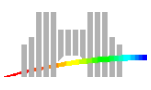
Génération de Code Parallèle

Paul Feautrier

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

2 décembre 2008



Génération de code parallèle

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

Un nid de boucle peut être représenté par un polyèdre :

```
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 les cours précédents.

Classification

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

- Transformation unimodulaire : un polyèdre
  - Une transformation unimodulaire est caractérisée par une matrice de déterminant $+1$ ou -1.
  - T^{-1} est entière (et unimodulaire)
  - $T(D)$ est un Z-polyèdre
- Transformation non unimodulaire : Z-polyèdre.
  - $T(D)$ est un polyèdre "à trous"
- Nid de boucle imparfait : union de Z-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 Z-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.

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.

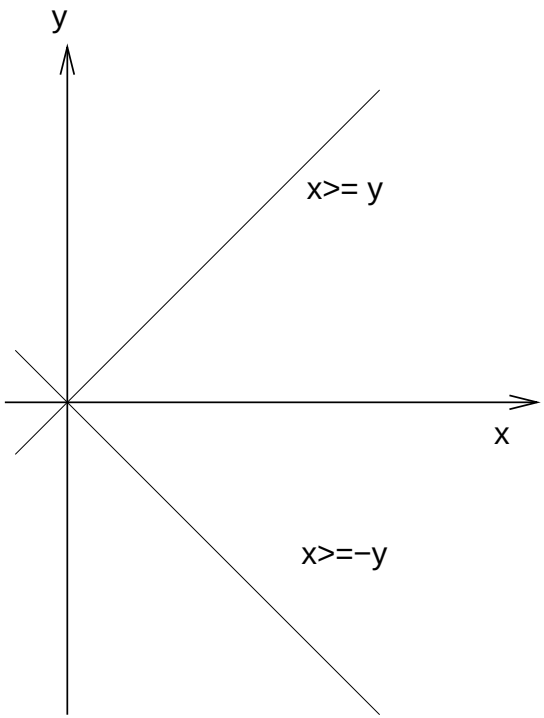
Programmation linéaire paramétrique

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.

Logiciel PIP : `www.piplib.org`

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 LBLs 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 cette union
- 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.

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

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

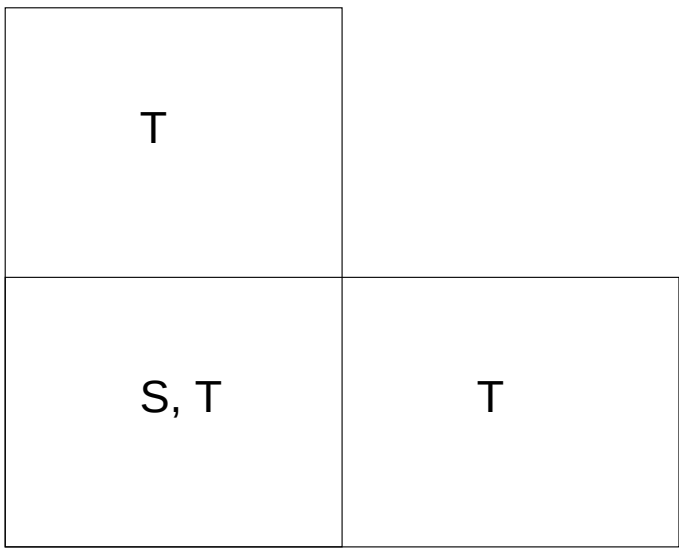
Objectif : minimiser le nombre de gardes.

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.

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.

Logiciel CLooG : `http ://www.cloog.org`

La méthode de P. Boulet

- Comme chacun sait, les processeurs n'exécutent pas directement les boucles ; le compilateur doit les traduire en tests et `GOTO`.
- On cherche donc à écrire directement du code de bas niveau. On fabrique ainsi un itérateur comme en Java ou C++.
- L'information nécessaire est donnée par la fonction `next` qui calcule l'opération qui suit immédiatement une opération donnée.

Référence : P. Boulet et P. Feautrier *Scanning Polyedra Without Do Loops*, PACT'98

Synopsis

Soit à parcourir dans l'ordre lexicographique $\ll$ le polyèdre :

$$\mathcal{P} = \{x \mid Ax + b \geq 0\},$$

en éxécutant l'instruction S en chaque point.

- On calcule les deux fonctions :

$$\text{first}() = \min_{\ll}\{x \mid Ax + b \geq 0\},$$

$$\text{next}(x) = \min_{\ll}\{y \mid Ay + b \geq 0, x \ll y, Ax + b \geq 0\}.$$

- Le programme (schéma) :

