Parcours de polyèdre paramétré avec l'élimination de Fourier-Motzkin

Marc Le Fur

▶ **To cite this version:**

Marc Le Fur. Parcours de polyèdre paramétré avec l'élimination de Fourier-Motzkin. [Rapport de recherche] RR-2358, INRIA. 1994. inria-00074319

HAL Id: inria-00074319
https://inria.hal.science/inria-00074319

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parcours de polyèdre paramétré avec l'élimination de Fourier-Motzkin

Marc Le Fur

N° 2358

Septembre 1994

PROGRAMME 1

ISSN 0249-6399

Parcours de polyèdre paramétré avec l'élimination de Fourier-Motzkin

Marc Le Fur*

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet PAMPA

Rapport de recherche n° 2358 — Septembre 1994 — 25 pages

Résumé : Ce rapport présente deux algorithmes calculant une structure de contrôle dont l'exécution énumère les vecteurs entiers d'un polyèdre paramétré dans un certain contexte. Les algorithmes reprennent la méthode des projections successives, basée sur l'élimination par paires de Fourier-Motzkin, définie par Ancourt et Irigoin dans le cas non paramétré. La politique de suppression des contraintes redondantes générées par l'élimination par paires est revue afin de réduire le temps de synthèse des parcours pour des polyèdres de complexité géométrique supérieure et améliorer le temps d'exécution des parcours produits. Les algorithmes présentés servent de base à la génération de code dans le prototype de compilateur Pandore développé à l'IRISA ; une comparaison entre ces algorithmes et celui défini par Ancourt et Irigoin est donnée sur la classe des polyèdres manipulés par le compilateur Pandore.

Mots-clé : Elimination par paires, parcours de polyèdre, test de redondance, test de faisabilité, simplexe.

(Abstract: pto)

*. mlefur@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Scanning Parameterized Polyhedron using Fourier-Motzkin Elimination

Abstract: This report presents two algorithms computing a control structure whose execution enumerates the integer vectors of a parameterized polyhedron defined in a given context. Both algorithms reconsider the successive projection method, based on Fourier-Motzkin pairwise elimination, defined by Ancourt and Irigoin in the non-parameterized case. The way the redundant constraints generated by pairwise elimination are removed is revisited in order to improve the computation time for the scanning of higher order polyhedrons but also their execution time. The algorithms presented in this report are at the root of the code generation in the Pandore compiler developed at IRISA; a comparison between these algorithms and the one defined by Ancourt and Irigoin is given on the class of polyhedrons manipulated by the Pandore compiler.

Key-words: Pairwise elimination, polyhedron scanning, redundancy test, feasibility test, simplex.

1 Introduction

Les motivations pour l'étude du parcours des vecteurs entiers d'un polyèdre sont à rechercher dans la parallélisation et la transformation de boucles [Wolf et al.91] ainsi que la compilation de boucles pour machines parallèles à mémoire partagée hiérarchisée [Ancourt91] ou bien encore à mémoire distribuée [Amarasinghe et al.93, LeFur et al.93]. Les transformations classiques de boucles (*reversal*, *permutation*, *skewing*) passent par l'application d'une transformation linéaire unimodulaire à un domaine d'itération polyédrique ; le domaine d'itération image est encore un polyèdre dont on peut énumérer les vecteurs à l'aide d'un nid de boucles parfaitement imbriquées. Dans la compilation de boucles pour machines parallèles à mémoire partagée hiérarchisée ou bien à mémoire distribuée, les codes de calcul et d'échange de données (resp. entre mémoire cache et mémoire globale et entre mémoires locales de processeurs) peuvent être modélisés par des polyèdres dont le parcours sert de base à la génération de code.

La méthode la plus connue permettant de synthétiser le code de parcours des vecteurs entiers d'un polyèdre est basée sur l'élimination par paires de Fourier-Motzkin [Schrijver86, Duffin74]. Ancourt et Irigoin [Irigoin et al.91] ont été les premiers à définir un algorithme utilisant l'élimination par paires ; il consiste en une suite de projections successives du polyèdre selon les différents axes, phase qui est suivie d'une élimination des contraintes redondantes générées par les différentes projections. Deux autres réponses au problème du parcours d'un polyèdre ont été proposées ; elles ont comme caractéristique commune de ne pas générer de contraintes redondantes. Feautrier propose l'utilisation d'un simplexe paramétré (PIP : Parametric Integer Programming) pour le calcul des bornes inférieures et supérieures dans chaque dimension du polyèdre [Feautrier89]. L'utilisation de PIP est complétée par une restructuration des bornes calculées [Collard et al.93]. La deuxième technique, basée sur l'algorithme de Chernikova [Lv92a], est exposée dans [LV et al.94]. La méthode procède également par projections successives mais repose sur le calcul de la représentation sous forme de rayons et sommets du polyèdre.

Ce rapport reprend la méthode des projections successives basée sur l'élimination par paires mais revoit complètement la politique de suppression des contraintes redondantes présentée par Ancourt et Irigoin. Le but visé est double. D'une part, améliorer le temps de production des parcours afin de rendre la méthode des projections successives applicable sur des polyèdres de dimension et de complexité géométrique supérieures [Amarasinghe et al.93, LeFur et al.93] et d'autre part, réduire le temps d'exécution des parcours produits. Deux algorithmes sont présentés dans ce rapport. Le premier revoit la deuxième phase d'élimination des contraintes redondantes de l'algorithme de Ancourt et Irigoin. Le deuxième propose un entrelacement des projections et des éliminations d'inégalités redondantes pour permettre la génération de parcours quand les algorithmes précédents échouent où s'avèrent trop coûteux. Une extension de ces algorithmes à l'énumération des vecteurs d'un polyèdre paramétré est également présentée ; nous montrerons que les vecteurs de $P(y) = \{x \ / \ Ay + Bx + c \geq 0\}$ défini dans le contexte $Q = \{y \ / \ My + h \geq 0\}$ peuvent être décrits par une conditionnelle dont le corps est un nid de boucles parfaitement imbriquées.

2 Définitions et propriétés

Les définitions et les propriétés qui suivent sont données pour l'espace $\mathbb{Q}^n$ même si elles sont également valides dans $\mathbb{R}^n$.

Soit $S = A\,(x_1, \ldots, x_n)^t + b \geq 0$ un système de contraintes affines (A est une matrice entière et b un vecteur entier) et $P = \{x = (x_1, \ldots, x_n)^t \ / \ Ax + b \geq 0\}$ le polyèdre associé à S. L'élimination par paires de Fourier-Motzkin [Schrijver86, Duffin74] vise à calculer la projection du polyèdre P selon un axe donné. Considérons par exemple la projection P_{x_n} du polyèdre P selon l'axe x_n et notons S_{x_n} le système de contraintes associé à P_{x_n} :

$$S_{x_n} = Nul(x_n) \ \cup \ Elim(x_n, Min(x_n), Max(x_n))$$

où $(Min(x_n), Max(x_n), Nul(x_n))$ définit la partition de S:

