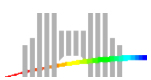
Génération de Code Parallèle

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

15 décembre 2011



Génération de code parallèle

- Paul Feautrier,
- basé sur des travaux de J. Fourier, C. Ancourt, F. Irigoin, A. Darte, JingLing Xue, T. Risset, P. Boulet, etc.

Plan

- Présentation du problème
- Parcours d'un polyèdre
- Parcours d'un Z-polyèdre
- Parcours d'une union de Z-polyèdres
- La méthode de Boulet et Feautrier

Présentation du problème, I

Parallélisation à l'ancienne

```
for(i=0; i<=n-1; i++){
  c[i] = 0.0;
  for(j=0; j<=n-1; j++)
    c[i] += a[i][j]*b[j];
}
```

$\Rightarrow$

```
//for(i=0; i<=n-1; i++){
  c[i] = 0.0;
  for(j=0; j<=n-1; j++)
    c[i] += a[i][j]*b[j];
}
```

Le code parallèle est "isomorphe" au code séquentiel.

Référence Allen et Kennedy, RR. Rice U. 1982
ACM Toplas, 1987

Présentation du problème, II

Parallélisation polyédrique

```
for(i=0; i<=n-1; i++)
  for(j=0; j<=n-1; j++)
    a[i][j] = (a[i-1][j]+a[i][j]+a[i][j-1])/3.0;
```

- Aucune boucle n'est paralléle
- Pour paralléliser, on doit appliquer la transformation

$$t = i + j, \quad p = j$$

- Où est le code parallèle ?

Référence Lamport, CACM, 1974

Un autre exemple

```
for(i=0; i<n; i++)
 for(j=0; j<i; j++)
 S : c[i] += a[i][j]*b[j];
```

$$D_S = \{[i,j] \mid 0 \leq i < n, 0 \leq j < i\}$$

- Pour paralléliser, on applique une transformation qui regroupe autant de dépendances que possible sur la boucle extérieure.

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad T(D_S) = \{[y,x] \mid 0 \leq x < n, 0 \leq y < x\}$$

- Le problème est d'écrire un nid de boucle qui parcourt $T(D_S)$ dans l'ordre lexicographique. La première boucle sera séquentielle et la seconde parallèle.
- Pour le choix de T, voir l'exposé d'Alain Darte.

Classification des Objets Polyédriques

Un polyèdre est l'ensemble des points (ou des vecteuts) qui satisfont un système d'inégalités affines : en notation matricielle

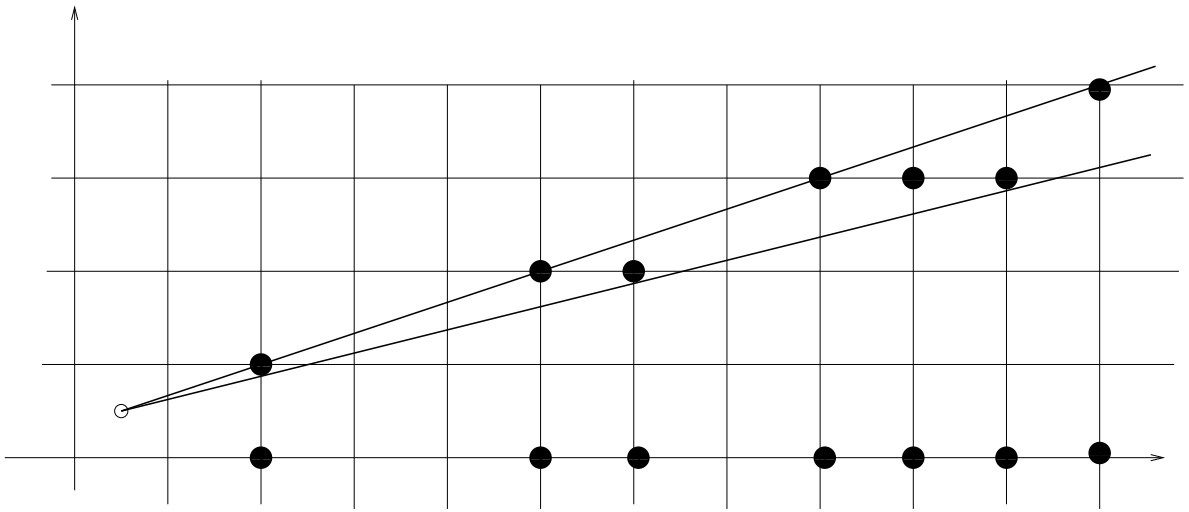
$$Ax + b \geq 0$$

- On dit polytope si les solutions sont bornées
- On précise parfois : polyèdre convexe
- Un polyèdre peut être également défini comme la coque convexe d'un nombre fini de points (théorème de Minkowsky)

Un Z-polyèdre est l'ensemble des point de coordonnées entières contenus dans un polyèdre.
L'image d'un polyèdre par une transformation affine est un polyèdre. L'image d'un Z-polyèdre par une transfomation affine n'est pas toujours un Z-polyèdre !

