Table des matières

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaïa
Faculté des Sciences Exactes
Département d'Informatique

École Doctorale Réseaux et Systèmes Distribués (ReSyD)

Mémoire de Magister

En Informatique

Option : Réseaux et Systèmes Distribués

Thème

Le dénombrement des points d'un réseau contenus dans un polytope dans les calculs parallèles

Présenté par

Meriama MAHAMDIOUA

Devant le jury composé de :

Président	Abdelnasser	DAHMANI	Professeur	Université de Bejaïa, Algérie
Rapporteur	Ali	MELIT	M.C.A	Université de Jijel, Algérie
Examinateur	Mohamed	BENMOHAMED	Professeur	Université de Constantine, Algérie
Examinateur	Abdallah	BOUKERRAM	M.C.A	Université de Sétif, Algérie
Invité	Alex	ESBELIN	M.C	Université de Clermont-Ferrand, France

Promotion 2007-2008

FIG. 0-1 –

Liste des figures

Introduction Générale

Les unités fonctionnelles indépendantes qui constituent les machines modernes permettent l'exécution de plusieurs instructions simultanément si les dépendances de données le permettent et si les ressources nécessaires sont disponibles.

Cependant, et malgré la capacité du matériel, et avant l'exigence en performance ainsi que la puissance de calcul demandée par les applications modèrnes, certains programmes ont un comportement qui ne leur permet pas de bénéficier pleinement des performances matérielles qui leurs sont offertes, ce qui exige de se diriger vers les applications parallèles. Le programmeur est devant deux situations :

- *Développement des programmes parallèles en utilisant des langages à parallélisme explicite.* Dans ce cas, c'est au programmeur de paralliliser les données, faire la communicatiion entre les processeurs, la synchronisation,...etc. Ces opérations nécessitent des connaissances approfondies de l'architecture et réduisent notablement la portabilité.

- *Parallélisation automatique d'un langage séquentiel de haut niveau.* Dans cette approche, le programmeur se débarrasse de la programmation parallèle, tout est réalisé par le compilateur, qui doit alors générer un code parallèle, en prenant en compte les caractéristiques de l'architecture.

Or, les caractéristiques des architectures du matériel parallèle deviennent plus compliqués, il est plus difficile pour que le programmeur puisse donner les solutions optimales. Ce que augmente le besoin d'outils de parallélisation automatique, qui ont la grande promesse en améliorant l'exécution des programmes parallèles .

Le plus souvent, ces outils automatiques sont centrées sur l'analyse et la transformation des structures de contrôle itératives -*nids de boucles* - qui présentent généralement les parties de programme qui portent plus de parallélisme, qui utilisent la plus grande partie de la mémoire, et qui consomment le plus de temps de calcul.

Par nature, la plupart des nids de boucles qui apparaissent dans des programmes sont affines, c'est-à-dire qu'ils s'expriment par des indices bornés par des expressions affines, et des accès à des tableaux par des fonctions affines des indices. Le domaine d'itération d'un nid de boucles affines peut être décrit par les points entiers appartenant à un polytope (polyèdre borné).

De nombreuses méthodes d'optimisation de programmes formalisent les problèmes d'analyse et de transformation de nids de boucles par le calcul du nombre de points entiers dans des polytopes. Lorsque les nids de boucles sont paramétrés, i.e., dépendent d'un ensemble de constantes symboliques non connues au moment de la compilation, les problèmes d'analyse et d'optimisation de nids de boucles se réduisent souvent au comptage de points entiers dans des polytopes paramétrés. Dans ce cas, le résultat du comptage est

donné par un ou plusieurs polynômes ayant des coefficients périodiques. Ces polynômes sont connus sous le nom de quasi-polynômes d'Ehrhart.

Dans ce travail, on s'intéresse aux dénombrement de points entiers dans un polytope, en expliquant son intérêt dans la parallélisation automatique.

Ce mémoire est organisé comme suit :

- Chapitre 1 : Contient des notions mathématiques qui sont nécessaires pour le reste du mémoire.

- Chapitre 2 : Expose des notions de parallélisation et localise le modèle polyédrique dans le domaine de parallélisation. Ainsi que l'utilisation de nombre de points entiers dans un polytope pour des intéréts de parallélisation.

- Chapitre 3 : Dans la première section de ce chapitre, nous parlons des différentes méthodes de dénombrement de points entiers dans un polytope, et dans la deuxième section, nous donnons des exemples d'utilisation de ce nombre dans des recherches actuelles dans le contexte de parallélisation.

- Chapitre 4 : Ce chapitre contient le travail réalisé. Nous proposons des algorithmes de dénombrement de points entiers dans un contexte bien précis, en terminant par un exemple d'utilisation de notre résultat.

Chapitre 1

Notions de base

1.1 Introduction

Le modèle polyédrique est un formalisme qui permet une représentation des calculs parallèles. Ce modèle est basé sur les systèmes d'équations et d'inéquations affines.

Ce chapitre contient des définitions mathématiques nécessaires à la compréhension des méthodes de dénombrement de points entiers dans un polytope traitées après. Il est organisé comme suit : La section 1.2 donne des définitions des notions mathématiques tels que affine, convexe, linéaire...etc. La section 1.3 introduit le modèle polyédrique avec ses différentes présentations tels que présentation implicite, paramétrique...etc. En fin, on termine par la définition du polynôme d'Ehrhart qui représente le nombre de points entiers dans un polytope.

1.2 Définitions mathématiques

Notation 1.2.1 *Soit $M_{m,n}(K)$ l'ensemble des matrices ayant m lignes et n colonnes à coefficients dans K. On note cet ensemble par $K^{m\times n}$.*

Définition 1.2.2 *Fonction linéaire et fonction affine*
- Une fonction k-dimension est linéaire ssi elle peut être écrite sous la forme

$$f(\overrightarrow{v}) = M_f \quad \overrightarrow{v}$$

où $\overrightarrow{v} = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_d \end{pmatrix}$ *et* $M_f \in \mathbb{R}^{k \times d}$ *est une matrice avec* k *lignes et* d *colonnes [13].*

- *Une fonction k-dimension est affine ssi elle peut être écrite sous la forme*

$$f(\overrightarrow{v}) = M_f \quad \overrightarrow{v} + \overrightarrow{f_0}$$

où $\overrightarrow{v} = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_d \end{pmatrix}$ *et* $M_f \in \mathbb{R}^{k \times d}$ *est une matrice avec* k *lines et* d *colonnes,* $\overrightarrow{f_0} \in \mathbb{R}^k$ *est un* $k-$*dimension vecteur [13].*

Définition 1.2.3 *Combinaisons*

Soit le vecteur $x = (x_1, x_2, ..., x_d)$ *, et le vecteur de coefficients* $\lambda = (\lambda_1, ..., \lambda_d)$*. La combinaison :*

- $\sum_{i=1}^{d} \lambda_i x_i$ *est une combinaison linéaire.*
- $\sum_{i=1}^{d} \lambda_i x_i$ *est une combinaison positive si* $\lambda_i \geq 0$*.*
- $\sum_{i=1}^{d} \lambda_i x_i$ *est une combinaison affine si* $\sum_{i=1}^{d} \lambda_i = 1$*.*
- $\sum_{i=1}^{d} \lambda_i x_i$ *est une combinaison convexe si* $\lambda_i \geq 0$ *et* $\sum_{i=1}^{d} \lambda_i = 1$*.*

Définition 1.2.4 *Système linéaire diophantien*

Un système linéaire diophantien est un ensemble d'inéquations linéaires à coefficients entiers $Ax \leq 0$ *dont on cherche les solutions entières, où* A *est une matrice d'entiers à* m *lignes et* n *colonnes,* b *un vecteur de* $\mathbb{Z}^m$ *et* x *est un vecteur de* n *indéterminées.*

Définition 1.2.5 *Espace affine*

Un ensemble de vecteurs est un espace affine ssi il est fermé sous combinaisons affines, c-a-d, si $\overrightarrow{x}, \overrightarrow{y}$ *sont dans l'espace, tous les points trouvés sur la ligne liée* $\overrightarrow{x}$ *et* $\overrightarrow{y}$ *sont dans l'espace [13].*

Définition 1.2.6 *Hyperplan affine*

Un hyperplan affine [13] est un sous espace affine de dimension $(n-1)$ *d'un espace de*

dimension n. Dans notre contexte, l'ensemble de tous les vecteurs $v \in \mathbb{Z}^n$ tel que $h.\overrightarrow{v} = k$ pour $k \in \mathbb{Z}$, forme un hyperplan affine. Un hyperplan $h.\overrightarrow{v} = k$ divise l'espace en deux demi-espace, comme illustré dans la figure.

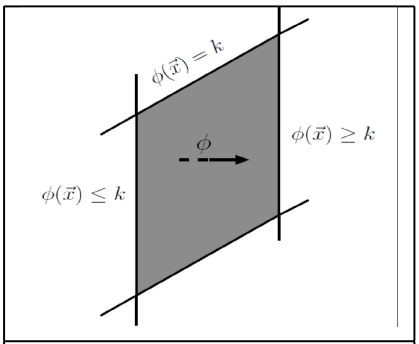


Fig.1.1 : Hyperplan affine

Définition 1.2.7 *Enveloppe linéaire*
L'enveloppe linéaire d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$est l'ensemble donné par la combinaison linéaire

$$LinX = \left\{ \sum_{i=1} \lambda_i x_i \mid xi \in X, \lambda_i \in \mathbb{R} \right\}$$

Définition 1.2.8 *Enveloppe positive*
L'enveloppe positive d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison positive

$$pos\ X = \left\{ \sum_{i=1} \lambda_i x_i \mid xi \in X, \lambda_i \geq 0 \right\}$$

Définition 1.2.9 *Enveloppe affine*
L'enveloppe affine d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison affine [35] [33]

$$aff\ X = \left\{ \sum_{i=1} \lambda_i x_i \mid x_i \in X, \sum_i \lambda_i = 1 \right\}$$

Définition 1.2.10 *Enveloppe convexe*
L'enveloppe convexe d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison convexe[33]

$$convX = \{ \sum_i \lambda_i\ x_i \mid x_i \in X, \lambda_i \geq 0, \sum_i \lambda_i = 1 \}$$

- Dans le plan, l'enveloppe convexe est le plus petit polygone contenant un ensemble de points.

Définition 1.2.11 *Enveloppe convexe inférieur et enveloppe convexe supérieure*
L'enveloppe convexe (noté $EC(S)$ dans l'image suivante) peut être considérée comme l'intersection de deux ensembles convexes non bornés qu'on appelle enveloppe convexe supérieure (noté $ECS(S)$ dans l'image suivante), et enveloppe convexe inférieure (noté $ECI(S)$ dans l'image suivante)

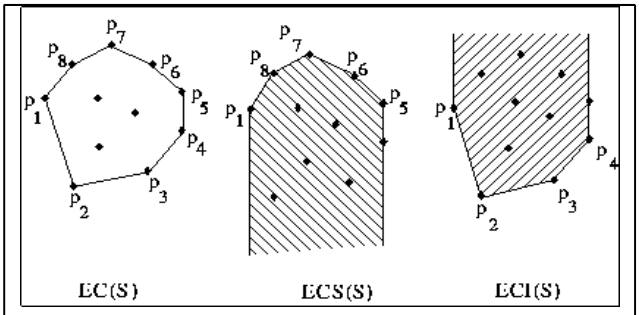


Fig.1.2 : Enveloppe convexe, enveloppe convexe supérieure et enveloppe convexe inférieur

Définition 1.2.12 *Polynôme d'interpolation de Lagrange*
Soient n points distincts $x_0, x_1, \ldots, x_n$ d'un intervalle fermé borné $[a, b]$, et une fonction f définie sur $[a, b]$ à valeurs dans $\mathbb{R}$, il existe un unique polynôme P de degré inférieur ou égal n, tel que $P(x_i) = f(x_i)$ pour $i = 0, 1, \ldots, n$. Ce polynôme est donné par :

$$P(X) = \sum_{i=0}^{n} \left(f(x_i) \prod_{j=0,\ j\neq i}^{n} \frac{X - x_j}{x_i - x_j} \right)$$

Les polynômes L_i définies par $L_i = \prod_{j=0,\ j\neq i}^{n} \frac{X - x_j}{x_i - x_j}$ sont appelés les polynômes de Lagrange.

Définition 1.2.13 *Formule de Presburger*
Une formule de Presburger se compose d'égalité affine et/ou des contraintes d'inégalité connectés via les opérateurs logiques : $\neg$ (non), $\wedge$ (et), $\vee$ (ou) et des quantificateurs $\forall$(quelque soit), $\exists$ (Il existe) [28].

1.3 Modèle polyédrique

Définition 1.3.1 *Polyèdre rationnel*
Un polyèdre rationnel P est l'intersection d'un nombre fini de demi espaces fermés [18].

Un polyèdre est défini par un système d'égalités et d'inégalités :

$$P = \left\{ x \in \mathbb{Q}^d |\ Ax \geq c \wedge A'x = c' \right\}$$

avec $A \in \mathbb{Z}^{m \times d}, A' \in \mathbb{Z}^{m' \times d}, c \in \mathbb{Z}^m, c' \in \mathbb{Z}^{m'}$ *, ou* m *est le nombre des inégalités et* m' *est le nombre des égalités [33].*

Définition 1.3.2 *Polytope*
Un polytope est un polyèdre borné ou fini.
L'image suivante présente des exemples des polytopes :

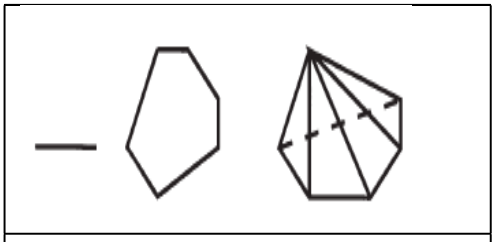


Fig 1.3 : Exemples des polytopes

Définition 1.3.3 *Dimension (géométrique) d'un polyèdre rationnel*
La dimension (géométrique) d'un polyèdre rationnel $P \subset \mathbb{Q}^d$ *est de la dimension de son enveloppe affine, ou d'autre manière, la dimension* d *de son espace ambiant* $\mathbb{Q}^d$ *moins le nombre d'équations (implicites), linéairement indépendantes. Un polyèdre de dimension* d *est appelé un* d*-polyèdre [33].*

Définition 1.3.4 *Sommet*
Un sommet de polytope est un point extrême de polytope, c-à-d, il ne peut pas être exprimé par une combinaison convexe de ses autres points [32].

Définition 1.3.5 *Rayon*
Un rayon d'un polyèdre rationnel P *est un vecteur non nul* r*, tel que* $(x + \mu r) \in P$ *quels que soient* $x \in P$ *et* $\mu \in \mathbb{Q}$ *avec* $\mu \geq 0$*. Un rayon représente une direction vers laquelle le polyèdre va à l'infini [33].*

Définition 1.3.6 *Ligne*
Une ligne d'un polyèdre rationnel P *est un vecteur non nul* l*, tel que* $(x + \lambda l) \in P$ *quels que soient* $x \in P$ *et* $\lambda \in \mathbb{Q}$*. Une ligne est aussi appelée rayon bidirectionnel [33].*

Définition 1.3.7 *Face, facette*
Une face F *d'un polyèdre rationnel* P *est l'intersection de* P *avec* $\{x \in \mathbb{Q}^d\ | A'x = c'\}$*, où* $A'x \geq c'$ *est un sous système de* $Ax \geq c$*.*

Les facettes d'un polyèdre rationnel sont ses $(n-1)-$faces (faces de dimension $n-1$), où n est la dimension (géométrique) du polyèdre. Les faces de dimension 0 sont les sommets du polyèdre [33].

Définition 1.3.8 **m**- *simplexe*
Si $x_1, x_2, .., x_{m+1}$ sont des points affinement indépendant (c-à-d, x_2-x_1 $x_3-x_1 \ldots x_{m+1}-x_1$ sont linéairement indépendants), alors l'enveloppe convexe de $x_1, x_2, .., x_{m+1}$ est nommé **m**- *simplexe (ou simplexe de dimension m) avec les sommets $x_1, x_2, .., x_{m+1}$. [18]*

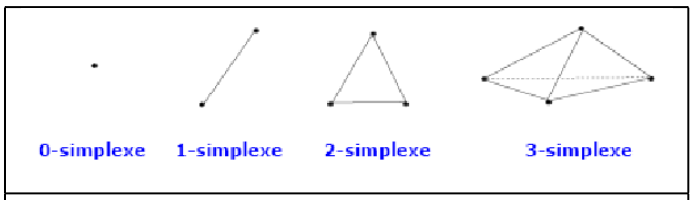


Fig 1.4 : Exemples de m-simplexes

Définition 1.3.9 *Un polytope paramétré borné*
Un polytope paramétré est borné dans $\mathbb{Q}^n$ s'il est défini par un ensemble de contraintes linéaires à coefficients entiers et si chaque coordonnée de ses sommets s'exprime comme une fonction affine de ses paramètres [43].

Définition 1.3.10 *Polytope entier, polytope rationnel*
Un Polytope est dit entier si tous ses sommets sont entiers, et dit rationnel si au moins un sommet possède des coordonnées rationnelles [18].

Théorème 1.3.11 *Un polytope est l'enveloppe convexe de ses sommets.*

Définition 1.3.12 *Réseau-polytope*
On appelle réseau-polytope l'intersection d'un réseau régulier de points avec un polytope [43].

Définition 1.3.13 *$\mathbb{Z}$-polytope*
Un $\mathbb{Z}$-polytope de dimension n est l'intersection du réseau standard $\mathbb{Z}^n$ avec un polytope. C'est donc un réseau-polytope particulier [43].

Exemple 1.3.14 *Le polytope P de la figure suivante est un polygone. L'intersection de ce polytope avec le réseau $\mathbb{Z}^2$ donne les résultats suivants :*

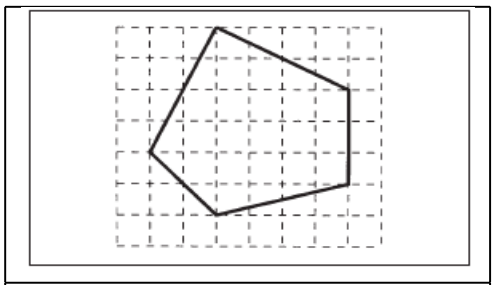


Fig 1.5 :Exemple de z-polytope

Le nombre de points entiers dans l'intérieur de P égale à 20.
Le nombre de points entiers sur le bord de P égale à 10.
Le nombre de points entiers de réseau polytope : $card(P \cap \mathbb{Z}^2) = 30$.

Définition 1.3.15 *Dénominateur d'un polytope*
On appelle dénominateur d'un polytope le plus petit commun multiple des dénominateurs des coordonnées rationnelles des sommets du polytope.

Définition 1.3.16 *Forme implicite, forme non paramétrique*
Un polyèdre paramétré $P(p)$ est un polyèdre dépend de vecteur des paramètres $p \in Q^m$, sa représentation implicite donnée en terme des équations et des inéquations :

$$P(p) = \{x \in Q^n | Ax = Bp + b \ \wedge \ Cx \geq Dp + d\}$$

Un tel polyèdre peut être associé à un polyèdre non paramétré P' :

$$P' = \left\{ \begin{bmatrix} x \\ p \end{bmatrix} \in Q^{n+m} | A' \begin{bmatrix} x \\ p \end{bmatrix} = b \ \wedge \ C' \begin{bmatrix} x \\ p \end{bmatrix} \geq d \right\}$$

d'où

$$A' = [A - B] \qquad C' = [C - D]$$

Définition 1.3.17 *$Proj_d$, $Proj_p$*
Deux projections peuvent être définies : $Proj_d$, $Proj_p$.
La foncrtion $Proj_d$ projète l'espace combiné $\mathbb{Q}^{n+m}$ dans l'espace de données $\mathbb{Q}^n$, et la fonction $Proj_p$ projète l'espace combiné $\mathbb{Q}^{n+m}$ dans l'espace de paramètres $\mathbb{Q}^m$.

Propriété : *Cette propriété relie un polyèdre P' dans l'espace combiné data/parameter à son polyèdre paramétrique équivalent $P(p)$. Soit $S(q)$ l'espace affine de $\mathbb{Q}^{n+m}$ qui vérifie*

$$S(q) = \left\{ \begin{bmatrix} x \\ q \end{bmatrix} \in Q^{n+m} \right\}$$

Alors, on a

$$P(p) = Proj_d(P' \cap S(q)) = \left\{ x \mid \begin{bmatrix} x \\ p \end{bmatrix} \in P' \right\}$$

Où p est le vecteur de paramètres.

Théorème 1.3.18 *Soit $v_i(p)$ un sommet paramétré défini sur m paramètres linéairement indépendant $\in \mathbb{Q}^m$. Le polyèdre P' peut être construit facilement de $P(p)$. Pour chaque sommet de $P(p)$, il existe $m - face$ $F_i^m(P')$ tel que $v_i(p) = Proj_d(F_i^m(P') \cap S(q))$.*

Définition 1.3.19 *Représentation homogène d'un système*
La définition homogène [42] d'un système peut être extraite de la représentation inhomogène en faisant la transformation suivante $x \rightarrow \begin{pmatrix} \xi x \\ \xi \end{pmatrix}$, ce qui change le système inhomogène P de dimension n en un système homogène P' de dimension $n+1$, de la manière suivante :

$$P = \{x \in |Ax = b \ \wedge \ Cx \geq d\} = \{x \in |Ax - b = 0 \ \wedge \ Cx - d \geq 0\}$$

$$\begin{aligned} C &= \left\{ \begin{pmatrix} \xi x \\ \xi \end{pmatrix} | \xi Ax - \xi b = 0, \ \xi Cx - \xi d \geq 0, \ \xi \geq 0 \right\} \\ &= \left\{ \begin{pmatrix} \xi x \\ \xi \end{pmatrix} | (A \mid -b) \begin{pmatrix} \xi x \\ \xi \end{pmatrix} = 0, \left(\begin{array}{c|c} C & -d \\ \hline 0 & 1 \end{array} \right) \begin{pmatrix} \xi x \\ \xi \end{pmatrix} \geq 0 \right\} \\ &= \left\{ \hat{x} \in |\hat{A} \ \hat{x} = 0 \ \wedge \ \hat{C} \ \hat{x} \geq 0 \right\} \end{aligned}$$

Définition 1.3.20 *Représentation de Mikowski*
Un polyèdre défini sous forme implicite peut être défini sous une forme appellée la forme de Minkowski, en terme des combinaisons linéaires des lignes (colonnes de la matrice L), combinaison convexe des sommets (les colonnes de la matrice V) et combinaison positive de rayons extrêmes (colonnes des matrice R)

$$P = \left\{ x \mid x = L\lambda + R\mu + V\nu \quad \mu, \nu \geq 0 \ , \ \sum \nu = 1 \right\}$$

Définition 1.3.21 *Contexte de sommet*
Les sommets d'un polyèdre paramétré sont paramétrés. Ils sont définis par des fonctions affines des paramètres. Un sommet paramétré est noté $v_i(p)$. Les sommets $v_i(p)$ d'un polyèdre paramétré ne sont pas définis pour toutes les valeurs possibles des paramètres, mais pour un sous ensemble des valeurs de paramètres. L'ensemble des paramètres où un sommet est défini nommé le contexte de sommet (domaine).