- $Min(x_n) = \{c_i x_n + f_i(x_1, \ldots, x_{n-1}) \geq 0\}_{i \in I}$ où les f_i sont des fonctions affines et $\forall i \in I \ \ c_i > 0$. Les inégalités de $Min(x_n)$ seront appelées *contraintes minimisantes* pour x_n,
- $Max(x_n) = \{c_j x_n + f_j(x_1, \ldots, x_{n-1}) \geq 0\}_{j \in J}$ avec $\forall j \in J \ \ c_j < 0$: l'ensemble des *contraintes maximisantes* pour x_n,
- $Nul(x_n) = \{f_k(x_1, \ldots, x_{n-1}) \geq 0\}_{k \in K}$: l'ensemble des contraintes de S dont le coefficient de x_n est nul,

et $Elim(x_n, Min(x_n), Max(x_n))$ dénote l'ensemble des éliminations de la variable x_n pour chaque paire d'inéquations de $Min(x_n) \times Max(x_n)$:

$$Elim(x_n, Min(x_n), Max(x_n)) = \begin{cases} \emptyset \ \text{ si } |I||J| = 0 \\ \{-c_j f_i(x_1, \ldots, x_{n-1}) + c_i f_j(x_1, \ldots, x_{n-1}) \geq 0\}_{(i,j) \in I \times J} \ \text{ sinon} \end{cases}$$

La figure 1 illustre la suppression d'une variable dans un ensemble de contraintes avec l'élimination de Fourier-Motzkin. La projection d'un polyèdre avec une élimination par paires est connue pour être coûteuse puisqu'elle peut générer un nombre d'inégalités de l'ordre de $(m/2)^2$ à partir d'un système comportant m contraintes. D'une manière générale, la projection d'un polyèdre selon l axes peut produire un nombre de contraintes de l'ordre de $m^{2^l}/2^{2^l+1}-2$. Le processus de projection d'un polyèdre avec une élimination par paires doit donc être utilisé avec précaution. L'élimination de Fourier-Motzkin possède d'autres propriétés remarquables :

Propriété 1 *$\tilde{x}$ vérifie S ssi $\tilde{x}$ vérifie $S_{x_n} \uplus Min(x_n) \uplus Max(x_n)$*

Propriété 2 *S est faisable ssi toutes les contraintes $e_i \geq 0$ ($e_i \in \mathbb{Z}$) du système obtenu après élimination de toutes les variables dans S sont trivialement faisables, c'est à dire telles que $e_i \in \mathbb{N}$.*

en PLNE se révèlent très coûteuses. Il faut alors s'orienter vers la résolution du programme linéaire du critère 1 en nombres rationnels et donc vers un test approché de redondance dans $\mathbb{Z}^n$ puisque seulement exact dans $\mathbb{Q}^n$.

Deuxième test de redondance

Le critère suivant [Lv92b] découle du lemme de Farkas exprimé dans sa forme affine [Schrijver86] et fournit un autre test exact de redondance dans $\mathbb{Q}^n$ et donc un autre test approché de redondance dans $\mathbb{Z}^n$:

Critère 2 *Dans* $\mathbb{Q}^n$, $ax+\beta \geq 0$ *est redondante dans* $Ax+b \geq 0$ *ssi* $\exists\ y \geq 0\ \ yA' = a\ \ yb' \leq \beta$

En d'autres termes, $ax + \beta \geq 0$ est redondante dans $Ax + b \geq 0$ si et seulement si $ax + \beta \geq 0$ est combinaison positive des inégalités de $A'x + b' \geq 0$ et de la contrainte $1 \geq 0$.

Troisième test de redondance

Nous retiendrons également le critère suivant [Cheng87] :

Critère 3 $ax + \beta \geq 0$ *est redondante dans* $Ax + b \geq 0$ *ssi* $A'x + b' \geq 0 \bigcup \{ax + \beta < 0\}$ *n'est pas faisable*

qui peut être raffiné pour donner un test approché de redondance dans $\mathbb{Z}^n$ qui n'est cette fois plus valide dans $\mathbb{Q}^n$: *dans* $\mathbb{Z}^n$, $ax + \beta \geq 0$ *est redondante dans* $Ax + b \geq 0$ *dès lors que* $A'x + b' \geq 0 \bigcup \{ax + \beta \leq -1\}$ *n'est pas faisable dans* $\mathbb{Q}^n$. Ce dernier test de redondance permet en effet de supprimer plus d'inégalités implicites que les tests 1 et 2 comme l'illustre la figure 5. Dans l'ensemble de contraintes $\{1, 2, 3\}$, la contrainte numéro 2 est redondante au sens du test 3 alors qu'elle n'est pas redondante dans $\mathbb{Q}^2$, c'est à dire avec les tests de redondance 1 et 2.

3.2.2 Mise en œuvre des tests de redondance

La *méthode du simplexe* [Papadimitriou et al.82] [Nemhauser et al.88] [Schrijver86] [Sakarovitch84] est actuellement la méthode de référence pour résoudre un programme linéaire $min\{cx + \delta \ / \ Ax + b \geq 0\}$ (ou bien encore $max\{cx + \delta \ / \ Ax + b \geq 0\}$, les deux problèmes étant de toutes façons duaux) en nombres rationnels. Cette méthode comporte deux phases : la première vise à rechercher une solution faisable au système $Ax + b \geq 0$ et la deuxième à calculer l'optimum quand celui-ci est borné. Ces deux phases sont résolues par un seul algorithme: *l'algorithme du simplexe*. Sa complexité temporelle a été prouvée exponentielle sur des exemples artificiels mais un comportement moyen polynomial, conforté par l'expérience pratique, a été établi. Cet algorithme travaille de plus à mémoire constante.

Dans une mise en œuvre efficace du test de redondance, on peut écarter le test 1 qui se traduit par deux simplexes, pour ne retenir que les tests 2 et 3 qui sont des problèmes de faisabilité d'un système et donc résolus par un seul simplexe (la phase I de la méthode du simplexe). On notera au passage que les tests de redondance 2 et 3 peuvent être mis en œuvre

variables dans la base J_0. Par abus de langage, on dira que la base J_0 est une *base réalisable*.

L'algorithme du simplexe consiste en une suite de pivotages de la matrice

$$\begin{pmatrix} A_0 & b_0 \\ c_0 & -\delta_0 \end{pmatrix}$$

des coefficients du programme linéaire (P_0). A chaque étape, le programme (P_q), écrit sous forme canonique par rapport à une base réalisable J_q, est transformé en un programme linéaire (P_{q+1}) équivalent, également écrit sous forme canonique par rapport à une base réalisable J_{q+1} ; chaque opération de pivotage faisant sortir une variable t de la base et rentrer une nouvelle variable s dans la base : $J_{q+1} = (J_q - \{t\}) \cup \{s\}$. L'algorithme termine à l'obtention d'une base J_m dont la solution (de base) associée ($x^{J_m} = (A_m^{J_m})^{-1} b_m,\ x^{\overline{J_m}} = 0$) est optimale (l'optimum est égal à δ_m) ou bien à la mise en évidence d'une solution non bornée au programme linéaire.

