graphe réduit** associé au système d'équations*

$$V_i(z) = f_i(V_{i_1}(z - d_{i_1,i}), \dots, V_{i_{m_i}}(z - d_{i_{m_i},i})) \ 1 \leq i \leq p$$

est le graphe orienté possédant:

- *p sommets s_i valués par les domaines D_i.*
- *une arête orientée de s_i vers s_j si $V_j(z) = f_j(\dots, V_i(z - d_{i,j}), \dots)$ et valuée par le vecteur $d_{i,j}$.*

Le qualificatif de "réduit" rappelle qu'en fait le graphe de dépendance complet est un graphe dont les sommets sont des points de $\mathbb{Z}^n$ et que le graphe réduit n'en est qu'une représentation condensée, représentation possible du fait de l'uniformité des dépendances.

2.3.2 Nids de boucles

Définition 11 *On appelle **nid de boucles parfait** la structure de programme suivante:*

```
DO x_1 = l_1(z) to u_1(z)
   DO x_2 = l_2(x_1, z) to u_2(x_1, z)
      ...
      DO x_n = l_n(x_1, x_2, ..., x_{n-1}, z) to u_n(x_1, x_2, ..., x_{n-1}, z)
      Program P:
         {Instruction 1}
         {Instruction 2}
              ⋮
         {Instruction k}
      ENDDO
      ...
   ENDDO
ENDDO
```

où $\vec{z}$ est le vecteur des paramètres de structure et où les bornes des boucles (l_i et u_i) sont des fonctions affines des indices des boucles englobantes et des paramètres.

Suivant le type des dépendances entre instructions au sein du nid de boucles, il peut être qualifié d'uniforme ou d'affine.

Un **nid de boucles** est dit **non-parfait** quand ses boucles constitutives ne sont pas parfaitement imbriquées. Exemple:

DO $x_1 = l_1(\vec{z})$ to $u_1(\vec{z})$
    DO $x_2 = l_2(x_1, \vec{z})$ to $u_2(x_1, \vec{z})$
        DO $x_3 = l_3(x_1, x_2, \vec{z})$ to $u_3(x_1, x_2, \vec{z})$
            {Instruction 1}
        ENDDO
        DO $x_4 = l_4(x_1, x_2, \vec{z})$ to $u_4(x_1, x_2, \vec{z})$
            {Instruction 2}
            {Instruction 3}
        ENDDO
    ENDDO
ENDDO

On peut définir un graphe de dépendance réduit pour les nids de boucles comme pour les équations récurrentes. Les sommets sont alors les instructions et non plus les variables.

La différence fondamentale entre les équations récurrentes et les nids de boucles se situe au niveau des vecteurs de dépendance qui sont lexicographiquement positifs pour les nids de boucles.

2.4 Ordonnancement

Les ordonnancements monodimensionnels (extension de la méthode de Lamport) sont relativement bien connus: existence, méthodes d'obtention, ...même s'il reste quelques questions ouvertes concernant leur optimalité (cf. chapitre 3). Ces ordonnancements correspondent en terme de parallélisation de boucles à la possibilité de réécrire les nids avec une seule boucle externe séquentielle.

Lorsqu'il n'existe pas d'ordonnancement monodimensionnel, c'est-à-dire lorsqu'il existe moins de parallélisme, on peut décrire l'imbrication de plusieurs boucles séquentielles au moyen d'un ordonnancement dit multidimensionnel. C'est l'objet principal de ce rapport.

2.4.1 Ordonnancement monodimensionnel

Définition 12 *On appelle **ordonnancement** une fonction qui à tout point du domaine associe une "date d'exécution" sous la forme d'un entier relatif, compatible avec les relations de dépendance.*

Une fonction t à valeurs dans $\mathbb{Z}$ est un ordonnancement si, et seulement si, pour tous points p et q, $t(p) \geq t(q) + 1$ si p dépend de q.

On étend la notion d'ordonnancement aux fonctions à valeurs dans $\mathbb{Q}$. De la même façon, on dira qu'une fonction t à valeurs dans $\mathbb{Q}$ est un ordonnancement si, et seulement si, pour tous points p et q, $t(p) \geq t(q) + 1$ si p dépend de q.

Lemme 1 *La partie entière d'un ordonnancement dans $\mathbb{Q}$ fournit un ordonnancement dans $\mathbb{Z}$.*

Preuve: Soient deux points p et q, p dépendant de q, et t un ordonnancement dans $\mathbb{Q}$.

$$t(p) \geq t(q) + 1 \Rightarrow \lfloor t(p) \rfloor \geq \lfloor t(q) + 1 \rfloor = \lfloor t(q) \rfloor + 1$$

Dans toute la suite, on ne s'intéressera qu'aux ordonnancements à valeurs dans $\mathbb{Q}$ plus simples à manipuler.

Définition 13 *On appelle **Ordonnancement au plus tôt** ou **ordonnancement libre** l'ordonnancement pour lequel chaque tâche est exécutée dès que possible. La date d'exécution d'une tâche avec cet ordonnancement est alors égale au plus long chemin de dépendance qui y aboutit.*

Définition 14 *On appelle **ordonnancement linéaire** un ordonnancement défini par une forme linéaire rationnelle:*

$$t(x) = \lfloor Xx \rfloor \textit{ pour } x \in D$$

où X est le vecteur de temps, vecteur à coordonnées rationnelles, et D est l'espace d'indiçage.

Un vecteur X satisfaisant le système d'inéquations $Xd \geq 1$ pour tout vecteur de dépendance d définit un ordonnancement linéaire.

Définition 15 *On appelle **ordonnancement affine** un ordonnancement dont la restriction à chaque sommet i du graphe réduit est une forme affine:*

$$p \mapsto X_i.p + c_i$$

Des vecteurs X_i et des constantes c_i satisfaisant le système d'inéquations

$$X_i p + X_i d_{j,i} + c_i \geq 1 + X_j p + c_j$$

pour tout vecteur de dépendance $d_{j,i}$ définissent un ordonnancement affine.

L' **ordonnancement affine** est dit **de partie linéaire unique** si la partie linéaire X_i est la même pour tous les sommets. Le système d'inéquations précédent ce réduit alors à:

$$X d_{j,i} + c_i \geq 1 + c_j$$

Définition 16 *On appelle **hyperplan faiblement séparateur** du graphe G un vecteur X tel qu'il existe un sous-graphe H de G et des rationnels ρ_i indicés par les sommets de H vérifiants:*

- *si* $e_i \in G - H \;\; w(e_i).x \geq \rho_l - \rho_k$
- *si* $e_i \in H \quad w(e_i).x \geq 1 + \rho_l - \rho_k$

Si $H = G$ l'hyperplan est un **hyperplan séparateur**, c'est le cas du vecteur de partie linéaire pour les ordonnancements affines valides de partie linéaire unique.

2.4.2 Ordonnancement multidimensionnel

Définition 17 *On appelle **ordonnancement multidimensionnel** [Fea92] une fonction qui associe à chaque couple (sommet, point du domaine) un vecteur de temps entier relatif compatible avec les relations de dépendance.*

La compatibilité avec les relations de dépendance s'exprime cette fois-ci par l'extension de l'ordre classique $<$ à un ordre multidimensionnel: l'ordre lexicographique (strict) $\gg_l$.

On définit $\gg_l$ par récurrence sur la dimension comme suit:

- $n = 1$: $(x_1) \gg_l (y_1)$ si, et seulement si $x_1 \geq y_1 + 1$
- $n > 1$: $(x_1, x_2, \ldots, x_n) \gg_l (y_1, y_2, \ldots, y_n)$ si, et seulement si,

$$\begin{cases} x_1 \geq y_1 + 1 \text{ ou} \\ x_1 = y_1 \text{ et } (x_2, \ldots, x_n) \gg_l (y_2, \ldots, y_n) \end{cases}$$

On étend la notion d'ordonnancement multidimensionnel aux fonctions à valeurs dans $\mathbb{Q}^n$. De la même façon, on dira qu'une fonction t à valeurs dans $\mathbb{Q}^n$ est un ordonnancement multidimensionnel si, et seulement si, pour tous points p et q, $t(p) \gg_l t(q)$ si p dépend de q.

Il est à noter que $\gg_l$ n'est pas un ordre total dans $\mathbb{Q}^n$, mais cette définition sera suffisante pour la suite.

Chapitre 3

Ordonnancement linéaire et ordonnancement affine de partie linéaire unique

On va s'intéresser ici aux nids de boucles uniformes et plus généralement aux équations récurrentes uniformes pour lesquels il existe un hyperplan séparateur.

3.1 Principaux résultats

Les résultats présentés ici le sont de manière schématique. Pour de plus amples informations et pour des démonstrations rigoureuses se référer à la thèse d'Alain Darte [Dar93].

3.2 Nids de boucles uniformes à une seule instruction

Dans le cas des nids de boucles uniformes les vecteurs de dépendances sont tous lexicographiquement positifs et il est aisé de construire un hyperplan séparateur. La méthode historique d'obtention d'un tel hyperplan est due au fameux Leslie Lamport [Lam74]. Trouver un hyperplan séparateur, c'est-à-dire un ordonnancement linéaire revient à trouver un vecteur d'ordonnancement X satisfaisant

$$XD \geq 1$$

où D est la matrice de dépendance (ou matrice de déplacement), c'est-à-dire la matrice dont les vecteurs colonne sont les différents vecteurs de dépendance.

On obtient l'ordonnancement linéaire de latence minimale sur le domaine $Ap \leq b$ par résolution du programme linéaire suivant:

$$\left\{\begin{array}{l} X_1.A = \quad X \\ X_2.A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ X.D \geq 1 \\ \min(X_1 + X_2).b \end{array}\right. \tag{3.1}$$

Dans le cas des nids de boucles uniformes à une seule instruction, l'ordonnancement linéaire est asymptotiquement optimal comme mes révérés directeurs de stage l'ont prouvé en association avec Leonid Khachiyan [DKR91]. Plus précisément, l'optimalité se traduit par la limite:

$$\lim_{N\to\infty} \frac{\mathrm{L}(N)}{\mathrm{F}(N)} = 1$$

sur une famille de domaines homothétiques de facteur d'homothétie N. $\mathrm{L}(N)$ est la latence du meilleur ordonnancement linéaire et $\mathrm{F}(N)$ celle de l'ordonnancement au plus tôt, c'est-à-dire la longueur du plus long chemin de dépendance.

En fait, on montre même:

$$\frac{\mathrm{L}(N)}{\mathrm{F}(N)} = 1 + O(\frac{1}{N}) \text{ et } \exists K \quad |\mathrm{L}(N) - \mathrm{F}(N)| \leq K$$

Autrement dit, la différence entre l'ordonnancement au plus tôt et l'ordonnancement linéaire optimal est bornée par une constante indépendante de la taille du domaine.

Nous aimerions établir un résultat similaire pour le cas des graphes uniformes fortement connexes à plusieurs sommets. Comme nous allons le montrer, la différence peut ne pas être bornée. Nous allons exhiber deux exemples, l'un à différence en $\log(N)$, l'autre en $\sqrt(N)$. Il semble néanmoins que la différence soit toujours en $o(N)$.

3.2.1 Graphes uniformes fortement connexes à plusieurs sommets

L'extension naturelle de l'ordonnancement linéaire pour le cas d'un graphe de dépendance uniforme à plusieurs sommets est l'ordonnancement affine de partie linéaire unique: un unique vecteur de temps X et une constante c_i par sommet. L'obtention d'un tel ordonnancement est similaire à celle d'un ordonnancement linéaire, elle se fait au moyen du programme linéaire suivant:

$$\left\{ \begin{array}{l} X_1 A = X \\ X_2 A = -X \\ X_1 \geq 0 \\ X_2 \geq 0 \\ \\ \forall (i,j) \text{ tel que } d_{i,j} \text{ existe} \\ X d_{i,j} + c_j - c_i \geq 1 \\ \min\,(X_1 + X_2) b \end{array} \right.$$

Dans ce programme linéaire, les contraintes de dépendances se traduisent par une inégalité par arête du graphe de dépendance. On peut en fait se ramener à une seule inégalité par cycle du graphe (par une méthode similaire à la méthode de resynchronisation de Leiserson et Saxe [LS83]). Dans ce cas, trouver l'ordonnancement affine de partie linéaire unique optimal revient à chercher l'ordonnancement linéaire optimal sur un graphe à un seul sommet et d'autodépendances les poids des cycles du graphe initial divisés par leur longueur.

On pourrait donc croire que des arguments analogues à ceux employés pour le cas à un seul sommet permettraient de démontrer que l'ordonnancement affine de partie linéaire unique est asymptotiquement optimal, au même sens que précédemment, mais pour les graphes fortement connexes à plusieurs sommets. En fait, il n'en est malheureusement rien comme nous allons le montrer dans la partie suivante.

3.2.2 Obstacles à la généralisation

Rappelons tout d'abord le principe général de la démonstration de l'optimalité de l'ordonnancement linéaire dans le cas d'un seul sommet.

Le programme linéaire 3.1 admet pour dual:

$$\begin{cases} Y \geq 0 \\ Ap \leq b \\ Aq \leq b \\ DY = p - q \\ \max \sum_{i=1}^{n} y_i \end{cases}$$

Ces deux programmes linéaires étant duaux, ils ont même solution optimale, égale donc à la fois à la latence de l'ordonnancement linéaire optimal et à la longueur d'un "chemin" de dépendance de q à p utilisant y_i fois le vecteur d_i. Le problème d'arrondir les valeurs p, q, y_i a priori rationnelles en des entiers n'est pas le problème majeur. La difficulté est d'exhiber un vrai chemin de dépendance entre q et p, c'est-à-dire une chaîne de points appartenant au domaine et correspondant à l'égalité vectorielle $DY = p - q$. L'idée pour y parvenir est d'approcher au mieux le segment de droite reliant p à q, et donc d'utiliser à chaque étape le vecteur d_i qui nous éloigne le moins de ce segment. Ce principe aboutit car les vecteurs de dépendance sont tous des vecteurs d'autodépendance issus d'un même sommet et qu'ils peuvent donc être utilisés dans un ordre arbitraire.

Dans le cas d'un graphe de dépendance à plusieurs sommets fortement connexe, il existe un cycle qui permet de relier tous les sommets entre eux. L'utilisation d'un tel cycle permettrait de simuler la construction précédente (sans oublier que l'utilisation d'un d_i est en fait ici l'utilisation d'un cycle), mais elle a l'inconvénient d'entraîner un surcoût qui peut a priori être linéaire en N. Sans rentrer dans les détails, disons simplement qu'une utilisation linéaire fait avorter la démonstration. Il faut minimiser le nombre d'utilisations artificielles de ce cycle, c'est-à-dire minimiser le nombre de changement de vecteur d_i.

Contrairement au cas à un seul sommet où il suffit d'approcher le segment de droite $[p,q]$, il faut ici construire un chemin qui rebondit sur les parois du domaine. Le nombre de rebonds est de l'ordre du surcoût dû aux utilisations artificielles d'arêtes de liaison et est également de l'ordre de la différence entre la latence de l'ordonnancement au plus tôt et celle de l'ordonnancement affine de partie linéaire unique optimal.

Ainsi, nous allons construire deux exemples, l'un pour lequel le plus long chemin de dépendance rebondit un nombre logarithmique de fois, l'autre pour lequel il rebondit plus de $\sqrt(N)$ fois.

Donc, il n'est pas vrai que:

$$\exists K \quad |\mathrm{AL}(N) - \mathrm{F}(N)| \leq K$$

mais nous n'avons pas de contre-exemple à:

$$\lim_{N \to \infty} \frac{\mathrm{AL}(N)}{\mathrm{F}(N)} = 1$$

où $\mathrm{AL}(N)$ est la latence du meilleur ordonnancement affine de partie linéaire unique.