Exemple : la Projection d'un Z-polyèdre

La projection est une transformation affine très fréquente.



La projection rationelle du polyèdre contient des points entiers qui ne sont pas projection de points entiers du polyèdre.
On parle de "Linearly Bounded Lattice" (Lothar Thiele) :

$$P = \{y \mid \exists x, y = Tx, Ax + b \geq 0, x, y \in \mathbb{N}\}$$

Classification, suite

$T(D)$ n'est pas toujours un Z-polyèdre : celà dépend de T.

- Un polyèdre ; transformation unimodulaire.
  - Une transformation unimodulaire est caractérisée par une matrice de déterminant $+1$ ou -1.
  - T^{-1} est entière.
  - $T(D)$ est un Z-polyèdre.
- Transformation non unimodulaire :
  - $T(D)$ est un polyèdre "à trous" ou LBL.
- Nid de boucle imparfait : union de LBL-polyèdres.
  - Dans un nid de boucles imparfait, il y a autant de domaines d'itération et de transformations que d'instructions.
  - Il faut parcourir $T_1(D_1), \ldots T_n(D_n)$.
  - $T_i(D_i)$ est un LBL-polyèdre.

Cas d'un seul polyèdre ; la méthode de Fourier-Motzkin

Un polyèdre est défini par un système d'inégalités :

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 &\geq a_{1,0}, \\ a_{2,1}x_1 + a_{2,2}x_2 &\geq a_{2,0}, \\ a_{3,1}x_1 + a_{3,2}x_2 &\geq a_{3,0}. \end{aligned}$$

dans le cas d'un polyèdre à deux variables et trois contraintes. On suppose que x_1 est le compteur de la boucle la plus interne.

- Si $a_{11} > 0$, on a la borne inférieure $x_1 \geq (a_{10} - a_{12}x_2)/a_{11}$.
- Si $a_{21} = 0$, on n'a aucune information sur x_1.
- Si $a_{31} < 0$ on a la borne supérieure $x_1 \leq (a_{30} - a_{32}x_2)/a_{31}$.

Dans le cas général :

$$\lceil \max_{a_{i1}>0}(a_{i0} - a_{i2}x_2 - \ldots)/a_{i1} \rceil \leq x_1 \leq \lfloor \min_{a_{i1}<0}(a_{i0} - a_{i2}x_2 - \ldots)/a_{i1} \rfloor.$$

- On élimine x_1 en écrivant :

$$(a_{i0} - a_{i2}x_2 - \ldots)/a_{i1} \leq (a_{j0} - a_{j2}x_2 - \ldots)/a_{j1}$$

  pour toute paire $a_{i1} > 0, a_{j1} < 0$
- On ajoute les contraintes correspondant à $a_{i1} = 0$ et on recommence pour x_2.

.

Complexité : $n(\frac{m}{2})^{2^n}$.
Le résultat n'est pas toujours le plus simple possible.

Propriétés de l'algorithme de Fourier-Motzkin

- L'algorithme peut échouer, si on arrive à fabriquer une inégalité absurde, du genre $1 \leq 0$. Dans ce cas, la boucle "ne tourne pas".
- Sinon, en remontant les étapes d'élimination, on obtient un intervalle non vide pour la variable éliminée. On choisit un point dans cet intervalle et on passe à la variable précédente.
- Si on choisit systématiquement la borne inférieure de l'intervalle, on obtient le minimum lexicographique du polyèdre.
- Si à chaque étape on "arrondit" les bornes (dans le bon sens) on peut énumérer les points entiers du polyèdre.

Exemple : l'inversion de boucle

$$D = \{[y, x] \mid 0 \leq x \leq n-1, 0 \leq y \leq x-1\}$$

On sélectionne les bornes pour x.

$$0 \leq x \leq n-1,$$

$$y+1 \leq x.$$

D'où les bornes :

$$\max(0, y+1) \leq x \leq n-1$$

On élimine x :

$$0 \leq n - 1,$$
$$0 \leq y \leq n - 2.$$

D'où les bornes :

$$0 \leq y \leq n - 2.$$

ce qui permet de simplifier la borne inférieure de x. D'où le programme :

```
for(y=0 ; y<n-1 ; y++)
    for(x =y+1 ; x<n ; x++)
        . . .
```

Référence C. Ancourt, F. Irigoin : *Scanning Polyhedra with DO loops* , PPoPP, 1991.