Initialisation pour le problème de faisabilité du test 2

Le problème que l'on cherche ici à résoudre est le suivant : $\exists\ y = (y_1, \ldots, y_p) \geq 0\ \ yA' = a\ \ yb' \leq \beta$ soit encore $\exists\ y = (y_1, \ldots, y_p) \geq 0\ \ \exists\ y_0 \geq 0\ \ yA' = a\ \ yb' + y_0 = \beta$. La variable y_0, qui mesure l'écart entre β et yb', est appelée *variable d'écart*. Le problème posé revient donc à rechercher la faisabilité d'un système de la forme

$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Notons m le nombre de lignes de la matrice A, I la $m \times m$-matrice identité et e le m-vecteur ligne $(1, 1, \ldots, 1)$. Quitte à multiplier par -1 les équations $a_i x_i = b_i$ de $Ax = b$ telles que $b_i < 0$, on peut toujours supposer $b \geq 0$ dans $Ax = b$. Le problème initial revient donc à montrer que le programme linéaire

$$(P) \begin{cases} Ax & +Iv & = b & x, v \geq 0 \\ & \sum_{i=1}^m v_i & = z(min) & \end{cases}$$

admet 0 comme solution optimale (qui existe puisque $\sum_{i=1}^m v_i \geq 0$). Dans le programme linéaire (P), les variables v_i sont appelées *variables artificielles*. L'égalité $e(Ax + Iv) = eb$ soit encore $eAx + \sum_{i=1}^m v_i = eb$ nous permet enfin d'écrire (P) sous forme canonique par rapport à la base réalisable composée des variables v_i :

$$(P) \begin{cases} Ax & +Iv & = b & x, v \geq 0 \\ -eAx & & = z(min) - eb & \end{cases}$$

Initialisation pour le problème de faisabilité du test 3

Le test de redondance 3 consiste en la recherche d'une solution faisable d'un système $S = Ax+b \geq 0$. Après avoir opéré un éventuel changement de variable $x_i = \tilde{x_i} - \tilde{\tilde{x_i}}$ $\tilde{x_i}, \tilde{\tilde{x_i}} \geq 0$ pour chaque variable x_i qui n'est pas astreinte à être positive dans S, on peut toujours supposer que le système S est de la forme

$$\begin{cases} Ax + b \geq 0 \\ x \geq 0 \end{cases}$$

S est alors partitionné en $S_1 = A_1x + b_1 \geq 0$ $(b_1 \geq 0)$ et $S_2 = A_2x + b_2 \geq 0$ $(b_2 < 0)$. Notons m_i le nombre de lignes de A_i, e_i le m_i-vecteur ligne $(1, 1, \ldots, 1)$ et I_i la $m_i \times m_i$-matrice identité pour i = 1, 2. Si $m_2 = 0$ alors S admet une solution triviale : $x = 0$. Sinon $Ax + b \geq 0$ peut être transformé en un système d'égalités en utilisant le procédé d'introduction de variables d'écart vu au paragraphe précédent :

$$\begin{cases} -A_1x & & +I_1v_1 & = b_1 & x, v_1, v_2 \geq 0 \\ A_2x & -I_2v_2 & & = -b_2 & \end{cases}$$

Le dernier système est faisable si et seulement si le programme linéaire

$$(P) \begin{cases} -A_1x & & +I_1v_1 & & = b_1 & x, v_1, v_2, v_2' \geq 0 \\ A_2x & -I_2v_2 & & +I_2v_2' & = -b_2 & \\ & & & e_2v_2' & = z(min) & \end{cases}$$

admet 0 comme solution optimale (qui existe puisque $e_2v_2' \geq 0$). En utilisant la technique de sommation des égalités contenant des variables artificielles vue au paragraphe précédent, le programme linéaire (P) peut être finalement mis sous forme canonique par rapport à la base réalisable composée des variables de v_1 et v_2' :

$$(P) \begin{cases} -A_1x & & +I_1v_1 & & = b_1 & x, v_1, v_2, v_2' \geq 0 \\ A_2x & -I_2v_2 & & +I_2v_2' & = -b_2 & \\ -e_2A_2x & +e_2v_2 & & & = z(min) + e_2b_2 & \end{cases}$$

Optimisation du simplexe pour le problème de faisabilité

A chaque étape de l'algorithme du simplexe, l'opération de pivotage se fait autour d'un élément $A_q(r, s) > 0$ d'une colonne hors-base ($s \in \overline{J_q}$). Cette opération a pour conséquence de faire rentrer s dans la base (s devient le r-ième vecteur de la base canonique) et sortir la colonne t égale au r-ième vecteur de la base canonique, les autres colonnes de J_q restant inchangées. Lorsque seule la valeur de la fonction objective à l'optimum nous intéresse (c'est le cas dans un problème de faisabilité), on peut alors représenter la matrice des coefficients du programme linéaire par ses colonnes hors-base ainsi qu'une colonne intermédiaire qui va servir au calcul de la nouvelle colonne hors-base. Cette représentation compacte de la matrice des coefficients permet de réduire le coût des opérations de pivotage, et ce d'autant plus que

le nombre de lignes de la matrice A_q du programme linéaire est grand devant le nombre de ses colonnes. Notons m le nombre de lignes de la matrice A_0 du programme linéaire (P_0) en entrée du simplexe. La matrice des coefficients de (P_q) a maintenant la forme

$$\begin{pmatrix} A_q^{\overline{J_q}} & \begin{pmatrix} d_1 \\ \vdots \\ d_m \end{pmatrix} & b_q \\ c_q^{\overline{J_q}} & d_{m+1} & -\delta_q \end{pmatrix}$$

où $(d_1, \ldots, d_{m+1})^t$ est la colonne intermédiaire qui va servir au calcul de la nouvelle colonne hors-base. La matrice des coefficients de (P_{q+1}) est alors calculée de la manière suivante (on ne décrit ici que le cas général de l'algorithme du simplexe) :

1. Choisir le pivot $A_q^{\overline{J_q}}(r, s) > 0$
2. Remplacer la colonne $(d_1, \ldots, d_{m+1})^t$ par le r-ième vecteur de la base canonique ($d_r := 1,\ \forall i \neq r\ d_i := 0$)
3. Pivoter la matrice des coefficients autour de l'élément $A_q^{\overline{J_q}}(r, s)$
4. Remplacer la colonne s (qui est rentrée dans la base) par la colonne d (qui est devenue hors-base)

3.2.3 Test de redondance retenu dans Pandore

S'agissant de l'élimination des contraintes redondantes dans un système $Ax + b \geq 0$ issu d'une élimination par paires et vus les polyèdres considérés, on peut supposer que le cadre dans lequel on se place est $m \gg n$ si A est une matrice $m \times n$. L'expérience pratique a montré qu'une élimination basée sur le test de redondance 3 est plus efficace en temps qu'une élimination reposant sur le test 2. Ce résultat semble pourtant contraire à l'intuition. En effet, le simplexe optimisé sous-jacent au test 2 travaille sur une matrice des coefficients dont la taille est de l'ordre de $n \times m$ alors qu'elle est de l'ordre de $m \times m/2$ pour le simplexe du test 3 (si $m/2$ est le nombre d'inégalités $a_i x_i + b_i \geq 0$ de $Ax + b \geq 0$ telles que $b_i < 0$). Dans le cadre $m \gg n$, le simplexe du test 3 opère donc sur des matrices de coefficients plus grandes que celles du test 2 ; ll semble que l'algorithme du simplexe soit plus sensible à l'augmentation du nombre de variables que du nombre d'inégalités. C'est donc le test de redondance 3 qui a été retenu dans Pandore, ce test pouvant en outre éliminer plus d'inégalites implicites que le test 2.