Conjecture 1 *Si le graphe de dépendance est fortement connexe, le théorème d'optimalité asymptotique est vrai pour les ordonnancements affines de partie linéaire unique.*

3.3 Exemples

Afin de mieux cerner le problème de l'optimalité voici deux exemples, le premier pour lequel la différence entre l'ordonnancement au plus tôt et l'ordonnancement linéaire est logarithmique en la taille du domaine, et le deuxième où elle est en la racine carrée de cette même taille.

3.3.1 Exemple à différence logarithmique

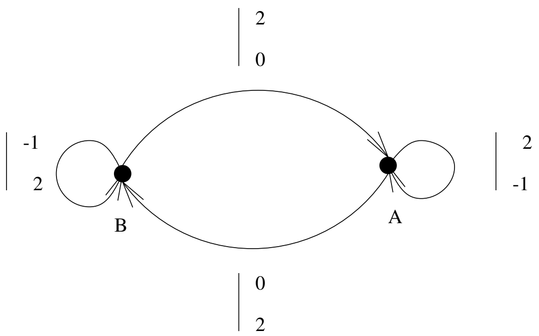

FIG. 3.1 - *Exemple à différence logarithmique*

Système d'équations récurrentes:

$$\begin{cases} a(i,j) = a(i-2, j+1) + b(i-2, j) \\ b(i,j) = b(i+1, j-2) + a(i, j-2) \end{cases}$$

Domaine:

$$\begin{cases} 0 \leq i \leq N \\ 0 \leq j \leq N \end{cases}$$

Nous allons étudier l'exemple de la figure 3.1 et prouver que dans ce cas la différence entre l'ordonnancement au plus tôt et l'ordonnancement linéaire est en $O(log(N))$, N étant la taille du domaine supposé carré.

Soit $\mathcal{D}$ la longueur du plus long chemin de dépendance, et soit $\mathcal{C}$ un chemin de dépendance de taille $\mathcal{D}$. On note n_1 (resp. n_2) le nombre d'utilisation du vecteur $L_1 = (-1, 2)$ (resp. $L_2 = (2, -1)$) dans $\mathcal{C}$, et μ la somme des nombres d'utilisations des vecteurs $(2, 0)$ et $(0, 2)$.

Il sera montré successivement que:

i. il existe un chemin de dépendance de longueur au moins $2 * N - k * log(N) + c$, où c est une constante;

ii. $\mu = o(N)$, c'est-à-dire que le nombre d'utilisations des arêtes reliant A et B est sublinéaire;

iii. $(n_1 + n_2) \sim 2 * N$, le chemin de dépendance n'utilise quasiment que les autodépendances;

iv. le chemin avec le minimum de changement de vecteur pour le graphe de la figure 3.2 est celui qui rebondit sur les parois du domaine;

v. le chemin du graphe de la figure 3.2 qui rebondit sur les parois du domaine a au moins $O(log(N))$ rebonds;

vi. $\mu \geq O(log(N))$, le nombre d'utilisations des arêtes reliant A et B est en $O(log(N))$;

vii. $\exists K > 0 \ | \ \mathcal{D} \leq 2 * N - K * log(N)$, autrement dit: la différence est logarithmique!

FIG. 3.2 - *Exemple à différence logarithmique simplifié*

Chemin de dépendance en $2 * N - k * log(N)$

Le principe de construction de ce chemin est l'utilisation maximale de la boucle d'auto-dépendance du sommet courant.

Première partie du chemin: Dans cette première partie, le chemin rebondit sur les plans $x = 0$ et $y = 0$. On définit le chemin grâce aux points $P_n = (\lambda_n, 0)$:

$(\lambda_n, 0) \xrightarrow{(0,2)} (\lambda_n, 2) \xrightarrow{\lambda_n*(-1,2)} (0, 2 * \lambda_n + 2) \xrightarrow{(2,0)} (2, 2 * \lambda_n + 2) \xrightarrow{(2*\lambda_n+2).(2,-1)} (4 * \lambda_n + 6, 0)$

D'où $\lambda_{n+1} = 4 * \lambda_n + 6$ et $\lambda_n = (2 + \lambda_0) * 4^n - 2$ pour $\lambda_n \leq N$.

Soit l_n la longueur du chemin entre les points P_n et P_{n+1}, $l_n = 3 * \lambda_n + 4$. Soit p l'indice du dernier point de la forme P_n, c'est-à-dire, soit p tel que $P_p \leq N < P_{p+1}$ et soit L la longueur du chemin entre les points P_0 et P_p, alors

$$L = -2 * p + (2 + \lambda_0) * (4^p - 1)$$

.

Deuxième partie du chemin: Dans cette deuxième partie, le chemin rebondit sur les plans $x = N$ (ou $x = N - 1$)[1] et $y = N$ (ou $y = N - 1$). On définit maintenant le chemin grâce aux points $Q_n = (N, \mu_n)$

$(N - \epsilon_n, \mu_n) \xrightarrow{(0,2)} (N - \epsilon_n, \mu_n + 2) \xrightarrow{\alpha_n*(-1,2)} (N - \alpha_n - \epsilon_n, y) \xrightarrow{(2,0)} (N - \alpha_n + 2 - \epsilon_n, y) \xrightarrow{\beta_n*(2,-1)} (N - \epsilon_{n+1}, y - \beta_n)$

où $\alpha_n = \lfloor N/2 - \mu_n/2 - 1 - \epsilon_n/2 \rfloor$, $\beta_n = \lfloor \alpha_n/2 - 1 + \epsilon_n/2 \rfloor$, $y = \mu_n + 2 + 2 * \alpha_n$, $\epsilon_n \in \{0, 1\}$ et $N - \epsilon_{n+1} = N - \alpha_n + 2 + 2 * \beta_n - \epsilon_n$

Contrairement à la première partie, nous ne pouvons faire de calculs exacts que sur un encadrement du chemin.

D'où $3/4 * N + \mu_n/4 + 1/4 \leq \mu_{n+1} \leq 3/4 * N + \mu_n/4 + 3$ et $(N + 1/3)(1 - 1/4^n) + \mu_0/4^n \leq \mu_n \leq (N + 4)(1 - 1/4^n) + \mu_0/4^n$.

Soit l'_n la longueur du chemin entre les points Q_n et Q_{n+1}, $l'_n = \alpha_n + \beta_n + 2$, donc $3/4N - 3/4 * \mu_n - 13/4 \leq l'_n \leq 3/4N - 3/4 * \mu_n - 1/2$. Soit q l'indice du dernier point de la forme Q_n, c'est-à-dire, soit q tel que $Q_p \geq N - 1 > Q_{q+1}$ et soit L' la longueur du chemin entre les points Q_0 et Q_q, alors

$$(N + 4 - \mu_0) * (1 - 1/4^q) - 25/4 * q \leq L' \leq (N + 1/3 - \mu_0) * (1 - 1/4^q) - 3/4 * q$$

1. à cause des problèmes d'arrondis

Partie intermédiaire: Les deux parties précédemment décrites sont reliées par le chemin suivant:

$(0, \lambda_q) \xrightarrow{(2,0)} (2, \lambda_q)(2, \lambda_q) \xrightarrow{\lfloor N/2-1 \rfloor} (N + \epsilon, \lambda_q - \lfloor N/2 - 1 \rfloor)$

On pose alors, $\mu_0 = \lambda_q - \lfloor N/2 - 1 \rfloor$ et $\epsilon_0 = \epsilon$.

Longueur totale du chemin Soit $\mathcal{L}$ cette longueur, $\mathcal{L} = L + L' + 1 + \lfloor N/2 - 1 \rfloor$, soit
$2 * N - 3/2 * N/4^q - 25/4 * q - 2 * p - (2 + \lambda_0) * (1 - 4^p/4^q) + 1 - (3/2 * N + 3)/4^q \leq \mathcal{L} \leq 2 * N - 3/2 * N/4^q - 3/4 * q - 2 * p + 7/3(1 - 1/4^q) - (2 + \lambda_0) * (1 - 4^p/4^q)$

Or, par définition de p et de q, on a $p = (log(N)))$ et $q = (log(N)))$. De l'encadrement $N \geq \mu_q \geq N - 1$ on tire que les rapports $N/4^q$ et $4^p/4^q$ sont bornés. Par conséquent, il existe deux constantes strictement positives k_1 et k_2, et deux autres constantes quelconques c_1 et c_2 telles que

$$2 * N - k_1 * log(N) + c_1 \leq \mathcal{L} \leq 2 * N - k_2 * log(N) + c_2$$

$\mu = o(N)$ **et** $(n_1 + n_2) \sim (2 * N)$

Par définition, $\mathcal{D} = n_1 + n_2 + \mu$. $n_1 + n_2 + 2 * \mu \leq 2 * N$, le vecteur $(1, 1)$ définissant un ordonnancement linéaire valide pour le graphe étudié.

Montrons $\mu = o(N)$ par l'absurde; supposons pour cela: $\exists k \mid \forall N_0 \ \ \exists N \geq N_0 \mu_N \geq k * N$, d'où $\mathcal{D}_N = n_1 + n_2 + 2 * \mu - \mu \leq (2 - k) * N$. Or $\mathcal{L} \geq 2 * N - k_1 * log(N) + c_1$[2]. Soit N_0 tel que $k_1 * log(N_0) - c_1 < k * N_0$, alors pour tout N, $N \geq N_0 \Rightarrow k_1 * log(N) - c_1 < k * N$ et donc $\mathcal{L}_N > \mathcal{D}_N$ ce qui est absurde par maximalité de D_N.

La négation de l'hypothèse étant $\forall k \mid \exists N_0 \forall N \geq N_0 \mu_N < k * N$, on a le résultat escompté.

Comme $2 * N - k_1 * log(N) + c_1 \leq D_N \leq 2 * N$, on déduit du résultat précédent que $n_1 + n_2$ est équivalent à $2 * n$.

Chemin maximal avec minimum de changement de vecteur du graphe simplifié

Le chemin Le chemin de longueur maximale, minimal pour le nombre de changement de vecteurs du graphe simplifié, est celui pour lequel on ne change de vecteur utilisé que lorsque le bord du domaine est atteint, d'où l'expression *chemin qui rebondit.*

En effet, supposons que l'on ait un chemin CM, minimal pour le nombre de changement de vecteur, pour lequel on change à un instant donné de vecteur sans rebondir, par exemple qui utilise L_1 pour atteindre un point $p = (x, y)$, puis n fois L_2 (donc $(0, 0) \leq p + n * (2, -1) \leq (N, N)$) alors que L_1 pouvait encore être utilisé (soit $(0, 0) \leq p + (-1, 2) \leq (N, N)$). On peut alors construire un chemin CM' identique à CM jusqu'au point p mais qui utilise alors une fois de plus L_1 puis n fois L_2 avant de recoincider avec CM.

Preuve:

$((0, 0) \leq p + (-1, 2) \leq (N, N)) \Rightarrow (y + 2 \leq N) \Rightarrow (y + 2 - n \leq N)$
$((0, 0) \leq p + (-1, 2) \leq (N, N)) \Rightarrow (0 \leq x - 1) \Rightarrow (0 \leq x + 2 * n - 1)$
$((0, 0) \leq p + n * (2, -1) \leq (N, N)) \Rightarrow (x + 2 * n \leq N) \Rightarrow (x + 2 * n - 1 \leq N)$
$((0, 0) \leq p + n * (2, -1) \leq (N, N)) \Rightarrow (0 \leq y - n) \Rightarrow (0 \leq y - n + 2)$

Ce chemin CM' est de même longueur que CM. En appliquant cette transformation à un chemin maximal à partir du premier changement de vecteur "hâtif" jusqu'au dernier, on obtient un chemin *qui rebondit.*

2. en reprenant les notations de la partie précédente

Le graphe simplifié Nous allons maintenant calculer la longueur du *chemin qui rebondit* du graphe simplifié et montrer qu'elle est en $2 * N' - k * log(N')$, où N' est la taille du domaine considéré. Comme μ est supérieur au nombre de changements de vecteur dans $\mathcal{C}$ à deux unités près, que ce nombre de changements de vecteur dans $\mathcal{C}$ est supérieur au nombre de rebonds dans le *chemin qui rebondit* dans un domaine de taille $N' = N - \mu$ note[3], si nous montrons que le nombre de rebonds est en $O(log(N'))$, nous aurons montré que $\mu \geq O(log(N))$ car $(\mu = o(N)) \Rightarrow (N \sim N')$

Chemin qui rebondit du graphe simplifié

On note l_{2i} le nombre d'utilisations du vecteur $L_2 = (2, -1)$ entre les $(2i-1)^{ème}$ et $2i^{ème}$ rebonds et l_{2i+1} le nombre d'utilisations du vecteur $L_1 = (-1, 2)$ entre les $2i^{ème}$ et $(2i+1)^{ème}$ rebonds. On note en outre (x_i, y_i) les coordonnées du $i^{ème}$ rebond, d'où le système:

$$\left\{ \begin{array}{rcccc} x_{2i} & = & x_{2i-1} & + & 2 * l_{2i} \\ y_{2i} & = & y_{2i-1} & - & l_{2i} \\ x_{2i+1} & = & x_{2i} & - & l_{2i+1} \\ y_{2i+1} & = & y_{2i} & + & 2 * l_{2i+1} \end{array} \right.$$

On cherche à montrer que ce nombre de rebonds est en $O(log(N'))$, N' étant la taille du domaine. Les quantités l_i doivent être entière, c'est pourquoi les rebonds ont en fait lieu sur l'un des huit plans suivants: $x = 0$, $x = 1$, $x = N'$, $x = N' - 1$, $y = 0$, $y = 1$, $y = N'$, $y = N' - 1$, l'imprécision se traduisant dans les équations ci-dessous pas l'introduction de la quantité δ_i.

Soit $p \in \mathbb{N}$ tel que (x_p, y_p), soit les coordonnées du dernier rebond pour lequel on a $x * y = 0$. Alors, $\forall i \mid 2i + 1 \leq p$, $x_{2i+1} = \delta_{2i+1}$ et $\forall i \mid 2i \leq p$, $y_{2i} = \delta_{2i}$, où $\delta_i \in \{0, 1\}$.

D'où $l_{2i-1} = ((\delta_{2i} - \delta_{2i-2}) + l_{2i})/2$ si $2i \leq p$ et $(2i - 2) \leq p$ et $l_{2i} = ((\delta_{2i+1} - \delta_{2i-1}) + l_{2i+1})/2$ si $(2i - 1) \leq p$ et $(2i + 1) \leq p$. Par conséquent, $l_i \leq (1 + l_{i+1})/2$ si $i \leq p - 1$, et donc $l_i \leq 1 + l_p/2^{p-i}$ si $i \leq p - 1$.

La longueur de la première partie du chemin (jusqu'au $(p-1)^{ème}$ rebond) peut alors être majorée comme suit:

$$\sum_{i=0}^{p-1} l_i \quad \leq \quad p + (1 - 1/2^p) * l_p$$

Soit $q \in \mathbb{N}$ tel que (x_q, y_q), soit les coordonnées d'un rebond pour lequel on a $x_q \geq (N' - 1)$ ou $y_q \geq (N' - 1)$, alors pour tout $i \geq q$ on aura $x_i \geq (N' - 1)$ ou $y_i \geq (N' - 1)$. Alors, $\forall i \mid 2i \geq q$, $x_{2i} = N' - \delta_{2i}$ et $\forall i \mid 2i + 1 \geq q$, $y_{2i+1} = N' - \delta_{2i+1}$, où $\delta_i \in \{0, 1\}$.

De même que précédemment, on montre que pour tout i supérieur à $(q+1)$, on a $l_{i+1} \leq (1+l_i)/2$ puis, $l_{i+1} \leq 1 + l_{q+1}/2^{i-q}$.