Exemple 1.3.22 *Soit le polytope paramétrisé suivant :*

$$P(p) = \{(x_1, x_2) \in \mathbf{Q}^2 | 0 \leq x_2 \leq 4 \ \wedge x_2 \leq x_1 \leq x_2 + 9 \ \wedge x_1 \leq p \wedge p \leq 40\}$$

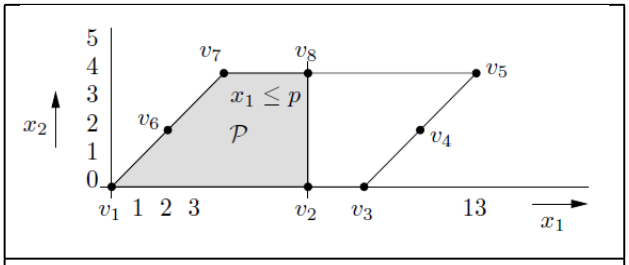


Fig 1.6 : Polytope paramétré P(p)

On définit les sommets et leurs contexte selon la valeur de paramètre p :

$$\begin{aligned}
v_1(p) &= (0,0) \quad si \quad 0 \leq p \leq 40 \\
v_2(p) &= (p,0) \quad si \quad 0 \leq p \leq 9 \\
v_3(p) &= (9,0) \quad si \quad 9 \leq p \leq 40 \\
v_4(p) &= (p,p-9) \quad si \quad 9 \leq p \leq 13 \\
v_5(p) &= (13,4) \quad si \quad 13 \leq p \leq 40 \\
v_6(p) &= (p,p) \quad si \quad 0 \leq p \leq 4 \\
v_7(p) &= (4,4) \quad si \quad 4 \leq p \leq 40 \\
v_8(p) &= (p,4) \quad si \quad 4 \leq p \leq 13
\end{aligned}$$

Définition 1.3.23 *Cône simple rationnel (ou cône polyédrique rationnel)*
Soient $u_1, ..., u_d \in \mathbb{Z}^d$ des vecteurs entiers linéairement indépendants. Un cône simple rationnel [5](ou un cône polyédrique rationnel [33]) généré par les vecteurs $u_1, ..., u_d$ est l'ensemble :

$$K = K(u_1, ..., u_d) = \left\{ \sum_{i=1}^{d} \alpha_i u_i : \alpha_i \geq 0 \ \ pour \ i = 1, ..., d \right\}$$

Un cône polyédrique peut aussi être donné par un système d'inégalités homogènes

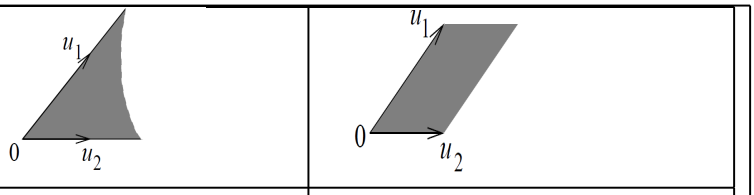
$$\{x \mid Ax \geq 0\}$$

Définition 1.3.24 *Parallélépipède fondamentale*
Soient $u_1, ..., u_d \in \mathbb{Z}^d$ des vecteurs entiers linéairement indépendants. Le Parallélépipède

fondamental de $u_1, ..., u_d$ *[5] est l'ensemble défini par :*

$$\Pi = \Pi(u_1, ..., u_d) = \left\{ \sum_{i=1}^{d} \alpha_i u_i \ , \ avec\ 0 \leq \alpha_i \leq 1 \ \ pour\ i = 1, ..., d \right\}$$

Exemple 1.3.25 *On donne ici un cône et son Parallélépipède*



Cône rationnel simple	*Parallélépipède fondamentale*

Fig 1.7 : Un cône rationnel simple et son Parallélépipède fondamentale

Définition 1.3.26 *Cône supportant d'un* $d-$*polyèdre*
Un cône supportant d'un $d-$*polyèdre* P *en un sommet* v *est la région de* Q^d *bornée par les contraintes de* P *qui sont saturées par* v*. Autrement dit, c'est la région bornée par les contraintes* $\langle a, x\rangle \geq b$*, avec* $\langle a, v\rangle = b$*, où* $\langle ., .\rangle$ *est le produit scalaire des vecteurs [33].*
Le cône supportant $K(P, v)$ *du polyèdre* P *au sommet* v *est la translation au sommet* v *d'un cône polyédrique* K_v *(d'origine* O *comme sommet)* $K(P, v) = K_v + v$*.*

Exemple 1.3.27 *Exemple de polytope et ses cônes supportants*

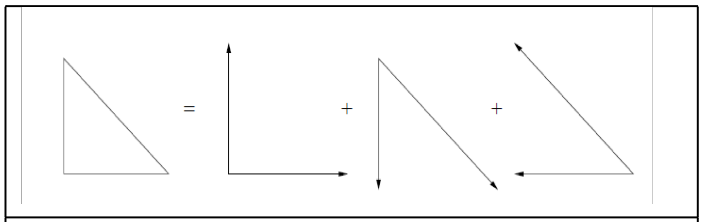


Fig 1.8 : Polytope et ses cônes supportants

Définition 1.3.28 *Cône polyédrique simplicial*
Un cône polyédrique simplicial de dimension d *est un cône polyédrique qui peut être engendré par* d *demi-droites, ou autrement dit, c'est un cône polyédrique ayant un nombre d'arêtes (générateurs) égal à sa dimension* d*.*

Définition 1.3.29 *Cône simplicial unimodulaire*
Un cône simplicial unimodulaire de dimension d *est un cône polyédrique dont les vecteurs générateurs* $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_d \in \mathbb{Z}^d$*, tels que la matrice formée par ses vecteurs générateurs*

est unimodulaire (son déterminant est égal à ± 1), ou d'une manière équivalente, son parallélépipède fondamental comporte un point entier unique (l'origine O).

Définition 1.3.30 *Fonction génératrice*
La fonction génératrice d'un ensemble $A \subset \mathbb{Q}^d$ est sous la forme d'une série des exposants avec un terme pour chaque point entier dans A et peut être écrite comme suit :

$$f(A, x) = \sum_{a \in A \cap \mathbb{Z}^d} x^a$$

1.4 Polynôme d'Ehrhart

Les équations qui composent les systèmes linéaires paramétriques considérés par Ehrhart sont de la forme générale suivante :

$$\sum_i a_i x_i < bp + c \quad , \quad \sum_i a_i x_i = bp + c \quad , \quad \sum_i a_i x_i \leq bp + c$$

où les a_i, b et c sont des constantes entières, les x_i sont les variables du système, et p est un paramètre entier positif.
Le résultat majeur d'Ehrhart est d'avoir démontré que les nombres de points entiers de tels polytopes, sont des polynômes en p, de degré égal à la dimension du plus petit espace contenant le polytope, si le polytope est entier (c-à-d, les coordonnés de leurs sommets sont tous entiers), et sont des pseudo-polynômes (quasi-polynômes) de même caractéristiques si le polytope est rationnel (au moins un sommet possède des coordonnées rationnelles) [18].

Ce résultat est ensuite généralisé pour des systèmes linéaires paramétrés avec plusieurs paramètres, comme on va voir avec le théorème fondamental d'Ehrhart ci après.

Définition 1.4.1 *Pseudo-polynôme*
Pseudo-polynôme (quasi-polynôme) est un polynôme dont les coefficients sont des nombres périodiques [43].

Définition 1.4.2 *Nombre périodique*
Soit le vecteur des paramètres $P = (p_j)_{j=1...m}$ où m est un entier naturel. Un nombre périodique U_P est défini par un tableau U de dimension m et de taille $s_1 \times s_2 \times ... \times s_m$ de la manière suivante :
$U_P = U[i_1, i_2, ..., i_m]$ si pour tout j de 1 à m, $i_j = p_j \ mod \ s_j$. Le vecteur $s = (s_j)$ est appelé la période de U_P.
Dans le cas d'un seul paramètre p, un nombre périodique $u_p = [u_0, u_1, ..., u_{m-1}]$ égal à l'élément dont le rang est égal à p modulo m, m étant appelé la période de u_p.

Exemple 1.4.3 *Soit* $U(p_1,p_2) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}_{p_1,p_2}$ *un nombre périodique où la période* $s = (2,4)$.
Ce nombre est interprété comme suit :

$$U(p_1,p_2) = \begin{cases} [1\ ,\ 0\ ,\ 1\ ,\ 0]_{p2} \ si\ \ P_1 \equiv 0 \bmod 2 \\ \begin{cases} 1 & si\ \ p_1 \equiv 0\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 0\ \ mod\ \ 4 \\ 0 & si\ \ p_1 \equiv 0\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 1\ \ mod\ \ 4 \\ 1 & si\ \ p_1 \equiv 0\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 2\ \ mod\ \ 4 \\ 0 & si\ \ p_1 \equiv 0\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 3\ \ mod\ \ 4 \end{cases} \\ [1\ ,\ 0\ ,\ -1\ ,\ 0]_{p2} \ si\ \ P_1 \equiv 1 \bmod 2 \\ \begin{cases} 1 & si\ p_1 \equiv 1\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 0\ \ mod\ \ 4 \\ 0 & si\ p_1 \equiv 1\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 1\ \ mod\ \ 4 \\ -1 & si\ p_1 \equiv 1\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 2\ \ mod\ \ 4 \\ 0 & si\ p_1 \equiv 1\ \ mod\ \ 2\ \ et\ \ p_2 \equiv 3\ \ mod\ \ 4 \end{cases} \end{cases}$$

Remarque 1.4.4 *Un nombre périodique multidimensionnel peut être représenté par un ensemble*
$U(p_1,p_2) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}_{p_1,p_2} = \Big[[1\ ,0\ ,\ 1\ ,\ 0]_{p2}\ ,\ [1\ ,\ 0\ ,\ -1\ ,0]_{p2}\Big]_{p_1}$.

Définition 1.4.5 *Période maximale*
La période maximale d'un coefficient périodique en P du polynôme d'Ehrhart associé à un polytope borné est égale au plus petit commun multiple des dénominateurs des coefficients en P des coordonnées des sommets du polytope. De manière plus générale, la période des coefficients d'un polynôme d'Ehrhart est bornée par le dénominateur du polytope

Définition 1.4.6 *Degré du polynôme d'Ehrhart*
Le degré du polynôme d'Ehrhart est borné par la dimension du polytope.

Théorème 1.4.7 *Théorème fondamental d'Ehrhart*
Soit P_P, $P = (p_1,p_2,...,p_m)$, *un polytope dans un espace de dimension* n, *et tel que les coordonnées de ses sommets sont des expressions affines en* P. *Le nombre de point entiers de* P_P *est un polynôme à plusieurs variables* $p_1,p_2,\ldots.p_m$ *de degré* n , *si* P_P *est entier, et est un pseudo-polynôme à plusieurs variables* $p_1,p_2,\ldots.p_m$ *de degré* n *dans la pseudo période est le dénominateur de* P_P, *si* P_P *est rationnel. Ce pseudo-polynôme(quasi-*

polynôme ou simplement polynôme d'Ehrhart) est de la forme

$$PE(p_1, p_2, \ldots . p_m) = \sum_{i_1=0}^{n} \sum_{i_2=0}^{n-i_1} \cdots \sum_{i_p=0}^{n-i_1-i_2-\ldots-i_{p-1}} c_{i_1,i_2,\ldots i_p} p_1^{i_1} p_2^{i_2} \ldots p_m^{i_m}$$

Exemple 1.4.8 *Soit le pseudo-polynôme en p :*

$$f(p) = [0, 1, 2]\, p^2 + \left[1, \frac{1}{2}\right] p + \left[0, \frac{1}{4}\right]$$

Selon les valeurs de p modulo 3 et p modulo 2, $f(p)$ prendra les valeurs suivantes :

$$\begin{aligned}
\text{Si } p \bmod 3 &= 0 \text{ et } p \bmod 2 = 0, \text{ alors } f(p) = p \\
\text{Si } p \bmod 3 &= 0 \text{ et } p \bmod 2 = 1, \ f(p) = \frac{1}{2}p + \frac{1}{4} \\
\text{Si } p \bmod 3 &= 1 \text{ et } p \bmod 2 = 0, \ f(p) = p^2 + p \\
\text{Si } p \bmod 3 &= 1 \text{ et } p \bmod 2 = 1, \ f(p) = p^2 + \frac{1}{2}p + \frac{1}{4} \\
\text{Si } p \bmod 3 &= 2 \text{ et } p \bmod 2 = 0, \ f(p) = 2p^2 + p \\
\text{Si } p \bmod 3 &= 2 \text{ et } p \bmod 2 = 1, \ f(p) = 2p^2 + \frac{1}{2}p + \frac{1}{4}
\end{aligned}$$

1.5 Conclusion

Dans ce chapitre nous avons donné des notions mathématiques, afin de simplifier la compréhension de différentes méthodes expliquées dans ce mémoire.

Avant d'entrer dans les détails de ces méthodes, le prochain chapitre localise le modèle polyédrique dans le domaine de parallélisme. Il contient des définitions concernant : nid de boucles, parallélisation automatique basée sur modèle polyédrique...etc.

Chapitre 2

Nid de boucles, modèle polyédrique et parallélisation automatique

2.1 Introduction

Dans la parallélisation automatique, les espaces des itérations des nids de boucles peuvent être représentés par des polyèdres, ce qui permet de trouver un grand nombre d'informations concernant ces programmes comme l'expression des dépendances de données, des calculs d'ordonnancement et de placement de tâches parallèles, des optimisations de génération de code...etc.

Dans ce chapitre, nous commençons par des définitions, puis nous traitons la parallélisation automatique en utilisant le modèle polyèdrique. Ensuite nous expliquons l'optimisation de mémoire dans le calcul parallèle, et nous terminons par l'utilisation de nombre de points entiers dans le calcul parallèle.

2.2 Définitions

Définition 2.2.1 *Nid de boucles*
Un nid de boucles est un ensemble fini de structures itératives imbriquées [10].
En d'autres termes, un nid de boucles est un ensemble de boucles emboitées contenant des instructions exécutables [21].

Définition 2.2.2 *Instruction*
Une instruction est une structure de programme qui ordonne à l'ordinateur d'exécuter une action spécifique (expression arithmétique, instance de variable... etc) [10].

Définition 2.2.3 *Opération*
Chaque instance d'une instruction est appelée une opération [10].

Chaque instruction engendre autant d'opérations que d'itérations.
Une opération est caractérisée par le nom de l'instruction exécutée et par la valeur courante du vecteur d'itération [21].

Définition 2.2.4 *Vecteur d'itération*
Un nid de boucles peut être représenté par un vecteur colonne de taille n appelé le vecteur d'itération

$$x = \begin{pmatrix} i_1 \\ i_2 \\ ... \\ i_n \end{pmatrix}$$

où i_k est le $k^{ième}$ itérateur (l'indexe de la $k^{ième}$ boucle), n est appelé le profondeur de la boucle, qui présente le nombre de boucles qui l'entourent [21].

Définition 2.2.5 *Nid de boucle parfait*
Le nid de boucles est parfait si toutes les instructions sont à la profondeur maximale [21].

Définition 2.2.6 *Espace d'itération*
l'espace d'itération est $\mathbb{N}^d$, où d est le nombre de boucles englobantes.

Définition 2.2.7 *Domaine d'itération*
L'ensemble des valeurs possibles du vecteur d'itération pour une instruction donnée est appelé le domaine d'itération D de l'instruction [10].
En d'autres termes, le domaine d'itération est l'ensemble des valeurs légales du vecteur d'itération. Le domaine d'itération est délimité par les bornes des boucles.

Exemple 2.2.8 *Dans cet exemple, on donne un nid de boucle parfait et le domaine d'itération de l'instruction S :*

for $(i = 0 \ ; \ i < N \ ; \ i++)$
  for $(j = 0 \ ; j < N \ ; \ j++)$
    $S \ : A[i,j] \ = \ A[i,j] \ + \ u1[i] * v1[j] + u2[i] * v2[j]$

$$D^S : \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq 0$$

Domaine d'itération de l'instruction S

Définition 2.2.9 *Ordre lexicographique*
Chaque instance d'instruction est caractérisée par ses coordonnées dans le domaine d'itération correspondant. Quand chaque pas de boucle d'un programme est une variable positive, on appelle cet ordre l'ordre lexicographique [10].
De manière formelle : soient x et y deux vecteurs de dimension d, on dit que x est lexicographiquement plus petit que y, noté $x \prec\prec y$, ssi il existe k, $1 \leq k \leq d$ tel que $x_i = y_i$ pour $i < k$ et $s_k < y_k$ [35].

2.3 Parallélisation Automatique en utilisant le modèle polyédrique

2.3.1 Principe

La parallélisation automatique est le processus de conversion automatique de programme séquentiel à une version qui peut être directement exécutée en parallèle sans toucher la sémantique du programme.[13].

La plupart des nids de boucles qui apparaissent dans des programmes sont affines, c'est-à-dire qu'ils s'expriment par des indices bornées par des expressions affines, et des accés à des tableaux par des fonctions affines des indices. On basant sur le modèle polyédrique, chaque instruction entourée par n boucles est présentée par un polytope de dimension n. Les coordonnées d'un point dans le polytope (vecteur d'itération) correspondent aux valeurs des indices des boucles commençant par la boucle extérieure. Chaque point dans le polytope correspond à une instance de l'exécution d'instruction S .

Exemple 2.3.1 *L'image suivante présente un nid de boucles, le système des inéquations et le polytope présente ce nid de boucles :*

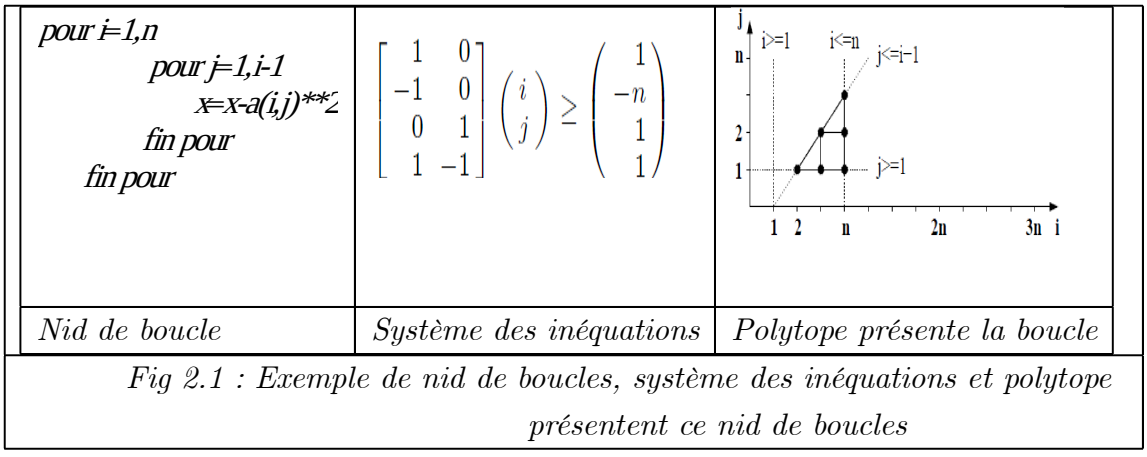


Fig 2.1 : Exemple de nid de boucles, système des inéquations et polytope présentent ce nid de boucles

Le passage de système d'inéquations au nid de boucles peut être fait par l'élémination de Fourier-Motzkin

Définition 2.3.2 *Elimination de Fourier-Motzkin*
Elimination de Fourier-Motzkin est un algorithme mathématique pour éliminer les variables à partir d'un système d'inégalités linéaires. Si on élimine toutes les variables d'un système d'inégalités linéaires, on obtient un système d'inégalités constant . Ce système a des solutions si et seulement si le système d'origine a des solutions. En conséquence, l'élimination de toutes les variables peuvent être utilisées pour détecter si un système d'inégalités a des solutions ou non.
Cette méthode est utilisée aussi pour trouver "les boucles de parcours " du polytope, comme illustré dans l'exemple suivant :

Exemple 2.3.3 *soit le polytope défini par :* $D = \{(y, x)|0 \leq x \leq n-1, 0 \leq y \leq x-1\}$
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 :

$$\begin{array}{l} for\ (y = 0; y < n-1; y++) \\ \qquad for\ (x = y+1; x < n; x++) \\ \qquad\qquad \ldots \end{array}$$

2.3.2 Démarche

La démarche de parallélisation automatique en utilisant le modèle polyédrique [23] [13] basée sur trois étapes principales :

- Modélisation et analyse de dépendances
- Parallélisation
- Génération de code

- Modélisation et analyse de dépendances

Regroupe la représentation de nid de boucle par un système d'inéquations (qui définе un polytope) et l'analyse de dépendances entre les itérations de boucle. Les bornes du système sont définies à partir de bornes de boucles.

Il y a des méthodes pour la génération de ces dépendances automatiquement à partir de programme source[23].

- Parallélisation

Après la détection de dépendances, les algorithmes de parallélisation utilisent des techniques de transformations afin de transformer le programme séquentiel en un programme parallèle.

Les transformations de boucles sont utilisées afin d'optimiser le temps d'exécution et l'utilisation de mémoires par les programmes. En général, une transformation de boucles modifie l'ordre dont lequel les instructions dans les boucles sont exécutés.

L'utilisation combinée des techniques d'*ordonnancement* et d'*allocation* donne une description *spatio-temporelle* de la répartition des calculs des données qui se traduit par une réécriture du nid de boucles initial lors de la phase de génération de code.

- *Ordonnancement*

L'ordonnancement est une étape de parallélisation. Une fonction d'ordonnancement d'un ensemble d'opérations qui associe à chaque opérations une valeur prise dans un ensemble totalement ordonné. De plus, elle doit satisfaire les contraintes de précédence définies par les dépendances : si l'opération T dépend de l'opération S, alors S doit être exécuté avant T [17].

Les fonctions d'ordonnancement sont généralement des transformations affines.