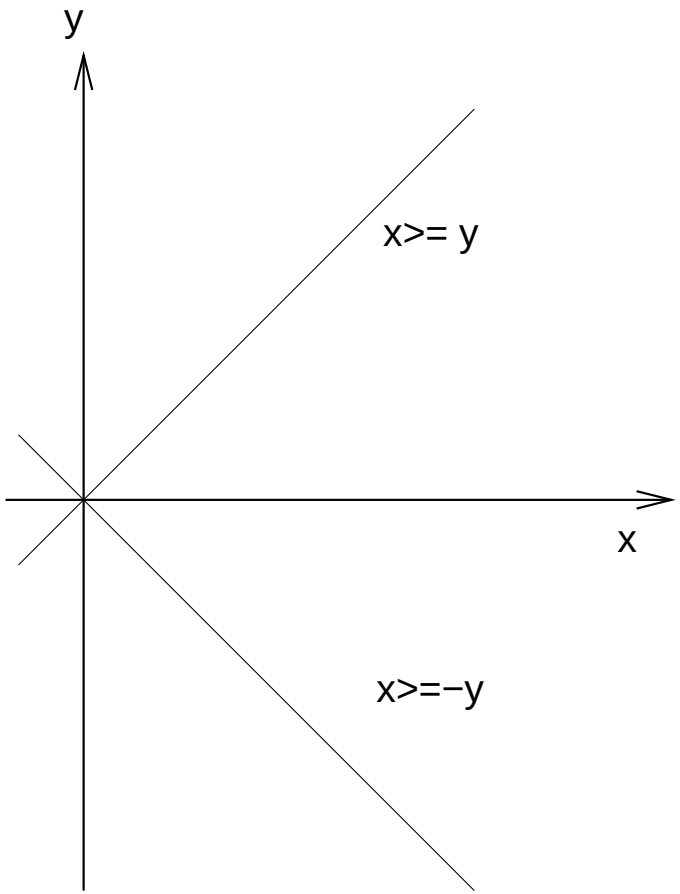
Programmation linéaire paramétrique

Trouver le minimum lexicographique d'un polyèdre dépendant de paramètres :

$$Q(y) = \min_{\ll}\{x \mid Ax + By + c \geq 0\} \tag{1}$$

- x et y sont des vecteurs, A et B des matrices, c un vecteur.
- La position du minimum dépend de la valeur du paramètre.
- Découpages successifs du domaine des paramètres. La solution se présente comme une conditionnelle.

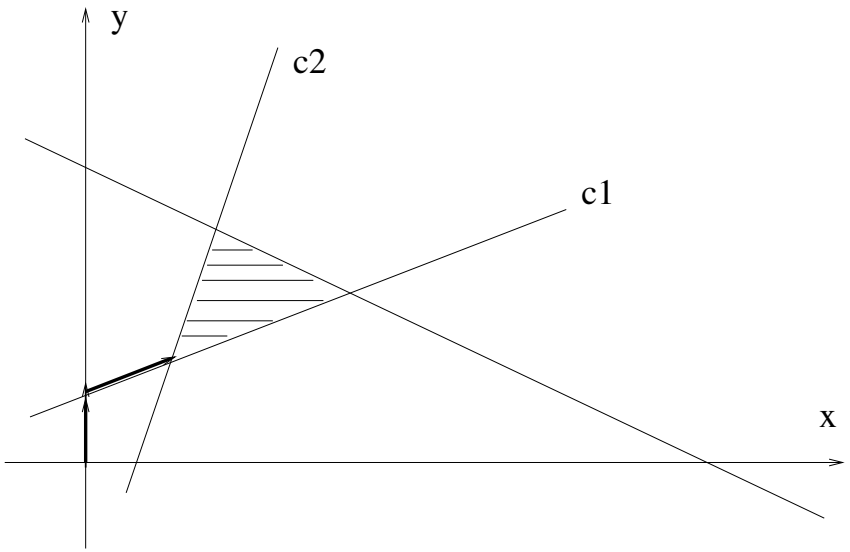
Exemple



$$\text{If } y \geq 0 \text{ Then } x \text{ Else } -x$$

Références P. Feautrier, *Parametric Integer Programming* RAIRO-RO, 1988.