La longueur de la deuxième partie du chemin (à partir du $(q+1)^{ème}$ rebond et jusqu'au $(r)^{ème}$ et dernier) peut alors être majorée comme suit:

$$\sum_{i=q+1}^{r} l_i \quad \leq \quad r - q + (1 - 1/2^{r-q}) * l_{q+1}$$

Longueur du chemin entier: Le plus petit entier q vérifiant les hypothèses énoncées plus haut vaut, suivant les cas, p ou $p + 1$, on fixe donc q à $p + 1$.

3. On pourrait affiner l'approximation en passant en $N' = N - \mu/2 + \epsilon$

Alors $\sum_{i=0}^{r} l_i = \sum_{i=0}^{p-1} l_i + l_p + l_{p+1} + \sum_{i=p+2}^{r} l_i$ et donc $\sum_{i=0}^{r} l_i \leq p + (1 - 1/2^p) * l_p + l_p + l_{p+1} + r - p - 1 + (1 - 1/2^{r-p-1}) * l_{p+2}$, soit

$$\sum_{i=0}^{r} l_i \leq r - 1 + 2 * l_p + l_{p+1} + l_{p+2} - l_p/2^p - l_{p+2}/2^{r-p-1}$$

Evaluons $2 * l_p + l_{p+1} + l_{p+2}$. Par définition, l_p, l_{p+1} et l_{p+2} sont tels qu'il existe x, y, x', et y' quatre points du domaine, et δ, δ', δ'',et δ''' éléments de $\{0, 1\}$ tels que:

- $(x, \delta) + l_p.(-1, 2) = (\delta', y)$ où $y = \delta + 2 * (x - \delta')$
- $(\delta', y) + l_{p+1}.(2, -1) = (N' - \delta'', y')$ où $y' = y - 1/2 * (N' - \delta' - \delta'')$
- $(N' - \delta'', y') + l_{p+2}.(-1, 2) = (x', N' - \delta''')$ où $x' = N' - \delta'' - 1/2 * (N' - \delta''' - y')$

D'où $l_p \leq x + 1$, $l_{p+1} \leq N'/2$ et $l_{p+2} \leq 3/4 * N' - x + 3/2$, et donc, $2 * l_p + l_{p+1} + l_{p+2} \leq 7/2 + 5/4 * N' + x \leq 7/2 + 2 * N'$ comme $x \in [N'/4, 3N'/4]$ (domaine de validité des équations).

En conclusion [4]:

$$\sum_{i=0}^{r} l_i \leq 2 * N' + 5/2 + r - 2 * N'/2^r$$

D'autre part, $\sum_{i=0}^{r} l_i = 2 * N - f(N) - \mu$ avec $f(N) = o(N)$ et $\mu \geq r - 2$, et $N' = N - g(N)$ avec $g(N) = o(N)$ d'où
$2 * N - f(N) - \mu \leq 2 * N - g(N) + 9/2 - 2 * N'/2^\mu + \mu$. Par conséquent,

$$\mu \geq \log_2(N') + 1 - ln(9/2 + 2 * \mu + f(N) - g(N))$$

or nous avons vu que $\mu = o(N)$ et $N \sim N'$, donc,

$$\mu = O(log(N))$$

Conclusion

Comme $\mathcal{D} = n_1 + n_2 + \mu = n_1 + n_2 + 2 * \mu - \mu \leq 2 * N - \mu$ où $\mu = \Omega(log(N))$, donc

$$\exists K > 0 \quad | \quad \mathcal{D} \leq 2 * N - K * log(N)$$

Lemme 2 La différence entre l'ordonnancement linéaire (s'il existe) et l'ordonnancement au plus tôt peut-être logarithmique en la taille du domaine pour les equations récurrentes uniformes

3.3.2 Exemple à différence en racine carrée

En fait, il peut même être prouvé que:

Lemme 3 La différence entre l'ordonnancement linéaire (s'il existe) et l'ordonnancement au plus tôt peut-être en racine carrée de la taille du domaine pour les equations récurrentes uniformes

Preuve: Laissée au lecteur en guise d'exercice [5]...

Système d'équations récurrentes:

4. car $-2^p \leq -2^q$, $l_p \leq N'$ et $l_{p+2} \leq N'$
5. Indication: utiliser l'exemple présenté ci-dessous

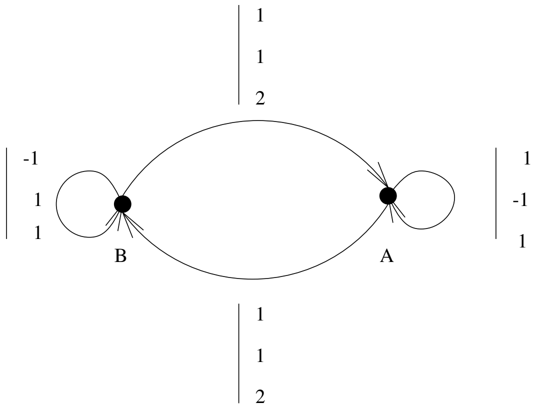

FIG. 3.3 - *Exemple à différence en racine carrée*

$$\begin{cases} a(i,j,k) = a(i-1,j+1,k-1) + b(i-1,j-1,k-2) \\ b(i,j,k) = b(i+1,j-1,k-1) + a(i-1,j-1,k-2) \end{cases}$$

Domaine:

$$\begin{cases} 0 \leq i \leq N \\ 0 \leq j \leq N \\ 0 \leq k \leq N \\ 0 \leq i+j+k \leq N \end{cases}$$

Le problème provient ici du fait que pour $(i+j)$ petit, les deux autodépendances ne peuvent être utilisées qu'un très petit nombre de fois avant que le chemin n'ait recours aux arêtes reliant A et B. D'autre part, l'utilisation de ces arêtes reliant A et B diminue irrévocablement le nombre d'utilisation total des arêtes d'autodépendance en rapprochant le chemin du "plafond" du domaine, le plan $i+j+k=N$.

Chapitre 4

Ordonnancement multi-dimensionnel

4.1 Insuffisance de l'ordonnancement linéaire

L'ordonnancement linéaire et l'ordonnancement affine de partie linéaire unique deviennent insuffisants lorsque l'on passe du cadre des nids de boucles uniformes à celui des Equations Récurrentes Uniformes.

Dans l'exemple présenté ci-dessous, il n'existe pas d'ordonnancement linéaire mais le système d'équations n'en est pas moins parallélisable puisqu'il possède un parallélisme d'ordre N. Nous verrons que l'absence d'un tel ordonnancement est caractéristique de la présence dans le graphe de dépendance réduit d'un multicycle de poids nul et que ce problème se raccroche donc directement à celui de la calculabilité des équations récurrentes uniformes.

L'étude des équations récurrentes uniformes est importante car intermédiaire entre celles des nids de boucles uniformes et celle des nids de boucles affines.

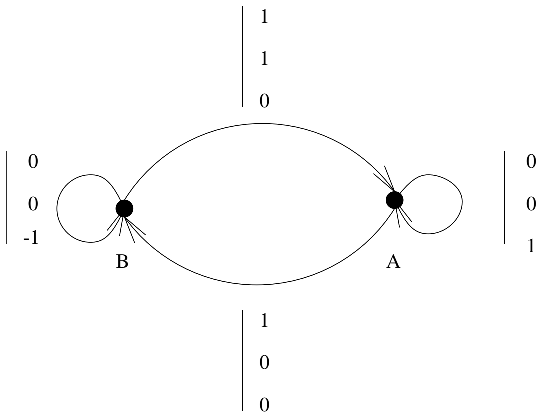

Fig. 4.1 - *Exemple sans ordonnancement linéaire*

Système d'équations récurrentes:

$$\begin{cases} a(i,j,k) = b(i-1,j-1,k) + a(i,j,k-1) \\ b(i,j,k) = a(i-1,j,k) + b(i,j,k+1) \end{cases}$$

Version parallélisée:

```
For i=0 to n Do
    For k = n to 0 Do
        Forall j=0 to n do
            b(i, j, k) = a(i-1, j, k) + b(i, j, k+1)
        Endforall
    Endfor
    For k = 0 to n Do
        Forall j=0 to n do
            a(i, j, k) = b(i-1, j-1, k) + a(i, j, k-1)
        Endforall
    Endfor
Endfor
```

4.2 Détection de cycles de poids nul dans les E.R.U.

Soit G le graphe de dépendance réduit d'un *système d'équations récurrentes uniformes*. L'espace d'indiçage étant supposé borné, les équations sont calculables si et seulement si le graphe de dépendance développé ne contient pas de cycle, c'est-à-dire, si et seulement si le graphe de dépendance réduit G ne contient pas de cycle de poids nul, quelle que soit la taille du domaine[1].

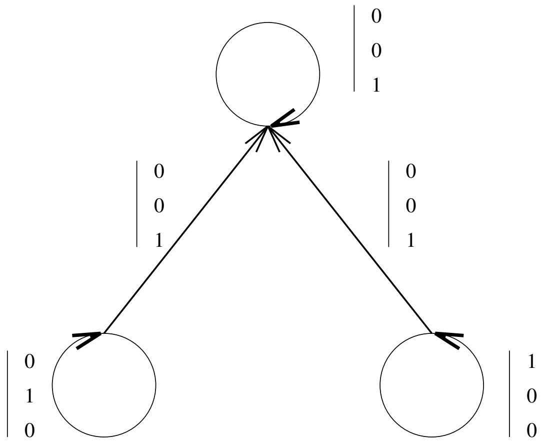

FIG. 4.2 - *Graphe de dépendance réduit de l'algorithme de multiplication de matrices ci-dessous*

Algorithme de multiplication de matrices:

```
for i=1 to N do
    for j=1 to N do
        for k=1 to N do
            a(i, j+1, k) = a(i, j, k)
            b(i+1, j, k) = b(i, j, k)
            c(i, j, k+1) = c(i, j, k)+a(i, j, k)*b(i, j, k)
        endfor
```

1. Un domaine trop petit peu faire disparaître artificiellement certains vecteurs de dépendance

endfor
endfor

4.2.1 Matrices de connexion et de déplacement

Soit C la *matrice de connexion* du graphe réduit, appelée aussi *matrice d'incidence*. Chacune de ses lignes correspond à un nœud de G et chacune de ses colonnes à un arc. Si le $j^{ème}$ arc de G est orienté du $i^{ème}$ nœud vers le $k^{ème}$, alors $c_{i,j} = -1$, $c_{k,j} = +1$ et pour $l \notin \{i, k\}$ $c_{l,j} = 0$, sauf si ce $j^{ème}$ arc est un arc d'autodépendance auquel cas pour tout l, $c_{l,j} = 0$

Soit D la *matrice de déplacement* du graphe réduit. Chacune de ses colonnes contient le vecteur de dépendance (ou de déplacement) de l'arc associé [2].

Exemple: Considérons l'exemple de la figure 4.2 contenant 3 nœuds et 5 arcs dont les poids sont éléments de $\mathbb{Z}^3$:

$$C = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} C \\ D \end{bmatrix}$$

S'il existe q un vecteur entier à composantes positives du noyau à droite de la matrice de connexion C (*i.e.* si $Cq = 0$), alors G contient un multicycle obtenu en prenant q_i fois le $i^{ème}$ arc de G (théorème sur les chemins eulériens). Si q appartient de plus au noyau à droite de la matrice de déplacement D (*i.e.* si $Bq = 0$), alors le graphe G contient un multicycle de poids nul. Si ce multicycle est en fait un cycle, le *système d'équations récurrentes uniformes* n'est pas calculable. Tout le problème est donc de savoir si le multicycle ainsi défini est ou non un cycle.

4.2.2 Algorithme de décomposition

Pour déterminer la calculabilité d'un système d'équations récurrentes uniformes, on applique au graphe réduit de dépendance l'algorithme suivant.

Algorithme

i. Décomposer le graphe G de départ en composantes fortement connexes maximales G_1, G_2, ..., G_s, et appeler l'étape (ii) sur chacun des G_i.

ii. Construire le sous-graphe G' du graphe G courant par suppression des arêtes de G qui ne participent d'aucun multicycle de poids nul; si G' est fortement connexe alors le graphe G n'est pas calculable, sinon appeler l'étape (i) sur G'.

Dans toute la suite de ce rapport on notera G' le graphe obtenu par action de l'étape (ii) sur un graphe fortement connexe G.

Preuve de la validité de l'algorithme

Lemme 4 *G contient un multicycle de poids nul si, et seulement si, G' en contient un.*

2. Les arcs de G sont bien évidemment considérés dans le même ordre dans C que dans D

Preuve: G' est le sous-graphe de G engendré par l'ensemble des arêtes de G participant à au moins un multicycle de poids nul, il contient donc tous les cycles de poids nul de G. Si G contient un cycle de poids nul il en va alors de même de G'.

La réciproque est évidente. □

Corollaire 2 *Si le sous-graphe G' d'un graphe G est vide, alors G ne contient pas de cycle de poids nul.*

Corollaire 3 *G contient un cycle de poids nul si, et seulement si, une de ses composantes connexes G'_i de G' en contient un.*

Preuve: Un cycle est une composante connexe. □

Lemme 5 *Si le sous-graphe G' d'un graphe G est fortement connexe, alors G contient un cycle de poids nul.*

Preuve: Si le sous-graphe G' d'un graphe G est fortement connexe, alors G' contient un cycle de poids nul, et G aussi par conséquent. En effet, si G' est fortement connexe, il existe un cycle $\mathcal{C}$ de G' passant par toutes ses arêtes. Notons e_1, e_2,..., e_m les arêtes successives de $\mathcal{C}$. Chaque arête e_i de $\mathcal{C}$ est contenue par définition de G' dans un multicycle $\mathcal{C}_i$ de poids nul de G et donc de G'. Ce multicycle peut se décomposer sous la forme $\mathcal{C}_i = e_i + \pi_i + \mathcal{C}'_i$, où $\mathcal{C}'_i$ est un multicycle et π_i un cycle de G' orienté du sommet sur lequel pointe e_i vers celui dont e_i est issue. Il est alors possible de construire un chemin de poids nul de G' comme suit: le cycle traverse dans l'ordre les arêtes e_1, e_2,..., e_m, puis il revient le long des chemins π_m,..., π_2, π_1; sur ce cycle support sont greffés les multicycles $\mathcal{C}'_i$ restant ce qui est tout à fait possible puisque ce cycle support passe par tous les sommets de G'.

La construction d'un tel cycle de poids nul est illustrée par la figure 4.3. *Avertissement:* ce dessin est une vue de l'esprit, puisque le cycle initial est sensé emprunter toutes les arêtes... □

Corollaire 4 *L'algorithme de décomposition est valide.*

Preuve:

Si l'algorithme se termine parce qu'un des G'_i est fortement connexe, ce sous-graphe G'_i contient un cycle de poids nul, par conséquent G contient aussi un cycle de poids nul et alors le graphe de départ correspond à un système d'E.R.U. non calculable. Si par contre l'algorithme se termine parce que tous les appels récursifs ont fini par déboucher sur des graphes F_i de fils F'_i vides, alors le système correspondant est calculable. □

Preuve de la terminaison de l'algorithme Lorsque l'étape (ii) est appliquée à un graphe F, soit l'algorithme termine par forte connexité de F' (découverte d'un cycle de poids nul), soit l'étape (ii) est appelée sur des sous-graphes du graphe F qui ont chacun un nombre d'arêtes strictement inférieur au nombre d'arêtes de F qui est lui bien évidemment fini, d'où la terminaison.