- *Allocation(Placement).*

Une fonction d'allocation d'un ensemble d'opérations associe à chaque opération les coordonnées spatiales du processeur sur lequel elle est exécutée. Afin de ne pas réduire le parallélisme potentiel induit par l'ordonnancement, cette fonction ne doit pas allouer au même processeur deux instructions possèdent la même date[17].

Cette location peut être ensuite optimisée par le met des calculs qui accèdent au

même cellule mémoire, dans le même processeur logique[23].

- Génération de code

Les solutions trouvées dans le modèle doivent être convertir en code efficace. Le résultat de la génération de code est un programme parallèle.

Dans la section suivante, nous expliquons : la détection de dépendances et leurs présentations, la transformation de programme et nous terminons par un exemple en expliquant le processus de parallélisation basé sur le modèle polyédrique.

2.3.3 Dépendances

Une dépendance de données existe entre deux instructions S_1 et S_2, où S_1 est exécutée avant S_2, si elles accèdent à la même zone mémoire.

Types de dépendances

Il existe quatre types de dépendances [41] :

- *Dépendance de flot* : On dit qu'il y a une dépendance de flot entre S_1 et S_2, si S_1 écrit dans la cellule mémoire qui est plus tard lue par S_2 et il n'y a pas une autre instruction S_3 qui accède à cette cellule mémoire après S_1 et avant S_2. Cette dépendance est dénotée par $S_1\ \delta^f\ S_2$.

- *Anti-dépendance* : On dit qu'il y a une Anti-dépendance entre S_1 et S_2, si S_1 lit la cellule mémoire qui est plus tard écrite par S_2 et il n'y a pas une autre instruction S_3 qui accède à cette cellule mémoire après S_1 et avant S_2. Cette dépendance est dénotée par $S_1\ \delta^a\ S_2$.

- *Dépendance de sortie* : On dit qu'il y a une dépendance de sortie entre S_1 et S_2, si S_1 écrit dans la cellule mémoire qui est plus tard écrite par S_2 et il n'y a pas une autre instruction S_3 qui accède à cette cellule mémoire après S_1 et avant S_2. Cette dépendance est dénotée par $S_1\ \delta^o\ S_2$.

- *Dépendance d'entrée* : On dit qu'il y a une dépendance d'entrée entre S_1 et S_2,si S_1 lit la cellule mémoire qui est plus tard lue par S_2 et il n'y a pas une autre instruction S_3 qui accède à cette cellule mémoire après S_1 et avant S_2. Cette dépendance est dénotée par $S_1\ \delta^i S_2$.

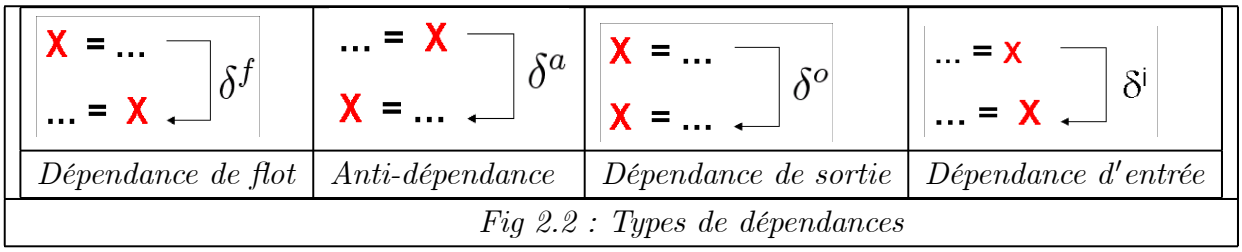


Fig 2.2 : Types de dépendances

Graphe de dépendance

Les dépendances sont présentés par plusieurs types de graphes. Parmi ces graphes, on trouve le graphe de dépendance (DDG).

- *Graphe de dépendance (DDG)*

Le graphe de dépendance (DDG) est un graphe orienté $G = (V, E)$, où chaque sommet représente un vecteur d'itération. Un arrêt $e \in E$ du nœud S_i au nœud S_j représente une dépendance [13].

Le graphe de dépendance a pour sommets les opérations et pour arcs les dépendances entre opérations.

Exemple 2.3.4 *On donne dans cet exemple un nid de boucle et son graphe de dépendance :*

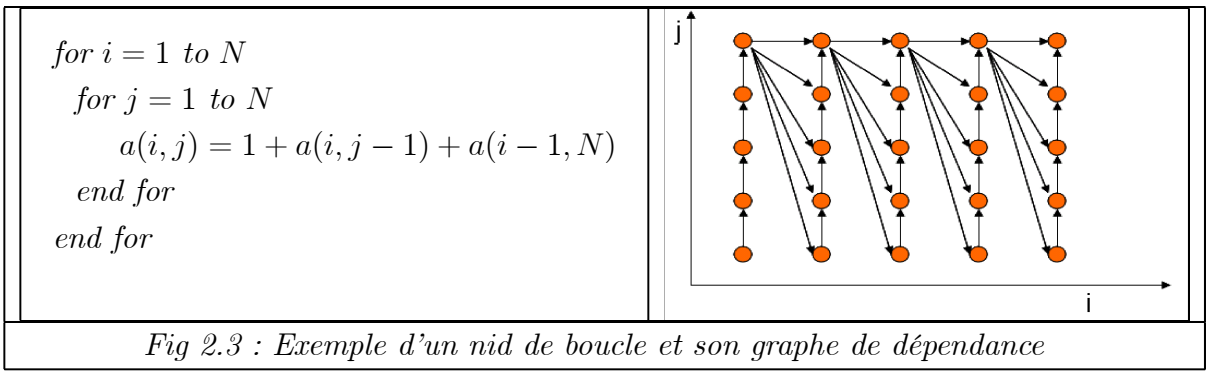


Fig 2.3 : Exemple d'un nid de boucle et son graphe de dépendance

2.3.4 Transformation de programme

Les transformations appliquées sur les nids de boucles modifient les polyèdres sources en des polyèdres cibles contenant les mêmes points mais dans un nouveau système de coordonnées, donc avec un nouvel ordre lexicographique.

Exemples de techniques de transformations

- *Eclatement de la boucle*

Une boucle avec plus d'une instruction peut être éclatée s'il n y a pas de dépendances entre ses instructions.

Exemple 2.3.5 *Il n'y a pas de dépendances entre S_1 et S_2, on peut éclater la boucle comme suit :*

for $(i = 0, i \leq n, i++)$ *S1* : $L1 = a(i) + L1;$ *S2* : $L2 = b(i) + d(i)$	*devient*	*for* $(i = 0, i \leq n, i++)$ *S1* : $L1 = a(i) + L1;$ // *for* $(i = 0, i \leq n, i++)$ *S2* : $L2 = b(i) + d(i)$

- *Fusion*

C'est la transformation inverse de l'éclatement de boucle.

Deux nids de boucle adjacents l_1, l_2 dont les boucles sont identiques peuvent être fusionnés en un seul nid de boucle, s'il n'existe pas de relations de dépendance des instructions des deux boucles.

- *Permutation (L'échange) de boucles*

La permutation de boucles modifie l'ordre de parcours de l'espace des itérations en échangeant l'ordre d'exécution de deux boucles.

Exemple 2.3.6 *Cet exemple illustre la permutation de boucle*

```
for i=                          for j=
      for j=                          for i=
        S            ==>                S
      end for                         end for
  end for                         end for
```

-*Tilling*

Tilling est une technique de transformation. Elle divise l'espace définie par structures de boucle en tiles (des petits blocs) réguliers en créant des blocs de données de tableau. Tiling a été étudiée à partir de deux perspectives [13] : optimisation de la localité de données et parallélisation

- Tilling pour la localité : exige le regroupement des points dans un espace d'itération en blocs plus petits (*tiles*), permettant la réutilisation dans de multiples directions lorsque le bloc s'adapte dans une mémoire plus rapide pourrait être registres, caches ... Lors de l'exécution produit par tile, la réutilisation des distances ne sont plus en fonction de la taille du problème, mais en fonction de la taille de tile. On peut répéter l'opération de tilling plusieurs fois, une par niveau de la hiérarchie mémoire.

- Tilling de gros grains de parallélisme : implique le partitionnement de domaine d'itération en tiles qui seront exécutés simultanément sur plusieurs processeurs avec une diminution de la fréquence et le volume de communication inter-processeur : un tile ato-

miquement exécuté sur un processeur, et la communication nécessaire seulement avant et après l'exécution.

- *Transformation affine*

Le but d'une transformation est de modifier l'ordre d'exécution originale des opérations. Une façon convenable pour exprimer le nouveau ordre est de donner à chaque opération une date d'exécution. Donner toutes les date d'exécution séparément résulte un ordonnancement très large. Alors les compilateurs utilisent des ordonnancements fondés sur des fonctions spécifiant un temps d'exécution pour chaque instance des instructions correspondantes. Ces fonctions sont généralement choisies affines, et elles ont la forme suivante $\theta_S(\overrightarrow{x}_S) = T_S\overrightarrow{x}_S + \overrightarrow{t}_S$ où $\overrightarrow{x}_S$ est le vecteur d'itération, T_S est une matrice constante dite de transformation, t_s est un vecteur constant. Il est prouvé que avec ces transformations affines, il est possible d'exprimer la majorité des transformations usuelles.

Pour mieux comprendre l'utilité de modèle polyédrique dans la parallélisation automatique, on donne l'exemple suivant :

Exemple 2.3.7 *Exemple tiré de [23]*
Soit le système suivant :

```
for (i = 0, n)
   for (j = 0, n)
      C(i+j)=C(i+j) +A(i)*B(j)
```

Dépendances

Les dépendances viennent de l'accès au tableau C. Les itérations de la boucle qui accèdent les mêmes locations mémoires sont les itérations ayant la même valeur de $i+j$ pour les différentes instances de i et j. Les itérations (i_0, j_0) qui vérifient l'équation $i_1+j_1 = i_0+j_0$ sont en dépendance. Commençant par l'itération (i_0, j_0), on calcule les itérations (i_1, j_1) qui précèdent (i_0, j_0), et qui satisfait $i_1 + j_1 = i_0 + j_0$. On prend ces contraintes en considération on obtient $(i_1, j_1) = (i_0 - 1, j_0 + 1)$.
La figure suivante illustre ces dépendances :

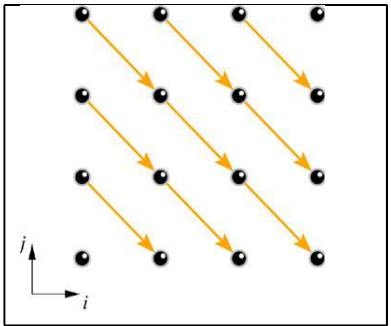


Fig 2.4 : Les dépendances

Parallélisation

On voit que les itérations pour $j = n$ peuvent être exécutées premièrement et simultanément. Une fois sont calculées, les itérations de la ligne suivante peuvent être calculées, et ainsi de suite. Alors, une fonction d'ordonnancement possible est $t(i, j) = n - j$.
Une fonction d'allocation essai de mettre les calculs dépendants dans le même processeur, alors une fonction d'allocation possible est : $p(i, j) = i + j$. La figure suivante présente le nouveau polytope :

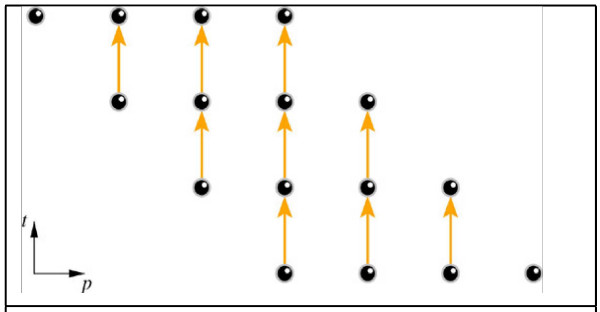


Fig 2.5 : Nouveau polytope en appliquant la fonction d'allocation

Génération de code

Pour générer le code à partir du résultat précédent, où p est la variable de la boucle extérieure et t est la variable de la boucle intérieure, on a :

$$\begin{cases} 0 \leq i \leq n \\ 0 \leq j \leq n \\ t = n - j \\ p = i + j \end{cases}$$

cela donne :

- $\begin{cases} 0 \leq i \leq n \\ 0 \leq j \leq n \\ p = i + j \end{cases} \Longrightarrow 0 \leq j + i \leq 2n \Longrightarrow \boxed{0 \leq p \leq 2n}$

- $\begin{cases} 0 \leq j \leq n \\ t = n - j \end{cases} \Longrightarrow \boxed{0 \leq t \leq n}$

- $\begin{cases} 0 \leq i \leq n \\ t = n - j \\ p = i + j \end{cases} \Longrightarrow \begin{cases} 0 \leq i \leq n \\ t = n + i - p \\ j = p - i \end{cases} \Longrightarrow n \leq n + i \leq 2n \Longrightarrow n - p \leq n + i - p \leq 2n - p$
$\Longrightarrow \boxed{n - p \leq t \leq 2n - p}$

Alors, Le code généré est :

$$
\begin{aligned}
&PAR\ for\ p = 0; 2 * n \\
&\quad for\ t = max(0, n-p); min(n, 2*n-p) \\
&\qquad C(p) = C(p) + A(t+p-n) * B(n-t)
\end{aligned}
$$

Les indices des tableaux sont calculés comme suit :
- $p = i + j \Longrightarrow C(i+j) = C(p)$
- $t = n - j \Longrightarrow j = n - t \Longrightarrow B(j) = B(n-t)$
-$p = i + j$, $j = n - t \Longrightarrow i = t + p - n \Longrightarrow A(i) = A(t+p-n)$

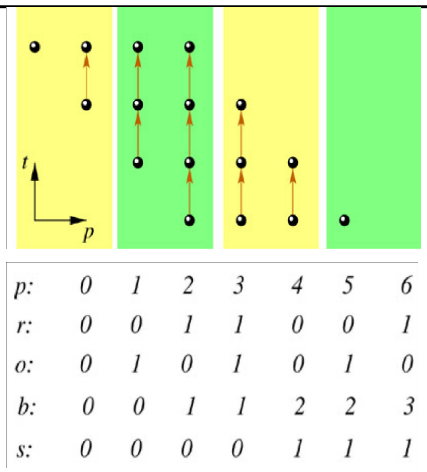
Extension : tilling

Le code résulte implique l'utilisation de $2n+1$ *processus. Le tilling permet de regrouper le travail sur un nombre plus petit de processeurs, en basculant le travail d'un nombre de processeurs virtuels parmi les* $2n+1$ *processeurs virtuels vers un seul processeurs physique*
On présente un processeur virtuel par le triple (r, b, o) *, tel que :*
- r *est le numéro de processeur réel (physique)*
- b *est le numéro de bloc*
- o *l'offset dans le bloc, où*

- $0 \leq r \leq NP - 1$	NP *est le nombre de processeurs réels*
- $0 \leq o \leq B - 1$	B *la taille de bloc*
- $p = b.B + o$	
- $b = s.NP + r$	*cette fonction spécifie la distribution cyclique de blocs vers les processeurs*

Pour la génération de nid de boucles, on utilise une version d'élimination de Fourier Motzkin.
On met $NP = 2$*, le résultat est dans la figure suivante, sachant que :*
$floor(x) = \lfloor x \rfloor = max\{n \in \mathbb{Z} | n \leq x\}, ceiling(x) = \lceil x \rceil = min\{n \in \mathbb{Z} | n \geq x\}$.

```
Do PAR r=0, NP-1
 Do t=0,n
  Do  s=ceiling(-t/NP*B)-r/NP+(n-B+1)/(NB*B),
        floor(-t/NP*B)-r/NP+(2*n)/(NP*B)
    Do o=max(0,-t-r*B-s*NP*B+n),
          min(B-1,-t-r*B-s*NP*B+2*n)
      Do p=max(n-t,r*B+s*NP*B+o),
            min(2*n-t,r*B+s*NP*B+o)
        C(p)=C(p)+A(t+p-n)*B(n-t)
```



p:	0	1	2	3	4	5	6
r:	0	0	1	1	0	0	1
o:	0	1	0	1	0	1	0
b:	0	0	1	1	2	2	3
s:	0	0	0	0	1	1	1

Fig 2.6 : Programme parallèle en utilisant le tilling

2.4 Optimisation de la mémoire et le calcul parallèle

Les techniques de parallélisation permettent l'exploitation des capacités des machines puissantes afin d'améliorer l'exécution des programmes. La distribution de la mémoire sur les processeurs et le coût élevé des communications entre ces processeurs posent un problème de placement des données et de minimisation des communications. La performance d'un programme est très dépendante de ses accès en mémoire. Les mémoires caches, d'accès beaucoup plus rapide que la mémoire principale est la solution matérielle apportée à ce problème. Or sa taille est limitée, les données sont chargées par bloc. L'absence de donnée dans la cache cause un chargement d'un bloc mémoire dans la cache. Un ordinateur verra ses performances pleinement exploitées s'il n'accède qu'à sa mémoire cache, car un défaut de cache est coûteux en termes de cycles processeur.

La minimisation des défauts de cache permet d'avoir des programmes performants. Pour cette minimisation, en essayant de regrouper les réutilisations d'une même donnée dans le temps. Il existe deux types de réutilisation de données : [17]

- *la réutilisation temporelle* : il s'agit de la réutilisation d'une seule et même donnée durant l'exécution.

- *la réutilisation spatiale* : elle concerne la réutilisation de même bloc de données chargé dans la mémoire cache .

De ces deux types de réutilisations, on déduit deux objectifs d'optimisation des performances : l'optimisation de la localité temporelle et l'optimisation de la localité spatiale.

2.5 Utilisation de nombre de points entiers dans un polytope dans la parallélisation automatique

Les techniques de parallélisation, essaient d'exploiter les capacités des machines puissantes afin de donner un bon résultat d'exécution des programmes. Afin de développer des programmes parallèles qui répondent aux besoins des utilisateteurs, les développeurs recourent aux techniques d'analyse des nids de boucles affines qui peuvent répondre à un ensemble de questions tels que :

– Quel est le nombre de localisations mémoire touchées par une boucle ?

– Quel est le nombre d'éléments d'un tableau accédés entre deux instants ?

– Quel est le nombre d'éléments d'un tableau qui sont en vie (qui ont déjà été utilisées et qui seront encore utilisés) à une itération donnée ?

– Quel est le nombre de défauts de cache générés par une boucle ?

– ...

La réponse à ces différentes questions et autres, est formalisée par le nombre de solutions entières de systèmes d'inégalités linéaires et de leurs transformations par des fonctions affines entières. Le nombre de ces solutions est dual au nombre de points entiers de polytopes, éventuellement paramétrés, ou de leurs transformations.[33]

L'exemple suivant explique une des utilisations de nombre de points entiers dans un polytope :

Exemple 2.5.1 *Soit le nid de boucle suivant*

pour i *de* 1 *à* N *faire*
  pour j *de* 1 *à* N *faire*
    pour k *de* 1 *à* N *faire*
      $c[i,j] := c[i,j] + a[i,k] * b[k,j]$
    fin pour
  fin pour
fin pour

Les itérations de ce nid de boucle sont présenté par le système d'inéquations suivant :

$$1 \leq i \leq N$$
$$1 \leq j \leq N$$
$$1 \leq k \leq N$$

Le nombre des opérations exécutées par le programme est donné par le nombre de points

entiers contenu dans le polytope présenté par le système d'inéquations précédent. Il est égale dans ce cas à N^3.
Cette information est utilisée dans plusieurs cas, par exemple pour évaluer le temps d'exécution du programme, ou pour réaliser l'équilibrage entre les processeurs (en les allouant de nombres égales des instructions)

2.6 Conclusion

Le modèle polyédrique est un outil mathématique fort dans la parallélisation.

Dans ce chapitre, nous avons présenté une simple introduction aux notions de parallélisme en utilisant le modèle polyédrique, afin de pouvoir localiser ce modèle dans le contexte de parallélisation automatique.

Chapitre 3

Méthodes de dénombrement de points entiers dans un polytope et exemples de l'utilité de ce nombre (État de l'art)

3.1 Introduction

Le dénombrement des solutions entières n'a pas de sens que si ce nombre est fini c-à-d dans le cas des polytopes.

Dans ce chapitre, on va utiliser les notions de base données dans le premier chapitre, pour décrire des méthodes de dénombrement des points entiers en mettant l'accent sur la méthode d'interpolation car notre travail est basé sur cette méthode. Puis on donne des exemples d'utilisations de nombre de points entiers dans la parallélisation automatique.

3.2 Méthodes de dénombrement des points dans un réseau polytope

Dans cette section, nous expliquons des méthodes de dénombrement de points entiers dans un polytope.

3.2.1 Méthode des sommes

La méthode des sommes [33] basée sur les formules standards de sommes de puissances de nombres entiers, peut être utilisée pour calculer le nombre de points entiers dans des

polytopes issus de l'analyse de nids de boucles.

La formule générale utilisée est donnée par :

$$\left(\sum i_1, ..., i_d : l_1(p) \leq i_1 \leq u_1(p), ..., l_d(i_1, ..., i_{d-1}, p) \leq i_d \leq u_d(i_1, ..., i_{d-1}, p) : f(i_1, ..., i_d, p)\right)$$

Cela signifie la somme (sur les i_j) de $f(i_1, ..., i_d, p)$ avec les contraintes $l_j(p) \leq i_j \leq u_j(p)$, où l_j, u_j sont des fonctions affines et $f(i_1, ..., i_d, p)$ est une fonction polynomiale des variables et des paramètres p. Par exemple :

$$\left(\sum i : 1 \leq i \leq n : i^3\right) = \left(1 \leq n : \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2\right)$$

Où ($\sum i : 1 \leq i \leq n : i^3$) signifie la somme (sur i) de i^3, avec $1 \leq i \leq n$;

Exemple 3.2.1 *Nous donnons ici deux exemples des sommes standards :*

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Les étapes principales de la méthode des sommes sont :

- On procède à un ensemble de simplifications et de règles de réécriture pour se ramener aux fonctions de sommes de base.

- A chaque étape, la somme est calculée par rapport à une variable en considérant le reste des variables comme des constantes.

- En remplaçant le terme $f(i_1, \cdots, i_d, p)$ par 1 dans la formule ci-dessus, nous obtenons la formule de sommes correspondant au nombre de points entiers dans le polytope défini par les bornes inférieures et supérieures sur les variables $i_1, \cdots, i_d$.

Des expressions polynomiales apparaissent dans le terme $f(i_1, \cdots, i_d, p)$ au fur et à mesure que les variables sont éliminées.

L'exemple suivant permet de comprendre mieux cette méthode.