```
    x = first();
1:  if(x != NULL){
        S(x);
        x = next(x);
        goto 1;
    }
```

Détails techniques

- Astuce : on peut calculer les fonctions first et next à la compilation
- PIP calcule directement les minima lexicographique
- Pour le calcul de next, il faut décomposer la clause

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

- Chaque terme engendre une fonction $\mathrm{next}_0, \mathrm{next}_1, \ldots$ et on doit calculer

$$\mathrm{next}(x) = \min_{\ll}(\mathrm{next}_0(x), \mathrm{next}_1(x), \ldots).$$

- Mais on observe que l'on a toujours $\mathrm{next}_{k+1}(x) \ll \mathrm{next}_k(x)$ sauf si $\mathrm{next}_{k+1}(x) = \perp$:

$$\begin{aligned} \mathrm{next}(x) \quad = \quad & \textbf{if } \mathrm{next}_p(x) \neq \perp \textbf{ then } \mathrm{next}_p(x) \\ & \textbf{else if } \mathrm{next}_{p-1}(x) \neq \perp \textbf{ then } \mathrm{next}_{p-1}(x) \\ & \textbf{else } \ldots \end{aligned}$$

Détails techniques, suite

- Les problèmes à résoudre sont en nombres entiers et paramétriques
- Outre les paramètres de structure du programme, x est un paramètre pour le calcul de next
- Le résultat est donc une conditionnelle, qui peut contenir des divisions entières (un quast)
- Le quast se traduit directement en code conditionnel.

Exemple : l'inversion de boucle

Soit à parcourir le polyèdre $\{j, i \mid 0 \leq i \leq n-1, 0 \leq j \leq i-1\}$ dans l'ordre lexicographique inverse $j < j' \vee (j = j' \ \& \ i < i'$.

$$\begin{aligned} \mathrm{first}(n) &= \min\{j, i \mid 0 \leq i \leq n-1, 0 \leq j \leq i-1\} \\ &= \textbf{if } n \geq 2 \textbf{ then } (0,1) \textbf{ else } \bot \end{aligned}$$

$$\begin{aligned} \mathrm{next}_1(i,j,n) &= \min\{j', i' \mid 0 \leq i' \leq n-1, 0 \leq j' \leq i'-1, j = j', i < i'\} \\ &= \textbf{if } i+1 <= n-1 \textbf{ then } (j, i+1) \textbf{ else } \bot \end{aligned}$$

$$\begin{aligned} \mathrm{next}_0(i,j,n) &= \min\{j', i' \mid 0 \leq i' \leq n-1, 0 \leq j' \leq i'-1, j < j'\} \\ &= \textbf{if } j+2 <= n-1 \textbf{ then } (j+1, j+2) \textbf{ else } \bot \end{aligned}$$

Exemple, II

$$\begin{aligned} \text{next}(i,j,n) \quad = \quad & \textbf{if } i+1 \leq n-1 \textbf{ then } (j, i+1) \\ & \textbf{else if } j+1 \leq n-1 \textbf{ then } (j+1, j+2) \textbf{ else } \perp \end{aligned}$$

```
if(n >= 2){
    j = 0; i = 1;
    }
else goto fin;
corps_de_boucle : ....
if (i+1<=n-1){
  j=j; i = i+1 ;
} else if(j+1<=n-1){
  i = j+2; j = j+1 ;
} else goto fin ;
goto corps_de_boucle;
fin : ....;
```

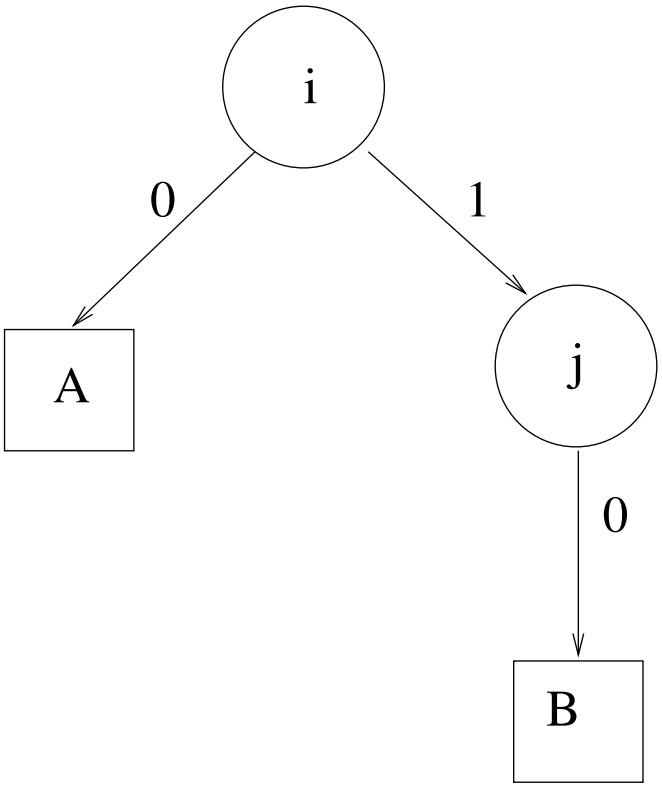
Le cas d'un programme ordonnancé

Notations

- Un programme $\mathcal{P}$. On écrit $A \in \mathcal{P}$ pour dire que A est une instruction de $\mathcal{P}$
- D_A est le domaine d'itération de A
- $E = \cup_{S \in \mathcal{P}} \langle S, D_S \rangle$ est l'ensemble des opérations du programme.
- On note θ la fonction d'ordonnancement, qui peut être multidimensionelle $\theta : E \rightarrow \mathbb{N}^d$
- L'ordre d'exécution séquentiel $<_{\text{seq}}$ peut être représenté comme un ordonnancement multidimensionel.

Ordonnancement séquentiel