4.2.3 Implémentation

L'étape (ii) est réalisée par programmation linéaire: soit G un graphe de matrice de connexion C et de matrice de déplacement D, alors les arêtes de G qui participent à au moins un multicycle

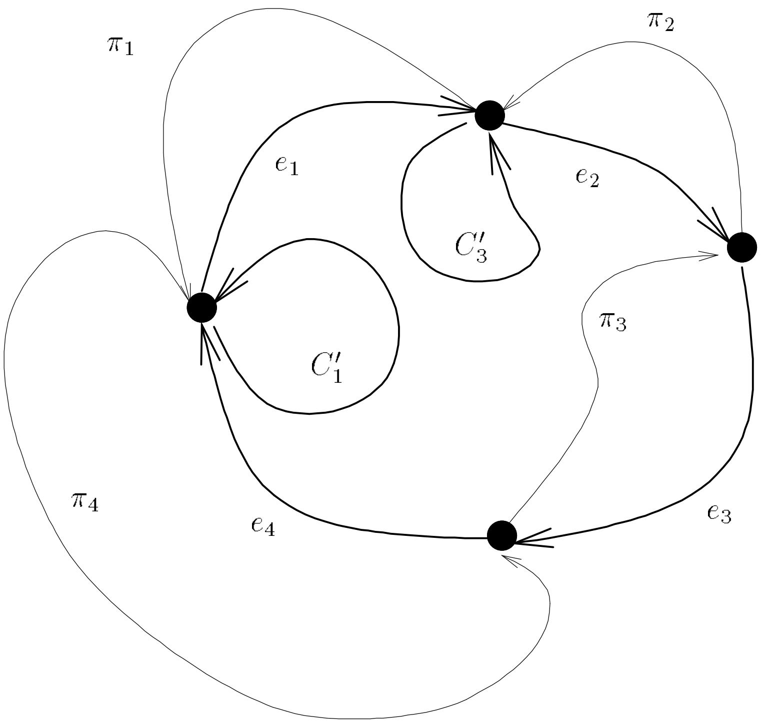

FIG. 4.3 - *Exemple schématique de construction d'un cycle de poids nul*

de poids nul correspondent exactement aux valeurs de i pour lesquelles le système suivant est satisfiable:

$$\begin{cases} Bq = 0 \\ q_i = 1 \\ \forall j \neq i \quad q_j \geq 0 \end{cases}$$

Le fait que la solution exhibée puisse être rationnelle ne pose aucun problème puisqu'à partir d'une solution rationnelle on peut toujours se ramener par homothétie à une solution entière.

Le problème de la décomposition d'un graphe G peut être résolu en une seule fois par le programme linéaire suivant, comme le montrent les lemmes 6 et 7:

$$\begin{cases} q \geq 0 \\ v \geq 0 \\ q + v \geq 1 \\ Bq = 0 \\ min(\sum_i v_i) \end{cases} \tag{4.1}$$

Lemme 6 *Les solutions optimales de ce programme linéaire sont telles que les v_i valent, soit 0, soit 1.*

Remarque préliminaire: la fonction objective du programme linéaire 4.1 est bornée par le nombre d'arêtes du graphe de dépendance. Sa valeur est donc finie puisque $q = 0$ est solution. **Preuve:**

Considérons une solution optimale du problème linéaire. Il est évident que les v_i valent au plus 1 car q est positif et la somme des v_i minimale.

Posons $v = max\{v_i \mid v_i < 1\}$ et $q = min\{q_i \mid v_i < 1\}$, $q > 0$ car $q_i + v_i \geq 1$. Supposons que l'on ait $v > 0$. On construit une nouvelle solution (q'_i, v'_i) du système en posant $q'_i = \frac{1}{q} * q_i$, $v'_i = 1$ si $v_i = 1$ et 0 sinon. Si $v_i = 1$, alors $q'_i + v'_i \geq v'_i = 1$ et si $v_i < 1$, $q'_i + v'_i \geq q'_i = \frac{1}{q} * q_i \geq 1$. D'autre part, $Bq' = \frac{1}{q} * Bq = 0$, et enfin $\sum_i v'_i < \sum_i v_i$, il y a contradiction. □

Lemme 7 *Les solutions optimales du programme linéaire 4.1 sont telles que: $v_i = 0 \Leftrightarrow e_i \in G'$*

Preuve: Toute solution du programme linéaire précédent définit un multicycle de poids nul. En effet, soit e_i une arête appartenant à l'un des multicycles de poids nul C_i de G'. Si $v_i = 1$, alors on ajoute à q le multicycle C_i pour obtenir un nouveau multicycle q'. On définit alors v' à partir de v: $v' = v$ sauf pour la $i^{ème}$ coordonnée pour laquelle v' est nul ($v'_i = 0$). Par définition q' et v' vérifient les équations du programme linéaire ci-dessus 4.1 et $\sum_j v_j = 1 + \sum_j v'_j$. Il y a donc contradiction, et par conséquent $v_i = 0$. Réciproquement si $v_i = 0$ alors $q_i \geq 1$ et e_i appartient à un multicycle de poids nul. □

Par conséquent le programme linéaire ci-dessus fournit directement G', G' étant formé des arêtes pour lesquelles v_i est nul.

4.2.4 Complexité de l'algorithme

Théorème 4 *L'algorithme de décomposition s'exécute en $O(n.Z)$ où Z est la complexité d'un programme linéaire à $(2 * |E|)$ variables et $(3 * |E| + n + |V|)$ équations ou inéquations.*

Preuve: Il sera montré ultérieurement (corollaire 6) que la profondeur de récursion de la décomposition est bornée par la dimension de l'espace d'indiçage (n). Par ailleurs, les arguments des différents appels récursifs à la profondeur i sont disjoints, et la complexité totale de ces appels est du même ordre que celle de l'appel initial (pour plus de précisions cf. [KS88]). Le calcul de G' à partir de G se ramenant à la résolution du programme linéaire précédent et la décomposition de G' en composante fortement connexe maximale étant de complexité négligeable ($O(|V|(|E|+|V|))$), on obtient le résultat escompté. $\square$

Remarque: la complexité proposée ici semble nettement meilleure que celle proposée par Kosaraju et Sullivan [KS88] pour leur propre algorithme. Cette différence s'explique en partie par leur méconnaissance de la borne sur la profondeur de la récursion (eux la borne par $|V|$), et, en partie, par l'utilisation d'un seul programme linéaire pour déterminer G'.

4.2.5 Profondeur de décomposition et latence

Sailesh Rao a énoncé le résultat suivant [Rao85]:

Théorème 5 *Si d est la profondeur de l'arbre de décomposition obtenu par action de l'algorithme 4.2.2 sur le graphe de dépendance réduit d'un programme, la latence du programme est alors en* $\Omega(N^{d+1})$

i. Dans la suite on considérera que le domaine d'indiçage est l'hypercube de taille N du quadrant positif de l'espace d'indiçage qui admet l'origine $0_{\mathbb{N}}$ pour sommet.

   Si le domaine d'indiçage est différent, on considérera le plus grand hypercube qu'il contient. Une fois l'hypercube fixé, le domaine et l'hypercube croisseront homothétiquement avec le facteur d'homothétie N.

ii. Dans ce qui suit, un nœud de l'arbre de décomposition d'un graphe G et le sous-graphe $H_{i,j}$ de G qu'il représente seront identifiés. Dans la notation $H_{i,j}$, i est la profondeur du nœud correspondant dans le graphe et j son numéro dans l'ensemble des nœuds de profondeur i.

Idée de la preuve de Sailesh Rao:

Pour chaque nœud $H_{i,j}$ de l'arbre de décomposition d'un graphe G, on construit un multicycle de poids nul $Q_{i,j}$ passant par toutes les arêtes de $H'_{i,j}$ ($Q_{i,j}$ est classiquement le résultat du programme linéaire 4.1 présenté ci-dessus éventuellement multiplié par une constante qui le rende entier). Ce multicycle $Q_{i,j}$ définit un cycle $C_{i+1,j'}$ pour chacun des fils $H_{i+1,j'}$ de $H_{i,j}$. La somme des cycles $C_{i+1,j'}$ sur l'ensemble des fils $H_{i+1,j'}$ de $H_{i,j}$ est donc de poids nul, puisque cette somme est exactement $Q_{i,j}$.

G étant par hypothèse fortement connexe, il existe un cycle $C_{0,1}$ passant par chacune de ses arêtes. Rao parcourt alors ce cycle $C_{0,1}$ d'une façon quelconque et "pour cette traversée détermine le premier nœud rencontré pour chacun des fils. Quand ces nœuds sont atteints pour la première fois, se brancher sur le cycle local et le parcourir cN fois pour une certaine constante c choisie telle que le chemin équivalent dans le graphe de dépendance (complet) reste dans le domaine d'indiçage tout au long du parcours"

L'existence de la constante c étant sujette à caution, je propose ici une preuve totalement constructive.

Preuve du théorème 5

Soit $G = (V, E)$ un graphe orienté fortement connexe aux arcs valués par des vecteurs de $\mathbb{Z}^n$. Pour tout chemin l de G, $l = \sum_{i=1}^{r} e_i$ où $head(e_i) = tail(e_{i+1})$, on définit deux entiers $m(l)$ et $M(l)$ comme suit:

$$m(l) = min\{t \in \mathbb{N} \mid -t.1_{\mathbb{N}} \leq \sum_{i=1}^{s} w(e_i), 1 \leq s \leq r\}$$

$$M(l) = min\{t \in \mathbb{N} \mid \sum_{i=1}^{s} w(e_i) \leq t.1_{\mathbb{N}}, 1 \leq s \leq r\}$$

Soit d la hauteur de l'arbre de décomposition de G par l'algorithme 4.2.2. On note $G_{i,j}$ le sous-graphe de G identifié au $j^{ème}$ nœud à la profondeur i de l'arbre de décomposition, pour i variant de 0 à d. Par exemple $G_{0,1} = G$

A tout graphe $G_{i,j}$ qui n'est pas une feuille ($i < d$) on associe un multicycle $Q_{i,j}$ passant par toutes ses arêtes participant d'un de ses multicycles de poids nul ($Q_{i,j}$ est classiquement le résultat du programme linéaire 4.1 présenté ci-dessus éventuellement multiplié par une constante qui le rende entier). Un multicycle $Q_{i,j}$ définit un cycle non vide $C_{i+1,j'}$ pour chacun des fils $G_{i+1,j'}$ de $G_{i,j}$. De plus, $\sum_{G_{i+1,j'} fils de G_{i,j}} w(C_{i+1,j'}) = 0$ par construction.

G étant supposé fortement connexe, il admet un cycle $C_{0,1}$ passant par toutes ses arêtes. On parcourt ce cycle $C_{0,1}$ à partir d'un sommet quelconque de G fixé $p_{0,1}$, et, pour ce parcours, on détermine le premier nœud de $G_{1,j}$, noté $p_{1,j}$, rencontré pour chaque fils $G_{1,j}$ de $G_{0,1} = G$. On procède de même pour chacun des nœuds de l'arbre de décomposition, le parcours de $G_{i+1,j'}$ débutant par le sommet $p_{i+1,j'}$ déterminé lors du parcours de $G_{i,j}$ père de $G_{i+1,j'}$. On note $m_{i,j} = m(C_{i,j})$ et $M_{i,j} = M(C_{i,j})$, où le point de départ de $C_{i,j}$ pour le calcul de m et de M est $p_{i,j}$.

Le but du jeu est de construire un vrai chemin de G de longueur $O(N^{d+1})$ où N.

Le chemin initial: On parcourt $C_{0,1}$ à partir de $p_{0,1}$ et à chaque fois qu'en parcourant un cycle $C_{i,j}$ on atteint un sommet $p_{i+1,j'}$ on se branche sur le cycle s'il n'a pas déjà été visité (on parcourt $C_{i+1,j'}$ en entier avant de finir de traverser $C_{i,j}$).

Tous les points *pt* de l'espace d'indiçage touchés lors du parcours de ce chemin, noté $\mathcal{C}^1$, à partir du point de l'espace d'indiçage $P^{(1)}$, tous ces points *pt* sont tels que:

$$P^{(1)} + \left(- \sum_{i,j} m_{i,j} \right) .1_{\mathbb{N}} \leq pt \leq P^{(1)} + \left(\sum_{i,j} M_{i,j} \right) .1_{\mathbb{N}}$$

On prend alors $P^{(1)} = \sum_{i,j} m_{i,j}$ et pour $N \geq \mathcal{N} = \sum_{i,j}(m_{i,j} + M_{i,j})$ tous les points *pt* de l'espace d'indiçage touchés sont dans le domaine.

Soit $N = k * \mathcal{N}, k \in \mathbb{N}^*$. Dans ce nouveau domaine hypercubique, on construit le chemin suivant.

Le chemin de k*N: On parcourt k fois $C_{0,1}$ à partir de $p_{0,1}$ et à chaque fois qu'en parcourant un cycle $C_{i,j}$ on atteint un sommet $p_{i+1,j'}$ non encore visité lors de **ce** parcours de $C_{i,j}$, on se branche sur le cycle $C_{i+1,j'}$ que l'on parcourt k fois (avant de finir de traverser $C_{i,j}$). On note ce chemin $\mathcal{C}^k$.

Dans toute la suite de cette construction, $G_{i+1,j'}$ désignera un graphe fils du graphe $G_{i,j}$.

Dans ce nouveau chemin, le cycle $C_{i,j}$ sera donc parcouru k^{i+1} fois.

Soit $L(l)$ la longueur du chemin l et soit $\mathcal{L} = \sum_j L(C_{d,j})$, alors $L(C^k) \geq k^{d+1} * \mathcal{L}$.

Le résultat promis sera démontré si le chemin correspondant dans l'espace d'indiçage ne sort pas du nouveau domaine, le point de départ dans l'espace d'indiçage ($P^{(k)}$) étant bien choisi. Pour cela, prenons un point *pt* quelconque du chemin et prouvons qu'il appartient bien au domaine. Le point *pt* est atteint lors du parcours d'un cycle $C_{i,j}$, il s'écrit sous la forme:

$$pt = P^{(k)} + w \left(\sum_{i,j} \theta(i,j).C_{i,j} + \sum_{i,j} l_{i,j} \right)$$

où $\theta(i,j)$ est le nombre de fois que $C_{i,j}$ a été entièrement parcouru et $l_{i,j}$ est le morceau de $C_{i,j}$ restant.

Il nous faut alors montrer: $0 \leq w\left(\sum_{i,j} \theta(i,j).C_{i,j} + \sum_{i,j} l_{i,j}\right) + P^{(k)} \leq N.1_{\mathbb{N}}$

Il est immédiat que $l_{i,j}$ est un sous-chemin de $C_{i,j}$ car un nouveau parcours de $C_{i,j}$ n'est commencé qu'après la fin du parcours précédent. $l_{i,j} = \emptyset$ signifie que le point *pt* n'a pas été atteint lors du parcours du cycle $C_{i,j}$.

Considérons $G_{i,j}$ et ses fils $G_{i+1,j'}$:

- si $l_{i,j} = \emptyset$ *(cas 1)* alors le point *pt* n'a pas été atteint lors du parcours du cycle $C_{i,j}$, et non plus que lors du parcours d'un des cycles $C_{i+1,j'}$ par conséquent. Le cycle $C_{i,j}$ a été parcouru en tout $\theta(i,j)$ fois, et pour chacun de ces parcours, chacun des cycles de ses fils a été parcouru k fois. Donc, soit si $l_{i,j} = \emptyset$ alors $\theta(i+1,j') = k * \theta(i,j)$;
- si $l_{i,j} \neq \emptyset$, c'est-à-dire si le point *pt* a été atteint lors du parcours de $C_{i,j}$, alors:
  - si $l_{i+1,j'} = \emptyset$, le point *pt* n'a pas été atteint lors du parcours de $C_{i+1,j'}$. Deux cas sont de nouveau possibles: soit *pt* a été atteint avant que le chemin n'atteigne le point $p_{i+1,j'}$ de branchement sur le cycle $C_{i+1,j'}$, soit *pt* a été atteint après.