Exemple 3.2.2 *On considère le nid de boucle suivant*

$$\begin{array}{l} for\ (i = 1;\quad i <= n;\quad i++) \\ \quad for\ (j = 1;\quad j <= i;\quad j++) \\ \quad\quad for(k = j;\quad k <= m;\quad k++) \\ \quad\quad\quad S : ... \end{array}$$

Le nombre de fois que l'instruction S est exécutée est donné par le nombre de points entiers dans un polytope

$$P = \left\{(i,j,k) \in \mathbb{Q}^3 \mid 1 \leq j \leq i \leq n \ \wedge \ j \leq k \leq m\right\}$$

La formule de sommes correspondante à ce polytope est

$$\left(\sum i,j,k : 1 \leq j \leq i \leq n \ \wedge j \leq k \leq m : 1\right)$$

- On faisant la somme sur k, nous obtenons

$$\left(\sum i,j : 1 \leq j \leq i \leq n \ \wedge j \leq m : m-j+1\right)$$

- Puis sur i, on trouve

$$\left(\sum j : 1 \leq j \leq n, m : (n-j+1)\,(m-j+1)\right)$$

- A ce niveau, il faut séparer les deux bornes supérieures de j, ce qui donne

$$\left(\sum j : 1 \leq j \leq n \leq m : (n-j+1)\,(m-j+1)\right) + \left(\sum j : 1 \leq j \leq m < n : (n-j+1)\,(m-j+1)\right)$$

- Puis on fait la somme sur j, on obtient le résultat suivant

$$\left(1 \leq n \leq m : \frac{mn^2}{2} - \frac{n^3}{6} + \frac{mn}{2} + \frac{n}{6}\right) + \left(1 \leq m < n : \frac{m^2n}{2} - \frac{m^3}{6} + \frac{mn}{2} + \frac{m}{6}\right)$$

Implémentation

Dans les cas simples, la méthode des sommes donne des résultats exacts et intéressants. Cependant elle reste difficile à automatiser dans le cas général.

3.2.2 Méthode d'interpolation de Clauss et Loechner

Dans cette méthode, le nombre de points entiers dans un polytope issu d'analyses de boucle est calculé par le polynôme d'Ehrhart. La méthode est basée sur le calcul des sommets de polytope paramétré, car le nombre maximum de valeurs prises par les coefficients périodiques de polynôme d'Ehrhart est donné par le dénominateur du polytope, qui est le plus petit commun multiple des dénominateurs des coordonnées de sommets du polytope.

Or, les sommets ne sont pas tous valides pour chaque valeur de paramètres, on doit calculer leurs domaine de validité.

Alors, on doit calculer les sommets et leurs domaines de validité, et pour chaque domaine (qui est un polytope aussi) on calcule un polynôme d'Ehrhart.

Les différentes étapes de la méthode d'interpolation sont :

- Lecture de système d'inégalités paramétriques (correspond au polytope $P(p)$ où p est le vecteur des paramètres, $p = p_1, p_2, ..., p_m$).

- Calcul des coordonnées paramétriques des sommets (*Algorithme Wilde et Loechner*) :

♦ convertir la présentation homogène de domaine paramétré écrit sous la forme implicite à la forme de Minkowiski (on la note P').

♦ calculer les $m - faces$, où m est le nombre des paramètres, en utilisant l'algorithme de calcul de $k - face$ (de *Wilde et Loechner).*

♦ pour chaque sommet paramétrique $v_i(p)$ dans $P(p)$ il y a une $m - face$ correspondante dans P'. Ses coordonnées correspondent à une projection de cette $m - face$.

- Partitionner le domaine du polytope en trouvant la liste des sommets qui sont valides sur chaque domaine de polytope. (*Algorithme Clauss et Loechner*). Les coordonnées des sommets sur un domaine donné sont des combinaisons affines des paramètres.

- Pour chaque domaine de validité, calculer du polynôme d'Ehrhart correspondant :

♦ Le degré de polynôme d'Ehrhart est la dimension de polytope ;

♦ Le calcul de ces polynômes sur chaque domaine de validité se fait en fonction des dénominateurs des sommets du polytope sur le domaine considéré et d'un certain nombre d'énumérations initiales de polytopes non paramétrés, obtenus par substitution de paramètres en utilisant la méthode de *Fourier-Motzkin* pour obtenir les boucles de parcours. Si $P(p)$ est entier alors, le polynôme d'Ehrhart est un polynôme à plusieurs variables (les m paramètres) $p_1, p_2, ..., p_m$ de degré n (n est la dimension de l'espace contenant le polytope). Et si $P(p)$ est rationnel alors, le polynôme d'Ehrhart est un pseudo polynôme à plusieurs variables $p_1, p_2, ..., p_m$ de degré n . Ce pseudo polynôme est

de la forme :

$$PE(p_1, p_2, ..., p_m) = \sum_{i_1=0}^{n} \sum_{i_2=0}^{n-i_1} ... \sum_{i_p=0}^{n-i_1-i_2-...-i_{p-1}} c_{i_1,i_2,...i_m} p_1^{i_1} p_2^{i_2} ... p_m^{i_m}$$

La figure suivante résume les étapes principales de la méthode d'interpolation de Clauss et Loechner :

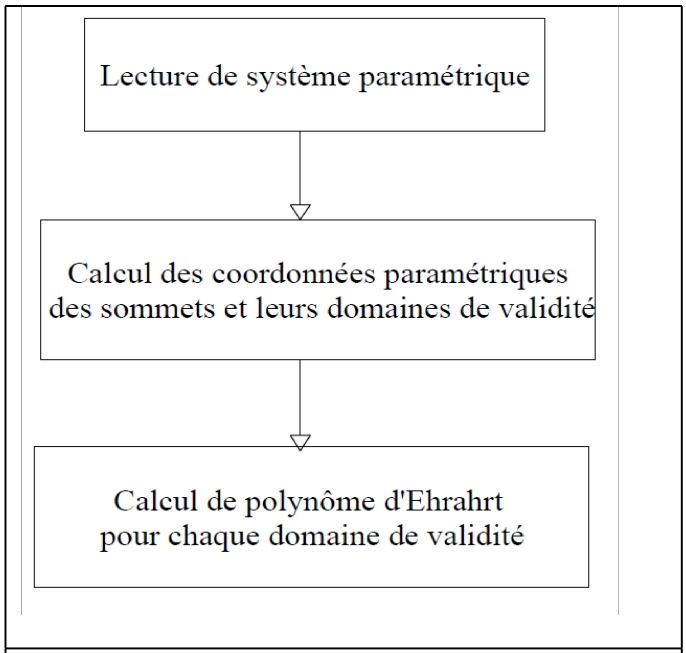


Fig.3.1 : Résumé de la méthode d'interpolation de Claus

Dans ce qui suit, nous représentons les différents algorithmes utilisés dans cette méthode.

Algorithmes utilisés dans la méthode d'interpolation de Clauss et Loechner

a) **Algorithme de calcul des coordonnées paramétriques des sommets**

Le détail de cet algorithme est dans [24]. L'idée est d'augmenter l'espace de données en espace données-paramètres (on peut toujours trouver le polyèdre initial par une projection dan l'espace de données). Les sommets de polyèdres initiaux (domaine de données) sont des $m - faces$ dans le nouveau polyèdre (domaine données-paramètres). Si la projection d'une $m - face$ dans l'espace de paramètres n'est pas de dimension m, c'est que cette $m - face$ ne correspond pas à un sommet paramétré de $P(p)$.

On résume les étapes dans le tableau suivant. Noter que les différentes présentations illustrées ici sont pour des polyèdres, et sont applicables pour les polytopes (car un poly-

tope est un polyèdre borné)

Présentation implicite On démarre par une présentation implicite paramétrique	$P(p) = \left\{ \begin{array}{c} x \in Q^n \mid Ax = Bp + b \wedge \\ Cx \geq Dp + d \end{array} \right\}$
Présentation non paramétrique : La présentation sous forme implicite paramétrique est écrite sous forme non paramétrique	$P' = \left\{ \begin{pmatrix} x \\ p \end{pmatrix} \in \mathbb{Q}^{n+m} \mid \hat{A}x = b, \hat{C}x \geq d \right\}$ $\hat{A} = [A \quad \mid \quad -B]$ $\hat{C} = [C \quad \mid \quad -D]$
Présentation homogène (non paramétrique) : La présentation non paramétrée est ensuite transformée sous la présentation homogène	$P' = \left\{ \begin{array}{c} \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} \in \mathbb{Q}^{n+m+1} \mid \\ \hat{A} \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} = 0, \\ \hat{C} \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} \geq 0 \end{array} \right\}$ $\hat{A} = [A \quad \mid \quad -B \quad \mid \quad -b]$ $\hat{C} = \begin{bmatrix} C & \mid & -D & \mid & -d \\ 0..0 & \mid & 0..0 & \mid & 1 \end{bmatrix}$
Présentation de Minkowski : La présentation de Minkowski sous la forme homogène est ensuite calculée.	$P' = \left\{ \begin{array}{c} \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} \in \mathbb{Q}^{n+m+1} \mid \\ \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} = \hat{L}\ \lambda + \hat{R}\ \mu,\ \mu \geq 0 \end{array} \right\}$ $\hat{L} = \begin{bmatrix} L \\ 0..0 \end{bmatrix}$ $\hat{R} = \begin{bmatrix} R & \mid & V \\ 0..0 & \mid & 1..1 \end{bmatrix}$ Où L : la matrice des vecteurs qui génèrent l'espace linéaire R : les rayons extrêmes V : les sommets dans la représentation de Minkowski de la présentation implicite.

Trouver les m-faces - Appliquer l'algorithme $m-faces$. - Chaque $m-faces$ F_i^m de P' est présentée par un ensemble de lignes, rayons et sommets, ou dans l'espace homogène par un ensemble de lignes et rayons	Chaque $m-face$ est présentée comme suit : $\left\{ \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} \in \mathbb{Q}^{n+m+1} \mid \begin{pmatrix} \xi x \\ \xi p \\ \xi \end{pmatrix} = \hat{L}\ \lambda + \hat{R}_i\ \mu, \quad \mu \geq 0 \right\}$ Où chaque colonne de $\hat{L}$ est un rayon bidirectionnel et chaque Colonne de $\hat{R}_i$ est un rayon unidirectionnel. Les colonnes de $\hat{R}_i$ sont un sous ensemble de $\hat{R}$
Trouver les sommets Soit la matrice $M_i = [\hat{L} \mid \hat{R}_i]$. Pour chaque $m-face$ F_i^m on applique ces deux derniers étapes : *a*) - Si la matrice $Proj_p(M_i)$ n'a pas d'inverse droite, alors cette $m-face$ ne correspond pas à un sommet de $P(p)$. Passer à la $m-face$ suivante et refaire *a*) - Calculer $T_i = Proje_d(M_i)Proj_p(M_i)^{-R}$ *b*) - Calculer le domaine de validité $Projp(F_i^m)$. - Passer à la $m-face$ suivante et revenir à *a*)	La projection de $m-face$ définie par $\hat{L}$ et $\hat{R}_i$ dans l'espace de paramètre $\mathbb{Q}^m : (T_i\ , Proj_p(F_i^m))$ est un sommet de $P(p)$.

Remarque 3.2.3 *Le passage entre la présentation implicite et la représentation de Mikowiski fait en utilisant un algorithme décrit dans [42].*

b) Algorithme de calcul de k-face

Selon [24], cet algorithme basé sur les propriétés suivantes :

- Chaque $k - face$ contient au moins $k + 1$ sommets, rayons et lignes qui doivent inclure tous les lignes et au moins un sommet,

- Chaque sommet, rayon, et ligne dans un $k - face$ sature au moins $d - k$ équations et inéquations, qui doivent inclure tous les équations.

Dans ce qui suit, on donne le principe de l'algorithme. Son détail est dans [24]

Algorithme

Soit P un polytope de dimension d.

1. Pour chaque combinaison de $(d-k)$ contraintes, on compte le nombre de sommets, rayons et lignes saturés par tous ces contraintes sélectionnés. Si ce nombre est égale ou grand que $k+1$ et l'ensemble contient au moins un sommet, alors l'ensemble de sommets, rayons et lignes saturés constitue un $k - face$ de polyèdre.

2. Élimination de $k - faces$ redandants :

Si un autre contrainte qui est déjà considérée sature le même ensemble de sommets, rayons et lignes, alors cette face est déjà signalée. Si non cette face est nouveau et doit être signalée.

3. Continuer la combinaison suivante des contraintes.

Le détail de cet algorithme est dans [24].

c) Algorithme de partitionnement de domaine des paramètres (Clauss / Loechner)

Le détail de l'algorithme est dans [18]

L'algorithme décrit ci après calcule un ensemble $D = \{d_j \mid 1 \leq j \leq v\}$ v est le nombre de domaines de validité.

Cet ensemble de domaines de validités est obtenu par partitionnement de l'espace de paramètres à partir de la projection de $m - faces$ de P'.

L'algorithme est :

Input : Ensemble des faces de dimension m : F^m

Output : Ensemble des domaines de validité D

Begin

$D = \{\}$

for all $F_i^m \in F^p$

$f = \prod(F_i^m)$ // *projection de* F_i^m *dans l'espace de paramètres*

$NewD = \{\}$

for $j = 1$ *to* $Card(D)$ *do*

if *Dimension* $(d_j \cap f) = p$ *then*

$NewD = NewD \cup \{dj \cap f\} \cup \{dj - f\}$

$f = f - d_j$

else

$NewD = NewD \cup \{dj\}$

endif

endfor

if $f \neq \phi$ *then*

$D = NewD \cup \{f\}$

endif

$D = NewD$

endfor

end

Nous donnons ici un exemple de calcul de polynôme d'Ehrhart par la méthode d'interpolation.

Exemple 3.2.4 *Soit le polytope paramétré suivant*

$$P(p) = \left\{(x_1, x_2) \in \mathbf{Q}^2 | 0 \leq x_2 \leq 4 \ \wedge x_2 \leq x_1 \leq x_2 + 9 \ \wedge x_1 \leq p \wedge p \leq 40\right\}$$

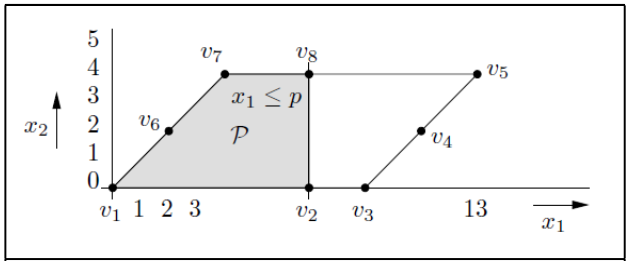


Fig.3.2 :Polytope paramétré P(p)

- On cherche les $m-faces$ (ici $1-faces$ car on a un seul paramètre p) (appliquant l'algorithme pour la recherche de $m-faces$) :

La figure suivante donne le polytope dans l'espace combiné data-parameter

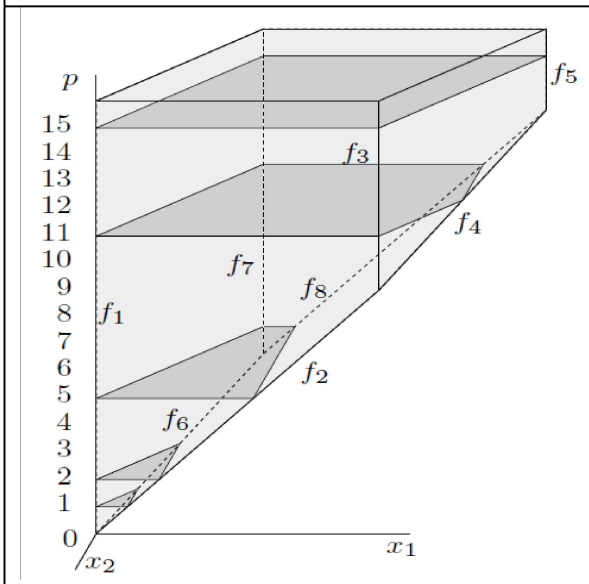


Fig 3.3 : Polytope dans l'espace combiné data-parameter

Tant que la dimension de l'espace de paramètre est 1, *les* $1-faces$ *(qui sont les arrêts) de* P' *sont projetés dans* $\mathbb{Q}^1$.
Tant que la projection de quatre $1-faces$ *en haut de la figure dans* $\mathbb{Q}^1$ *sont des points, alors de dimension* 0,
ne correspondent pas aux sommets paramétriques de $P(p)$.
Les huits $1-faces$ *considérées sont étiquettes* $f_1, f_2, ..., f_8$.

- Par projection des $1-faces$, *on trouve les sommets et leurs contexte selon la valeur de paramètre* p *sont :*

$$
\begin{aligned}
v_1(p) &= (0,0) \quad si \quad 0 \leq p \leq 400 \\
v_2(p) &= (p,0) \quad si \quad 0 \leq p \leq 9 \\
v_3(p) &= (9,0) \quad si \quad 9 \leq p \leq 40 \\
v_4(p) &= (p,p-9) \quad si \quad 9 \leq p \leq 13 \\
v_5(p) &= (13,4) \quad si \quad 13 \leq p \leq 40 \\
v_6(p) &= (p,p) \quad si \quad 0 \leq p \leq 4 \\
v_7(p) &= (4,4) \quad si \quad 4 \leq p \leq 40 \\
v_8(p) &= (p,4) \quad si \quad 4 \leq p \leq 13
\end{aligned}
$$

- Appliquant l'algorithme de calcul de domaines de validité, on trouve les domaines de validités des sommets paramétriques suivants :

$$\begin{aligned} 0 &\leq p \leq 4 : V_1(p) = \{v_1, v_2, v_6\} = \{(0,0), (p,0), (p,p)\} \\ 4 &\leq p \leq 9 : V_{\acute{e}}(p) = \{v_1, v_2, v_7, v_8\} = \{(0,0), (p,0), (4,4), (p,4)\} \\ 9 &\leq p \leq 10 : V_3(p) = \{v_1, v_3, v_4, v_7, v_8\} = \{(0,0), (9,0), (p,p-9), (4,4), (p,4)\} \\ 13 &\leq p \leq 40 : V_4(p) = \{v_1, v_3, v_5, v_7\} = \{(0,0), (9,0), (13,4), (4,4)\} \end{aligned}$$

- Maintenant, on calcule les polynômes d'Ehrhart pour chaque domaine de validité, on trouve :

• $\boldsymbol{0 \leq p \leq 4}$

Les sommets sont

$$V_1(p) = \{(0,0), (p,0), (p,p)\}$$

(Toutes les valeurs $v \in V_1(p)$ sont des entiers) $\Longrightarrow$ *(toutes les pseudo périodes du polynôme d'Ehrhart sont* 1*).*

La dimension de l'espace affine contenant le polytope est 2*, alors la forme générale de polynôme d'Ehrhart est*

$$PE_1(p) = c_2 p^2 + c_1 p + c_0$$

On cherche les c_i en construisant un système d'équations. On donne des valeurs initiales pour p, puis on calcule le nombre de points dans le polytope pour cette valeur de p.

↪ *pour $p = 0$. Le système est* $P(0) = \begin{cases} (x_1, x_2) \in \mathbf{Q}^2 | 0 \leq x_2 \leq 4 \wedge \\ x_2 \leq x_1 \leq x_2 + 9 \wedge \\ x_1 \leq 0 \end{cases}$

Le seul point entier dans le polytope est $(0,0)$*. Alors le nombre de points entiers*

$PE_1(0) = 1$

↪ *De la même manière : pour $p = 1$ on trouve $E_1(1) = 3$.*

↪ *Et pour $p = 2$ on trouve $E_1(2) = 6$.*

Alors, nous construisons le système suivant :

$$\begin{cases} c_0 = 1 \\ c_2 + c_1 + c_0 = 3 \\ 4c_2 + 2c_1 + c_0 = 6 \end{cases} \Leftrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}$$

On résoudre ce système, on trouve $(c_0, c_1, c_2) = (1, \frac{1}{2}, \frac{1}{2}) \Rightarrow \boxed{PE_1(p) = \frac{1}{2}p^2 + \frac{1}{2}p + 1}$

• $\boldsymbol{4 \leq p \leq 9}$

Les sommets sont $V_2(p) = \{(0,0), (p,0), (4,4), (p,4)\}$.

Nous suivons la même démarche que celle utilisée pour $0 \leq p \leq 4$, nous trouvons $(c_0, c_1, c_2) = (-5, 5, 0) \Rightarrow$ $\boxed{PE_2(p) = 5p^2 - 5}$

• ***$9 \leq p \leq 13$***

Les sommets sont $V_3(p) = \{(0,0), (9,0), (p, p-9), (4,4), (p,4)\}$.

On trouve $(c_0, c_1, c_2) = (-41, \frac{13}{2}, -\frac{1}{2}) \Rightarrow$ $\boxed{PE_3(p) = -\frac{1}{2}p^2 + \frac{13}{2}p - 41}$

• ***$13 \leq p \leq 40$***

Les sommets sont $V_4(p) = \{(0,0), (9,0), (13,4), (4,4)\}$.

On trouve $(c_0, c_1, c_2) = (50, 0, 0) \Rightarrow$ $\boxed{PE_4(p) = 50}$.

Remarque 3.2.5 *Tous les sommets sont des entiers, c'est pour ça tous les coefficients de polynôme d'Ehrhart $EP_i(p)$ sont des scalaires. L'exemple suivant donne un polynôme d'Ehrhart où les coefficients sont des nombres périodiques.*

Exemple 3.2.6 *Soit S_n le système linéaire suivant :*

$$(S_n) \begin{cases} x \geq 0 \\ x \leq \frac{1}{2}n \\ y \geq 0 \\ y \leq \frac{1}{2}n \\ n \geq 0 \end{cases}$$

Ce système définit un polytope rationnel paramétrique P_n et plus précisément, un carré dont les sommet ont pour coordonnées $(0,0)$, $\left(\frac{1}{2}n, 0\right)$, $\left(0, \frac{1}{2}n\right)$ *et* $\left(\frac{1}{2}n, \frac{1}{2}n\right)$.