4 Parcours de polyèdre non paramétré avec Fourier-Motzkin

Notons $P = \{x = (x_1, \ldots, x_n)^t \ / \ Ax + b \geq 0\}$ (où A est une matrice entière et b un vecteur entier) le polyèdre dont on veut énumérer les vecteurs entiers et $S = Ax + b \geq 0$ le système d'inégalités associé.

4.1 Algorithme de Ancourt et Irigoin

L'algorithme décrit dans [Irigoin et al.91] procède en deux phases :

Phase I Calcul d'un système équivalent à S obtenu en appliquant récursivement l'élimination par paires dans la propriété 1 :

$$S' = Min(x_1) \uplus Max(x_1) \biguplus Min(x_2) \uplus Max(x_2) \biguplus \ldots \biguplus Min(x_n) \uplus Max(x_n)$$

Phase II Elimination de contraintes redondantes dans S'. Cette phase produit un système équivalent à S' (et donc à S) :

$$S'' = Min'(x_1) \uplus Max'(x_1) \biguplus Min'(x_2) \uplus Max'(x_2) \biguplus \ldots \biguplus Min'(x_n) \uplus Max'(x_n)$$

où $Min'(x_i)$ (resp. $Max'(x_i)$) est le système d'inégalités issu de l'élimination de contraintes redondantes dans $Min(x_i)$ (resp. $Max(x_i)$). Le calcul de S'' commence par l'élimination de contraintes redondantes dans $Min(x_n)$ et $Max(x_n)$ puis $Min(x_{n-1})$ et $Max(x_{n-1})$ ... jusqu'à la simplification des inégalités de $Min(x_1)$ et $Max(x_1)$. Notons S_nred_i le système $Min'(x_n) \uplus Max'(x_n) \biguplus \ldots \biguplus Min'(x_{i+1}) \uplus Max'(x_{i+1})$ résultant de l'élimination de contraintes redondantes dans $Min(x_n) \uplus Max(x_n) \biguplus \ldots \biguplus Min(x_{i+1}) \uplus Max(x_{i+1})$ et S_red_i le sous-ensemble $Min(x_{i-1}) \uplus Max(x_{i-1}) \biguplus \ldots \biguplus Min(x_1) \uplus Max(x_1)$ des contraintes de S non encore considérées. Pour i=n,n-1,...,2 dans cet ordre, les ensembles $Min'(x_i)$ et $Max'(x_i)$ sont calculés séquentiellement de la manière suivante :

1. $Min'(x_i)$ = élimination de presque toutes les contraintes redondantes de $Min(x_i)$ dans le contexte $S_nred_i \uplus S_red_i \uplus Max(x_i)$
2. $Max'(x_i)$ = élimination de presque toutes les contraintes redondantes de $Max(x_i)$ dans le contexte $S_nred_i \uplus S_red_i \uplus Min'(x_i)$

ou bien encore

1. $Max'(x_i)$ = élimination de presque toutes les contraintes redondantes de $Max(x_i)$ dans le contexte $S_nred_i \uplus S_red_i \uplus Min(x_i)$
2. $Min'(x_i)$ = élimination de presque toutes les contraintes redondantes de $Min(x_i)$ dans le contexte $S_nred_i \uplus S_red_i \uplus Max'(x_i)$

suivant que l'on veut éliminer (presque toutes) les inégalités redondantes de $Min(x_i)$ avant celles de $Max(x_i)$ ou non. D'après la remarque formulée au 3.1, on note au passage que les deux séquences précédentes ne conduisent pas au même résultat. Les ensembles $Min(x_1)$ et $Max(x_1)$ étant de la forme $Min(x_1) = \{c_j x_1 - \alpha_j \geq 0\}_{j \in J}$ et $Max(x_1) = \{-c_k x_1 + \alpha_k \geq 0\}_{k \in K}$ $(c_j, c_k > 0)$, le calcul des singletons $Min'(x_1)$ et $Max'(x_1)$ est plus trivial : $Min'(x_1) = \{c_{j_0} x_1 - \alpha_{j_0} \geq 0\}$, $Max'(x_1) = \{-c_{k_0} x_1 + \alpha_{k_0} \geq 0\}$ avec $\alpha_{j_0}/c_{j_0} = max\{\alpha_j/c_j \ / \ j \in J\}$ et $\alpha_{k_0}/c_{k_0} = min\{\alpha_k/c_k \ / \ k \in K\}$.

La dernière phase de l'algorithme est commune à la plupart des méthodes utilisées en synthèse de parcours d'un polyèdre et concerne l'extraction de bornes inférieures et supérieures $Binf_i$ et $Bsup_i$ à partir des ensembles de contraintes $Min'(x_i) = \{c_q x_i + f_q((x_1, \ldots, x_{i-1})) \geq 0\}_{q \in Q_i}$ et $Max'(x_i) = \{c_r x_i + f_r((x_1, \ldots, x_{i-1})) \geq 0\}_{r \in R_i}$. Dans sa thèse [Ancourt91], Ancourt montre que $\alpha\beta + \gamma \geq 0 \Longleftrightarrow \beta \geq div(-\gamma + \alpha - 1, \alpha)$ et $-\alpha\beta + \gamma \geq 0 \Longleftrightarrow \beta \leq div(\gamma, \alpha)$ si α est un entier strictement positif et *div* la division euclidienne. De ces équivalences, on peut déduire l'ensemble des bornes inférieures et supérieures pour chaque x_i : $Binf_i = \{div(-f_q((x_1, \ldots, x_{i-1})) + c_q - 1, c_q)\}_{q \in Q_i}$ et $Bsup_i = \{div(f_r((x_1, \ldots, x_{i-1})), -c_r)\}_{r \in R_i}$. La structure de contrôle énumérant les vecteurs entiers de P est alors donnée par le nid de boucles parfaitement imbriquées

$$\begin{array}{l} \textbf{for } x_1 = \textbf{max } Binf_1, \ \textbf{min } Bsup_1 \\ \quad \ddots \\ \qquad \textbf{for } x_n = \textbf{max } Binf_n, \ \textbf{min } Bsup_n \end{array}$$

4.2 Première amélioration : l'algorithme *descendant*

A la lecture de l'algorithme, on peut observer que des inégalités implicites de $Min(x_i)$ et $Max(x_i)$ sont éliminées en tenant compte des contraintes de S_nred_i c.a.d des inégalités associées aux indices de boucles les plus internes. De part la nature même de l'élimination par paires (les contraintes minimisantes ou maximisantes pour x_i sont pour la plupart des combinaisons positives et donc des conséquences d'inégalités de $Min(x_n) \uplus Max(x_n) \biguplus \ldots \biguplus Min(x_{i+1}) \uplus Max(x_{i+1})$), ceci permet d'éliminer la plupart des contraintes redondantes de $Min(x_i)$ et $Max(x_i)$, simplifiant d'autant les bornes de x_i dans la boucle. La contre-partie de la méthode est qu'elle supprime des contraintes minimisantes ou maximisantes pour x_i qui sont non redondantes dans le contexte S_red_i associé aux boucles englobant x_i, ce qui se traduit par la génération de nids de boucles *à trous* c.a.d des boucles

$$\begin{array}{l} \textbf{for } x_1 = \alpha_1, \beta_1 \\ \quad \ddots \\ \qquad \textbf{for } x_n = \alpha_n, \beta_n \end{array}$$