Algorithme du Simplex, Interprétation géométrique



- Le minimum est nécessairement en un sommet, et un sommet est l'intersection de n contraintes (ici $n = 2$).
- Comme on a les contraintes $x \geq 0$ et $y \geq 0$ (implicite) on teste d'abord l'origine, qui n'est pas dans le polyèdre.
- On décide alors de remplacer l'axe des x par la contrainte c_1. le nouveau sommet n'est pas dans le polyèdre.
- on remplace alors l'axe des y par c_2. Le nouveau sommet est dans le polyèdre, et c'est le minimum cherché.
- il existe des règles pour choisir la nouvelle contrainte qui garantissent que l'on ne passe jamais deux fois par le même sommet (donc que l'algorithme termine), et que le premier sommet "faisable" est le minimum.
- L'algorithme s'étend au cas paramétrique.

Application au parcours d'un polyèdre

Pour chaque niveau de boucle, on détermine un minimum et un maximum, les compte-tours des boucles englobantes étant considérés comme des paramètres.

$$D_0(n) = \{[y, x] \mid 0 \leq x \leq n-1, 0 \leq y \leq x-1\}$$

$$\min_{\ll} D_0(n) = \textbf{if } n \geq 2 \textbf{ then } [0,0] \textbf{ else } \bot.$$

$$\max_{\ll} D_0(n) = \textbf{if } n \geq 2 \textbf{ then } [n-2, n-1] \textbf{ else } \bot.$$

$$\min_{\ll} D_1(y, n) = \textbf{if } 0 \leq y \leq n-2 \textbf{ then } [y+1] \textbf{ else } \bot.$$

$$\max_{\ll} D_1(y, n) = \textbf{if } 0 \leq y \leq n-2 \textbf{ then } [n-1] \textbf{ else } \bot.$$

- On retrouve la même boucle que ci-dessus.
- La condition $n \geq 2$ n'a pas besoin d'être explicitée.
- Il en est de même pour la condition $0 \leq y \leq n-2$. On peut éliminer celle-ci directement en utilisant le système de *contexte* de PIP. Les simplifications deviennent automatiques.

Parcours d'un Z-polyèdre

Il y a deux façons de définir un Z-polyèdre :

- $$P = \{y \mid \exists x, y = Tx, Ax + b \geq 0, x, y \in \mathbb{N}\}$$
- $$Q = \{y \mid \exists x, y = Tx, Ay + b \geq 0, x, y \in \mathbb{N}\}$$
- La première définition (les LBL de Lothar Thiele) est plus générale et correspond mieux aux problèmes rencontrés en génération de code.
- L'utilisation de la deuxième forme n'apporte pas de simplification significatives.

Forme Normale de Hermite

Toute matrice non singulière à coefficients entiers T peut être mise sous forme normale de Hermite :

$$T = HU,$$

où U est unimodulaire et où H :

- Est triangulaire inférieure à éléments positifs,
- Les éléments diagonaux dominent les autres éléments.

Méthode On applique à T une succession de transformations élémentaires (changement de signe, échange de lignes, torsion) jusqu'à obtenir la forme désirée. Toutes ces transformations sont unimodulaires. On calcule U en appliquant à la matrice unité les transformations inverses.

Références Newman : *Integral Matrices*
Alain Darte : Thèse, Lyon, 1993.

Parcours d'un Z-polyèdre

- On calcule H et U.
- $U(D)$ est un polyèdre parce que U est unimodulaire. On pose :

$$x \in D,\ y = Tx = HUx,\ z = Ux,\ y = Hz.$$

- H respecte l'ordre lexicographique parce qu'elle est triangulaire inférieure et que ses coefficients diagonaux sont positifs.
- On écrit la boucle qui fait parcourir à z le polyèdre $U(D)$ par l'une des méthodes précédentes.
- On calcule la transformée $y = Hz$,
- Ou bien on applique H directement aux bornes des boucles. Les éléments diagonaux de H donnent les pas des boucles.

Soit le programme :

```
for(i=0; i<n; i++)
    for(j=0; j<m j++)
       S;
```

à qui on doit appliquer la transformation

$$T = \begin{pmatrix} 1 & -1 \\ 1 & 2 \end{pmatrix}$$

La décomposition de Hermite de T est :

$$H = \begin{pmatrix} 1 & 0 \\ 1 & 3 \end{pmatrix}.$$

$$U = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}.$$

Le domaine $U(D)$ est :

$$U(D) = \{[z_1, z_2] \mid 0 \leq z_1 + z_2 \leq n-1, 0 \leq z_2 \leq m-1\}$$

La méthode de Fourier donne la boucle :

```
for(z1=1-m; z1 < n; z1++)
    for(z2=max(0,-z1); z2 < min(m, n-z1); z1++) {
        y1 = z1;
        y2 = z1 + 3 * z2;

    }
```

que l'on peut également écrire :

```
for(y1 = 1-m; y1<n; y1++)
  for(y2=y1+3*max(0,-y1); y2 < y1 + 3*min(m, n-y1); y2 += 3)
```