Le dénominateur de P_n est 2, *la dimension de l'espace le contenant est* 2, *et on est sûr que S_n possède au moins une solution entière. Par conséquent, le pseudo-polynôme d'Ehrhart de P_n a donc la forme générale suivante :*

$$PE(n) = c_1 n^2 + [c_2, c_3]\, n + [c_4, c_5]$$

Afin de déterminer les coefficients c_i, 5 *valeurs numériques de $pe(n)$, $n \in [0,4]$, permettent de poser un système d'égalités linéaires dont les solutions sont leurs valeurs numériques :*

$$\left.\begin{array}{r} PE(0) = c_5 = 1 \\ PE(1) = c_1 + c_2 + c_4 = 1 \\ PE(2) = 4c_1 + 2c_3 + c_5 = 4 \\ PE(3) = 9c_1 + 3c_2 + c_4 = 4 \\ PE(4) = 16c_1 + 4c_3 + c_5 = 9 \end{array}\right\} \Longrightarrow PE(n) = \tfrac{1}{4}n^2 + \left[\tfrac{1}{2}, 1\right] n + \left[\tfrac{1}{4}, 1\right].$$

Implémentation

La méthode d'interpolation est implémentée dans une bibliothèque en C, c'est la bibliothèque *Polylib.* La bibliothèque polyédrique *Polylib*, est une bibliothèque de calcul sur les polyèdres convexes. Elle a été développée initialement par Hervé Le Verge et Doran Wilde à l'Inria Rennes.

3.2.3 Méthode des fractions partielles

Dans cette méthode[11], on s'intéresse à calculer le nombre de points entiers $\phi_A(b)$ de système suivant :

$$\begin{aligned} &x \in \mathbb{R}^d_{\geq 0} \\ &Ax = b \end{aligned}$$

où A est une $(m \times d)-$matrice d'entiers non négatifs et $b \in \mathbb{Z}^m$.

On suppose que A est constante et on étudie $\phi_A(b)$ en fonction de b, où :

- $\phi_A(b)$ nommée "*fonction de partition vectorielle* (vector partition function)"

- $\phi_A(b) = \#\{x : Ax = b, x \geq 0,\ x\ est\ entier\}$(# : est le nombre des éléments d'un ensemble).

- Les colonne de A nommés a_i.

Lemme 3.2.7 *Lemme d'Euler*
La fonction de partition vectorielle $\phi_A(b)$ est égale au coefficient de $z^b = z_1^{b_1}...z_m^{b_m}$ de la fonction $f(z) = \dfrac{1}{(1-z^{a_1})...(1-z^{a_d})}$ développée en séries entières centrées à $z = 0$, où $a_1,\cdots,a_d$ sont les vecteurs colonnes de la matrice A [11].

- Le coefficient de z^b dans $f(z)$ égale au terme constant de $\dfrac{f(z)}{z^b}$ (coefficient de z^0) noté par $const\dfrac{f(z)}{z^b}$.

- On exprime $\dfrac{f(z)}{z^b}$ en fonctions partielles pour compter son terme constant et donc $\phi_A(b)$

- Alors, d'après ce lemme d'Euler, on a $\phi_A(b) = const\dfrac{1}{(1-z^{a_1})...(1-z^{a_d})z^b}$

- Pour calculer cette constante, on procède au développement de la fonction ci-dessus en fractions partielles. Ce qui permet d'éliminer, à chaque étape, une variable z_i. Le terme constant est calculé une fois que la fonction n'est définie qu'en fonction d'une seule variable z_n.

Prenant l'exemple suivant pour mieux comprendre :

Exemple 3.2.8 *On cherche le nombre de solutions entières du système*

$$\begin{cases} x_1 + 2x_2 + x_3 = a \\ \\ x_1 + x_2 + x_4 = b \end{cases}, \ x_1, x_2, x_3, x_4 \geq 0$$

Pour trouver ce nombre, on cherche la fonction de partition vectorielle $\phi_A(b)$*, avec*

$$A = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

et

$$b = (a, b)$$

$$\phi_A(a,b) = const \frac{1}{(1 - z_1 z_2)...(1 - z_1^2 z_2)(1 - z_1)(1 - z_2) z_1^a z_2^b}$$

- On développe premièrement en fonctions partielles en z_2

$$\phi_A(a,b) = const_{z1} \left(\frac{1}{(1 - z_1) z_1^a} const_{z_2} \left(\frac{1}{(1 - z_1 z_2)(1 - z_1^2 z_2)(1 - z_2) z_2^b} \right) \right)$$

On a

$$\begin{aligned} \frac{1}{(1 - z_1 z_2)(1 - z_1^2 z_2)(1 - z_2) z_2^b} &= -\frac{\frac{z_1^{b+1}}{(1 - z_1)^2}}{1 - z_1 z_2} + \frac{\frac{z_1^{2b+3}}{(1 - z_1)(1 - z_1^2)}}{1 - z_1^2 z_2} + \frac{\frac{1}{(1 - z_1)(1 - z_1^2)}}{1 - z_2} \\ &\quad + \sum_{k=1}^{b} \frac{...}{z_2^k} \end{aligned}$$

Si on prend le terme constant donne

$$\begin{aligned}\phi_A(a,b) &= const\left(\frac{1}{(1-z_1)\,z^a}\left(-\frac{z_1^{b+1}}{(1-z_1)^2}+\frac{z_1^{2b+3}}{(1-z_1)(1-z_1^2)}+\frac{1}{(1-z_1)(1-z_1^2)}\right)\right)\\ &= const\left(-\frac{z_1^{b-a+1}}{(1-z_1)^3}+\frac{z_1^{2b-a+3}}{(1-z_1)^2(1-z_1^2)}+\frac{1}{(1-z_1)^2(1-z_1^2)z_1^a}\right)\end{aligned}$$

On calcule chaque terme séparément :
- Si $b-a+1>0$ (équivalent à $b\geq a$), on a :

$$const\left(\frac{z_1^{b-a+1}}{(1-z)^3}\right)=0$$

- Si $b<a$, on a

$$const\left(\frac{1}{(1-z_1)^3 z_1^{a-b-1}}\right) = \frac{(a-b)^2}{2}+\frac{a-b}{2} \qquad \left(on\ a\ \frac{1}{(1-z_1)^3}=\sum_{k\geq 0}C_{k+2}^2\ z_1^k\right)$$

Alors pour le premier terme, on a

$$const\left(\frac{1}{(1-z_1)^3 z_1^{a-b-1}}\right)=\begin{cases}0 & \text{Si } b\geq a\\ \dfrac{(a-b)^2}{2}+\dfrac{a-b}{2} & \text{Si } b\leq a+1\end{cases}$$

Pour le deuxième terme, on a

$$Si\ 2b-a+2\geq 0 : const\left(\frac{z_1^{2b-a+3}}{(1-z_1)^2(1-z_1^2)}\right)=0$$

$$\begin{aligned}Si\ a &\geq 2b+3 : const\left(\frac{z_1^{2b-a+3}}{(1-z_1)^2(1-z_1^2)}\right)\\ &= const\left(\frac{1/2}{(1-z_1)^3}+\frac{\frac{a-2b-3}{2}+\frac{1}{4}}{(1-z_1)^2}+\frac{\frac{(a-2b-3)^2}{4}+\frac{a-2b-3}{2}+\frac{1}{8}}{1-z_1}+\frac{(-1)^{a+1}/8}{1+z_1}\right)\\ &= \frac{(a-2b)^2}{4}+\frac{2b-a}{2}+\frac{1+(-1)^{a+1}}{8}\end{aligned}$$

Alors pour le deuxième terme on a

$$const\left(\frac{z_1^{2b-a+3}}{(1-z_1)^2(1-z_1^2)}\right) = \begin{cases} 0 & \text{Si } a \leq 2b+2 \\ \frac{(a-2b)^2}{4} + \frac{2b-a}{2} + \frac{1+(-1)^{a+1}}{8} & \text{Si } a \geq 2b \end{cases}$$

pour le troisième terme on trouve :

$$\begin{aligned} const\left(\frac{1}{(1-z_1)^2(1-z_1^2)z_1^a}\right) &= const\left(\frac{1/2}{(1-z_1)^3} + \frac{\frac{a}{2}+\frac{1}{4}}{(1-z_1)^2} + \frac{\frac{a^2}{4}+\frac{a}{2}+\frac{1}{8}}{1-z_1} + \frac{(-1)^a/8}{1+z_1}\right) \\ &= \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} \end{aligned}$$

On sommant les termes, on trouve :

$$\begin{aligned} \phi_A(a,b) &= \begin{cases} \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} & \text{Si } a \leq b \\ -\frac{(a-b)^2}{2} - \frac{a-b}{2} + \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} & \text{Si } \frac{a}{2}-1 \leq b \leq a+1 \\ -\frac{(a-b)^2}{2} - \frac{a-b}{2} + \frac{(a-2b)^2}{4} + \frac{2b-a}{2} + \frac{1+(-1)^{a+1}}{8} + \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} & \text{Si } b \leq \frac{a}{2} \end{cases} \\ &= \begin{cases} \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} & \text{Si } a \leq b \\ ab - \frac{a^2}{4} - \frac{b^2}{2} + \frac{a+b}{2} + \frac{7+(-1)^a}{8} & \text{Si } \frac{a}{2}-1 \leq b \leq a+1 \\ \frac{b^2}{2} + \frac{3b}{2} + 1 & \text{Si } b \leq \frac{a}{2} \end{cases} \end{aligned}$$

3.2.4 Méthode de comptage par automates à états finis déterministes

L'ensemble des entiers satisfait une formule de Presburger peut être reconnait par un automate à états finis (en représentations binaires).

Il existe plusieurs algorithmes pour la construction d'un automate à partir d'une contraintes arithmétique linéaire [27], [12], [3].

Dans ce qui suit, nous donnons un algorithme proposé dans [3] pour le calcul de $\sum a_i x_i = c$:

Algorithme pour construire l'automate pour $\sum a_i x_i = c$

- Créer un état étiqueté $(-c)$, on le considère l'état initial et on le marque *extensible* ;
- Créer un nouvel état étiqueté *puit*, les transitions commencent de cet état retournent à lui même ;
- Marquer ce nouveau état *non extensible ;*

Tant que il existe un état extensible nommé s

Pour chaque transition $b_1 b_2..b_v$ de cet état **faire**

Si $s\prime = (\sum_{i=1} a_i.b_i + s)/2$ est un entier **alors**

S'il n'existe pas un état étiqueté $s\prime$

Créer un nouveau état étiqueté s' et marque le *extensible* ;

Considérer cet état *destination* de la transition

Sinon

Considérer l'état étiqueté *puit* la destination de transition ;

Marquer l'état étiqueté s *non-extensible ;*

- Marquer l' état 0 comme état *accepté.*
- Marquer les autres états *refusés.*

La méthode de comptage de nombre de points par automates à états finis déterministes consiste à :

- Présenter les entiers vérifiant chaque contrainte linéaire

$$\begin{aligned}
\sum a_i x_{\ i} + b &= 0 \\
\sum a_i x_{\ i} + b &< 0 \\
\sum a_i x_{\ i} + b &\leq 0 \\
\sum a_i x_{\ i} + b &> 0 \\
\sum a_i x_{\ i} + b &\geq 0
\end{aligned}$$

d'une formule de *Presburger* par automate à état finis déterministe.

- Combiner les différents automates en utilisant les opérations sur les automate tel que : intersection, union, complément...etc.

- Le résultat est un automate où le langage reconnu est les solutions de la formule de Presbuger codé en binaire.

- Chaque chemin accepté dans l'automate est une solution.

- Alors, le comptage de solution de la méthode du formule de *Presburger* est le même que le comptage de chemins (de même langueur) accepté par l'automate.

Exemple 3.2.9 *Soit le polytope suivant*

$P = \{(x, y) \in \mathbb{Q}^2 \mid 2x + 3y - 6 \geq 0 \quad \wedge \quad -x + y + 3 \geq 0, \quad -x - 4y + 8 \geq 0\}$

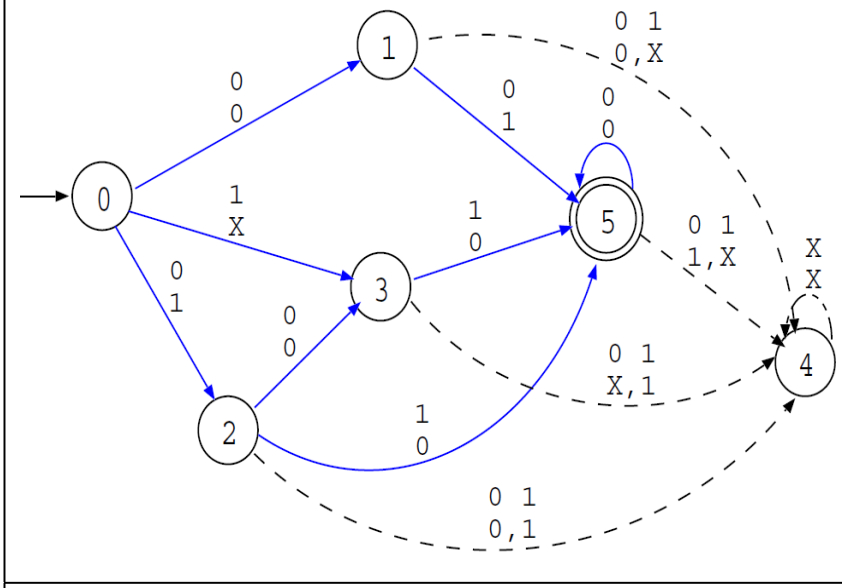


Fig 3.4 : Automate pour P

- Le caractère X indique un bit quelconque (0 ou 1).

- et les lignes en continu représentent les chemins qui donnent les mots (vecteurs entiers) acceptés par l'automate.

- Ces mots sont codés sur 3 *bits (pour chaque variable) puisque la longueur du plus long chemin de l'état initial* (0) *jusqu'à l'état final* (5) *est* 3*. Le nombre de solutions est donné par le nombre de chemins de longueur* 3 *reliant l'état initial à l'état final.*

- Dans notre exemple, ce nombre est égal à 5*.*

- Les mots acceptés par l'automate peuvent être explicitement obtenus, en concaténant, de droite à gauche, les bits reconnus

$$\begin{pmatrix} 000 \\ 010 \\ 010 \\ 001 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 011 \\ 000 \\ 100 \\ 001 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 011 \\ 001 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix},$$

Implémentation

- Cette méthode ne peut être généralisée au cas paramétré.

- La méthode de comptage par automates est de complexité proportionnelle au nombre de solutions de la formule de Presburger.

- La méthode de comptage par automates reste tout de même assez efficace lorsque

le nombre de solutions est relativement petit.

3.2.5 Méthode de Barvinok pour les polytopes non paramétriques

Barvinok a développé une méthode de calcul de points entiers pour les polytopes non paramétriques en utilisant la fonction génératrice de polytope. Dans cette méthode, on commence par le calcul de la fonction génératrice du polytope, puis on fait une évaluation de cette fonction génératrice pour obtenir le nombre de points du polytope.

La fonction génératrice est donnée par

$$f(P;x) = \sum_{m \in P \cap \mathbb{Z}^d} x^m$$

En évaluant cette fonction à $x = 1$, on obtient le nombre de termes de la série qui correspond au nombre de points entiers de P. Avec cette formule, on doit énumérer toutes les points, ce qui inefficace. Barvinok a utilisé une autre formule, qui est donnée par la somme sur l'ensemble des sommets $V(P)$ des fonctions génératrices de ses cônes supportants, en utilisant la théorème et la définition suivantes :

Théorème 3.2.10 *Théorème de Brion*
La fonction génératrice $f(P;x)$ d'un polyèdre P est égale à la somme des fonctions génératrices de ses cônes supportants

$$f(P;x) = \sum_{v \in V(P)} f(K(P,v);x)$$

tel que $V(P)$ est l'ensemble des sommets de P [31].

Définition 3.2.11 *La fonction génératrice [5] d'un cône unimodulaire est donnée par*

$$f(K,x) = \prod_{i=1}^{d} \frac{1}{1 - x^{u_i}}$$

où les $u_1, ..., u_d \in \mathbb{Z}^d$ sont les générateurs de K.

Alors, l'algorithme consiste en les transformations et les calculs suivants :

- **Les cônes supportants** : le polytope est décomposé en ses cônes supportants.

La fonction génératrice d'un polytope est égale à la somme des fonctions génératrices de ses cônes supportants (théorème précédente).

- **La triangulation des cônes supportants en cônes simpliciaux**. La triangulation se fait par plusieurs méthodes, parmi ces méthodes on a la triangulation de Delaunay. Son principe est donnée aprés.

- **La décomposition des cônes simpliciaux en un ensemble signé de cônes unimodulaires** en utilisant l'algorithme de Barvinok pour la décomposition en cônes unimodulaires.

$$\{(\epsilon_i, k_i)\} = B(K)$$

avec $\epsilon_i \in \{-1, 1\}$ signifie que la fonction génératrice de K est la somme signée des fonctions génératrices des cônes unimodulaires

- **La fonction génératrice :** pour chaque cône unimodulaire, on détermine la fonction génératrice.

Pour obtenir la formule générale de la fonction génératrice finale, on doit faire translation de fonctions génératrices de cônes unimodulaires k_i. Cela signifie que, pour chaque sommet v :

- Si v est entier, on fait une multiplication par x^v.

- Si v n'est pas entier, on a besoin de trouver un point entier $v' = E(v, K_i) = \sum \lceil \lambda_i \rceil u_j^i$, où λ est la solution rationnelle de $v = \sum \lambda_i \ u_j^i$ et tel que $(K_i + v') \cap \mathbb{Z}^d = (K_i + v) \cap \mathbb{Z}^d$.

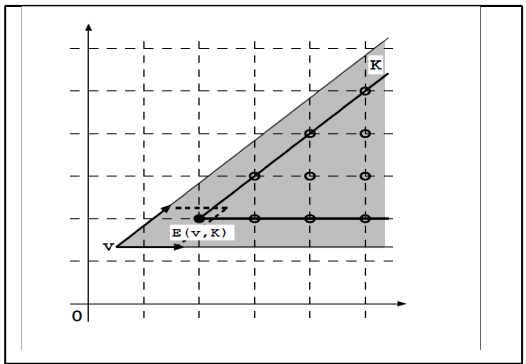


Fig 3.5 : Exemple de $E(v, K_i)$

Alors, la forme de la fonction génératrice

- $f(P; x) = \sum_{v \in V} \epsilon_i \dfrac{x^v}{\prod_{j=1}^{d} (1 - x^{u_i})}$, dont tous les sommets sont entiers et dont tous les cônes sont unimodulaires.

- $f(P; x) = \sum_{v \in V(P)} \sum_{i=1}^{|B(K_v)|} \epsilon_i \dfrac{x^{E(v,K_i)}}{\prod_{j=1}^{d} (1 - x^{u_j^i})}$ pour un polyèdre quelconque.

où K_v est la translation du cône supportant $K(P, v)$ à l'origine O, $B(K_v)$ est l'ensemble des cônes unimodulaires résultant de la décomposition de Barvinok, et u_j^i est le $j^{ème}$ vecteur générateur du $i^{ème}$ cône unimodulaire dans l'ensemble $B(K_v)$.

- **Le nombre de points entiers** est calculé en évaluant la fonction génératrice trouvée.

Algorithme de Triangulation

L'idée de cette triangulation est la suivante :

- On définie un nouveau cône K' de dimension d +1 a partir du cône polyédrique K de dimension d ayant les vecteurs générateurs $u_1, u_2, .., u_d$, en ajoutant une nouvelle coordonnée $w_i = \sum_{j=1}^{d} u_{ij}^2$ à chaque vecteur générateur u_i de K. Les vecteurs générateurs du nouveau cône K' sont donc : $(u_1, w_1), (u_2, w_2), \cdots, (u_d, w_d)$.

- Ensuite, on calcule l'enveloppe convexe inférieure de K' et on la projette sur l'espace original (de dimension d) pour obtenir la triangulation de K.

L'enveloppe convexe inférieure (par rapport à w) de K' est donnée par ses faces inférieures, où par face inférieure on désigne un système $\{x \in \mathbb{Q}^{d+1} |\ Ax = 0\}$ tel que :

i)$\{x \in \mathbb{Q}^{d+1} |\ Ax \geq 0\}$ est un sous-ensemble des inégalités définissant le cône K'

ii) tous les éléments de la $(d+1)ème$ colonne de A sont positifs.

Algorithme de décomposition de cône simplicial en un ensemble signé de cônes unimodulaires (Barvinok)

Après la triangulation, on a des cônes simpliciaux qui peuvent être non unimodulaires. (Rappelons que $C = \{x \in \mathbb{Q}^d | Ax \geq 0\}$ est un cône unimodulaire si A est une matrice unimodulaire i.e son déterminant soit ± 1), alors on doit les décomposés en cônes unimodulaires, L'idée principale est de décomposer le cône C où le déterminant de sa matrice K des générateurs est $det(K)$, en des cônes K_i avec $det(K_i) < \det(K)$.

On décompose le cône C en cherchant un vecteur w, qui peut être écrit comme une combinaison linéaire de générateurs de C avec des petites coefficients

$$w = \sum_{i=1}^{d} \alpha_i u_i \ |\alpha_i| \leq \ |det(K)|^{-\frac{1}{d}}$$

avec ce vecteur w, on décompose C en d cônes C_i avec des déterminants plus petits. Cette procédure est répétée récursivement sur chaque cône résultant jusqu'à ce que tous les cônes deviennent unimodulaires. On peut trouver plus de détails dans [33] [4][31]

Evaluation des fonctions génératrices

A ce niveau, la fonction génératrice est trouvée, on doit l'évaluer afin de pouvoir trouver sa valeur au point 1.

Dans cette partie, on va expliquer la méthode d'évaluation de la fonction génératrice à travers un exemple.