    Dans le premier cas *(cas 2)*, $l_{i,j}$ ne contient pas $p_{i+1,j'}$, et $C_{i+1,j'}$ n'a pas été parcouru lors du dernier parcours (inachevé) de $C_{i,j}$, donc $\theta(i+1,j') = k * \theta(i,j)$.

    Dans le deuxième cas *(cas 3)*, $l_{i,j}$ contient $p_{i+1,j'}$ et $C_{i+1,j'}$ a été parcouru k fois lors du dernier parcours (inachevé) de $C_{i,j}$, et donc: $\theta(i+1,j') = k * \theta(i,j) + k$.
  - si $l_{i+1,j'} \neq \emptyset$ *(cas 4)*, le point *pt* a été atteint lors du parcours de $C_{i+1,j'}$ et lors du dernier parcours (inachevé) de $C_{i,j}$, le cycle $C_{i+1,j'}$ a été parcouru moins de $k-1$ fois, donc: $k * \theta(i,j) \leq \theta(i+1,j') < k * \theta(i,j) + k$

Les différents cas sont illustrés sur la figure 4.4.

On définit alors la fonction $\delta(l_{i,j})$ qui vaut 1 si $l_{i,j} \neq \emptyset$ et 0 sinon. De l'étude précédente on tire l'inéquation:

$$\forall (i,j,j') \quad | \quad G_{i+1,j'} \text{ fils de } G_{i,j} 0 \leq \theta(i+1,j') - k * \theta(i,j) + \delta(l_{i+1,j'}) \leq k \tag{4.2}$$

On pose $\theta'(i+1,j') = \theta(i+1,j') - k * \theta(i,j)$ et $\theta'(0,1) = \theta(0,1)$.

Par construction, la somme sur l'ensemble des fils $G_{i+1,j'}$ de $G_{i,j}$ des cycles $C_{i+1,j'}$ vaut exactement $Q_{i,j}$ et est donc de poids nul ($\sum_{j' \mid G_{i+1,j'} \ fils \ de \ G_{i,j}} C_{i+1,j'} = 0$). Par conséquent:
$w\left(\sum_{j' \mid G_{i+1,j'} \ fils \ de \ G_{i,j}} \theta(i+1,j').C_{i+1,j'}\right) =$
$w\left(\sum_{j' \mid G_{i+1,j'} \ fils \ de \ G_{i,j}} (\theta'(i+1,j') + k * \theta(i,j)).C_{i+1,j'}\right) =$
$w\left(\sum_{j' \mid G_{i+1,j'} \ fils \ de \ G_{i,j}} \theta'(i+1,j').C_{i+1,j'}\right)$ En sommant cette équation sur tous les nœuds à la même profondeur, puis sur les profondeurs, on obtient:

$$w\left(\sum_{i \geq 1,j} \theta(i,j).C_{i,j} + \sum_{i \geq 1,j} l_{i,j}\right) = w\left(\sum_{i \geq 1,j} \theta'(i,j).C_{i,j} + \sum_{i \geq 1,j} l_{i,j}\right)$$

Equation qui peut elle même être étendue au cas i positif compte tenu de la définition de $\theta'(0,1)$. Comme $m_{i,j}.1_{\mathbb{N}} \leq C_{i,j} \leq M_{i,j}.1_{\mathbb{N}}$, $m_{i,j}.1_{\mathbb{N}} \leq l_{i,j} \leq M_{i,j}.1_{\mathbb{N}}$ et que l'équation 4.2 implique $0 \leq \theta'(i,j) + \delta(l_{i,j}) \leq k$, on déduit de tout ce qui précède que l'on a: $(-\sum_{i,j} k.m_{i,j}).1_{\mathbb{N}} \leq$

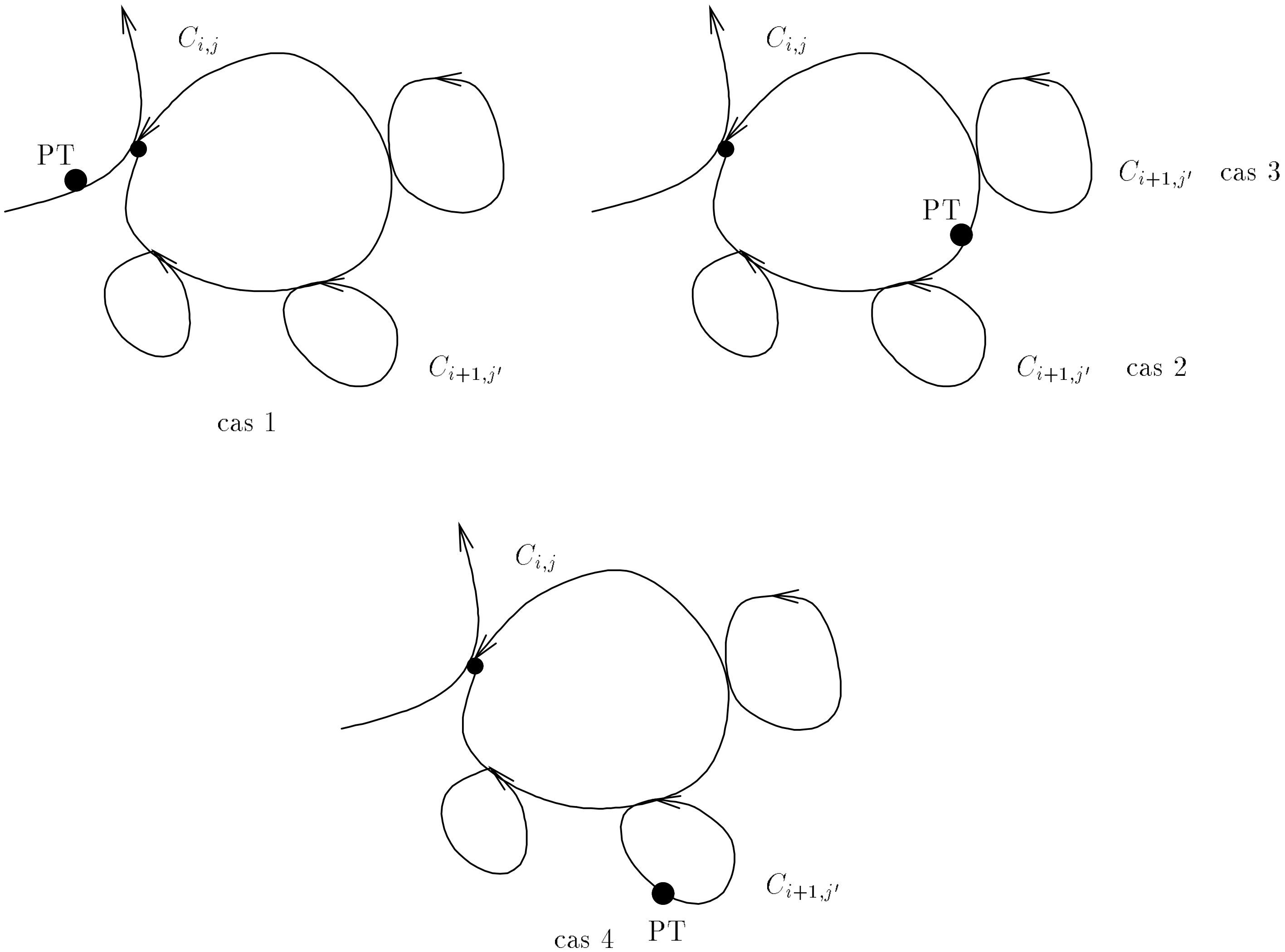

FIG. 4.4 - *Evaluation de la fonction theta*

$w\left(\sum_{i,j}\theta(i,j).C_{i,j}+\sum_{i,j}l_{i,j}\right) \leq (\sum_{i,j}k.M_{i,j}).1_{\mathbb{N}}$, d'où

$$P^{(k)}-k.(\sum_{i,j}m_{i,j}).1_{\mathbb{N}} \leq w\left(\sum_{i,j}\theta(i,j).C_{i,j}+\sum_{i,j}l_{i,j}\right) \leq P^{(k)}+k.(\sum_{i,j}M_{i,j}).1_{\mathbb{N}}$$

et comme $N=k*\mathcal{N}=k*\sum_{i,j}(m_{i,j}+M_{i,j})$, il suffit de prendre $P^{(k)}=k.P^{(1)}=k.\sum_{i,j}m_{i,j}$ pour être sûr que tous les points du chemin sont dans le domaine. □

4.3 Construction d'ordonnancements multi-dimensionnels

Nous allons construire ici pour tout système d'équations récurrentes uniformes calculable, un ordonnancement multidimensionnel affine de partie linéaire localement unique.

4.3.1 Résultats préliminaires et notations

Réduction au cas des graphes fortement connexes

Théorème 6 *Soit un système d'E.R.U. calculable de graphe de dépendance réduit G et d'ordonnancement multidimensionnel affine pour la $i^{ème}$ composante fortement connexe maximale G_i $\{(X_{i,j},C_{i,j})\}$ ($X_{i,j}$ est le vecteur des parties linéaires pour le $j^{ème}$ nœud, et $C_{i,j}$ le vecteur des parties constantes).*

Il existe alors des entiers α_i tels que le système d'E.R.U. considéré admette comme ordonnancement multidimensionnel affine le système: $(X_v,C_v)=((0,X_{i,j}),(\alpha_i,C_{i,j}))$ si le nœud v est le $j^{ème}$ nœud de la $i^{ème}$ composante fortement connexe.

Preuve: Dans le graphe de dépendance réduit on condense chaque composante fortement connexe maximale en un seul sommet, les seules arêtes du graphe résultant appelé *condensation* étant donc les arêtes reliant des composantes fortement connexes maximales distinctes. Le graphe obtenu est par construction acyclique et un ordonnancement peut être facilement obtenu: à chaque nœud i est associé un entier α_i égal à la longueur du plus long chemin d'une feuille du graphe condensé au nœud i.

Le système proposé étant notoirement un ordonnancement, le tour est joué. □

Remarque: vue la structure du problème, le graphe G ne contient pas de sommet isolé et il en va de même de G'.

Hyperplan faiblement séparateur

Lemme 8 *Soit G un graphe de dépendance. Il existe alors des rationnels ρ_1, ρ_2, ..., ρ_m et un vecteur x de dimension n tels que pour toute arête e_i dirigée de v_k vers v_l on ait:*

- *si $e_i \in G'$* $\quad w(e_i).x \geq \rho_k-\rho_l$
- *si $e_i \notin G'$* $\quad w(e_i).x \geq 1+\rho_k-\rho_l$

Preuve: Comme pour le lemme équivalent de Karp, Miller et Winograd [KMW67] où sont considérés les graphes sans multicycles de poids négatif, la démonstration repose sur l'utilisation du lemme de (Minkowski-) Farkas [Sch86] sur le système ci-dessous trivialement non satisfiable:

$$\begin{cases} (i) & q \geq 0 \\ (ii) & \sum_{i, e_i \notin G'} q_i = 1 \\ (iii) & Bq = 0 \end{cases}$$

□

Arbre de décomposition Considérons un graphe G calculable, il ne contient donc pas de cycles de poids nul. On lui applique l'algorithme de décomposition présenté ci-dessus. La décomposition peut être représentée par un arbre de racine G dont les nœuds sont exactement les composantes fortement connexes maximales du graphe obtenu par action de l'étape (ii) de l'algorithme, un nœud H étant relié aux composantes fortement connexes de son sous-graphe H'. Un tel arbre sera appelé **arbre de décomposition**. Contrairement à Sailesh Rao [Rao85], nous supposerons que le graphe G est fortement connexe, si ce n'était pas le cas, nous nous y ramènerions grâce au théorème 6 en se restreignant à l'étude des seules composantes fortement connexes de G. L'arbre de décomposition de Sailesh Rao admet pour racine le graphe G[3], pour fils de la racine les composantes fortement connexes de G, le reste de l'arbre étant défini comme celui ici choisi. Du fait de cette différence de construction, les arbres de Sailesh Rao sont d'une unité plus grand que les nôtres. Lorsqu'un résultat proposé dans [Rao85] a été reproduit ici, il a été adapté à la nouvelle définition de l'arbre.

4.3.2 Construction d'un ordonnancement multidimensionnel affine

On construit récursivement un ordonnancement multidimensionnel affine de partie linéaire localement unique.

Considérons un graphe G calculable, et son arbre de décomposition. Les feuilles de cet arbre sont des sous-graphes de G ne contenant pas de multicycles de poids nul[4]. Par conséquent il existe pour chacun de ces graphes de dépendance un ordonnancement affine (lemme 8), donc multidimensionnel affine de dimension un.

Considérons maintenant un nœud interne de l'arbre, noté H. Ce nœud H a pour fils H_1, H_2, $\ldots$, H_s dont chaque sommet v admet, par hypothèse de récurrence, un ordonnancement multidimensionnel affine de dimension k_v défini par deux vecteurs x_v et c_v, les vecteurs x_v et c_v sont normalisés en deux vecteurs de dimension $k = max_v(k_v)$ par ajoût du nombre idoine de zéros terminaux. Appliquons par ailleurs le lemme 8, nous obtenons un vecteur x, et des rationnels ρ_1, ρ_2, $\ldots$, ρ_m tels que:

- si $e_i \in G'$ $\quad w(e_i).x \geq \rho_k - \rho_l$
- si $e_i \notin G'$ $\quad w(e_i).x \geq 1 + \rho_k - \rho_l$

Associons maintenant à chaque sommet v de H de nouveaux vecteurs x'_v et c'_v d'ordonnancement comme suit: $x'_v = (x, x_v)$ et $c'_v = (\rho_v, c_v)$. Il reste à montrer que nous avons bien construit un ordonnancement, c'est-à-dire que pour tout point p de l'espace d'indiçage et tout couple (u, v) de sommets de H reliés par une arête e de H de poids w orientée de u vers v (v dépend de u), $x'_v.p + c'_v \gg_l x'_u.(p - w) + c'_u$. Or si $e \in H'$, $x_v.p + c_v \gg_l x_u.(p - w) + c_u$ par hypothèse de récurrence

3. Qu'il soit ou non fortement connexe

4. on ne considère pas pour la construction de l'arbre les graphes vides obtenus pas action de l'algorithme sur les graphes terminaux ne contenant pas de multicycles de poids nul

et $x.p + \rho_v \geq x.(p - w) + \rho_u$ par construction; sinon $e \notin H'$ et $(x.p + \rho_v) - (x.(p - w) + \rho_u) = x.w + \rho_v - \rho_u \geq 1$ par construction, et donc $x.p + \rho_v \geq 1 + x.(p - w) + \rho_u$.

Nous construisons donc de la sorte par récurrence un ordonnancement multidimensionnel affine, de partie linéaire localement unique.

4.3.3 Complexité de l'ordonnancement obtenu

Théorème 7 *Soit un graphe G de dépendance uniforme calculable quelconque de composantes fortement connexes maximales G_1, $G_2 \ldots$, G_k. Soit d la plus grande profondeur du graphe de décomposition d'un G_i, alors l'ordonnancement obtenu est en $O(N^{d+1})$*

Preuve:

Supposons dans un premier temps le graphe est fortement connexe. Aucun des hyperplans faiblement séparateur généré par l'algorithme n'est nul. Pour s'en convaincre il suffit de considérer une composante fortement connexe maximale de G et un des sous-graphes issus de sa décomposition. Ce sous-graphe est fortement connexe, on prend un cycle passant par chacune de ses arêtes. Le produit scalaire de l'hyperplan faiblement séparateur et du poids de ce cycle doit être supérieur par définition au nombre d'arêtes du cycle ne participant à aucun multicycle de poids nul du sous-graphe considéré. G étant calculable, il y a au moins une arête de ce type dans le cycle et par conséquent l'hyperplan n'est pas nul. Un hyperplan non nul génère une boucle de taille $O(N)$, et une famille libre de k hyperplans (faiblement) séparateurs génère une boucle de taille $O(N^k)$. Les différents hyperplans faiblement séparateurs associés à un sommet du graphe formant une famille libre comme il le sera démontré ultérieurement (cf. 8), le théorème étudié est démontré dans ce cas.