où $\exists \ i \in 1..n \ \ \exists \ (x_1, \ldots, x_{i-1}) \ \alpha_1 \leq x_1 \leq \beta_1 \ \ldots \ \alpha_{i-1} \leq x_{i-1} \leq \beta_{i-1}$ tel que $\alpha_i > \beta_i$. On dira qu'une telle boucle comprend un *trou situé à la profondeur* x_i. Ainsi pour le polyèdre

possédant 50600 vecteurs entiers défini par le système

$$\begin{cases} i & -i+7 & j & \\ -j+13 & k & -k+19 & \\ l-1 & -l+100 & m-1 & \\ 10*l-m+1 & -500*i+5*l & 500*i-5*l+499 & \geq 0 \\ -300*j+l+m & 300*j-l-m+299 & -l-m+3999 & \\ -200*k+2*l+m & 200*k-2*l-m+199 & & \end{cases}$$

comprenant 5 variables i, j, k, l, m et 17 inégalités, l'algorithme de Ancourt et Irigoin produit le parcours

```
for i = 0 , 1
  for j = div(i,3) , div(2750*i+2747,750)
    for k = div(3*j,2) , 19
      for l = max(100*i,1) , min(div(500*i+499,5),100)
        for m = max(1,300*j-l,200*k-2*l) ,
                min(10*l+1,300*j-l+299,200*k-2*l+199)
```

qui comporte 6759 trous tous situés à la profondeur m, générant ainsi beaucoup de calculs de bornes inutiles à l'exécution. La deuxième remarque concerne l'ordre de simplification des contraintes du système S' issu de la phase I. L'élimination de contraintes redondantes est faite de manière *ascendante* c.a.d des contraintes minimisantes et maximisantes pour x_n jusqu'à celles calculées pour x_1. Le système S_red_i associé aux boucles englobant x_i comprend donc par définition beaucoup d'inégalités implicites, rendant la simplification des contraintes minimisantes et maximisantes pour x_i très coûteuse.

Ces deux remarques mènent à l'algorithme *descendant* qui améliore la phase II de l'algorithme de Ancourt et Irigoin. La phase va toujours produire un système équivalent à S' et donc à S

$$S''' = Min''(x_1) \uplus Max''(x_1) \biguplus Min''(x_2) \uplus Max''(x_2) \biguplus \ldots \biguplus Min''(x_n) \uplus Max''(x_n)$$

où $Min''(x_i)$ et $Max''(x_i)$ sont les ensembles de contraintes minimisantes et maximisantes pour x_i issus de l'élimination des contraintes redondantes dans $Min(x_i)$ et $Max(x_i)$ respectivement. Les ensembles $Min(x_1)$ et $Max(x_1)$ sont tout d'abord simplifiés comme vu au 4.1 pour donner $Min''(x_1)$ et $Max''(x_1)$ puis pour i=2,3,...,n dans cet ordre, les ensembles $Min''(x_i)$ et $Max''(x_i)$ sont calculés en parallèle :

- $Min''(x_i)$ = élimination des contraintes redondantes de $Min(x_i)$ dans le contexte $Min''(x_1) \uplus Max''(x_1) \biguplus \ldots \biguplus Min''(x_{i-1}) \uplus Max''(x_{i-1})$
- $Max''(x_i)$ = élimination des contraintes redondantes de $Max(x_i)$ dans le contexte $Min''(x_1) \uplus Max''(x_1) \biguplus \ldots \biguplus Min''(x_{i-1}) \uplus Max''(x_{i-1})$

4.3 Deuxième amélioration : l'algorithme *entrelacé*

Le système S' issu de la phase I de l'algorithme donné par Ancourt et Irigoin peut contenir beaucoup d'inégalités ce qui peut rendre l'élimination des contraintes redondantes

de la phase II trop complexe voire impossible faute de mémoire suffisante. Pour le système vu au 4.2 par exemple, S' est composé de 370 contraintes. Cette observation justifie le dernier algorithme, dit *entrelacé*, incorporé dans le compilateur Pandore : les projections selon un axe et les éliminations de contraintes redondantes sont entrelacées afin d'éviter le risque d'explosion combinatoire du nombre de contraintes de la phase I. De même que dans l'algorithme descendant, l'élimination des contraintes redondantes minimisantes ou maximisantes pour x_i ne tient plus compte des inégalités associées aux indices de boucles les plus internes. La figure 6 décrit le calcul d'un système $S'''' = sys_eq\ S\ n$ équivalent à S de la forme

$$S'''' = Min'''(x_1) \uplus Max'''(x_1) \biguplus Min'''(x_2) \uplus Max'''(x_2) \biguplus \ldots \biguplus Min'''(x_n) \uplus Max'''(x_n)$$

où $Min'''(x_i)$ (resp. $Max'''(x_i)$) est un ensemble de contraintes minimisantes (resp. maximisantes) pour x_i, qui va servir à la synthèse des bornes de la boucle parcourant le polyèdre. De

---

sys_eq $(S,\ i)$ =
  soit $Min_x_i \uplus Max_x_i \uplus Nul_x_i$ la partition de S
  dans si $i = 1$
    alors $Min'''_x_1 \uplus Max'''_x_1$
      où $Min'''_x_1 = \{c_{j_0} x_1 - \alpha_{j_0} \geq 0\}$ avec $\alpha_{j_0}/c_{j_0} = max\{\alpha_j/c_j\ /\ j \in J\}$
      si $Min_x_i = \{c_j x_1 - \alpha_j \geq 0\}_{j \in J}\ (c_j > 0)$
      et $Max'''_x_1 = \{-c_{k_0} x_1 + \alpha_{k_0} \geq 0\}$ avec $\alpha_{k_0}/c_{k_0} = min\{\alpha_k/c_k\ /\ k \in K\}$
      si $Max_x_i = \{-c_k x_1 + \alpha_k \geq 0\}_{k \in K}\ (c_k > 0)$
    sinon soit $Min'''_x_i =$ elim_red_ctxt $(Min_x_i,\ Max_x_i \uplus Nul_x_i)$
      dans soit $Max'''_x_i =$ elim_red_ctxt $(Max_x_i,\ Min'''_x_i \uplus Nul_x_i)$
        dans soit $Nul'''_x_i =$ elim_red_ctxt $(Nul_x_i,\ Min'''_x_i \uplus Max'''_x_i)$
          dans soit $S_x_i = Elim(x_i, Min'''_x_i, Max'''_x_i) \cup Nul'''_x_i$
            dans sys_eq $(S_x_i,\ i-1) \biguplus Min'''_x_i \uplus Max'''_x_i$

FIG. 6 - *Algorithme entrelacé dans le cas non paramétré*

même que dans l'algorithme de Ancourt et Irigoin, on note que l'ordre de calcul de Min'''_x_i et Max'''_x_i peut être inversé dans la fonction *sys_eq* :

1. $Max'''_x_i =$ elim_red_ctxt $(Max_x_i,\ Min_x_i \uplus Nul_x_i)$
2. $Min'''_x_i =$ elim_red_ctxt $(Min_x_i,\ Max'''_x_i \uplus Nul_x_i)$

les deux séquences ne produisant pas le même nid de boucles au final.