Exemple 3.2.12 *Considérons le polytope montré par la figure suivante :*

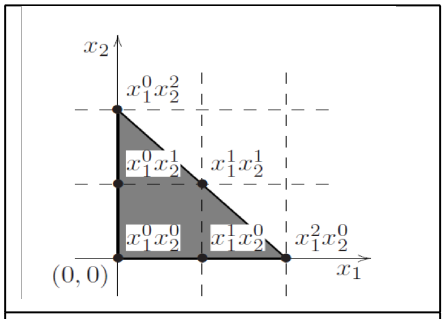


Fig 3.6 : Exemple de polytope

Les points entiers de P sont$(0,0),(1,0),(2,0),(0,1),(1,1)$, *et* $(0,2)$. *La fonction génératrice de P correspondant à ces points est :*

$$f\ (P;x) = 1 + x_1 + x_1^2 + x2 + x_1x_2 + x_2^2$$

Cette fonction est donnée par l'algorithme de Barvinok sous forme d'une somme de fonctions rationnelles qui ne nécessitent pas de parcourir tous les points entiers du polytope

$$f(P,x) = \frac{x_1^2}{(1-x_1^{-1})(1-x_1^{-1}x_2)} + \frac{x_2^2}{(1-x_2^{-1})(1-x_1x_2^{-1})} + \frac{1}{(1-x_1)(1-x_2)}$$

C'est la fonction génératrice à évaluer :
- On cherche un vecteur u qui n'est pas orthogonal à tous les vecteurs générateurs, par l'application de l' algorithme de Barvinok ou de préférence, on le prendre aléatoire [33]. Ici, $u = (u_1,u_2) = (1,-1)$
- On substitue $x_1 = (s+1)^{u_1}$ *et* $x_2 = (s+1)^{u_2}$ *dans* $f(P;x)$, *on obtient :*

$$f\ (P;s) \quad = \quad \frac{(s+1)^2}{(1-(s+1)^{-1})(1-(s+1)^{-2})} + \frac{(s+1)^{-2}}{(1-(s+1))(1-(s+1)^2)} + \frac{1}{(1-(s+1))(1-(s+1)^{-1})}$$

Afin d'obtenir des puissances positives seulement, dans le dénominateur, on multiple le numérateur et le dénominateur pour chaque terme par $(s+1)^{-c}$ *avec c est la somme de puissances négatives dans le dénominateur. Alors, on obtient*

$$f\ (T;s) \ = \ \frac{(s+1)^5}{(1-(s+1))(1-(s+1)^2)} + \frac{(s+1)^{-2}}{(1-(s+1))(1-(s+1)^2)} - \frac{(s+1)}{(1-(s+1))(1-(s+1))}$$

- Réécrire chaque terme sous la forme $\frac{1}{s^2}\frac{N(s)}{D(s)}$ *(à travers Taylor Expansion et division polynomial) on trouve*

$$f\ (T;s) = \frac{1}{s^2}\left(\frac{1}{2} + \frac{9}{4}s + \frac{31}{8}s^2 + ...\right) + \frac{1}{s^2}\left(\frac{1}{2} - \frac{5}{4}s + \frac{17}{8}s^2 + ...\right) - \frac{1}{s^2}\left(1 + 0s^2 + ...\right)$$

Finalement, le nombre des points entiers dans le polytope est donné par la somme des coefficients de s^2 *dans* $\frac{N(s)}{D(s)}$ *i.e* $\frac{31}{8} + \frac{17}{8} - 0 = 6$.

Les détails de la méthode et les bases mathématique utilisés sont décrit dans [33] [4] [38].

Implémentation

L'algorithme proposé par Barvinok a été récemment implémenté dans la bibliothèque Barvinok et LattE. [33].

3.2.6 Généralisation de Méthode de Barvinok pour les polytopes paramétrés

Cette méthode est proposée par *Rachid SEGHIR [33].* C'est une généralisation de l'algorithme de Barvinok.

La fonction génératrice (paramétrée) du polytope est donnée par la somme des fonctions génératrices des cônes supportants en ses différents sommets. Pour cela, on doit calculer :

- Les sommets paramétrés du polytope
- Les vecteurs générateurs de ses cônes supportants en ces sommets.

Les sommets d'un polytope paramétré sont des expressions affines de paramètres, qui peuvent n'être valides que sur des sous-ensembles de valeurs des paramètres (domaines de validité).

Clauss et Loechner [18] ont montré (comme nous avons vu dans la méthode d'interpolation) que tout polytope paramétré peut être décomposé en un ensemble de polytopes,

chacun d'entre eux étant défini sur un domaine de validité, où tous ses sommets sont valides. Par conséquent, la fonction génératrice d'un polytope paramétré quelconque est donnée par un ensemble de fonctions génératrices, chacune d'entre elles correspond à un polytope valide sur un sous-ensemble de valeurs des paramètres (domaine de validité) .

L'algorithme est complètement décrite dans [33]. Avant tous, on utilise la méthode de Clauss et Loechner pour calculer les sommets paramétriques et leurs domaines de validités, ensuite, on calcule une fonction génératrice pour chaque domaine de validité.

Seghir [33] a prouvé que le dénominateur de la fonction génératrice d'un polytope paramétré est calculé de la même façon que pour un polytope non paramétré. Il ne reste donc qu'à définir son numérateur. Deux cas sont possibles : - Sommets entiers

- Sommets rationnels

Sommets entiers

Un sommet paramétré v(p) est dit entier si toutes ses coordonnées sont des fonctions affines à coefficients entiers, i.e., $v(p) = (g_1(p), g_2(p), \cdots, g_d(p))$, où $g_i(p) = a_0 + \sum_{j=1}^{n} a_j p_j$, avec $a_j \in \mathbb{Z}, \forall j \in \{0, 1, ..., n\}$ [33].

Dans ce cas, la fonction génératrice sur un domaine de validité D est donc donnée par :

$$f_D(P;x) = \sum_{v(p)\in V_D(P)} \sum_{i=1}^{|\mathrm{B}(\mathrm{K}_{\mathrm{v(p)}})|} \in_i \frac{x_1^{g_1(p)} x_2^{g_2(p)} ... x_d^{g_d(p)}}{\prod_{J=1}^{d} (1 - x^{u_j^i})}$$

tel que $X^{v(p)} = x_1^{g_1(p)} x_2^{g_2(p)} ... x_d^{g_d(p)}$.

Sommets rationnels

Un sommet paramétré est dit rationnel si au moins une de ses coordonnées n'est pas entier. Dans ce cas,

$$f_D(P;x) = \sum_{v(p)\in V_D(P)} \sum_{i=1}^{|\mathrm{B}(\mathrm{K}_{\mathrm{v(p)}})|} \in_i \frac{X^{E(v(p),K_i)}}{\prod_{J=1}^{d} (1 - x^{u_j^i})}$$

où $E(v(p), K_i)$ est donné en fonction du sommet paramétré $v(p) = (g_1(p), \cdots, g_d(p))$ et des vecteurs générateurs u_j^i du cône K_i, comme suit :

$$E(v(p), K_i) = \sum_{j=1}^{d} \lceil \lambda_j(p) \rceil \, u_j^i$$

où les $\lambda_j(p)$ sont des fonctions affines rationnelles des paramètres. Ce sont les solutions

du système $v(p) = \sum_{j=1}^{d} \lambda_j(p) u_j^i$

La fonction génératrice dans ce cas est donnée par :

$$f_D(P;x) = \sum_{v(p) \in V_D(P)} \sum_{i=1}^{|\mathrm{B}(\mathrm{K}_{\mathrm{v(p)}})|} \in_i \frac{x_1^{\sum_{j=1}^{d} \lceil \lambda_j(p) \rceil u_{j1}^i} \; x_2^{\sum_{j=1}^{d} \lceil \lambda_j(p) \rceil u_{j2}^i} \; \ldots x_d^{\sum_{j=1}^{d} \lceil \lambda_j(p) \rceil u_{jd}^i}}{\prod_{J=1}^{d} (1 - x^{u_j^i})}$$

On résume l'algorithme dans les points suivants :

1. Pour chaque sommet $vi(p) \in V(P)$:
   (a) Calculer le cône supportant $cone(P, v_i(p))$
   (b) Soit $K = cone(P, v_i(p)) - v_i(p)$ la translation à l'origine de $cone(P, v_i(p))$
   (c) Soit $\{(c_j, K_j)\} = B(K)$ la décomposition unimodulaire de K
   (d) Pour chaque cône unimodulaire K_j
      i. Calculer la fonction génératrice $f(K_j; x)$
   (c) $f(cone(P, vi(p); x) = \sum_j \in_j x^{E(v_i(p), K_j)} f(K_j; x)$
2. Pour chaque domaine de validité C_k de P :
   (a) $f_{C_k}(P;x) = \sum_{vi \in V_{C_k}(P)} f(cone(P, v_i(p); x)$
   (b) Evaluer $f_{C_k}(P;1)$

Implémentation

Cet algorithme de comptage de points entiers dans des polytopes paramétrés a été implémenté, essentiellement dans la librairie Barvinok, en utilisant la procédure de calcul de la décomposition unimodulaire de Barvinok implémentée dans LattE, et la librairie Polylib pour réaliser les opérations polyédriques.

3.2.7 Méthodes approximatives de dénombrement des points entiers dans un polytope

Certaines applications ont besoins d'un calcul approximatif de nombre de points entiers dans un polytope. Il existe des applications qui ne sont besoin qu'à une limite (max ou min) de nombre de points entiers dans un polytope.

Les méthodes approximatives permettent de calculer le nombre de points entiers d'une façon approximative en évitant la complexité des méthodes de calcul exacte.

Le principe est de trouver pour chaque polynôme d'Ehrhart périodique un polynôme d'Ehrhart non périodique approximatif en utilisant différentes techniques, par exemple :

- Remplacer chaque coefficient périodique de polynôme d'Ehrhart par sa valeur moyenne.

- Une autre méthode est d'utiliser une transformation linéaire entière L afin de trouver un autre polytope tel que $P\prime = LP$ et le nombre de point entier donné par

$$\#(Pp \cap \mathbb{Z}^d) \approx \frac{\#(P'_p \cap \mathbb{Z}^d)}{\det(L)}$$

On donne pas ici des détails sur les méthodes approximatives, puisqu'on s'intéresse dans ce mémoire par les méthodes exactes. Pour d'autres détailles sur les méthodes approximatives, le lecteur est invité à consulter [26].

Exemple 3.2.13

$$P1 = \begin{cases} i > 0 \\ j \geq 0 \\ 2i \leq n \\ 2j + 1 \leq n \end{cases}$$

Le polynôme d'Ehrhart est

$$\varepsilon_{p_1}(n) = \frac{1}{4}n^2 + \frac{1}{2}n + \left[0 \quad \frac{1}{4}\right]_n$$

Le polytope précédent $P1$ *peut être développé par :*

$$\begin{pmatrix} i\,' \\ j\,' \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

en un polytope $P\,'_1$ *:*

$$P\,'_1 = \begin{cases} i\,' \geq 0 \\ j\,' \geq 0 \\ i\,' \leq n \\ j\,' + 1 \leq n \end{cases}$$

Le polynôme d'Ehrhart de $P\,'_1$ *est*

$$\varepsilon_{p'_1}(n) = n.(n-1) = n^2 - 2n + 1$$

Le taux approximatif entre P *et* P' *est* 4 *(d'après la matrice de développement), donc*

$$\varepsilon_{p'_1}(n)/4 = \frac{n^2 - 2n + 1}{4}$$

est un approximatif pour ε_{p_1}*.*

3.3 Comparaison entre les méthodes

Dans cette section, nous comparons entre les différentes méthodes de dénombrement des points entiers dans un polytope discutées précédemment :

• **Méthodes des sommes** :

- Méthode de sommes semble pratique dans des cas simples, mais elle reste très difficile à automatiser dans le cas général.

- Dans le pire des cas, elle est exponentielle.

• **Méthode d'interpolation de Clauss et Loechner**

- Le temps de calcul par interpolation d'un quasi-polynôme peut être exponentiel dans le pire des cas, et ce même pour une dimension fixée.

- Puisque la méthode d'interpolation représente les nombres périodiques par des tableaux multidimensionnels dont la taille dépend des périodes, la taille du résultat (quasi-polynômes) peut aussi être exponentielle dans le pire des cas.

- Dans certains cas, la méthode échoue.

• **Méthode des fractions partielles** :

- Baldoni et al. [33] ont proposé une implémentation des fractions partielles. Ils ont mentionné que leurs programmes ont été conçus spécialement pour ce type de polytopes

- La méthode est théoriquement générale, et pourrait être appliquée à tout polytope paramétré. Mais aucune analyse de complexité ni expériences n'ont été fournies. A noter que les techniques standard de calcul des fractions partielles sont exponentielles [33].

• **Méthode de comptage par automates**

- C'est une méthode assez efficace lorsque le nombre de solutions de la formule de Presburger est relativement petit.

- Cette méthode est de complexité proportionnelle au nombre de solutions de la formule de Presburger. Plus ce nombre est grand, plus les automates sont volumineux, ce qui influe considérablement sur le temps de calcul.

• **Méthode de Barvinok**

Cette algorithme de calcule de nombre de points entiers dans un polytope est en un temps polynomial (pour une dimension fixée) [4].

• **Méthode de Barvinok paramétré** :

Lorsque la dimension est fixée [33] :

- La taille de la décomposition en domaines de validité est polynomiale en la taille du système de contraintes définissant le polytope paramétré, et elle peut être calculée en un temps polynomial.

- La taille des quasi-polynômes représentant le nombre de points entiers d'un polytope paramétré (output) est polynomiale en la taille des contraintes linéaires le définissant, et ils sont calculés en un temps polynomial.

3.4 Exemples d'utilisation de dénombrement des points entiers dans un polytope dans le parallélisation automatique

Le nombre de points entiers est utilisé dans plusieurs cas pour répondre aux besoins de parallélisation. Il est utilisé par exemple :

- Evaluation du temps d'exécution du programme.
- Réalisation d'équilibrage entre les processeurs.
- Utilisation des techniques d'optimisation de mémoire afin d'améliorer la parallélisation.
-...etc.

Dans ce qui suit, nous donnons des exemples d'utilisation de *dénombrement de point entiers dans un polytope* dans le domaine de parallélisation automatique.

3.4.1 Optimisation de la localité spatiale par réorganisation des données

Clauss a proposé dans [17] des optimisations de la localité spatiale par réorganisation des données, afin qu'un bloc de données chargé dans la mémoire cache sera complètement utilisé par le nid de boucles pour une itération et des itérations successives (avant qu'il soit retiré de la mémoire cache). Alors, on doit déterminer une réorganisation des éléments de tableau afin de pouvoir une utilisation complète de bloc de données chargé dans la mémoire cache. Cette méthode d'optimisation se fait en calculant l'ensemble d'itérations référencent un élément de tableau, puis fait des décisions selon cet ensemble.

Pour illustrer ses résultats, on prend l'exemple suivant :

Exemple 3.4.1 *Soit le nid de boucles suivant*

$$
\begin{array}{l}
\textit{for } i_1 = 1 \textit{ to } p \\
\quad \textit{for } i_2 = i_1 \textit{ to } 2 * i_1 - 1 \\
\qquad \textit{for } i_3 = i_1 - i_2 \textit{ to } i_1 + i_2 \\
\qquad\quad X(i_1, i_2, i_3) := Y(i_1 + i_2 - 1, i_1 + i_3 + 2, i_2 + i_3) + 10
\end{array}
$$

où p est un paramètre entier positif.
Les accès au tableau Y sont définis par une fonction affine du vecteur d'itération $\overrightarrow{I} = (i_1, i_2, i_3)$, $\quad f(\overrightarrow{I}) = R\overrightarrow{I} + \overrightarrow{\sigma}$

$$f(i_1, i_2, i_3) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} * \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix}$$

R est appelée matrice d'accès et $\overrightarrow{\sigma}$ *est appelé vecteur de déplacement.*

Prenant $Y(\overrightarrow{I_0}) = Y(i\ 0_1, i\ 0_2, ..., i\ 0_n)$, un élément de tableau Y, (regarder l'exemple), référencé par un nid de boucles. L'*ensemble des itérations référençant* cet élément $Y(\overrightarrow{I_0})$ est $P(\overrightarrow{I_0}) = \left\{ \overrightarrow{I} \in P \mid R * \overrightarrow{I} + \overrightarrow{\sigma} = \overrightarrow{I_0} \right\}$, où P est l'espace d'itération. Si $P(\overrightarrow{I_0})$ est vide, alors $Y(\overrightarrow{I_0})$ est jamais référencé par le nid de boucle, alors ne doit jamais être chargé dans la mémoire cache. Cette information est donnée en calculant **le polynôme d'Ehrhart** $EP(\overrightarrow{I_0})$ de $P(\overrightarrow{I_0})$. Si $P(\overrightarrow{I_0})$ est réduit à une seule itération, alors le tableau n'est pas réutilisable temporellement, et $Y(I)$ sera placé en mémoire principale à l'adresse physique $b + q \times w$ (où b est l'adresse de base du tableau Y, et w est la taille en octets d'un élément physique) pour la $q - ième$ itération de $Y(I)$. Si $P(\overrightarrow{I_0})$ défini plusieurs itérations, que signifie une réutilisation temporelle, la première itération selon l'ordre lexicographique de cet ensemble est choisie. Cette valeur minimale peut être calculée en utilisant la bibliothèque *PolyLib*.

Selon la fonction affine qui définit l'accès au tableau, il y a des élément qui ne seront jamais accédés, alors il faut pas les chargés en mémoire cache. L'*ensemble des données effectivement* utilisé est un sous ensemble des données de tableau. On le définit par :

$$D_Y = \left\{ \overrightarrow{I}_0 \mid \overrightarrow{I_0} = A * \overrightarrow{I} + \overrightarrow{\sigma},\ \ \overrightarrow{I} \in P \right\}$$

Clauss trouve que le calcul de l'enveloppe convexe $conv(DY)$ est suffisant, en utilisant son programme pour le calcul des sommets et domaine de validation [18].

Placement des données en mémoire

Après cette présentation des notions utilisés dans les travaux de Clauss, on expose ici sa méthode de la réorganisation des données pour l'optimisation de la localité spatiale.

La première itération qui est défini par le minimum lexicographique de $P(\overrightarrow{I_0})$ fait référencer à un élément $Y(\overrightarrow{I_0})$. Soit $\overrightarrow{I}_{min}$ ce point, dont les coordonnées sont des fonctions affines de $\overrightarrow{I_0}$.

La nouvelle position de la donnée $Y(\overrightarrow{I_0})$ en mémoire principale est donnée par la

position de cette itération, selon l'ordre d'exécution, relativement à l'adresse de base b du tableau Y.

L'optimalité spatiale est garantie, car la donnée référencée juste avant ou juste après $Y(\overrightarrow{I_0})$, sera contiguë à $Y(\overrightarrow{I_0})$ en mémoire.

La position de $\overrightarrow{I}_{min}$ est déterminée en calculant le nombre d'itérations exécutées avant $\overrightarrow{I}_{min}$:

$$P(\overrightarrow{I}_{min}) = \left\{ \overrightarrow{I} \in P \mid \overrightarrow{I} \prec\prec \overrightarrow{I}_{min} \right\}$$

$\prec\prec$ est l'ordre lexicographique.

$$P(\overrightarrow{I}_{min}) = P_1(\overrightarrow{I}_{min}) \cup P_2(\overrightarrow{I}_{min}) \cup P3(\overrightarrow{I}_{min}) \cup ...P_n(\overrightarrow{I}_{min})$$

avec

$$\begin{aligned}
P_1(\overrightarrow{I}_{min}) &= \left\{ \overrightarrow{I} \in P \mid i_1 < i_{1,min} \right\} \\
P_2(\overrightarrow{I}_{min}) &= \left\{ \overrightarrow{I} \in P \mid i_1 = i_{1,min}, i_2 < i_{2,min} \right\} \\
P_3(\overrightarrow{I}_{min}) &= \left\{ \overrightarrow{I} \in P \mid i_1 = i_{1,min}, i_2 = i_{2,min}, i_3 < i_{3,min} \right\} \\
&.... \\
P_3(\overrightarrow{I}_{min}) &= \left\{ \overrightarrow{I} \in P \mid i_1 = i_{1,min}, i_2 = i_{2,min}, ..., i_{n-1} = i_{n-1,min}, i_n < i_{n,min} \right\}
\end{aligned}$$

Puisque $\overrightarrow{I}_{min}$ est défini comme une fonction affine de $\overrightarrow{I_0}$, $P(\overrightarrow{I}_{min})$ est une union de polyèdre paramétrés par $\overrightarrow{I_0}$.

Le nombre d'itérations définies par $P(\overrightarrow{I}_{min})$ et paramétré par $\overrightarrow{I_0}$ est donné par le *polynôme somme de polynômes d'Ehrhart* $EP_q(\overrightarrow{I}_0)$ de chacun de ces polyèdres. Ce polynôme définit une fonction de placement des éléments $Y(\overrightarrow{I_0})$ en mémoire principale, en donnant leur position relative à l'adresse de base du tableau Y.

Alors pour une localité spatiale optimale, au lieu d'évaluer classiquement les références d'un tableau pour un compilateur, on évalue le polynôme d'Ehrhart déterminé préalablement. Cette opération de compilateur peut être vue comme la production de code exécutable suivant :

for $i_1 = l_1$ to h_1
  for $i_2 = l_2(l_1)$ to $h_2(i_1)$
    ...
      for $i_n = l_n(i_1, i_2, ..;, i_{n-1})$ to $h_n(i_1, i_2, ..;, i_{n-1})$
        ... $Y(EP(A.(i_1, i_2, ..., i_n)^T + (\sigma_1, \sigma_2, ..., \sigma_n)^T))$....

où Y est vue maintenant comme un tableau unidimensionnel.