```
for(i=0; i<n; i++){
A: c[i] = 0.;
   for(j=0; j<n; j++)
B:   c[i] += a[i][j] * b[j];
}
```



$$\theta_{\text{seq}}(A, i) = (i, 0, 0, 0)^T$$
$$\theta_{\text{seq}}(B, i, j) = (i, 1, j, 0)^T$$

Domaine d'itération étendu

- La dimension du nouvel espace d'itération est la somme de la dimension de l'ordonnancement θ plus le nombre de dimensions de θ_{seq}. On note le vecteur d'itération $(t, i)^T$
- Les contraintes qui définissent le nouveau domaine d'itération sont :
  - Les contraintes initiales $i \in D_S$
  - La contrainte d'ordonnancement $t = \theta(S, i)$
  - Les contraintes qui fixent les parties constantes de i.
- On note D'_S le domaine d'itération étendu et $E' = \cup_{S \in \mathcal{P}} \langle S, D'_S \rangle$
- L'ordre d'exécution (parallèle) cherché est alors l'ordre lexicographique sur $(t, i)^T$.

Détails techniques, I

On peut maintenant poser :

$$\begin{aligned} \mathrm{first}() &= \min_{\ll} E', \\ \mathrm{next}(u) &= \min_{\ll}\{v \in E' \mid u \ll v\} \end{aligned}$$

On décompose la fonction next suivant les instructions :

$$\mathrm{next}_T(S, i) = \min_{\ll}\{\langle T, j\rangle \mid j \in D'_T, i \ll j\}$$

Comme il n'y a plus à manipuler que des ordres lexicographiques, on procède comme dans le cas simple.

Détails techniques, II

Pour calculer

$$\mathrm{next}(S, i) = \min_{\ll}\{\mathrm{next}_T(S, i) \mid T \in \mathcal{P}\}$$

on utilise les règles de réécriture :

$$\begin{aligned}
\min(\textbf{if } p \textbf{ then } x \textbf{ else } y, z) &= \textbf{if } p \textbf{ then } \min(x, z) \textbf{ else } \min(y, z) \\
\min(\bot, x) &= x \\
\textbf{if } p \textbf{ then } x \textbf{ else } x &= x
\end{aligned}$$

ainsi que de la définition de l'ordre lexicographique.

Le programme objet

On transforme chaque expression next en un code `Next(S)` :

- On conserve les tests
- On remplace une feuille $\bot$ par un `goto fin`
- Soit maintenant une feuille $(t, i)^T$:
  - On ignore t
  - On extrait les termes non constants de i qui donnent les nouvelles valeurs du vecteur d'itération, et on écrit les affectations correspondantes
  - Les termes constants de i permettent d'identifier l'instruction S par descente dans l'AST. On écrit un `goto S`.
- Le programme se compose d'un paragraphe par instruction :

```
S: corps de l'instruction;
   Next(S);
```

- Il est en général possible d'optimiser (affectations inutiles, `goto` inutiles).

Et le parallélisme ?

Il y a parallélisme lorsque une feuille de la fonction next a les mêmes composantes t que l'argument.

- On peut essayer de collecter les opérations *isochrones* par itération de la fonction next
- Difficile, sauf si l'ordonnancement est à parallélisme borné
- Peut-être utilisable en synthèse de haut niveau

Conclusions

- Il y a encore des progrès à faire en génération de code parallèle
- Dès qu'il y a plusieurs paramètres de structure, le code est encombré par des tests, qu'il faudrait éliminer par des méthodes de type *évaluation partielle* suivies d'une compilation *just in time*.
- Cependant, le générateur CLooG se perfectionne en permanence ; il est utilisé dans le monde entier !
- La méthode de P. Boulet offre beaucoup de souplesse, mais elle pose des problèmes d'optimisation difficiles (élimination des sous-expressions communes, des affectations redondantes, des `goto` redondants, etc.).