Supposons dans un deuxième temps que G n'est pas fortement connexe. Les vecteurs d'ordonnancement obtenus sont alors de taille $d+2$ s'ils ont été normalisés par ajoût de zéros. La première composante de chacun de ces vecteurs d'ordonnancement a été généré par le théorème 6. La première composante de chaque vecteur de parties linéaires est donc nul. Le théorème 6 revient en fait à ordonner entre elles les composantes fortement connexes maximales de G et la complexité de l'ordonnancement est donc égale à la complexité maximale de l'ordonnancement sur les G_i. □

4.3.4 Propriétés et corollaires

Des corollaires immédiats découlent du théorème 5:

Corollaire 5 *L'ordonnancement multidimensionnel affine 4.3.2 est asymptotiquement optimal: si l'ordonnancement construit est de dimension d, l'arbre de décomposition est de hauteur $d-1$ et il existe dans le programme traité un chemin de dépendance en $\Theta(N^d)$.*

Preuve: comparer les théorèmes 7 et 5 ...

Question: Peut-on être plus précis, peut-on estimer l'efficacité effective de cette algorithme c'est-à-dire la constante en facteur du "(N^d)"?

Corollaire 6 *Si d est la profondeur de l'arbre de décomposition obtenu par action de l'algorithme 4.2.2 sur un graphe de dépendance réduit dont l'espace d'indiçage est de dimension n, alors $d \leq n-1$.*

Dans leur article [KMW67], Karp *et al.* prétendent que si l'arbre de décomposition est de hauteur d et n la dimension de l'espace d'indiçage, alors $d \leq n$, mais pour eux la profondeur d'un arbre est le nombre de sommets sur sa plus grande branche, d'où la différence d'une unité avec le corollaire ci-dessus (dans [KMW67], le résultat est simplement énoncé mais non démontré).

Le corollaire précédent peut être obtenu plus directement à partir du théorème suivant:

Théorème 8 *Soit G un graphe réduit de dépendance calculable et soit v un sommet de ce graphe, alors, les hyperplans faiblement séparateurs obtenus pour v lors de la décomposition de G sont linéairement indépendants.*

Ce théorème sera prouvé dans la partie suivante (4.4.4) après que certains résultats indispensables à sa preuve ait été établis.

Ce théorème est très important pour l'utilisation pratique de l'algorithme en vue de la réécriture des équations récurrentes: les vecteurs étant indépendants ils forment le noyau valide d'un changement de base.

4.4 Comparaison avec l'algorithme de Paul Feautrier

L'algorithme de décomposition repose sur le programme linéaire proposé en 4.2.3. Nous allons maintenant étudier ce programme linéaire pour montrer qu'en fait l'ordonnancement optimal que nous proposons ici avait déjà été présenté par Paul Feautrier [Fea92] mais comme une heuristique gloutonne.

4.4.1 Programme linéaire dual

Le programme linéaire précédent s'écrit sous forme canonique:

$$min\left\{\sum_i v_i \,\middle|\, [qvw] \geq 0, [qvw].\begin{bmatrix} I_d & {}^tC & {}^tD \\ I_d & 0 & 0 \\ -I_d & 0 & 0 \end{bmatrix} = [1 \;\; 0 \;\; 0]\right\}$$

ou plus généralement:

$$min\left\{u.\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \,\middle|\, u \geq 0, u.\begin{bmatrix} I_d & {}^tC & {}^tD \\ I_d & 0 & 0 \\ -I_d & 0 & 0 \end{bmatrix} = [1 \;\; 0 \;\; 0]\right\}$$

et admet pour programme dual le programme linéaire:

$$max\left\{[1 \;\; 0 \;\; 0].t \,\middle|\, \begin{bmatrix} I_d & {}^tC & {}^tD \\ I_d & 0 & 0 \\ -I_d & 0 & 0 \end{bmatrix}.t \leq \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}\right\}$$

qui peut s'écrire:

$$max\left\{\sum_i x_i \mid 0 \leq x \leq 1, \forall i \;\; x_i + z.w_i + y_{head(i)} - y_{tail(i)} \leq 0\right\}$$

avec $t = (x, y, z)$.

Cette dernière expression est celle du programme linéaire à la base de l'algorithme de construction de l'ordonnancement multidimensionnel présenté par Paul Feautrier dans l'article [Fea92], l'algorithme en question ne différant de celui proposé plus haut que par le programme linéaire invoqué. Dans les deux cas, le programme linéaire invoqué sert juste à déterminer un ensemble d'arêtes qui sera traité par un hyperplan faiblement séparateur et l'ensemble complémentaire sur lequel l'algorithme sera appelé récursivement. Les deux programmes linéaires étant duaux, les deux algorithmes sont identiques, comme nous allons le montrer.

4.4.2 Identité des deux algorithmes

La démonstration d'identité repose sur le théorème de programmation linéaire dit de *complementary slackness* ([Sch86]):

Théorème 9 *(Complementary slackness)*
Soit A une matrice, b un vecteur colonne et c un vecteur ligne. Considérons l'équation de dualité entre programmes linéaires suivante:

$$max\{cx \mid \; Ax \leq b\} = min\{yb \mid \; y \geq 0, yA = c\}$$

Supposons que les deux optima soient finis, alors pour chaque inégalité $a_i x \leq b_i$ dans $Ax \leq b$, exactement une des deux propriétés suivantes est vérifiée:

- *Le problème de maximisation admet une solution optimale x_0 avec $a_i x_0 < b_i$*
- *Le problème de minimisation admet une solution optimale y_0 de $i^{ème}$ composante strictement positive.*

On applique ce théorème à la $(|E| + i)^{ème}$ ligne de A, pour $1 \leq i \leq |E|$. Il nous donne alors qu'exactement une des deux propriétés suivantes est vérifiée:

- Le problème de maximisation admet une solution optimale $[xyz]$ avec $x_i < 1$
- Le problème de minimisation admet une solution optimale $[qvw]$ avec $v_i > 0$

Vu ce qui précède (lemme 6), le vecteur v est identique pour toutes les solutions optimales du problème de minimisation et il en va de même pour x et le problème de maximisation (sur ce point cf. [Fea92]). Sachant en outre que les x_i valent comme les v_i soit 1, soit 0 (cf. [Fea92]), le théorème de complémentarité se résume à l'équivalence:

$$\forall i \;\; x_i = 1 \Leftrightarrow v_i = 1$$

et les deux algorithmes sont équivalents.

4.4.3 Autre démonstration de l'identité des deux algorithmes

Soient G un graphe de dépendance, x un hyperplan faiblement séparateur quelconque et les rationnels ρ_i associés. On définit H un sous-graphe de G par ses arêtes:

- si $e_i \in G - H \quad w(e_i).x \geq \rho_l - \rho_k$
- si $e_i \notin H \qquad w(e_i).x \geq 1 + \rho_l - \rho_k$

Soit C un multicycle de G, par définition de H on a alors:

$$w(C) \geq card(\{e_i \in C \mid e_i \in H\})$$

Donc si C est un multicycle de poids nul, H ne contient pas d'arêtes de C, par conséquent H est inclus dans $G - G'$, or le lemme 8 nous assure de l'existence d'un hyperplan faiblement séparateur dont l'ensemble H défini comme ci-dessus est égal à $G - G'$. D'où l'identité des deux algorithmes.

4.4.4 Corollaire

Lemme 9 *Soient G un graphe calculable, x un hyperplan faiblement séparateur obtenu par l'algorithme de décomposition et les rationnels ρ_i associés, alors si e_i est une arête de G',*

$$x.w(e_i) - \rho_{tail(i)} + \rho_{head(i)} = 0$$

Preuve La preuve de ce lemme découle directement de la dualité des deux programmes linéaires étudiés ci-dessus et du fait que pour les solutions optimales du programme linéaire de Paul Feautrier:

$$max \left\{ \sum_i t_i \mid 0 \leq t \leq 1, \forall i \;\; t_i + z.w_i + y_{head(i)} - y_{tail(i)} \leq 0 \right\}$$

, les t_i valent 0 ou 1.

En effet, supposons le lemme faux et considérons une arête e_I de G' pour laquelle on a $x.w(e_I) + \rho_{head(I)} - \rho_{tail(I)} = \alpha > 0$. Les deux programmes linéaires étant duaux, on a vu qu'un hyperplan faiblement séparateur défini sur G' était solution optimale du programme linéaire de Paul Feautrier. Le couple $(1/\alpha * x,\ \{1/\alpha * \rho_i\}_i)$ est une autre solution du problème de Feautrier et pour laquelle la quantité:

$$\sum_i t_i \mid 0 \leq t \leq 1, \forall i \;\; t_i + (-x/\alpha).w(e_i) - \rho_{head(i)}/\alpha + \rho_{tail(i)}/\alpha \leq 0$$

est plus grande que l'optimal d'au moins une unité... il y a donc contradiction. □

Preuve du théorème 8:

Raisonnons par l'absurde et supposons qu'il existe un sommet v d'un graphe G calculable qui ait pour partie linéaire du vecteur d'ordonnancement multidimensionnel affine le vecteur $(x_1, \ldots, x_{k-1}, x_k, \ldots, x_l)$, tel que les vecteurs $x_1, \ldots, x_{k-1}$ forment une famille libre et les vecteurs $x_1, \ldots, x_{k-1}, x_k$ une famille liée. Il existe alors des rationnels non tous nuls, $\{\lambda_i\}_{1 \leq i \leq k-1}$ tels que $x_k = \sum_{1 \leq i \leq k-1} \lambda_i . x_i$.

Notons G_i le sous-graphe de G contenant v généré par l'algorithme de décomposition à l'étape i, et notons $\{\rho_j^{(i)}\}_j$ l'ensemble des constantes associées au vecteur x_i lors de la décomposition. Nous avons par construction:

$$\forall i \;\; 1 \leq i \leq k-1, \quad \forall j \;\; e_j \in G_k \quad x_i.w(e_j) - \rho^{(i)}_{head(e_j)} + \rho^{(i)}_{tail(e_j)} \geq 0 \tag{4.3}$$

et

$$\forall j \;\; e_j \in G_k \quad x_k.w(e_j) - \rho^{(i)}_{head(e_j)} + \rho^{(i)}_{tail(e_j)} \geq \delta(j) \tag{4.4}$$

où $\delta(j) = 1$ si $j \notin G'_k$ et $\delta(j) = 0$ si $j \in G'_k$.

D'après le lemme 9, l'inégalité 4.3 est en fait toujours une égalité.

Alors pour tout i et tout cycle C de G_k, $x_i.w(\mathcal{C}) = 0$, et donc $x_k.w(\mathcal{C}) = 0$ (par appartenance de x_k à la famille générée par $x_1, \ldots, x_{k-1}$). Or $x_k.w(\mathcal{C})$ est par définition supérieur au nombre des arêtes de C qui ne sont pas dans G'_k, la dernière égalité établie est donc en contradiction avec l'hypothèse de calculabilité de G.

□

Chapitre 5

Application aux nids de boucles non nécessairement uniformes

Nous allons présenter ici un algorithme construisant un ordonnancement pour des nids de boucles parfaits non nécessairement uniformes. Pour de tels nids on pourrait à l'instar de Paul Feautrier s'intéresser aux dépendances exactes, mais on peut aussi ne considérer qu'une approximation de ces dépendances. Le mieux serait d'avoir une description du cône de dépendance par ses rayons et sommets. Nous allons partir ici d'une description encore plus lâche: $(\mathbb{Z}, +, -, *)$, description déjà suffisante dans de nombreux cas pour extraire du parallélisme.

L'efficacité de l'ordonnancement obtenu dépend directement de la précision de l'analyse de dépendance. L'ordonnancement proposé extrait tout le parallélisme contenu dans les graphes qu'il traite, seule l'analyse et la description choisie peuvent faire perdre du parallélisme. L'exemple suivant montre l'importance de l'analyseur de dépendance: suivant qu'il déterminera l'auto-dépendance comme valant $(1, -1)$ ou $(+, -)$, c'est-à-dire suivant sa finesse, il sera ou non possible d'extraire ultérieurement du parallélisme. L'exemple est dû à Vincent van Dongen:

```
For i=0 to n do
    For j=0 to n do
        A(i, j) = A(j, i)
    Endfor
Endfor
```

5.1 Génération d'un graphe de dépendance uniforme à partir d'un graphe de dépendance dans $(\mathbb{Z}, +, -, *)$

5.1.1 Graphe de dépendance dans $(\mathbb{Z}, +, -, *)$

Les dépendances sont exprimées dans l'alphabet $(\mathbb{Z}, +, -, *)^n$ définit par Uptal Banerjee [Ban88]. Grossièrement, le symbole "+" signifie que la distance de dépendance est positive entière, le symbole "-" qu'elle est négative entière et le symbole "*" qu'elle est quelconque mais toujours entière.

Soit e une arête d'un graphe $G = (V, E)$ dont les dépendances sont exprimées dans cet alphabet $(\mathbb{Z}, +, -, *)^n$. On notera dans la suite $w(e)$ le poids de e sous la forme $u \oplus v \ominus w \otimes t$. On notera $(bc_1, \ldots, bc_n)$ la base canonique.

Dire que l'arête e est de poids $u \oplus v \ominus w \otimes t$ signifie que le sommet extrémité terminale de e dépend du sommet dont e est issue par tous les vecteurs de dépendance de la forme:

$$\sum_{0 \leq i \leq n} (u_i + \gamma_i * v_i - \lambda_i * w_i + (\mu_i - \nu_i) * t_i) * e_i$$

où γ, λ, μ et ν appartiennent à $\mathbb{N}^n$.

5.1.2 Génération du graphe de dépendance uniforme

Nous allons construire un graphe de dépendance uniforme $H = (W, F)$ à partir de $G = (V, E)$. Initialement $W = V$ et $F = \emptyset$.

Soit e une arête de E de poids $w(e) = u \oplus v \ominus w \otimes t$, orientée de a vers b où a et b sont deux sommets non forcément distincts de V, on applique alors l'algorithme suivant:

i. si $(v, w, t) = (0, 0, 0)$

   – on ajoute e à F;

ii. si $(v, w, t) \neq (0, 0, 0)$ et $t = 0$

   – on crée un nouveau[1] nœud a_1 que l'on ajoute à V;
   – si $v_i \neq 0$ on ajoute à E l'arête e_i^1, autodépendance de a_1 de poids e_i;
   – si $w_i \neq 0$ on ajoute à E l'arête e_i^2, autodépendance de a_1 de poids $-e_i$;
   – on ajoute à E l'arête e^3, de a vers a_1 de poids u (a_1 dépend de a);
   – on ajoute à E l'arête e^4, de a_1 vers b de poids $0_{\mathbb{Z}^n}$ (b dépend de a_1);

iii. si $t \neq 0$

   – on crée deux nouveaux nœuds a_1 et a_2 que l'on ajoute à V;
   – si $(v_i + t_i) \neq 0$ on ajoute à E l'arête e_i^1, autodépendance de a_1 de poids e_i;
   – si $(w_i + t_i) \neq 0$ on ajoute à E l'arête e_2, autodépendance de a_2 de poids $-e_i$;
   – on ajoute à E l'arête e^3, de a vers a_1 de poids u (a_1 dépend de a);
   – on ajoute à E l'arête e^4, de a_1 vers a_2 de poids $0_{\mathbb{Z}^n}$ (a_1 dépend de a_2);
   – on ajoute à E l'arête e^5, de a_2 vers b de poids $0_{\mathbb{Z}^n}$ (b dépend de a_2);

Si le graphe $H = (W, F)$ obtenu par action de l'algorithme décrit ci-dessus sur chacune des arêtes de G admet un cycle de poids nul, alors G en admet aussi un. F peut s'écrire comme la somme de trois ensembles F_1, F_2 et F_3 où F_j est l'ensemble des arêtes de F générées par l'étape (j) de l'algorithme ci-dessus. On distinguera aussi le sous-ensemble F_c de F constitué des arêtes de F_2 et F_3 de types e^3,e^4 et e^5.