Exemple 3.4.2 *suite de l'exemple précédent*
Prenons un élément $Y(i\ 0_1, i\ 0_2, i\ 0_3)$, *l'ensemble des itérations qui le référence défini par :*

$$P(i\ 0_1, i\ 0_2, i\ 0_3) = \{(i_1,i_2,i_3) \in P \mid i_1 + i_2 - 1 = i\ 0_1, i_1 + i_2 + 2 = i\ 0_2, i_2 + i_3 = i\ 0_3\}$$

$$= \left\{ \begin{array}{c} (i_1,i_2,i_3) \in P \mid i_1 = \dfrac{i0_1 + i0_2 - i0_3 - 1}{2},\ i_2 = \dfrac{i0_1 - i0_2 + i0_3 + 3}{2}, \\ i_3 = \dfrac{-i0_1 + i0_2 + i0_3 - 3}{2} \end{array} \right\}$$

- Cette ensemble est vide si au moins une des fraction n'est pas entière ou est une valeur qui n'appartienne pas à l'espace d'itération P.
- De plus $P(i\ 0_1, i\ 0_2, i\ 0_3)$ *définit au plus une seule itération.*
- L'ensemble de données effectivement référencées est défini par :

$$D_Y = \left\{ \begin{array}{c} (i\ 0_1, i\ 0_2, i\ 0_3) \mid i\ 0_1 = i_1 + i_2 - 1, i\ 0_2 = i_1 + i_2 + 2, i\ 0_3 = i_2 + i_3, \\ 1 \le i_1 \le p, i_1 \le i_2 \le 2i_1 - 1, i_1 - i_2 \le i_3 \le i_1 + i_2 \end{array} \right\}$$

$$\begin{array}{rcl} Conv(D_Y) & = & \{(i\ 0_1, i\ 0_2, i\ 0_3) \mid i\ 0_1 + i\ 0_2 \le i\ 0_3 + 2p + 1, \\ 3i\ 0_3 + 7 & \le & i\ 0_1 + 3i\ 0_2,\ i\ 0_2 \le i\ 0_3 + 2, i\ 0_1 + i\ 0_2 \le 3i\ 0_3\ + 1, I02 + i\ 0_3 \le 3i\ 0_1 + 5\ \} \end{array}$$

- Le polytope $P(i\ 0_1, i\ 0_2, i\ 0_3)$ *définit au plus un point entier associé à une itération référençant un élément* $Y(i\ 0_1, i\ 0_2, i\ 0_3)$.
Cette itération est défini par :

$$\overrightarrow{I_{min}} = (\frac{i0_1 + i0_2 - i0_3 - 1}{2}, \frac{i0_1 - i0_2 + i0_3 + 3}{2}, \frac{-i0_1 + i0_2 + i0_3 - 3}{2})$$

Le nombre d'itération exécutées avant $\overrightarrow{I_{min}}$ *est déterminé par le polynôme d'Ehrhart de l'union des polyèdres suivants :*

$$\begin{array}{rl} & P(\frac{i0_1 + i0_2 - i0_3 - 1}{2}, \frac{i0_1 - i0_2 + i0_3 + 3}{2}, \frac{-i0_1 + i0_2 + i0_3 - 3}{2}) \\ = & \left\{(i_1,i_2,i_3) \in P \mid i_1 < \dfrac{i0_1 + i0_2 - i0_3 - 1}{2}, (i\ 0_1, i\ 0_2, i\ 0_3) \in conv(D_Y)\right\} \\ & \cup \left\{ \begin{array}{c} (i_1,i_2,i_3) \in P \mid i_1 = \frac{i0_1+i0_2-i0_3-1}{2}, i_2 < \frac{i0_1-i0_2+i0_3+3}{2}, \\ (i\ 0_1, i\ 0_2, i\ 0_3) \in conv(D_Y) \end{array} \right\} \\ & \cup \left\{ \begin{array}{c} (i_1,i_2,i_3) \in P \mid i_1 = \frac{i0_1+i0_2-i0_3-1}{2}, i_2 = \frac{i0_1-i0_2+i0_3+3}{2}, \\ i_3 \le \frac{-i0_1+i0_2+i0_3-3}{2}, (i\ 0_1, i\ 0_2, i\ 0_3) \in conv(D_Y) \end{array} \right\} \end{array}$$

Alors, les polynômes d'Ehrhart sur chaque polytope sont :

$$\boxed{\begin{array}{l} EP_1(i\ 0_1, i\ 0_2, i\ 0_3) = -\frac{1}{8}i0_3^3 + \frac{3}{8}i0_2i0_3^2 + \frac{3}{8}i0_1i0_3^2 - \frac{3}{4}i0_3^2 - \frac{3}{8}i0_2^2i0_3 - \frac{3}{4}i0_1i0_2i0_3 \\ +\frac{3}{2}i0_2i0_3 - \frac{3}{8}i0_1^2i0_3 + \frac{2}{3}i0_1i0_3 - \frac{11}{8}i0_3 + \frac{1}{8}i0_2^3 + \frac{3}{8}i0_1i0_2^2 - \frac{3}{4}i0_2^2 + \frac{3}{8}i0_1^2i0_2 \\ -\frac{3}{2}i0_1i0_2 + \frac{11}{8}i0_2 + \frac{1}{8}i0_1^3 - \frac{3}{4}i0_1^2 + \frac{11}{8}i0_1 - \frac{3}{4} \end{array}}$$

$$\boxed{EP_2(i\ 0_1, i\ 0_2, i\ 0_3) = i0_1i0_3 - i0_1i0_2 + i0_3 - i0_2 + 2i0_1 + 2}$$

$$\boxed{EP_3(i\ 0_1, i\ 0_2, i\ 0_3) = \frac{3}{2}i0_3 - \frac{1}{2}i0_2 - \frac{1}{2}i0_1 + \frac{3}{2}}$$

Donc, l'adresse mémoire de tout élément $Y(i\ 0_1, i\ 0_2, i\ 0_3)$*, résultant en une localité spatiale optimale est*

$$\boxed{\begin{array}{c} EP(i\ 0_1, i\ 0_2, i\ 0_3) * w + b = ((EP_1(i\ 0_1, i\ 0_2, i\ 0_3) + EP_2(i\ 0_1, i\ 0_2, i\ 0_3) + \\ EP_3(i\ 0_1, i\ 0_2, i\ 0_3)) * w + b \end{array}}$$

où b *est l'adresse de base du tableau* Y*.*
Soient les itérations successives $(3, 5, 8), (4, 4, 0), (4, 4, 1)$*. Les élément référencés par ces itérations sont* $Y(7, 13, 13), Y(7, 6, 4), Y(7, 7, 5)$.
En utilisant le résultat précédemment, on trouve des adresses mémoires contiguës pour ces éléments :
$EP(7, 13, 13) = 15 + 16 + 11 = 42$
$EP(7, 6, 4) = 42 + 0 + 1 = 43$
$EP(7, 7, 5) = 42 + 0 + 2 = 44$
Par conséquent, ces éléments seront mémorisés aux adresses $42 * w + b$,$43 * b + w$, $44 * w + b$.

3.4.2 Réutilisation des distances

Dans [38], le travail réalisé présente une technique d'analyse basée sur la réutilisation des distances (reuse distances), pour communiquer l'accès à une location mémoire au processeur.
La réutilisation des distances (Reuse distance) d'un certain accès dans une boucle à un élément mémoire A est le nombre de différents éléments mémoires accédés entre cet accès et l'accès suivant à A, et il est dépend de l'itération de boucle. Ce nombre est calculé par le calcul de nombre de points entiers dans un union fini des ensembles.

3.4.3 Optimisation de copier des données de mémoire globale en mémoire partagée et estimation de trafic mémoire dans GPU

Dans [8][13], le travail réalisé est concernant le développement d'un framework de compilation automatique pour la génération des programmes parallèles en basant sur le modèle polyédrique. Et comme architecture réelle, ils ont utilisé l'architecture GPU (Graphical Processing Unit) qui présente une classe d'architecture parallèle spécialisée dans les tâches graphique et 3D, avec une puissance de calcul énorme.

Avant d'exposer le travail réalisé dans [8][13], on commence par une description des notions et techniques utilisés dans ce framewarke tels que : l'architecture considérée, la technique de tilling utilisée et l'espace de données accédé.

Architecture

L'architecture considérée est composée d'un ensemble de multiprocesseurs (MIMD units), dont chaque multiprocesseurs composé d'un ensemble de cœurs processeurs (processors cores) exécutent en SIMD.

- L'ensemble de multiprocesseurs constituent le niveau externe. Ils communiquent à travers une mémoire globale lente.

- L'ensemble des cœurs processeurs dans chaque multiprocesseurs constitue le niveau interne. Ils communiquent à travers une mémoire locale partagée.

Il arrive que plus qu'un processus est lancé au niveau externe, alors la mémoire locale doit être partagée de telle façon que les processus de niveau interne appartenant au même processus de niveau externe utilisent (partagent) la même partie de mémoire locale.

Approche de tilling utilisée

- Pour un nombre total spécifie de processus lancés dans le même processeur de niveau externe, il y a un plafond supérieur pour la partie de la mémoire locale valable pour chaque processus. Cette limite est donnée par la division de la taille totale de mémoire locale dans le processeur par le nombre de processus attribués à ce processeur.

- On fixe le nombre de processus parallèles dans le niveau externe et interne pour être une multiple de nombre de processeurs parallèles physiques dans chaque niveau.

- Premièrement on effectue le tiling de domaine de boucles au niveau externe afin de distribuer les tiles à travers les processus parallèles de niveau externe.

- Si le *tile* dans un niveau externe est plus large que la mémoire locale qu'elle est

permise, il devient nécessaire d'introduire un niveau de plus de *tilling*, dans le but de limiter la quantité de mémoire local nécessaire.

Dans ce cas, on divise le *tile* dans le processus de niveau externe en *sous-tile*, tel que chaque *sous-tile* consomme une mémoire locale ne dépasse pas la taille permise. On trouve un ensemble optimal de tailles de *tile* qui définit une unité atomique de calcul dans un *tile* de niveau externe sous les contraintes de mémoire locale limitée. Une fois le *tiling* de niveau extérieur est fait, on effectue un *tiling* de niveau interne de l'espace de boucles qui distribue également les itérations d'une unité atomique de calcul exécutée dans un processus parallèle de niveau externe entre les processus parallèles de niveau interne.

Espace de données accédées

Le framework prend comme input l'espace d'itérations de tous les instructions et les fonctions d'accès pour toutes les références.

pour chaque tableau A, soient $S_1, S_2, ..S_q$ les instructions dans un bloc de programme donné contient les références au A, et l_k représente le domaine d'itération de l'instruction S_k $(1 \leq k \leq q)$. $F_k^1, F_k^2, ..., F_k^p$ les matrices représentent les fonctions de lecture des références (the afine read reference functions) de A dans une instruction S_k, et G_k la matrice représente les fonctions d'écriture des références (the afine write reference functions) de A.

L'espace de données accédées par une référence (présenté par une fonction matrice d'accès F) dans une instruction représenté par un espace d'itération I est donné par FI. L'ensemble de tous les espaces de données accédés par toutes les lectures de références de A dans toutes les instructions dans le bloc de programme est

$$DS_r^A = \left\{ F_k^l \; I_k \;\; | \; 1 \leq l \leq p \;\wedge\; 1 \leq k \leq q \right\}$$

De façon similaire, l'ensemble de tous les éspaces de données accédés par toutes les écritures de références de A

$$DS_w^A = \left\{ G_k^l \; I_k \;\; | \;\; 1 \leq k \leq q \right\}$$

L'ensemble de tous les espaces accédés par toutes les références de A dans un bloc est

$$DS_{rw}^A = DS_w^A \cup DS_w^A$$

Après la présentation de ces notions, on passe au travaux, qu'utilisent le nombre

de points entiers dans un polytope ou les outils développés pour ce calcul, réalisés dans [8][13].

Optimisation de copier des données de mémoire globale en mémoire partagée (shared)

Les tableaux et les données réutilisables doivent efficacement copier de/vers mémoire partagée.

Soit I un polytope présente le domaine d'itérations et $F_1, F_2, ..., F_k$ un ensemble des fonctions d'accès au tableau . Les éléments accédés dans le domaine d'itération (nommé ci-après, l'espace de données accessibles) est données par : $DS = \bigcup_{j=1}^{k} F_j\ I$.

- Pour trouver l'espace de données pour chaque *tile*, les auteurs de [8] utilisent *Polylib* (la bibliothèque utilisée dans la méthode d'interpolation de Claus).

Modèle d'estimation du trafic de mémoire (Memory Traffic)

Dans ce paragraphe, on parle d'un modèle d'estimation du trafic de mémoire pendant l'exécution d'un tile. Cela est ensuite utilisé pour orienter la recherche sur la taille des tiles .

Dans le modèle de l'estimation de trafic mémoire, les auteurs utilisent une fonction f qui compte *le nombre de points entiers dans un polytope*, et cela peut être calculé en utilisant le polynôme d'Ehrhart.

Considérons un tile qui va être exécuté par un bloc de processus ou un processus.

Considérons un tile de n boucles avec les tailles de tile soient $t_1, t_2..t_n$. Considérons k tableaux $(a_1, a_2, ..., a_k)$ soient accédés dans le tile.

Soit r_i le nombre des références lues, wi le nombre de références écrites de tableau a_i. Soient $F_{i1}, F_{i2}, ..., F_{in}$ les accès pour la lecture d'un tableau a_i dans le tile et G_{i1}, G_{i2}, G_{iw} les accès pour l'écriture dans tableau a_i dans le tile.

I est le domaine d'itération du tile paramétré par les tailles de tile.

Soit f la fonction pour compter le nombre de points entiers dans un polytope en donnant les paramètres.

DS_{li} décrit l'espace de références de a_i accédés pour lecture. Le nombre de points entiers dans DS_{li} donne le nombre de chargements dues de tableau a_i

DS_{si} décrit l'espace de références de a_i accédés pour écriture. Le nombre de points entiers dans DS_{si} donne le nombre de sauvegardes dues de tableau a_i. Le modèle pour l'estimation de chargement et sauvegarde dans un tile peut caractérisé comme suit :

$$DS_{li} = \bigcup_{j=1}^{r_i} F_{ij}\ I \quad et \quad DS_{si} = \bigcup_{j=1}^{r_i} G_{ij}\ I$$

Le nombre de chargement (load) et sauvegarde (stores) dans un tile =

$$\sum_{i=1}^{k} f(DS_{l_i}, t_1, t_2, ..., t_n) + f(DS_{s_i}, t_1, t_2, ..., t_n)$$

Le trafic total de mémoire est estimé en basant sur le nombre des itérations dans l'espace d'itération dans le tile, c-à-d le nombre de tiles.

3.5 Conclusion

Le calcul de nombre de points entiers dans un polytope est très commandé dans le domaine de parallélisation afin de pouvoir répondre aux questions critiques du contexte de parallélisme. Les algorithmes de calcul des points entiers dans un polytope sont nombreuses. Dans ce chapitre, on a présenté les plus connus et répondus. On a distingué entre les algorithmes paramétrés, et ceux non paramétrés, ainsi ceux généraux et ceux développés pour des cas particuliers. Des exemples d'utilisation de ce nombre sont expliqués à la fin du chapitre.

Dans le chapitre suivant nous présentons notre proposition pour le calcul de nombre de points entiers dans un sous ensemble des polytopes.

Chapitre 4

Développement des algorithmes de dénombrement des points entiers dans un polytope

4.1 Introduction

Les travaux réalisés dans le contexte de développement des méthodes et des algorithmes de dénombrement des points entiers contenus dans un polytope ont pris différentes orientations. Il existe des algorithmes efficaces pour des dimensions fixes, des algorithmes sont appliqués dans des domaines bien spécifiés, et d'autres sont applicables dans des cas généraux. Cependant, le choix de telle ou telle technique dépend du but attendu de l'utilisation. Les calculs en utilisant les bibliothèques existant (Polylib, Barvinok...) donnent leurs résultats dans des temps d'exécution différents.

Concernant ceux applicables dans le cas général, Dans certains cas, il donnent un temps d'exécution important quand on les applique dans certains cas particuliers.

Dans ce chapitre, nous présentons l'importance de notre travail. Nous montrons ici que le développement des algorithmes pour un domaine restreint, améliore le temps de calcul de dénombrement de points, en utilisant les aspects mathématiques particuliers à ce domaine.

Notre travail est basé sur la méthode d'interpolation de Clauss et Loechner, tout en choisissant un ensemble de polytopes particulier.

Ce chapitre est organisé comme suit : Dans la section domaine de travail, nous spécifions la famille de polytopes concernée par notre étude. Dans la section suivante, nous expliquons la démarche à suivre en commençant par une comparaison entre notre méthode et la méthode de Clauss et Loechner. Dans la partie expérimentale, nous exposons

l'intérêt de notre travail par des comparaisons, en terme de temps de calcul et les input, entre notre méthode et la méthode de Clauss et Loechner (appliquée sur le domaine de travail considéré). On termine ce chapitre par un exemple d'utilisation de notre résultat dans un calcul parallèle.

4.2 Proposition

Si on choisit un sous ensemble de polytope, on peut extraire des contraintes spécifiques de ce sous ensemble, et utiliser ces particularités pour développer des algorithmes permettant l'amélioration de temps de calcul, la taille des données en entrée (input) et/ou la taille de données en sortie (output).

Nous proposons de choisir un sous ensemble des polytopes et tirer ses propriétés mathématiques. Nous les utilisons pour développer des algorithmes de calcul de points entiers dans ce sous ensemble des polytopes.

Nos algorithmes proposés ici, sont basés sur la méthode d'interpolation de Clauss et Loechner avec des modifications expliqués après.

4.3 Domaine de travail

Déterminer le nombre de points entiers dans un polytope a une importance majeure dans plusieurs domaines. Ce nombre peut être calculé par le polynôme d'Ehrhart. Cependant ce calcul dans le cas général prends beaucoup de temps.

On s'intéresse dans ce mémoire à un sous ensemble des polytopes paramétrés avec un seul paramètre, de la forme suivante :

$$P_n = \left\{ x \in (\mathbb{R}_+)^d \ / \ \sum_{i=1}^{d} a_i x_i = n \ , \quad a_i \in \mathbb{Z}_+, \ d, n \in \mathbb{N} \right\}$$

Le nombre de points entiers dans ce polytope est un quasi-polynôme (donné par un polynôme d'Ehrhart).

4.3.1 Intérêt du polytope

Les points entiers de ce polytope correspondent aux itérations de programme (nid de boucles) suivant :

$$\text{for } (x_1 = 0 \; ; x_1 \leq \frac{n}{a_1} \; ; x_1 + +)$$
$$\text{for } (x_2 = 0 \; ; x_2 \leq \frac{n - a_1 x_1}{a_2} \; ; x_2 + +)$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$\text{for } (x_d = 0 \; ; x_d \leq \frac{n - a_1 x_1 - ... - a_{d-1} x_{d-1}}{a_d} \; ; x_d + +)$$
Instructions....

4.4 Démarche à suivre

Notre proposition se base sur la méthode d'interpolation de Clauss et Loechner. Cependant, la démarche utilisée simplifie ce calcul et améliore son temps d'exécution. Pour cela on divise le problème en plusieurs parties, puis on les rassemble.

Nous proposons ici des algorithmes pour le dénombrement des points de polytope décrit dans la section précédente. Pour cela, nous utilisons le principe de la méthode d'interpolation de Clauss et Loechner pour le calcul des polynômes d'Ehrhart, avec les modifications citées dans le tableau suivant :

Méthode d'interpolation de Clauss et Loechner	**Notre méthode**
- Calcul des sommets paramétrés en appliquant l'algorithme de Wilde et Lochner. - Calcul de domaines de validité du polytope et la liste des sommets qui sont valides sur chaque domaine. - Pour chaque domaine de validité , calcul de quasi-polynôme d'Ehrhart correspondant, d'où : * Le calcul se fait en résolvant des systèmes d'équations linéaires symboliques. * Les valeurs initiales numériques nécessaires à la résolution de ces systèmes s'obtiennent par la détermination préalable de boucles de parcours du polytope, en utilisant la méthode de *Fourier-Motzkin*	- Les sommets sont donnés par une formule mathématique sans appliquer l'algorithme de Wilde et Lochner. - Le domaine de validité est connu, et on n'a pas besoin de le calculer. - Calcul de quasi-polynôme d'Ehrhart pour l'unique domaine de validité. * Calcul initial sur des polytopes non paramétrés en utilisant un algorithme par *balayage.*

4.5 Différentes étapes de notre travail

Pour simplifier la démarche proposée, nous subdivisons note travail en trois étapes :

1- Calcul sur le polytope défini par :

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ / \sum_{i=1}^{d} x_i = p \right\}$$

2 - Calcul sur le polytope défini par :

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ / \sum_{i=1}^{d} a_i x_i = p \ \times \ ppcm(a_i) \ , \ \ a_i \in \mathbb{Z}_+, \ d \in \mathbb{N} \right\}$$

3 - Calcul sur le polytope défini par :

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ / \sum_{i=1}^{d} a_i x_i = p \ \times \ ppcm(a_i) + R \ , \quad a_i \in \mathbb{Z}_+, \ d \in \mathbb{N}, \ 0 \leq R < ppcm(a_i) \right\}$$

Pour chaque calcul, nous proposons un ou plusieurs algorithmes. Le réassemblage des résultats des trois parties permet de donner un résultat général concernant le calcul de nombre de points entiers sur le polytope initial.