4.4 Comparaison des algorithmes

Nous avons implémenté l'algorithme de Ancourt et Irigoin dans le langage fonctionnel interprété CAML [Aponte et al.90], langage dans lequel le compilateur Pandore a été écrit, en

vue de le comparer à nos algorithmes sur la classe des polyèdres définis dans [LeFur et al.93]. Le test unitaire de redondance sous-jacent aux trois algorithmes est celui présenté dans la section 3.2. Les métriques qui vont nous permettre de comparer les parcours produits par les différents algorithmes pour chaque polyèdre sont les suivantes :

- le gain obtenu sur le temps de production du parcours avec les versions descendante et entrelacée par rapport à notre implémentation de l'algorithme de Ancourt et Irigoin,
- le nombre d'opérations élémentaires (*min* et *max* d'arité 2, *div*, $+$, $-$, $*$, incrément de compteur de boucle) effectuées lors de l'exécution du parcours,
- le temps d'exécution utilisateur du nid de boucles, mesuré sur une station de travail Sun Sparc 10 après compilation du parcours avec `gcc -O2`,
- les trous éventuels existant dans les nid de boucles.

Pour chaque polyèdre, le nombre de sommets rationnels calculé avec la bibliothèque polyédrique [LV et al.94] est donné.

4.4.1 Polyèdre P_1

Ensemble de contraintes

Il est composé de 12 inégalités et porte sur 4 variables i, j, k, l. Le polyèdre P_1 associé comprend 502500 vecteurs entiers et 25 sommets rationnels.

$$\begin{cases} i & -i+19 & j \\ -j+19 & k-1 & -k+1000 \\ -k+l & 2*k-l+1 & -200*i+k+l \\ 200*i-k-l+199 & -200*j-k+2*l & 200*j+k-2*l+199 \end{cases} \geq 0$$

Parcours produit avec l'algorithme de Ancourt et Irigoin

```
for i = 0 , 15
  for j = div(i,3) , 19
    for k = 1 , 1000
      for l = max(k,200*i-k,div(200*j+k+1,2)) ,
              min(2*k+1,200*i-k+199,div(200*j+k+199,2))
```

Parcours produit avec l'algorithme descendant

```
for i = 0 , 15
  for j = max(2*i-15,div(i,2)) , min(15,i+1)
    for k = max(div(200*i+1,3),div(400*i-200*j-197,3),div(200*j,3),1) ,
            min(div(200*i+199,2),200*j+199,div(400*i-200*j+398,3),1000)
      for l = max(div(200*j+k+1,2),200*i-k,k) ,
              min(div(200*j+k+199,2),200*i-k+199,2*k+1)
```

Parcours produit avec l'algorithme entrelacé

On obtient le même parcours qu'avec la version descendante.

Evaluation des parcours

Algorithme	gain	# op élémentaires	10 * tps d'exécution	trous
Ancourt-Irigoin	-	7710317	22.290s	276510 au niveau l
descendant	4.3	2224350	11.180s	10 au niveau k
entrelacé	3.6	””	””	””

4.4.2 Polyèdre P_2

Ensemble de contraintes

Il comprend 17 inégalités et 5 variables i, j, k, l, m. Le polyèdre P_2 associé comprend 50600 vecteurs entiers et 58 sommets rationnels.

$$\begin{cases} i & -i+7 & j \\ -j+13 & k & -k+19 \\ l-1 & -l+100 & m-1 \\ 10*l-m+1 & -500*i+5*l & 500*i-5*l+499 \\ -300*j+l+m & 300*j-l-m+299 & -l-m+3999 \\ -200*k+2*l+m & 200*k-2*l-m+199 & \end{cases} \geq 0$$

Parcours produit avec l'algorithme de Ancourt et Irigoin

```
for i = 0 , 1
  for j = div(i,3) , div(2750*i+2747,750)
    for k = div(3*j,2) , 19
      for l = max(100*i,1) , min(div(500*i+499,5),100)
        for m = max(1,300*j-l,200*k-2*l) ,
                min(10*l+1,300*j-l+299,200*k-2*l+199)
```

Parcours produit avec l'algorithme descendant

```
for i = 0 , 1
  for j = div(i,3) , 3
    for k = max(div(18*j,11),i,div(i+3*j,2)) ,
            min(div(6000*i+5993,1000),div(600*j+597,200),
                div(300*j+399,200))
      for l = max(100*i,1,-300*j+200*k-299,div(100*k+5,6),
                  div(300*j+9,11)) ,
              min(div(500*i+499,5),100,-300*j+200*k+199,300*j+298)
        for m = max(1,300*j-l,200*k-2*l) ,
                min(10*l+1,300*j-l+299,200*k-2*l+199)
```

Parcours produit avec l'algorithme entrelacé

```
for i = 0 , 1
  for j = 0 , min(3,div(2750*i+2747,750))
    for k = max(div(i+3*j,2),i,div(18*j,11)) ,
            min(div(300*j+399,200),div(6000*i+5993,1000))
      for l = max(div(300*j+9,11),div(100*k+5,6),-300*j+200*k-299,1,
                  100*i) ,
              min(300*j+298,100*k+99,-300*j+200*k+199,100,
                  div(500*i+499,5))
        for m = max(200*k-2*l,300*j-l,1) ,
                min(200*k-2*l+199,300*j-l+299,10*l+1)
```

Evaluation des parcours

Algorithme	gain	# op élémentaires	200 * tps d'exécution	trous
Ancourt-Irigoin	-	305410	7.390s	6759 au niveau m
descendant	36.5	163731	4.000s	0
entrelacé	93.5	164226	4.320s	0

4.4.3 Polyèdre P_3

Le système définissant P_3 comprend 20 contraintes et porte sur 6 variables i, j, k, l, m, n. Ce polyèdre contient 502500 vecteurs entiers et possède 192 sommets rationnels.

$$\left\{\begin{array}{lll} i & -i+3 & j \\ -j+3 & k & -k+3 \\ l & -l+3 & m-1 \\ -m+1000 & -m+n & 2*m-n+1 \\ -1000*i+m+n & 1000*i-m-n+999 & -1000*j-m+2*n \\ 1000*j+m-2*n+999 & -1000*k+3*m+1 & 1000*k-3*m+998 \\ -1000*l+2*m+n-2 & 1000*l-2*m-n+1001 & \end{array}\right. \geq 0$$

Pour cet exemple, seul l'algorithme entrelacé a pu produire un parcours, notre implémentation de l'algorithme de Ancourt et Irigoin ainsi que la version descendante ne pouvant pas produire le système S' résultant de la phase I, faute de mémoire suffisante sur la station de travail. Le parcours synthétisé avec la version entrelacée est le suivant :

```
for i = 0 , 3
  for j = max(2*i-3,0) ,
          min(div(3000*i+3001,2000),div(5000*i+7997,5000),3,
              div(5000*i+4989,1000))
    for k = i ,
            min(div(3000*i+2999,2000),div(1500*j+1499,500),
                div(2000*i-1000*j+1999,1000),3)
      for l = max(div(750*j+1250*k-3,1500),div(375*i+125*k-1,375),
                  div(250*k-1,250),div(500*j-2,375),
                  div(1250*i-250*j-251,750),0) ,
              min(div(120*j+200*k+319,240),div(600*j+599,200),
                  div(3000*i+1000*k+3989,3000),
                  div(5000*i-1000*j+4989,3000),div(3000*i+2993,2000),3)
        for m = max(div(1000*i+1,3),div(2000*i-1000*j-997,3),
                    div(1000*j,3),250*l+1,-1000*i+1000*l-997,
```

```
                -200*j+400*l-199,div(1000*k+1,3)) ,
            min(div(1000*i+999,2),1000*j+999,div(1000*l+1001,3),
                -1000*i+1000*l+1001,div(2000*i-1000*j+1998,3),
                div(-1000*j+2000*l+2002,5),1000,div(1000*k+998,3))
        for n = max(1000*l-2*m+2,div(1000*j+m+1,2),1000*i-m,m) ,
                min(1000*l-2*m+1001,div(1000*j+m+999,2),1000*i-m+999,
                    2*m+1)
```

Le nid de boucles ne contient que 2 trous situés à la profondeur m.