Il est possible de simplifier l'algorithme en ne considérant que deux cas au lieu de trois: en considérant le cas ii comme un cas particulier du cas iii qui n'a pas besoin d'être traité à part. De plus, la différence d'approche n'aurait aucune influence sur l'ordonnancement trouvé!

5.1.3 Exemples

Voir figures 5.1 et 5.2.

1. s'il existe deux arêtes orientées de a vers b telles que $(v, w, t) \neq (0, 0, 0)$, les nœuds intermédiaires rajoutés seront distincts

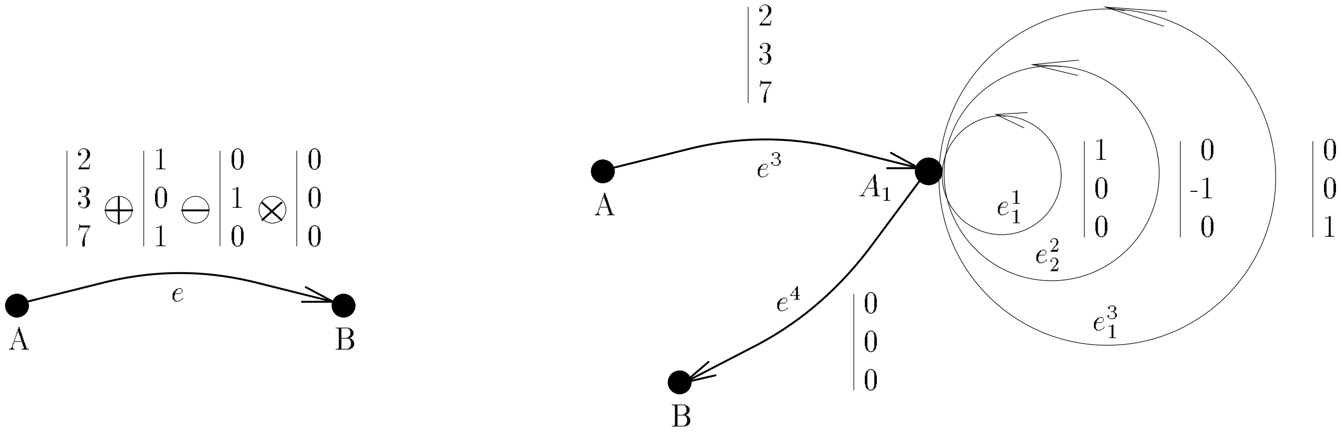

Fig. 5.1 - *Exemple pour le cas (ii)*

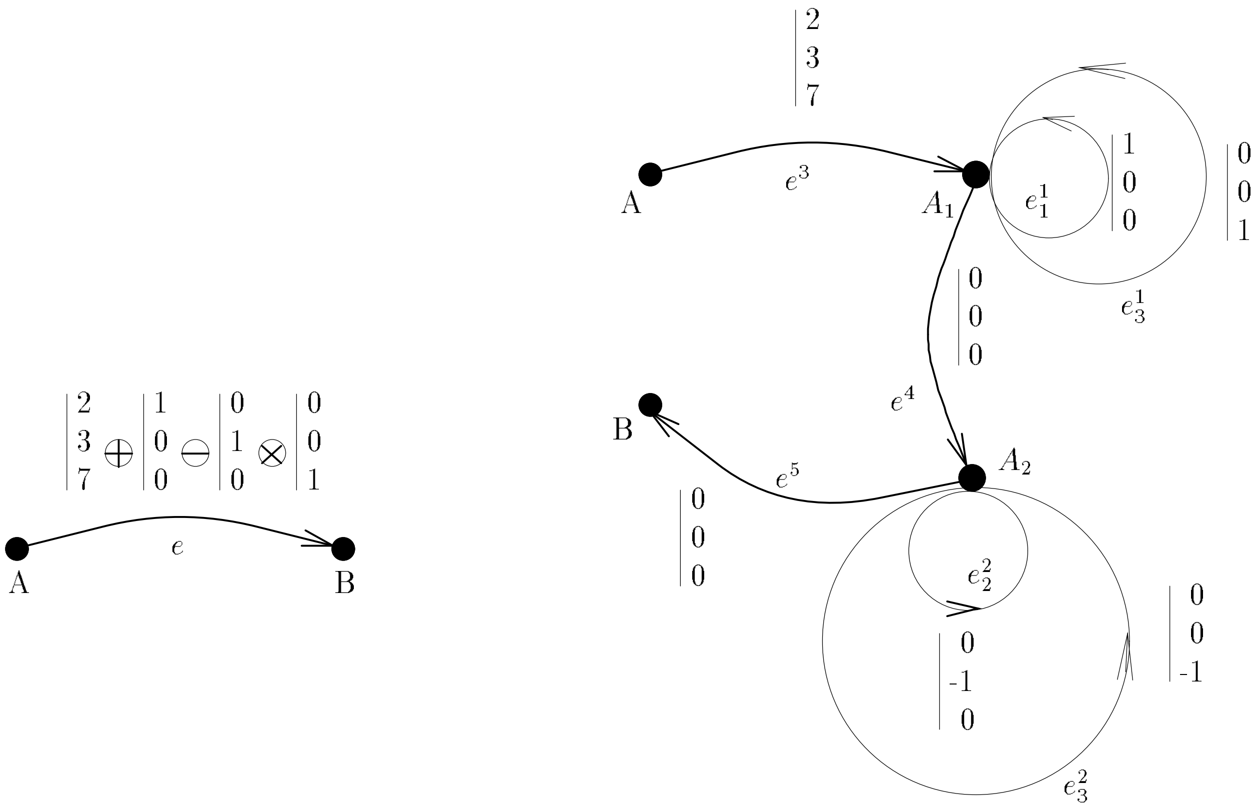

Fig. 5.2 - *Exemple pour le cas (iii)*

5.2 Construction d'un ordonnancement multidimensionnel affine de partie linéaire unique

5.2.1 Résultat préliminaire

La construction et la preuve de l'ordonnancement reposent sur le lemme suivant qui n'est qu'un affaiblissement du lemme 8.

Lemme 10 *Soit G un graphe de dépendance. Il existe alors des rationnels ρ_1, ρ_2, $\ldots$, ρ_m et un vecteur x de dimension n tels que pour toute arête e_i dirigée de v_k vers v_l on ait:*

- *si* $e_i \in G' \quad w(e_i).x + \rho_l - \rho_k = 0$
- *si* $e_i \notin G' \wedge e_i \in F_c \wedge e_i \in F_j \quad w(e_i).x + \rho_l - \rho_k \geq 1/j$
- *si* $e_i \notin G' \wedge e_i \notin F_c \quad w(e_i).x + \rho_l - \rho_k \geq 1$

L'affaiblissement de certaines des contraintes telle qu'il est fait ne change pas fondamentalement le résultat mais permet simplement de l'améliorer quelque peu. La modification de certains cœfficients ne change pas la décomposition car le programme linéaire dual présenté ci-dessous garde notoirement les mêmes solutions optimales si les variables λ_i qui y apparaissent sont strictement positives. Le lemme 8 correspond au cas où λ_i vaut 1 pour tout i.

$$min \left\{ \sum_i \lambda_i * v_i \middle| [qvw] \geq 0, [qvw]. \begin{bmatrix} I_d & {}^tC & {}^tD \\ I_d & 0 & 0 \\ -I_d & 0 & 0 \end{bmatrix} = [1 \;\; 0 \;\; 0] \right\}$$

5.2.2 Construction de l'ordonnancement

Soit $H = (W, F)$ le graphe réduit de dépendance obtenu.

On construit récursivement un ordonnancement comme suit:

i. Appliquer à $H = (W, F)$ l'algorithme de décomposition (cf. 4.2.2) et obtenir $H' = (W', F')$;

ii. Obtenir l'hyperplan faiblement séparateur promis par le lemme 10 et défini par un vecteur x et des rationnels ρ_1, ρ_2, $\ldots$, ρ_m;

iii. Pour chaque sommet de $G = (V, E)$ prendre pour partie linéaire de l'ordonnancement le vecteur x et pour partie constante le ρ_i associé au sommet correspondant dans $H = (W, F)$;

iv. Appeler récursivement l'algorithme sur $H'' = (W'', F'')$ où H'' est le sous-graphe de H' défini par $F'' = F' \setminus \{e_i^j, j \in \{1, 2\} | e^3 \notin H'\}$, si $H'' \neq \emptyset$.

Dans la suite de cette partie, on appellera *arbre de décomposition* d'un graphe H l'arbre associé à sa décomposition par l'algorithme ci-dessus, et non celui associé à sa décomposition par l'action de l'algorithme 4.2.2.

Il reste à prouver que nous avons bien construit un ordonnancement.

5.3 Preuve de l'ordonnancement

L'algorithme récursif de construction de l'ordonnancement $(\mathcal{X}, \mathcal{C})$ finit bien puisque l'algorithme de décomposition (4.2.2) finit bien lui-même.

Il faut prouver que pour toute arête e du graphe de départ $G = (V, E)$ orientée de a vers b, on a $\mathcal{X}_a.w(e) + \mathcal{C}_a \gg_l \mathcal{X}_b + \mathcal{C}_b$.

Soient $H = (W, F)$ un graphe d'image $H' = (W', F')$ par l'algorithme de décomposition (étape(i)), le vecteur x et les rationnels ρ_1, ρ_2, ..., ρ_m associés par l'étape (ii). Prenons une arête e du graphe de départ $G = (V, E)$. On note $w(e) = u \oplus v \ominus w \otimes t$ le poids de e dans $G = (V, E)$. Au parcours de cette arête de G correspond un certain nombre (infini si $(v, w, t) \neq (0, 0, 0)$) de chemins possibles dans $H = (W, F)$. On note suivant le cas, e, $e^3 + e^4 + \sum_i \alpha_i * e_i^1 + \sum_i \beta_i * e_i^2$ ou $e^3 + e^4 + e^5 + \sum_i \alpha_i * e_i^1 + \sum_i \beta_i * e_i^2$ le parcours dans H étudié.

Comme pour l'algorithme 5.1.2, trois cas se présentent suivant le type de l'arête e, cas qui seront énumérés et examinés ci-dessous.

Pour prouver la validité de l'ordonnancement multidimensionnel produit on procède récursivement sur la composante de l'ordonnancement étudiée, en commençant par la première... Chaque cas se décompose lui-même en sous-cas. Dans un sous-cas, toutes les arêtes du parcours de H étudié appartiennent à H' et la composante concernée du délai sur e est nulle (cf. lemme 9). Comme le graphe G est supposé calculable, il en va de même de H et il arrivera un moment dans la décomposition où au moins une des arêtes du parcours de H considéré ne participera d'aucun multicycle de poids nul du sous-graphe de H alors étudié. Il arrivera donc avant que toutes les composantes de l'ordonnancement aient été passées au crible que l'on se trouve dans un autre de ces sous-cas, or dans tous les autres sous-cas la composante concernée du délai est supérieure à 1. On pourra donc conclure à la validité de l'ordonnancement.

Voici les trois cas:

i. $(v, w, t) = (0, 0, 0)$ L'arête e est alors élément de $H = (W, F)$, et donc $\mathcal{X}.w + \mathcal{C}_a - \mathcal{C}_b \gg_l O$. En effet, ou e n'est pas dans H' et $x.w + \rho_a - \rho_b \geq 1$, ou e est dans H' et alors $x.w + \rho_a - \rho_b = 0$ (cf. lemme 9) et la propriété doit être vérifiée à un niveau en-dessous.

ii. $(v, w, t) \neq (0, 0, 0)$ et $t = 0$

Deux cas de figure apparaissent alors:

(i) si $(e^3 \notin H' \wedge e^4 \notin H' \wedge e^5 \notin H')$[2] alors
$(x.W(e^3) \geq 1/2 + \rho_{a_1} - \rho_a)$ et
$(x.W(e^4) = x.0 = 0 \geq 1/2 + \rho_b - \rho_{a_1})$ et
$(\forall j \in \{1, 2\} \forall i \; x.W(e_i^j) \geq 0)$[3].
On en déduit $x.W(e^3) \geq 1 + \rho_b - \rho_a$ donc
$x.W(e^3) + x.W(e^4) + \sum_i \alpha_i * x.W(e_i^1) + \sum_i \beta_i * x.W(e_i^2) + \rho_a - \rho_b \geq 1$ et l'ordonnancement est valide sur l'arête e de départ. De plus est justifié l'appel récursif non pas sur H' mais sur H''.

(ii) si $(e^3 \in H')$ alors
$x.W(e^3) = \rho_{a_1} - \rho_a$ et
$x.W(e^4) = x.0 = 0 = \rho_{a_2} - \rho_{a_1}$
et donc $x.W(e^3) = \rho_b - \rho_a$. Si une des arêtes e_i^j n'est pas dans H', pour tout parcours de

2. en fait $e^3 \notin H' \Leftrightarrow e^4 \notin H'$ par connexité
3. si $e_i^j \in H'$, on a même $x.W(e_i^j) \geq 1$

e la contenant on a $x.W(e^3)+x.W(e^4)+\sum_i \alpha_i * x.W(e_i^1)+\sum_i \beta_i * x.W(e_i^2)+\rho_a-\rho_b \geq 1$, sinon $x.W(e^3)+x.W(e^4)+\sum_i \alpha_i * x.W(e_i^1)+\sum_i \beta_i * x.W(e_i^2)+\rho_a-\rho_b = 0$ et la propriété doit être vérifiée à un niveau en dessous.

iii. $t \neq 0$ Deux cas de figure apparaissent encore:

(i) si $(e^3 \notin H' \wedge e^4 \notin H' \wedge e^5 \notin H')$[4] alors
$(x.W(e^3) \geq 1/3 + \rho_{a_1} - \rho_a)$ et
$(x.W(e^4) = x.0 = 0 \geq 1/3 + \rho_{a_2} - \rho_{a_1})$ et
$(x.W(e^5) = x.0 = 0 \geq 1/3 + \rho_b - \rho_{a_2})$ et
$(\forall j \in \{1,2\} \forall i \; x.W(e_i^j) \geq 0)$[5].
On en déduit $x.W(e^3) \geq 1 + \rho_b - \rho_a$ donc
$x.W(e^3) + x.W(e^4) + x.W(e^5) + \sum_i \alpha_i * x.W(e_i^1) + \sum_i \beta_i * x.W(e_i^2) + \rho_a - \rho_b \geq 1$ et l'ordonnancement est valide sur l'arête e de départ. De plus est justifié l'appel récursif non pas sur H' mais sur H''.

(ii) si $(e^3 \in H')$ alors
$x.W(e^3) = \rho_{a_1} - \rho_a$ et
$(x.W(e^4) = x.0 = 0 = \rho_{a_2} - \rho_{a_1})$ et
$(x.W(e^5) = x.0 = 0 = \rho_b - \rho_{a_2})$
et donc $x.W(e^3) = \rho_b - \rho_a$. Si une des arêtes e_i^j n'est pas dans H', pour tout parcours de e la contenant on a $x.W(e^3)+x.W(e^4)+x.W(e^5)+\sum_i \alpha_i * x.W(e_i^1)+\sum_i \beta_i * x.W(e_i^2)+\rho_a - \rho_b \geq 1$, sinon $x.W(e^3)+x.W(e^4)+x.W(e^5)+\sum_i \alpha_i * x.W(e_i^1)+\sum_i \beta_i * x.W(e_i^2)+\rho_a-\rho_b = 0$ et la propriété doit être vérifiée à un niveau en dessous.