Parcours d'une union de Z-polyèdres

Le problème est très difficile dans le cas général :

- S'arranger pour que les images aient le même nombre de dimensions.
- Construire plus ou moins approximativement l'union des images (coque convexe, plus petit parallélipipéde rectangle englobant.
- Ecrire le code de parcours de ce parallélipipède.
- Le corps de boucle est composé de toutes les instructions du programme original avec une garde pour qu'elles ne soient exécutées que là où il faut.

$$\begin{aligned} \text{For} \quad y \quad & \in T_1(D_1) \cup T_2(D_2) \cup \ldots \\ & \textbf{if } y \in T_1(D_1) \textbf{ then } S_1; \\ & \textbf{if } y \in T_2(D_2) \textbf{ then } S_2; \end{aligned}$$

Inconvénients et améliorations

- Le calcul des *gardes* : $y \in T_k(D_k)$ est une perte de temps. Il peut suffire à annuler l'avantage obtenu par une optimisation ou une parallélisation.
- On peut tenter d'éviter ces gardes en les remontant dans le nid de boucle :

```
for(i=0; y<2*n; i++)
  for(j=0; j<n; j++){
    if(i<n) S1;
    else S2;
}
```

```
for(i=0; y<2*n; i++)
  if(i<n}
    for(j=0; j<n; j++)
      S1;
  else
    for(j=0; j<n; j++)
      S2;
}
```

On peut maintenant couper la boucle externe en deux :

```
for(i=0; y<n; i++)
  for(j=0; j<n; j++)
      S1;
for(i=n; i<2*n; i++)
  for(j=0; j<n; j++)
      S2;
```

La méthode est difficile à formaliser.

La Méthode de Quilleré-Bastoul

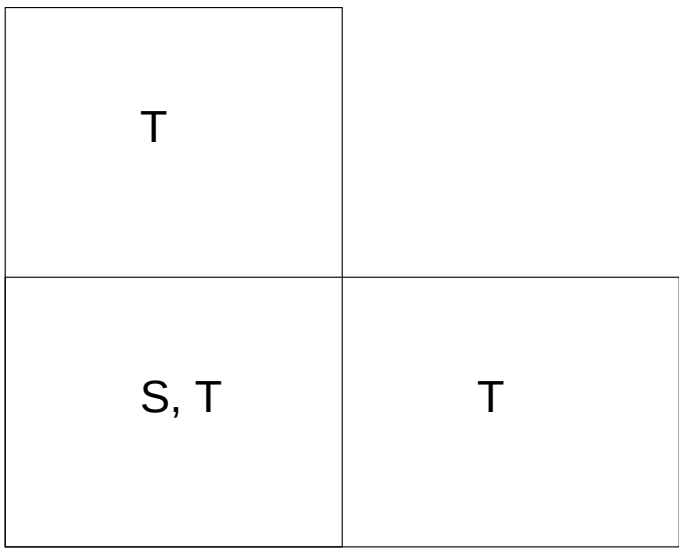
Objectif : minimiser le nombre de gardes.

Synopsis de l'algorithme :

- Au lieu d'appliquer la transformation T, on la considère comme un renommage. Au lieu de parcourir $T(D)$, on parcourt $\{\langle t, i\rangle \mid t = T(i), i \in D\}$. Si T est inversible, les boucles sur i ne font qu'un tour.
- On projette l'ensemble des polyèdres à parcourir sur la première dimension.
- On calcule toutes les intersections deux à deux jusqu'à obtenir une union de polyèdres (segments) disjoints. Dans chaque polyèdre, les instructions à exécuter sont fixées.
- On trie les segments dans l'ordre croissant.
- On connait tout ce qu'il faut pour écrire la séquence de boucles qui parcourt la première dimension.
- On recommence successivement pour chaque polyèdre et pour la dimension suivante.

Exemple

Soit une instruction S, dont le domaine est $\{0 \leq i \leq 100, 0 \leq j \leq 10 \mid \}$ et une instruction T, dont le domaine est $\{0 \leq i \leq 50, 0 \leq j \leq 20 \mid \}$.