4.4.4 Polyèdre P_4

Le dernier exemple est le polyèdre le plus complexe que nous avons généré à ce jour. Le système associé porte sur 8 variables i, j, k, l, m, n, o, p et comprend 28 contraintes :

$$\left\{\begin{array}{lll} i & j & k \\ l & m & n \\ -i+3 & -j+3 & -k+3 \\ -l+3 & -m+3 & -n+3 \\ o-1 & -o+1000 & -o+p \\ 2*o-p+1 & -1000*i+o+p & 1000*i-o-p+999 \\ -1000*j-o+2*p & 1000*j+o-2*p+999 & -1000*k+3*o+1 \\ 1000*k-3*o+998 & -1000*l+2*o+p-2 & 1000*l-2*o-p+1001 \\ -1000*m+o & 1000*m-o+999 & -1000*n+2*p-3 \\ 1000*n-2*p+1002 & & \end{array}\right. \geq 0$$

P_4 possède 976 sommets rationnels et 502499 vecteurs entiers. De même que pour le polyèdre P_3, seule la version entrelacée a pu produire un parcours (donné ici à titre d'illustration!) :

```
for i = 0 , 3
  for j = max(2*i-3,0) ,
          min(div(3000*i+3001,2000),div(5000*i+7997,5000),3,
              div(5000*i+4989,1000))
    for k = i ,
            min(div(3000*i+2999,2000),div(1500*j+1499,500),
                div(2000*i-1000*j+1999,1000),3)
      for l = max(div(750*j+1250*k-3,1500),div(375*i+125*k-1,375),
                  div(250*k-1,250),div(1250*i-250*j-251,750),
                  div(500*j-2,375),0) ,
              min(div(120*j+200*k+319,240),div(600*j+599,200),
                  div(3000*i+1000*k+3989,3000),
                  div(5000*i-1000*j+4989,3000),div(3000*i+2993,2000),3,
                  div(3000*j+1000*k+15983,6000))
        for m = max(div(-250*j-249,250),div(2*i-j-1,3),div(j,3),
                    div(l,4),-i+l-1,div(-j+2*l-1,5),div(k,3)) ,
                div(k,3)
          for n = max(div(375*j+125*k-1,375),div(250*k-2,375),
                      div(500*l-1000*m-999,250),
                      div(-1000*k+1500*l-997,750),2*l-4,
                      div(500*j+250*l-1,625),div(500*l-1,750),
                      div(500*j-2,375),div(1500*i-500*k-501,750),
                      div(250*i+250*j-1,375),div(500*i-250*l-251,125),
                      div(250*i-1,250),0) ,
                  min(div(-2000*k+3000*l+2999,1500),
```

```
            div(750*i-250*k+748,375),
            div(500*j+500*m+997,500),
            div(3000*j+1000*k+3983,3000),j+1,
            div(1000*i+2000*j+2989,2000),l,
            div(1000*i-500*l+997,250))
for o = max(div(1000*i+1,3),div(2000*i-1000*j-997,3),
            1000*i-500*n-501,div(1000*j,3),250*l+1,
            -1000*i+1000*l-997,-200*j+400*l-199,
            500*l-250*n-249,250*n+1,-1000*j+1000*n-995,
            div(1000*k+1,3),1000*m) ,
        min(div(1000*i+999,2),1000*j+999,
            div(1000*l+1001,3),500*n+501,
            -1000*i+1000*l+1001,div(2000*i-1000*j+1998,3),
            div(-1000*j+2000*l+2002,5),-1000*j+1000*n+1002,
            1000*i-500*n+997,div(1000*l-500*n+999,2),1000,
            div(1000*k+998,3),1000*m+999)
  for p = max(500*n+2,1000*l-2*o+2,div(1000*j+o+1,2),
              1000*i-o,o) ,
          min(500*n+501,1000*l-2*o+1001,
              div(1000*j+o+999,2),1000*i-o+999,2*o+1)
```

Le parcours contient un trou à la profondeur m et un trou à la profondeur o.

5 Extension pour le parcours d'un polyèdre paramétré

Soit $P(y) = \{x = (x_1, \ldots, x_n)^t \ / \ Ay + Bx + c \geq 0\}$ le polyèdre paramétré dans le contexte $Q = \{y \ / \ My + h \geq 0\}$ (A, B, M sont des matrices entières et c, h des vecteurs entiers). On note $S = Ay + Bx + c \geq 0$ et $C = My + h \geq 0$ les systèmes d'inégalités associés. Les algorithmes descendant et entrelacé peuvent naturellement être étendus afin de synthétiser le parcours de $P(y)$ dans le contexte Q. Dans les deux cas, la méthode va consister à calculer un système équivalent à S (dans le contexte C) de la forme

$$\Lambda(y) \ \biguplus \ Min(x_1) \uplus Max(x_1) \ \biguplus \ \ldots \ \biguplus \ Min(x_n) \uplus Max(x_n)$$

où $Min(x_i)$ et $Max(x_i)$ sont des ensembles de contraintes minimisantes et maximisantes pour x_i et $\Lambda(y)$ un système d'inégalités ne portant que sur le vecteur des paramètres y. . Si $Binf_i$ (resp. $Bsup_i$) est l'ensemble des bornes inférieures (resp. supérieures) pour x_i issues de $Min(x_i)$ (resp. $Max(x_i)$), la structure de contrôle énumérant les vecteurs de $P(y)$ dans le contexte Q devient alors

$$\begin{array}{l} \textbf{for } x_1 = \textbf{max } Binf_1, \ \textbf{min } Bsup_1 \\ \quad \ddots \\ \qquad \textbf{for } x_n = \textbf{max } Binf_n, \ \textbf{min } Bsup_n \end{array}$$

si $\Lambda(y) = \emptyset$ et

if $\bigwedge_{i=1}^{r} t_i y + g_i \geq 0$
then for $x_1 =$ **max** $Binf_1$, **min** $Bsup_1$
$\quad \ddots$
$\quad\quad$ **for** $x_n =$ **max** $Binf_n$, **min** $Bsup_n$
else skip

si $\Lambda(y) = \{t_1 y + g_1 \geq 0, \ldots, t_r y + g_r \geq 0\}$ (les vecteurs entiers du contexte Q qui ne vérifient pas les inégalités de $\Lambda(y)$ définissent des instances vides de $P(y)$).