5.4 Généralisation aux nids de boucles non parfaits

Nous allons proposer ici une extension de l'algorithme précédent aux nids de boucles non-parfaits. Cette extension repose sur la réécriture des nids de boucles non-parfaits en nids de boucles parfaits.

5.4.1 Schématisation d'un nid de boucles non-parfait

Les nids de boucles non parfaits peuvent être décrits à partir du schéma de boucle suivant dénoté T:

```
DO i⃗ = k⃗1 to t⃗1
   S1(i⃗)
   DO j⃗1 = l⃗1 to u⃗1
      T1(i⃗, j⃗1)
   ENDDO
   S2(i⃗)
   DO j⃗2 = l⃗2 to u⃗2
      T2(i⃗, j⃗2)
   ENDDO
```

4. en fait $e^3 \notin H' \Leftrightarrow e^4 \notin H' \Leftrightarrow e^5 \notin H'$ par connexité
5. si $e_i^j \in H'$, on a même $x.W(e_i^j) \geq 1$

```
    ...
    Sp(i)
    DO jp = lp to up
        Tp(i, jp)
    ENDDO
ENDDO
```

où les variables S_i désignent des NOP ou des blocs d'instructions ne contenant pas de boucles et où les variables T_i représentent soit des NOP [6], soit des blocs d'instructions ne contenant pas de boucles, soit des nids de boucles non-parfaits de la forme T.

5.4.2 Algorithme de transformation des nids de boucles non-parfaits

Pour réécrire un nid de boucles non-parfait en un nid de boucles parfait nous allons procder récursivement en partant des nids non-parfaits de plus grande profondeur de la forme T pour lesquels toutes les variables T_i désignent soient des NOP, soient des blocs d'instructions ne contenant pas de boucles.

Le nouveau nid de boucles a pour profondeur la somme de la profondeur P du nid englobant et des profondeurs des différents nids parfaits englobés, soit:

$$P = dim(\vec{i}) + \sum_{k=1}^{p} dim(\vec{j_k})$$

Dans ce nouveau nids les instructions:

$S'_k(\vec{i}, \vec{j_1}, \ldots, \vec{j_p}) = IF(\ (\vec{j_1} = \vec{u_1}) \wedge \ldots \wedge (\vec{j}_{k-1} = \vec{u}_{k-1}) \wedge (\vec{j_k} = \vec{u_k}) \wedge \ldots \wedge (\vec{j_p} = \vec{u_p}))$
THEN $S_k(\vec{i})$

remplacent les instructions $S_k(\vec{i})$, et les instructions:

$T'_k(\vec{i}, \vec{j_1}, \ldots, \vec{j_p}) = \text{IF}\ ((\vec{j_1} = \vec{u_1}) \wedge \ldots \wedge (\vec{j}_{k-1} = \vec{u}_{k-1}) \wedge (\vec{j}_{k+1} = \vec{u}_{k+1}) \wedge \ldots \wedge (\vec{j_p} = \vec{u_p}))$
THEN $T_k(\vec{i})$

remplacent les instructions $T_k(\vec{i})$.

Les instructions S'_k et T'_k sont ordonnées entre elles dans le nouveau nid de boucles exactement comme les instructions S_k et les nids de boucles appelant les T_k l'étaient dans le nid initial.

Il est facile de vérifier que la sémantique du nid de boucles initial est bien conservée.

Pour fixer les idées, voici l'image du nid de boucles non-parfait T par l'algorithme:

```
DO i = k1 to t1
    DO j1 = l1 to u1
        DO j2 = l2 to u2
            ...
            DO jp = lp to up
                IF  ((j1 = l1) ∧ (j2 = l2) ∧ ... ∧ (jp = lp))
                    THEN S1(i)
                ENDIF
                IF  ((j2 = l2) ∧ ... ∧ (jp = lp))
```

6. "No Operation":l'instruction vide

THEN $T_1(\vec{i}, \vec{j_1})$
ENDIF
IF $((\vec{j_1} = \vec{u_1}) \wedge (\vec{j_2} = \vec{l_2}) \wedge \ldots \wedge (\vec{j_p} = \vec{l_p}))$
THEN $S_2(\vec{i})$
ENDIF
IF $((\vec{j_1} = \vec{u_1}) \wedge (\vec{j_3} = \vec{u_3}) \wedge \ldots \wedge (\vec{j_p} = \vec{l_p}))$
THEN $T_2(\vec{i}, \vec{j_2})$
ENDIF
...
IF $((\vec{j_1} = \vec{u_1}) \wedge \ldots \wedge (\vec{j_{p-1}} = \vec{u_{p-1}}) \wedge (\vec{j_p} = \vec{l_p}))$
THEN $S_p(\vec{i})$
ENDIF
IF $((\vec{j_1} = \vec{u_1}) \wedge \ldots \wedge (\vec{j_{p-1}} = \vec{u_{p-1}}))$
THEN $T_p(\vec{i}, \vec{j_p})$
ENDIF
ENDDO
...
ENDDO
ENDDO
ENDDO

5.4.3 Expression des nouveaux vecteurs de dépendances

Les nouveaux vecteurs de dépendances se réécrivent simplement en fonctions des anciens:

- Une dépendance entre S_k et S_l (resp. T_l) de vecteur $(\vec{d})$, $\vec{d} \in (\mathbb{Z}, +, -, *)^{d}im(\vec{i})$, devient une dépendance entre S'_k et S'_l (resp. T'_l) de vecteur:
$(\vec{d}, \underbrace{*, \ldots, *}_{(\sum_{m=1}^{p} dim(\vec{j_m}))})$.
- Une dépendance entre T_k et T_l (resp. S_l) de vecteur $(\vec{d})$, $\vec{d} \in (\mathbb{Z}, +, -, *)^{d}im(\vec{i})$, devient une dépendance entre T'_k et T'_l (resp. S'_l) de vecteur:
$(\vec{d}, \underbrace{*, \ldots, *}_{(\sum_{m=1}^{p} dim(\vec{j_m}))})$.
- Une dépendance interne au bloc T_k de vecteur $(\vec{d})$, $\vec{d} \in (\mathbb{Z}, +, -, *)^{dim(\vec{i})+dim(\vec{j_k})}$, devient une dépendance au bloc T'_k de vecteur:
$(d_1, \ldots, d_{dim(\vec{i})}, \underbrace{*, \ldots, *}_{(\sum_{m=1}^{k-1} dim(\vec{j_m}))}, d_{dim(\vec{i})+1}, \ldots, d_{dim(\vec{i})+dim(\vec{j_k})}, \underbrace{*, \ldots, *}_{(\sum_{m=k+1}^{p} dim(\vec{j_m}))})$

5.4.4 Efficacité

Maintenant que nous nous sommes ramenés au cas précédent nous pouvons trouver un ordonnancement multidimensionnel. D'aucuns pourraient craindre qu'au pire cas l'ordonnancement obtenu soit de dimension P, ils auraient tort, car:

Théorème 10 *Dans le cas des nids de boucles non-parfaits l'ordonnancement multidimensionnel obtenu après action successives des algorithmes 5.4.2, 5.1.2 et 5.2.2 est au plus de dimension égale à la profondeur de l'instruction la plus profonde dans le nid non-parfait initial.*

Preuve: Soit d un vecteur de dépendance du nid de boucles perfectionné et soit x un hyperplan faiblement séparateur obtenu lors de l'ordonnancement, alors si la $k^{ème}$ composante de d est le symbole "*", elle peut prendre n'importe quelles valeurs infiniment grandes positives et négatives et donc la $k^{ème}$ composante de x doit être nulle. Considérons un bloc d'instruction T'_k, il était initialement de profondeur $dim(\vec{i}) + dim(\vec{j_k})$ et les $l^{èmes}$ composantes de ses vecteurs telles que $dim(\vec{i}) + 1 \leq l \leq dim(\vec{i}) + \sum_{m=1}^{k-1} dim(\vec{j_m})$ ou $dim(\vec{i}) + \sum_{m=1}^{k} dim(\vec{j_m} \leq l$ valent "*". L'espace engendré par le vecteur des parties linéaires de son ordonnancement multidimensionnel est donc au plus de dimension $dim(\vec{i}) + dim(\vec{j_k})$ d'où le résultat par récurrence et par utilisation du théorème 8 (indépendance des hyperplans faiblement séparateurs). □

Une objection possible à la transformation et à l'ordonnancement présentés ici est l'inflation du nombre d'instructions. En effet, partant au départ d'une instruction S de profondeur p on la remplace par une instruction S' de profondeur P. On notera I la partie du vecteur d'indiçage de S' correspondant aux coordonnées du vecteur d'indiçage qui pilotait S, et J le reste du vecteur d'indiçage. $S'(I, J)$ est alors de la forme: IF $J = c$ THEN $S(I)$, où c est un vecteur indépendant des indices de boucles.

En fait, l'inflation apparente du nombre d'instructions n'est que passagère, en effet, l'ordonnancement trouvé pour $S'(I, J)$ ne concerne que les coordonnées de I (toujours le problème des "*" qui induisent des zéros), et donc, ces coordonnées I fixées par l'ordonnancement, le parallélisme est total sur les coordonnées de J et les instructions apparemment rajoutées disparaissent par équivalence des deux nids suivants:

```
FOR conditions_sur(I) DO
    FORALL J DO
            S'(I, J)
    ENDFORALL
ENDFOR
```

```
FOR conditions_sur(I) DO
    S'(I, c)
ENDFOR
```

5.4.5 Autre approche

Une autre approche des nids de boucles non-parfaits aurait consisté à ordonnancer le nid de boucles le plus externe, puis récursivement les niveaux plus internes, c'est-à-dire de n'ordonnancer à chaque fois que sur les variables d'indiçage communes. En se replaçant dans la description par l'exemple dénoté T, cela revient à considérer chaque nid de boucles interne T_i comme un bloc d'instructions et à trouver un ordonnancement valide pour l'ensemble de ces blocs en ne considérant que les vecteurs de dépendances entre blocs d'instructions et ceux internes au blocs d'instruction qui ne mettent en jeu que des indices des boucles communes (les autres coordonnées des vecteurs de dépendance valent dans ce cas "*"), puis, ceci fait, de refaire le même travail pour le niveau d'imbrication suivant.

Cette approche qui est une sorte de généralisation de l'algorithme de Kennedy et Allen [KA84] est équivalente à la précédente où la présence systématique du symbole "*" dans les vecteurs de dépendance provoque exactement les mêmes restrictions que l'étude récursive sur les nids imbriqués.

5.5 Conclusion

L'algorithme d'ordonnancement proposé ici découle directement de celui qui avait été proposé pour traiter les équations récurrentes uniformes. De même il devrait facilement être transposable aux nids de boucles à dépendances non nécessairement uniformes dont les cônes de dépendances sont décrits par les rayons et sommets.

Il reste encore à prouver que l'algorithme utilisé pour les nids de boucles non-uniformes extrait tout le parallélisme de la description qui lui est donnée.

Chapitre 6

Conclusion

6.1 Paternité

L'algorithme de décomposition des graphes de dépendance des Equations Récurrentes Uniformes 4.2.2 présenté ici est une traduction de celui de Sailesh Rao [Rao85] qui s'inspirait lui-même de celui de Karp, Miller et Winograd [KMW67] qui étaient, eux, quelque peu plus généraux puisqu'ils s'intéressaient aux domaines infinis dans le quadrant positif et donc à l'existence de cycles de poids négatif.

La preuve de cet algorithme est une transposition de celle présentée par Karp pour son propre algorithme (la preuve de Rao est insuffisante et induit en erreur le lecteur).

La preuve du théorème sur la latence minimale (théorème 5) s'inspire directement du principe de celle fournie par Rao. La justification par Rao de l'appartenance du chemin construit au domaine est insuffisante. Ce théorème a donc été redémontré, la preuve présentée ici se voulant juste sur la question de l'appartenance du chemin au domaine, ce qui explique d'ailleurs son caractère quelque peu monstrueux.

Paul Feautrier [Fea92] prétendait présenter une *heuristique gloutonne*, nous avons montré ici qu'elle ́tait optimale.

6.2 Apports personnels

En rapprochant les travaux de Sailesh Rao de ceux de Paul Feautrier nous avons pu construire un algorithme d'ordonnancement multidimensionnel optimal pour les systèmes d'Equations Récurrentes Uniformes.

Après avoir fait le lien entre les dépendances affines décrites dans $(\mathbb{Z}, +, -, *)$ et les équations récurrentes uniformes, nous avons présenté ici un algorithme simple d'ordonnancement multidimensionnel des Equations Récurrentes Affines, et donc des nids de boucles affines. Cet algorithme ne nécessite pas de connaître les dépendances exactes mais simplement des dépendances approchées. Néanmoins il reste efficace dans la mesure où il ne perd pas de parallélisme. Il s'agit donc plus d'un algorithme que d'une heuristique.

6.3 Questions ouvertes

Nous avons montré que l'algorithme d'ordonnancement proposé ici pour les Equations Récurrentes Uniformes était optimal. Nous avons en fait montré que si le chemin le plus long du graphe de dépendance était en $O(N^d)$, il en allait de même de la complexité de l'ordonnancement généré. Il resterait à déterminer le rapport entre les deux constantes multiplicatives, s'il existe...

Il reste de même à affiner l'étude de l'ordonnancement pour les Equations Récurrentes Affines, en particulier dans le cas des nids de boucles non-parfaits. Il serait souhaitable que l'algorithme proposé ici soit étendu aux cas des dépendances approchées décrites par des cônes, peut-être avec le symbolisme de Vincent Van Dongen.

D'un autre côté il reste toujours à démontrer LA conjecture: l'optimalité asymptotique de l'ordonnancement affine de partie linéaire unique.

Remerciements

Je tiens avant tout à remercier Alain qui a eut la bonté de bien vouloir m'abandonner ce très joli sujet. Je lui en suis d'autant plus reconnaissant que je sais ce qu'il lui en a coûté...

Je me dois de remercier tout aussi chaleureusement Yves qui a eut le courage, l'abnégation et l'inconscience d'accepter dans son équipe un râleur de première classe qui passe la moitié de son temps à rouspéter et l'autre à avoir raison.

Je remercie tous ceux que j'ai pu embêter au cours de ce stage pour des raisons plus ou moins futiles, et tout particulièrement: Claire, Michèle et Sylvie, Eric, Hervé, Pierre, Pierre-Yves et Tanguy.

Bibliographie

[Ban88] Utpal Banerjee. An introduction to a formal theory of dependence analysis. *The Journal of Supercomputing*, 2:133–149, 1988.

[Dar93] Alain Darte. *Techniques de parallélisation automatique de nids de boucles.* PhD thesis, Ecole Normale Supérieure de Lyon, 1993.

[DKR91] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[Fea92] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II, multidimensional time. *Int. J. Parallel Programming*, 21(6):389–420, December 1992. Available as Technical Report 92-78, Laboratoire MASI, Université Pierre et Marie Curie, Paris, October 1992.

[KA84] Ken Kennedy and J.R. Allen. Automatic loop interchange. In SIGPLAN Notices, editor, *Proceedings of the SIGPLAN Symposium on Compiler Construction*, volume 19-6, pages 233–246, Montreal, Canada, june 1984.

[KMW67] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, jul 1967.

[KS88] S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In ACM Press, editor, *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 398–406, May 1988.

[Lam74] Leslie Lamport. The parallel execution of DO loops. *Communications of the ACM*, 17(2):83–93, feb 1974.

[LS83] C.E. Leiserson and J.B. Saxe. Optimizing synchronous systems. *Journal of VLSI Computers Systems*, 1:41–67, apr 1983.

[Rao85] Sailesh K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, oct 1985.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1986.

[Xuo92] Nguyen Huy Xuong. *Mathématiques Discrètes et Informatique.* Masson, 1992.