- Les deux projections sont $[0, 50]$ et $[0, 100]$.
- Les segments disjoints sont $[0, 50]$ et $[51, 100]$.
- D'où les boucles :

```
for(i=0; i<=50; i++)
    ...
for(i=51; i<=100; i++)
    ...
```

Exemple, suite

- Au niveau suivant, le deuxième intervalle conduit à une seule boucle.
- Le premier intervalle conduit à deux boucles, correspondant aux deux intervalles $[0, 10]$ et $[11, 20]$.

```
for(i=0; i<=50; i++){
  for(j=0; j<=10; j++){
    S;
    T;
  }
  for(j=11; j<=20; j++)
    T;
}
for(i=51; i<=100; i++)
  for(j=0; j<=10; j++)
    S;
```

Noter :

- l'absence de gardes,
- la duplication du code.

Gardes Résiduelles

La projection est une opération rationnelle. Tous les points entiers de la projection d'un polyèdre ne sont pas projection de points entiers du polyèdre.
Il faut donc construire des gardes résiduelles pour éliminer les fausses projections.

Références Cédric Bastoul : *Code Génération in the Polytope Model is Easier than You Think*, PACT 2004.

Exemple

Deux polyèdres :

$$\begin{pmatrix} t = 2i \\ 0 \leq i \leq n \end{pmatrix} \quad \begin{pmatrix} t = 2i+1 \\ 0 \leq i \leq n \end{pmatrix}$$

- On projette sur t et on découpe. On trouve les deux points $(0,0)$ et $(2n+1, n)$ et le segment $[1, 2n]$.
- Dans le segment, pour le premier polyèdre, on a $t/2 \leq i \leq t/2$. Mais $t/2$ n'est entier que si $t \mod 2 = 0$. Il faut une garde.
- Pour le deuxième polyèdre, la contrainte est $t \mod 2 = 1$.

```
S1(i<-0);
for(t=1; t<2*n; t++){
  if(t%2 ==1) S2(i<- (t-1)/2);
  if(t%2 ==0) S1(i<- t/2);
  }
S2(i<-n);
```

Cas particulier

Si les deux tests sont les mêmes, on peut les éliminer et les remplacer par un *pas*.

```
for(t=0; t<2*n; t+=2){
  S1(i<-t/2);
  S2(i<-t/2);
  }
```

Modifier l'ordonnancement

Mais il est plus efficace d'agir sur l'ordonnancement.

Par exemple, il vaut mieux prendre $\begin{pmatrix} i \\ 0 \end{pmatrix}$ et $\begin{pmatrix} i \\ 1 \end{pmatrix}$ que $2i$ et $2i+1$: on trouve directement le meilleur code possible.

Et le parallélisme ?

Comment retrouver le parallélisme dans le résultat de l'algorithme Quilleré-Bastoul ?

Règle Le parallélisme se déduit des itérateurs de boucles.

- Les boucles qui portent sur une variable d'ordonnancement sont séquentielles.
- Les boucles qui portent sur une variable de placement (ou allocation) sont parallèles.
- Les boucles qui portent sur les variables originale du programme, si elles subsistent, sont parallèles.

Difficultés L'implémenation standard de l'algorithme change les noms des comptes-tours.

Certaines boucles disparaissent.

La Méthode de Boulet-Feautrier, I

Principe :

- On doit parcourir les points entiers d'un certain domaine D dans l'ordre lexicographique.
- On détermine le premier point du domaine, $first() = \min_{\ll} D$.
- Sachant que l'on a atteint le point $x \in D$, on cherche quel est le point suivant :

$$next(x) = \min_{\ll}\{y \in D \mid x \ll y\}.$$

Référence P. Boulet et P. Feautrier : *Scanning Polyhedra Without DO Loops* PACT, 1998.

La Méthode de Boulet-Feautrier, II

- Programme résultant :

```
x = first();
while(x ≠ ⊥){
  body;
  x = next(x);
}
```

- Si D est une union de polyèdres ou de LBL, tous les calculs se font par Progammation Linéaire Paramétrique.

Quasts

La solution construite par PIP est un *quast* (quasi-affine selection tree)[1]. Soit x le vecteur des paramètres :

q ::= $\bot$
| if $f(x) \geq 0$ then q else q
| $\vec{f}(x)$

- $\bot$ indique que le polyèdre est vide,
- $f(x)$ est une forme affine (ou quasi-affine) en x,
- $f(\vec{x})$ est un vecteur dont les composantes sont des formes affines en x.

1. quasi-affine parce qu'il peut y avoir des divisions entières

Le Cas d'une seule Instruction, I

Le calcul de $first()$ se fait directement par PIP.
Attention, le résultat paut être un quast :

- $D = \{x \mid 1 \leq x \leq n\}$
- $first(n) =$ **if** $x \geq 1$ **then** 1 **else** $\perp$.