5.1 Algorithme descendant paramétré

Les deux phases de l'algorithme descendant deviennent :

Phase I Calcul d'un système équivalent à S obtenu en appliquant récursivement l'élimination par paires dans la propriété 1 :

$$S' = \Lambda(y) \biguplus Min(x_1) \uplus Max(x_1) \biguplus \ldots \biguplus Min(x_n) \uplus Max(x_n)$$

Phase II Elimination de contraintes redondantes dans S'. Le système S'' issu de cette phase a la forme :

$$S'' = \Lambda'(y) \biguplus Min'(x_1) \uplus Max'(x_1) \biguplus \ldots \biguplus Min'(x_n) \uplus Max'(x_n)$$

où $\Lambda'(y)$ est l'élimination des contraintes redondantes de $\Lambda(y)$ dans le contexte C et pour i=1,2,...,n dans cet ordre, les ensembles $Min'(x_i)$ et $Max'(x_i)$ sont calculés en parallèle comme suit :

- $Min'(x_i)$ = élimination des contraintes redondantes de $Min(x_i)$ dans le contexte $C \biguplus \Lambda'(y) \biguplus Min'(x_1) \uplus Max'(x_1) \biguplus \ldots \biguplus Min'(x_{i-1}) \uplus Max'(x_{i-1})$
- $Max'(x_i)$ = élimination des contraintes redondantes de $Max(x_i)$ dans le contexte $C \biguplus \Lambda'(y) \biguplus Min'(x_1) \uplus Max'(x_1) \biguplus \ldots \biguplus Min'(x_{i-1}) \uplus Max'(x_{i-1})$

5.2 Algorithme entrelacé paramétré

Le système équivalent à S qui va permettre de synthétiser la conditionnelle énumérant les vecteurs entiers de $P(y)$ dans le contexte Q est maintenant donné par l'appel *sys_eq_prm* S n C à la fonction décrite dans la figure 7.

6 Conclusion

Dans ce rapport, nous avons présenté les deux algorithmes implémentés dans le compilateur Pandore permettant la synthèse du parcours d'un polyèdre paramétré dans un contexte

```
sys_eq_prm (S, i, C) =
  si i = 0
  alors elim_red_ctxt (S, C)
  sinon soit Min_x_i ⊎ Max_x_i ⊎ Nul_x_i la partition de S
        dans soit Min'_x_i = elim_red_ctxt (Min_x_i, Max_x_i ⊎ Nul_x_i ⊎ C)
             dans soit Max'_x_i = elim_red_ctxt (Max_x_i, Min'_x_i ⊎ Nul_x_i ⊎ C)
                  dans soit Nul'_x_i = elim_red_ctxt (Nul_x_i, Min'_x_i ⊎ Max'_x_i ⊎ C)
                       dans soit S_x_i = Elim(x_i, Min'_x_i, Max'_x_i) ∪ Nul'_x_i
                            dans sys_eq_prm (S_x_i, i − 1, C) ⊎ Min'_x_i ⊎ Max'_x_i
```

FIG. 7 - *Algorithme entrelacé dans le cas paramétré*

donné. Ces algorithmes rendent la méthode de Ancourt et Irigoin applicable aux polyèdres utilisés dans la compilation de boucles pour machine parallèle à mémoire distribuée. Pour ces polyèdres en effet, la première phase de projections successives du polyèdre selon les différents axes peut produire beaucoup d'inégalités, au point d'être impossible faute de mémoire suffisante, rendant la phase d'élimination des contraintes redondantes qui s'en suit problématique.

Les deux algorithmes mettent l'accent sur la politique de suppression des inégalités redondantes ; ils reposent tous deux sur un test unitaire de redondance mis en œuvre avec un simplexe optimisé et prennent en compte les propriétés de l'élimination par paires afin d'améliorer le temps de production des parcours et au passage leur temps d'exécution. Le premier algorithme décrit reconsidère la deuxième phase d'élimination des contraintes redondantes dans l'algorithme donné par Ancourt et Irigoin. Dans Pandore, cette version vise les polyèdres pour lesquels le système issu de la première phase peut être produit en mémoire et possède une taille *raisonnable*, c'est à dire qui rend applicable la phase de suppression des contraintes redondantes qui s'en suit. La deuxième version présentée permet la génération de parcours pour les autres polyèdres ; elle réalise un entrelacement de projections et de suppressions de contraintes implicites afin d'éviter le risque d'explosion combinatoire du nombre d'inégalités possible dans la première version.

Remerciements

Ils vont à Yannick Saouter pour avoir consacré du temps à lire ce rapport.

Références

[Amarasinghe et al.93] S. M. Amarasinghe et M. Lam. Communication Optimization and Code Generation for Distributed Memory machines. *in ACM SIGPLAN'93 Conference on Programming Language Design and Implementation*, Juin 1993.

[Ancourt91] C. Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales.* Thèse, Université de Paris VI, Mars 1991.

[Aponte et al.90] M.V. Aponte, A. Lavaille, M. Mauny, A. Suarez, et P. Weis. *The CAML Reference Manual.* Rapport technique 121, INRIA, Rocquencourt, France, Septembre 1990.

[Cheng87] M.C. Cheng. General Criteria for Redundant and Nonredundant Linear Inequalities. *Journal of Optimization Theory and Applications*, 53(1):37–42, Avril 1987.

[Collard et al.93] J.F. Collard, P. Feautrier, et T. Risset. *Construction of DO Loops from Systems of Affine Constraints.* Rapport technique 93-15, LIP, Lyon, France, 1993.

[Duffin74] R.J. Duffin. On Fourier's Analysis of Linear Inequality Systems. *Mathematical Programming Study*, 1:71–95, 1974.

[Feautrier89] P. Feautrier. Semantical Analysis and Mathematical Programming. Application to Parallelization and Vectorization. *in* M. Cosnard et al., editeurs, *Parallel and Distributed Algorithms*, pages 309–320, Elsevier Science Publishers B.V. (North Holland), 1989.

[Irigoin et al.91] F. Irigoin et C. Ancourt. Scanning Polyhedra with DO Loops. *in Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 39–50, Avril 1991.

[LeFur et al.93] M. Le Fur, J.L. Pazat, et F. André. *Static Domain Analysis for Compiling Commutative Loop Nests.* Rapport technique 2067, INRIA, France, Octobre 1993.

[LV et al.94] H. Le Verge, V. Van Dongen, et D. K. Wilde. *Loop Nest Synthesis Using the Polyhedral Library.* Rapport technique 830, IRISA, Rennes, France, Mai 1994.

[Lv92a] H. Le Verge. *A Note on Chernikova's Algorithm.* Rapport technique 635, IRISA, Rennes, France, Février 1992.

[Lv92b] H. Le Verge. *Un environnement de transformation de programme pour la synthèse d'achitectures régulières.* Thèse, IFSIC, Université de Rennes I, Octobre 1992.

[Nemhauser et al.88] G.L. Nemhauser et Wolsey. L.A. *Integer and Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics and Optimization*, John Wiley and Sons, 1988.

[Papadimitriou et al.82] C. H. Papadimitriou et K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice-Hall, 1982.

[Sakarovitch84] M. Sakarovitch. *Optimisation combinatoire, méthodes mathématiques et algorithmiques, graphes et programmation linéaire*. Enseignement des sciences, Hermann, 1984.

[Schrijver86] A. Schrijver. *Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization*, John Wiley and Sons, 1986.

[Wolf et al.91] M. E. Wolf et M. S. Lam. A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *in IEEE Transactions on Parallel and Distributed Systems*, Octobre 1991.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399