Le Calcul de *next*()

Dans la définition :

$$next(x) = \min_{\ll}\{y \in D \mid x \ll y\}.$$

La contrainte $x \ll y$ n'est pas convexe. Il faut la décomposer en plusieurs disjunctions :

$$x \ll y \equiv x_1 < y_1 \vee (x_1 = y_1 \ \& \ x_2 < y_2) \vee \ldots.$$

Chaque terme produit un quast $next_0, next_1, \ldots$, et il faut en calculer le minimum lexicographique.
Astuce Il est facile de voir que si $p < q$, alors $next_q(x) \ll next_p(x)$, *sauf si* $next_q(x) = \perp$. On peut donc se contenter, en partant de l'indice maximum, de "greffer" $next_{p-1}(x)$ à la place des $\perp$ dans $next_p(x)$.

Après une Transformation

Comme dans la méthode de Quilleré Bastoul, on ajoute les coordonnées transformées au domaine. Si par exemple on a obtenu un ordonnancement θ, le domaine devient :

$$D' = \{t, x \mid x \in D \ \ \& \ \ t = \theta(x)\}.$$

On procède comme ci-dessus la coordonnée t étant la première dans l'ordre lexicographique.

Le cas de plusieurs instructions

- On obtient autant de $next(x)$ qu'il y a d'instructions. Il faut en calculer le minimum lexicographique.
- Auparavant, il faut "décorer" les feuilles des quasts ; dans $next_S$, remplacer une feuille $\vec{f}(x)$ par $\langle S, \vec{f}(x) \rangle$.
- On peut alors calculer le min en tenant compte des positions relatives des instructions.
- On procède par une succession de réécritures.

Règles de réécriture, I

- Un candidat est une expression conditionelle :

$$q ::= a \,|\, \textbf{if } p \textbf{ then } q_1 \textbf{ else } q_2 \,|\, \bot.$$

  où a est un repère affine, p est un prédicat affine, et $\bot$ indique que le polyèdre correspondant est vide.
- On doit calculer des expressions de la forme $\min_{\ll}(q, q')$.
- La fonction min est symmétrique.

Règles de réécriture, II

- On a les propriété :

$$\begin{aligned} \min(\bot, q) &= q, \\ \min(\textbf{if } p \textbf{ then } q_1 \textbf{ else } q_2, q') &= \textbf{if } p \textbf{ then } \min(q_1, q') \\ &\quad \textbf{else } \min(q_2, q'), \\ \textbf{if } p \textbf{ then } q \textbf{ else } q &= q. \end{aligned}$$

- On peut démontrer que ces règles de réécriture se terminent toujours.
- On peut encore simplifier en éliminant les branches inconsistantes.

$$\begin{aligned} \textbf{if} &\quad x > 0 \\ \textbf{then} &\quad (\textbf{if } x < 0 \textbf{ then } a \textbf{ else } b) \\ \textbf{else} &\quad c = \textbf{if } x > 0 \textbf{ then } b \textbf{ else } c. \end{aligned}$$

Règles de réécriture, III

Quand la réécriture se termine, il faut évaluer des termes de la forme :

$$\min_{\ll}(\langle R, \vec{f}_R(x)\rangle, \langle T, \vec{f}_T(x)\rangle).$$

On compare tout d'abord f_R et f_S dans l'ordre lexicographique, et s'il y a égalité, on consulte l'ordre de R et T dans le texte du programme.

Astuce Au lieu de préfixer f du nom de l'instruction R on peut "entrelacer" f et le vecteur de position π_R de R. π_R se déduit de l'AST du programme. Il est tel que

$$R \,\text{avant}\, S \equiv \pi_R \ll \pi_S$$

.

Un exemple, I

```
  for(i=0; i<n; i++){
Z :  c[i] = 0;
     for(j=0; j<n; j++)
M :    c[i] += ....
}
```

- Les ordonnancements sont $\theta(Z,i) = 0, \theta(M,i,j) = j+1$.
- Il est facile de voir que $\mathrm{first}() = \langle Z, 0\rangle$.
- On doit caculer $\mathrm{next}(Z,i)$ et $\mathrm{next}(M,i,j)$.
- On trouve facilement
  $\mathrm{next}(Z,i) =$ **if** $i+1 < n$ **then** $\langle Z, i+1\rangle$ **else** $\langle M, 0, 0\rangle$.

Exemple, II

- $\mathrm{next}(M, i, j)$ peut être une instance de Z ou de M.
- Mais les instances de Z sont toujours avant celles de M.
- On doit donc calculer :

$$\min\{\langle M, i', j'\rangle \mid \langle M, i, j\rangle \ll \langle M, i', j'\rangle\}.$$

- La contrainte s'écrit :

$$\begin{aligned} & j+1 = j'+1 \ \&\ i = i' \ \&\ j < j' \\ \vee\ & j+1 = j'+1 \ \&\ i < i' \\ \vee\ & j+1 < j'+1 \end{aligned}$$

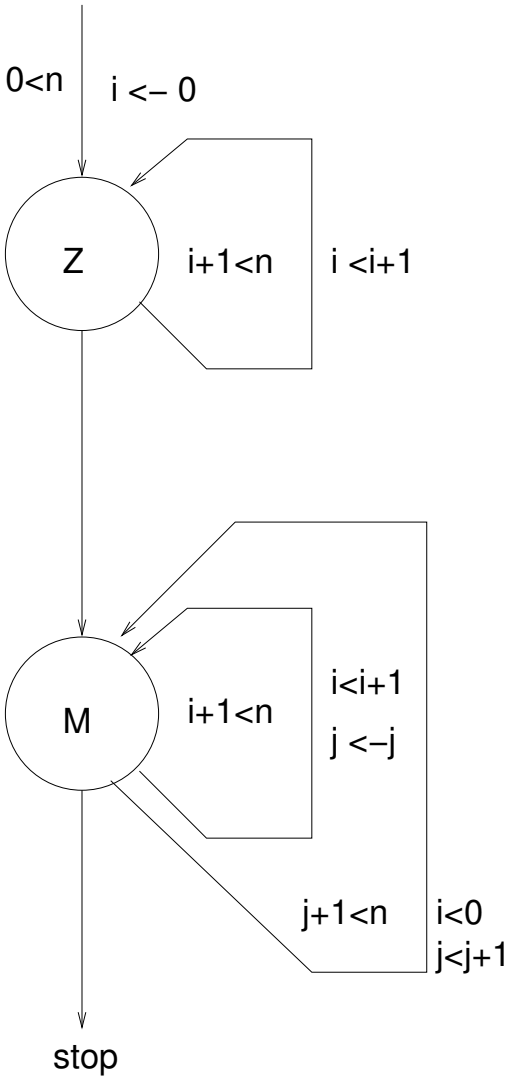
Exemple, III

- Ligne 1 : $\perp$.
- Ligne 2 : **if** $i+1<n$ **then** $\langle M, i+1, j\rangle$ **else** $\perp$.
- Ligne 3 : **if** $j+1<n$ **then** $\langle M, 0, j+1\rangle$ **else** $\perp$.

$$\begin{array}{rlll} \mathrm{next}(M,i,j) = & \textbf{if} & i+1<n & \\ & \textbf{then} & \langle M, i+1, j\rangle & \\ & \textbf{else} & \textbf{if } j+1<n & \\ & & \textbf{then } \langle M, 0, j+1\rangle & \\ & & \textbf{else } \perp & \end{array}$$

- Le dernier $\perp$ indique la fin du programme.

Exemple, IV




```
etat=I;
while(1){
  switch(etat){
    case I : i=0; suivant = Z;
             break;
    case Z : if(i+1<n) {i=i+1; suivant = Z;}
             else {i=0; j=0; suivant = M;
             }; break;
    case M : if(i+1<n) {i = i+1; suivant = M;}
             else if(j+1<n) {i=0; j=j+1; suivant = M;}
             else suivant = S;
             break;
    case S : exit;
    }
  etat = suivant;
 }
```

Où est le parallélisme ?

- L'automate ressemble à l'organigramme du programme parallèle.
- Certaines transitions sont *isochrones* : la valeur de l'ordonnancement ne change pas. C'est elles qui portent le parallélisme.
- Pour exprimer le parallélisme, on peut dérouler un état de l'automate le long d'un arc isochrone, puis regrouper les états obtenus.
- Facteur de déroulage = nombre de ressources disponibles.
- Noter que la méthode est moins puissante que le pipeline logiciel.

Conclusions provisoires

- Le problème de la génération de code à partir d'un ordonnancement reste un problème difficile.
- Deux méthodes : parcours de polyèdre ou construction d'automate.
- Parcours de polyèdre : adaptée au logiciel.
- Automates : adapté au matériel.
- La complexité peut être énorme : pour construire un automate, quadratique en nombre d'instruction et probablement exponentielle en profondeur des boucles.
- Nombreux problèmes encore à résoudre :
  1. Choix de la forme de l'ordonnancement.
  2. Elimination de tests redondants.
  3. Réduction de force (calculs de modulos).
  4. Un seul ou plusieurs automates / un seul ou plusieurs threads.
  5. Cas des opérations longues.