4.5.1 Etape 1

Cette étape consiste à développer un algorithme pour le polytope le plus simple définie par

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ / \sum_{i=1}^{d} x_i = p \right\}$$

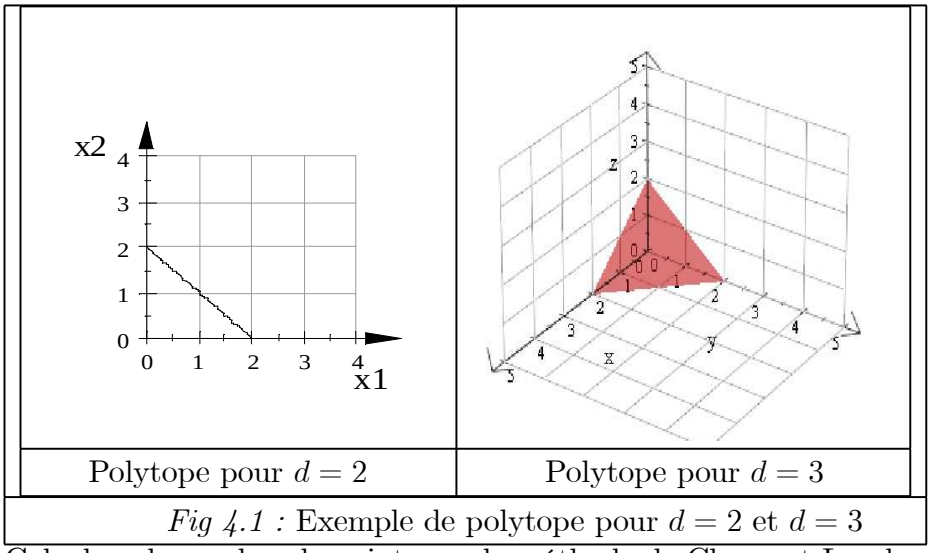
Le nombre de points entiers de ce polytope est calculé par la théorème suivante :

Théorème 4.5.1 *Le nombre des points entiers contenu dans ce polytope est donné par*

$$Card\left(P_p \cap \mathbb{Z}^d\right) = C_{p+d-1}^{d-1}$$

Preuve. Soient les polytopes pour $d = 2$ et $d = 3$:

$$\begin{aligned} P_p &= \left\{ x \in (\mathbb{R}_+)^2 \ / \ x_1 + x_2 = p \right\} \\ P_p &= \left\{ x \in (\mathbb{R}_+)^3 \ / \ x_1 + x_2 + x_3 = p \right\} \end{aligned}$$



Polytope pour $d = 2$	Polytope pour $d = 3$

Fig 4.1 : Exemple de polytope pour $d = 2$ et $d = 3$

Calculons le nombre de points par la méthode de Clauss et Loechner, on trouve :

▶ Pour $d = 2$:
Le nombre de points entiers est un polynôme d'Eharhart de degré 1

$$P(p) = c_1 p + c_0$$

On trouve que

$$\begin{cases} c_0 = 1 & \text{pour } p = 0 \\ c_1 + c_0 = 2 & \text{pour } p = 1 \end{cases}$$

Alors

$$P(p) = p + 1 = C_{p+1}^1$$

▶ Pour $d = 3$:
Le nombre de points entiers est un polynôme d'Ehrhart de degré 2

$$P(p) = c_2 p^2 + c_1 p + c_0$$

Le système à résoudre :

$$\begin{cases} c_0 = 1 & \text{pour } p = 0 \\ c_2 + c_1 + c_0 = 3 & \text{pour } p = 1 \\ 4c_2 + 2c_1 + c_0 = 6 & \text{pour } p = 2 \end{cases}$$

On trouve

$$c_0 = 1 \ , \ c_1 = \frac{3}{2} \ , \quad c_2 = \frac{1}{2}$$

et le polynôme est

$$\begin{aligned} P(p) &= \frac{1}{2}p^2 + \frac{3}{2}p + 1 \\ &= \frac{p^2 + 3p + 2}{2} \\ &= \frac{(p+2)(p+1)}{2} \\ &= \frac{(p+2)!}{2} \\ &= C_{p+2}^2 \end{aligned}$$

Généralisation :
On suppose que la propriété est juste pour d, et on la démontre pour $d+1$, c-à-d

$$\forall p, \sum_{j=1}^{d+1} x_j = p \ , \quad \#P_p \stackrel{?}{=} \ C_{p+d}^{d}$$

Pour $x \in P_p$, x_j peut prendre une valeur $(0, ..., p-j, ..., p)$. Posons $x_{d+1} = p - j$ donc

$$\sum_{i=1}^{d} x_i + p - j = p$$

d'où

$$\sum_{i=1}^{d} x_i = j$$

et

$$\#P_j = \ C_{j+d-1}^{d-1}$$

alors

$$\#P_p = \sum_{j=0}^{p} C_{j+d-1}^{d-1} = \sum_{j=0}^{p} C_{j+d-1}^{j}$$

D'autre part on a

$$\begin{aligned}
C_{p+d}^{d} &= \frac{(p+d)!}{p!\ d!} = \frac{(p+d-1)!\ (p+d)}{p!\ d!} \\
&= \frac{(p+d-1)!\ p}{p!\ d!} + \frac{(p+d-1)!\ d}{p!\ d!} \\
&= \frac{(p+d-1)!}{(p-1)!\ d!} + \frac{(p+d-1)!}{p!(d-1)!} \\
&= \frac{(p+d-2)!\ (p+d-1)}{(p-1)!\ d!} + \frac{(p+d-1)!}{p!(d-1)!} \\
&= \frac{(p+d-2)!\ (p-1)}{(p-1)!\ d!} + \frac{(p+d-2)!\ d}{(p-1)!\ d!} + \frac{(p+d-1)!}{p!(d-1)!} \\
&= \frac{(p+d-2)!}{(p-2)!\ d!} + \frac{(p+d-2)!}{(p-1)!\ (d-1)!} + \frac{(p+d-1)!}{p!(d-1)!} \\
&\quad ... \\
&= \frac{0!}{0!\ (d-1)!} + ... + \frac{(p+d-2)!}{(p-1)!\ (d-1)!} + \frac{(p+d-1)!}{p!(d-1)!} \\
&= \sum_{j=0}^{p} C_{j+d-1}^{j}
\end{aligned}$$

donc

$$\#P_p = \sum_{j=0}^{p} C_{j+d-1}^{j}$$

alors

$$\forall p, \sum_{j=1}^{d+1} x_j = p \ , \quad \#P_p = \ C_{p+d}^{d}$$

■

Nous nottons l'algorithme qui calcule ce nombre de points entiers par "Algorithme Calcul_N_p1".

Algorithme Calcul_N_p1

- Lire : Lecture des a_i
- Polynôme : Donner le polynôme C_{p+d}^{d}

4.5.2 Etape 2

Dans cette étape, nous calculons le nombre de points entiers pour le polytope défini par

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ \ / \sum_{i=1}^{d} a_i x_i = p \ \times \ ppcm(a_i) \ , \ \ a_i \in \mathbb{Z}_+, \ d \in \mathbb{N} \right\}$$

Les sommets de ce polytope est donné par :

$$S_i(0, 0, ..., \frac{p \ \times \ ppcm(a_i)}{a_i}, 0, ..., 0)$$

Dans ce qui suit, nous prouvons que ce polytope est l'enveloppe convexe de ses sommets.

Preuve. En effet, il faut démontrer l'équivalence

$$\begin{aligned} (x \ \ &\in \ \ P_p : a_1x_1 + a_2x_2 + ... + a_dx_d = p \ \times \ ppcm(a_i)) \ \overset{?}{\Longleftrightarrow} \\ (\exists \lambda_i \ &\geq \ 0, \sum_{i=1}^{d} \lambda_i = 1, \ x = \lambda_1 S_1 + \lambda_2 S_2 + ... + \lambda_d S_d) \end{aligned}$$

*)Commençons par l'implication :

$$\begin{aligned} (x \ \ &\in \ \ P_p : a_1x_1 + a_2x_2 + ... + a_dx_d = p \ \times \ ppcm(a_i)) \ \overset{?}{\Longrightarrow} \\ (\exists \lambda_i \ &\geq \ 0, \sum_{i=1}^{d} \lambda_i = 1, \ x = \lambda_1 S_1 + \lambda_2 S_2 + ... + \lambda_d S_d) \end{aligned}$$

Il suffit de prendre

$$\lambda_i = \frac{x_i \times a_i}{p \times \ ppcm(ai)}$$

d'où

$$\sum_{i=1}^{d} \lambda_i = \sum_{i=1}^{d} \frac{x_i \times a_i}{p \times \ ppcm(ai)} = \frac{1}{p \times \ ppcm(ai)} \sum_{i=1}^{d} x_i \times a_i = \frac{p \times \ ppcm(ai)}{p \times \ ppcm(ai)} = 1$$

**)Pour la deuxième implication :

$$[\exists \lambda_i \ \geq \ 0, \sum_{i=1}^{d} \lambda_i = 1, \ x = \lambda_1 S_1 + \lambda_2 S_2 + ... + \lambda_d S_d] \stackrel{?}{\Longrightarrow}$$
$$[x \in \ P_p : a_1 x_1 + a_2 x_2 + ... + a_d x_d = p \ \times \ ppcm(a_i)]$$

on a

$$x \quad = \quad \lambda_1 S_1 + \lambda_2 S_2 + ... + \lambda_d S_d \Longrightarrow \begin{cases} x_1 = \frac{\lambda_1 \ \times \ p \ \times \ ppcm(a_i)}{a_i} \\ . \\ . \\ . \\ x_d = \frac{\lambda_d \ \times \ p \ \times \ ppcm(a_i)}{a_d} \end{cases}$$

$$\Longrightarrow \begin{cases} x_1 \ \times \ a_i = \lambda_1 \ \times \ p \ \times \ ppcm(a_i) \\ . \\ . \\ . \\ x_d \ \times \ a_d = \lambda_d \ \times \ p \ \times \ ppcm(a_i) \end{cases}$$

alors

$$\sum_{i=1}^{d} ai \ \times \ xi = \sum_{i=1}^{d} \lambda_i \ \times \ p \ \times \ ppcm(a_i) = p \ \times \ ppcm(a_i)$$

■

Alors, ce polytope est l'enveloppe convexe de ses sommets

$$S_i(0, 0, ..., \frac{p \ \times \ ppcm(a_i)}{a_i}, 0, ..., 0)$$

Les sommets sont tous entiers (coefficients de p sont des entiers), alors le nombre de points entiers dans ce polytope est un polynôme de degré $d - 1$.

Les coefficients de ce polynôme sont donnés par l'algorithme suivant :

Algorithme Calcul_N_p2

- Lire : Lecture des a_i;

- Ppcm_p : Calcul du plus petit commun multiple des a_i;

- N_point : calcul initial de nombre de points ;

- System_equ : construction de système d'équations des coefficients de polynôme d'Ehrhart ;

- Gauss : résolution de système d'équations des coefficients. Le résultat est le polynôme cherché.

La procédure *N_point* permet de calculer les nombres de points entiers initiaux nécessaires au calcul des coefficients de polynôme est un algorithme de *balayage*, le principe est le suivant :

- Procédure N_point

$$\max x_1 = \frac{ppcm(a_i)\ p}{a_1}$$

pour $x_1 = 0$ *jusqu'à* $\max x_1$

$$\max x_2 = \frac{ppcm(a_i)\ p - a_1 x_1}{a_2}$$

pour $x_2 = 0$ *jusqu'à* $\max x_2$

.

.

.

$$\max x_d = \frac{ppcm(a_i)\ p - a_1 x_1 - ... - a_{d-1} x_{d-1}}{a_d}$$

pour $x_d = 0$ *jusqu'à* $\max x_d$

Instruction...

Lors de l'implmentation, nous avons utilisé une version récursive de cette procédure.

4.5.3 Étape 3

Dans cette étape, nous calculons le nombre de points entiers pour le polytope défini par :

$$P_p = \left\{ x \in (\mathbb{R}_+)^d \ / \sum_{i=1}^{d} a_i x_i = p \ \times \ ppcm(a_i) + R \ , \quad a_i \in \mathbb{Z}_+, \ d \in \mathbb{N}, \ 0 \leq R < ppcm(a_i) \right\}$$

Les sommets de ce polytope sont donnés par :

$$S_i(0, 0, ..., \frac{p \quad \times \quad ppcm(a_i) + R}{a_i}, 0, ..., 0)$$

Les sommets ne sont pas entiers, alors le nombre de points entiers de ce polytope est un quasi-polynôme d'Ehrhart avec les paramètres p et R.

Pour trouver le nombre de points entiers de ce polytope, nous avons procédé selon deux aspects :

- Dévelopement un algorithme qui trouve le quasi-polynôme d'Ehrhart en fonction de p et R,

- Dévelopement des algorithmes qui donnent le nombre de points entiers en donant des valeurs à p et R. L'intérêt de ces algorithmes dans ce cas, est de ne pas perdre le temps à calculer le quasi-polynôme d'Ehrhart (paramété) si nous voulons trouver le nombre de points entiers pour des valeurs données de p et R.

Algorithme calculant le quasi-polynôme d'Ehrhart

Dans cette section, nous développons un algorithme permet de donner le polynôme d'Ehrhart en p et R (les paramètres p et R sont les inconnus).

Le nombre de points entiers dans ce polytope est un quasi-polynôme d'Ehrhart de degré $d-1$ et de période de $ppcm(a_i)$. Les grandes lignes de cet algorithme sont données par :

Algorithme Calcul_N_p3

- Lire : Lecture des a_i;
- Ppcm_p : Calcul le plus petit commun multiple des a_i;
- N_point : calcul initial de nombre de points ;
- System_equ : construction de système d'équations des coefficients de polynôme d'Ehrhart, pour $0 \leq R < ppcm(a_i)$;
- Gauss : résolution du système d'équations en donnant le polynôme d'Ehrhart ;

La procédure *N_point* permet de calculer les nombres de points entiers initiaux nécessaires au calcul des coefficients de polynôme d'Ehrhart. Cette procédure est un algorithme de *balayage*, le principe est le suivant :

- Procédure N_point

$$\max x_1 = \frac{ppcm(a_i)\ p + R}{a_1}$$
$$pour \quad x_1 = 0 \quad jusqu'à \quad \max x_1$$

$$\max x_2 = \frac{ppcm(a_i)\ p + R - a_1 x_1}{a_2}$$
$$pour \quad x_2 = 0 \quad jusqu'à \quad \max x_2$$

.

.

.

$$\max x_d = \frac{ppcm(a_i)\ p + R - a_1x_1 - ... - a_{d-1}x_{d-1}}{a_d}$$
$$pour \quad x_d = 0 \quad jusqu'à \quad \max x_d$$
$$Instruction...$$

Lors l'implémentation, nous avons utilisé une version récursive de cette procédure.

Algorithmes en fonction des valeurs des p et R

Dans cette section, nous développons des algorithmes pour le calcul de nombre de points entiers de polytope définie précédemment, en donnant des valeurs pour p et R (ou de $n = ppcm\ (a_i).p + R$). Le résultat n'est pas un polynôme d'Ehrhart, mais un chiffre qui exprime le nombre de points entiers dans ce polytope pour telles valeurs de p et R. Les deux algorithmes développés sont :

- Version simplifiée de l'algorithme précédent

C'est une version simplifiée du programme précédent en fonction de p et R, c'est à dire le résultat du programme n'est pas un quasi-polynôme d'Ehrhart, mais la valeur de ce polynôme pour des valeurs de p et R données.

- Algorithme en utilisant l'interpolation de Lagrange

Ce ci est un algorithme qui donne la valeur de polynôme pour p et R donnés en utilisant l'interpolation de Lagrange donné par :

$$P(x) = \sum_{i=0}^{d} \left(f(x_i) \prod_{j=0,\ j\neq i}^{d} \left(\frac{x - x_j}{x_j - x_i} \right) \right)$$

Où $x_0, x_1, \ldots, x_d \in [a, b]$, et f est une fonction définie sur $[a, b]$ à valeurs dans $\mathbb{R}$, tel que $P(x_i) = f(x_i)$ pour $i = 0, 1, \ldots, d$.

Dans notre algorithme, nous travaillons sur des entiers q. Alors, nous utilisons une formule simplifiée, donnée par :

$$P(q) = \sum_{i=0}^{d} B_i\ C_n^{d+1} C_{d+1}^{i} \frac{d + 1 - i}{(-1)^{d-i}\ \ (q - i)}$$

Où B_i sont les valeurs des calculs initiaux du polytope (trouvé par la procédure *N_points*).

4.6 Complexité

Après l'étude de notre algorithme pour le calcul de polynôme d'Ehrhart basé sur la méthode d'interpolation de Clauss et Loechner, nous trouvons que la complexité de notre algorithme est :

- En temps d'exécution : $0(d^2 \times n^{d+1} \times (d-1) \times \log(n) \times \log(Max\,\{a_i\}))$

- En espace : $0(Max\{d^2 \times log(Max\,\{a_i\}\ , (d+1)log(n) + log(Max\,\{a_i\})\})$

Notre étude montre que l'algorithme est LINSPACE (en espace linéaire), mais la complexité théorique en temps est exponentielle. On voit que la plus grande partie (de loin) du temps est consommé dans le calcul initial du nombre de points N_point. Cependant, la taille de l'espace étant faible. Cet algorithme est fiable pour des grands nombres sans adaptation.

On peut trouver des explication sur la complexité de notre algorithme ajoutées en annexe.

4.7 Expérimentation

Dans cette partie, nous évaluons nos algorithmes expérimentalement en faisant les comparaisons suivantes :

- Comparaison de notre algorithme donnant le quasi-polynôme d'Ehrhart avec l'implémentation de la méthode d'interpolation dans la bibliothèque de Polylib et l'implémentation de la méthode de Barvinok paramétré dans la bibliothèque de Barvinok, en terme de la taille des données en entré.

- Comparaison de notre algorithme donnant le quasi-polynôme d'Ehrhart avec l'implémentation de la méthode d'interpolation dans la bibliothèque de Polylib, en terme de temps d'exécution.

Nous donnons aussi des exemples des cas incalculables ou prennent un temps insupportable en utilisant la méthode d'interpolation.

- Comparaison des deux algorithmes qui donnent directement le nombre de points entiers : la version simplifiée de l'algorithme de calcul de polynôme d'Ehrhart et l'algorithme utilisant l'interpolation de Lagrange.

4.7.1 Comparaison entre la présentation de données en input de notre algorithme donnant le polynôme d'Ehrhart et la présentation des données de Polylib et Barvinok

La façon de la représentation des données influe sur la taille de mémoire nécessaire pour le stockage de données. Nous comparons ici entre la présentation de données en entré de notre algorithme avec la présentation de données utilisée dans les bibliothèques *Polylib* et *Barvinok*.

Les deux bibliothèques *Polylib* et *Barvinok* utilisent la même présentation de données en entré :

- Chaque contrainte (équation ou inéquation) est présentée par "nombre de variables +2" éléments et qui a la forme suivante $(S, X_1, X_2, .., C)$, où :

- C est la constante d'équation ou d'inéquation ;

- S égale à zéro, s'il s'agit d'une équation et égale à un s'il s'agit d'une inéquation comme c'est indiqué :

* si $S = 0 : X_1 i + X_2 j + .. + X_n l + C = 0$

* si $S = 1 : X_1 i + X_2 j + ... + X_n l + C \geq 0$

Nous prenons l'exemple suivant pour illustrer comment utiliser ces algorithmes ainsi que leurs présentations des données :

Exemple 4.7.1 *Prenons le polytope suivant :*

$$P(p, r) = \{2x_1 + 3x_2 = 6p + r, 0 \leq r < 6\}$$

En utilisant l'implémentation de la méthode de l'interpolation de Clauss et Loechner dans la bibliothèque de polylib, ce polytope est présenté par une matrice de données et une matrice des paramètres, d'où

$$\begin{aligned}
&P(p, r) = \{2x_1 + 3x_2 = 6p + r, 0 \leq r < 6\} \Rightarrow \\
&P(p, r) = \{ \\
&\qquad 2x_1 + 3x_2 - 6p - r + 0.c = 0, \\
&\qquad x_1 \geq 0 \\
&\qquad x_2 \geq 0 \\
&\qquad P \geq 0 \\
&\qquad r \geq 0 \\
&\qquad r \leq 5 \\
&\qquad \} \quad \Rightarrow \\
&P(p, r) = \{ \\
&\qquad 2x_1 + 3x_2 - 6p - r + 0.c = 0, \\
&\qquad x_1 \geq 0 \\
&\qquad x_2 \geq 0 \\
&\qquad P \geq 0 \\
&\qquad r \geq 0 \\
&\qquad -r + 5 \geq 0 \\
&\qquad \}
\end{aligned}$$

La matrice des données est :

Equation/ inéquation	*coeff de x_1*	*coeff de x_2*	*coeff de P*	*coeff de R*	*coeff de C*	*signification*
0	*2*	*3*	*-6*	*-1*	*0*	$2x_1 + 3x_2 - 6p - r + 0.c = 0$
1	*1*	*0*	*0*	*0*	*0*	$x_1 \geq 0$
1	*0*	*1*	*0*	*0*	*0*	$x_2 \geq 0$

La matrice des paramètres est :

équation/inéquation	*coeff de P*	*coeff de R*	*coeff de C*	*signification*
1	*1*	*0*	*0*	$P \geq 0$
1	*0*	*1*	*0*	$r \geq 0$
1	*0*	*-1*	*5*	$-r + 5 \geq 0\}$

Le tableau suivant présente l'exécution de deux algorithmes pour calculer le polynôme d'Ehrhart pour ce polytope. Il illustre la différence entre les présentations des algorithmes :

Méthode de Clauss et Loechner (Claus et Barvinok paramétré utilisent la même présentation des entrés)		***Notre méthode de calcul de polynôme d'Ehrhart***
Entrées :		*Entrées :*
La matrice des données		*Le nombre de coefficients :*
3 6	*// nombre de // lignes et // colonnes // de la matrice // des données*	2 *Les ai* 2 3
0 2 3 -6 -1 0 *1 1 0 0 0 0* *1 0 1 0 0 0*	*// $2x_1 + 3x_2 - 6p - r + 0.c = 0$* *// $x_1 \geq 0$* *// $x_2 \geq 0$*	
La matrice des paramètres		
3 4	*// nombre de // lignes et // colonnes // de la matrice // des paramètres*	
1 1 0 0 *1 0 1 0* *1 0 -1 5*	*// $P \geq 0$* *// $r \geq 0$* *// $-r + 5 \geq 0\}$*	
Sorties : *Le polynôme d'Ehrhart*		*Sorties :* *Le polynôme d'Ehrhart*
$P + (\frac{1}{6}R + [1, \frac{-1}{6}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{6}]_{\_R})$		$P + [1, 0, 1, 1, 1, 1]_{\_R}$

Nous voyons bien que dans le cas des bibliothèques de Polylib et Barvinok, nous avons besoin d'espace mémoire plus important.

4.7.2 Comparaison de notre algorithme donnant le quasi-polynôme d'Ehrhart avec l'implémentation de la méthode d'interpolation dans la bibliothèque de Polylib en terme de temps d'excution

Le tableau suivant présente les résultats d'exécution de notre algorithme (qui donne le quasi-polynôme d'Ehrhart) et l'algorithme de l'interpolation de Clauss et Loechner. Les expériences ont été faites pour une dimension $d = 2$ et $d = 3$. Ces expériences ont été réalisées sur une machine Dell Latitude E6500, Intel (R) Core (TM) 2 Duo CPU, 2.39 GHz, 3.48 Go de RAM

Dimension	NB Chiffres de ai	Polytope			Temps notre méthode (secondes)	Temps méthode Clauss (secondes)
		a1	a2	PPCM		
2	1	2	3	6	0.000	0.0000
		4	5	20	0.0000	0.0000
		6	4	12	0.0000	0.0000
		9	8	72	0.0000	0.03100
		3	6	6	0.0000	0.01600
	2	11	12	132	0.0000	0.01500
		13	16	208	0.01600	0.06300
		11	33	33	0.0000	0.0000
		45	30	90	0.0000	0.1600
		80	70	560	0.03100	0.11000
	3	111	333	333	0.01500	0.04700
		112	400	2800	0.03100	0.42200
		520	600	7800	0.28100	0.79700
		700	820	28700	0.98500	4.73500
		900	225	900	0.03100	0.12500
	4	1111	5555	5555	0.18800	0.42200
		3100	6200	6200	0.2030	0.45300
		4000	8000	8000	1.42200	6.40600
		6000	3000	3000	0.18800	4.07800
		9000	8200	369000	**37.92200**	**1578.48**

Tab 4.1 : Comparaison entre notre méthode et méthode d'interpolation de Clauss et Loechner

Dimension	NB Chiffres de ai	Polytope a1	a2	a3	PPCM	Temps notre méthode (secondes)	Temps méthode Clauss (secondes)
3	1	1	3	4	12	0.000	0.45300
		2	3	5	30	0.01000	2.50100
		6	5	4	60	0.01600	3.92200
		6	8	5	120	0.04700	18.6500
		9	6	7	126	0.04700	18;98500
	2	11	10	12	660	**0.62500**	**1258.70300**
		20	30	40	600	0.01600	12.39100
		60	50	30	300	0.01600	5.48400
		80	60	90	720	0.10900	41.37500
		96	90	80	1440	0.21900	175.31200
	3	111	222	444	444	0.11000	3.17200
		222	333	666	666	0.47000	3.87500
		500	300	600	3000	**0.21900**	**99.87500**
		800	600	900	7200	0.53100	411.40600
		800	840	640	67200	**91.42200**	**Plus de 2 h**
	4	2000	3000	6000	6000	**0.31200**	**Plus de 2h**

Suite de Tab 4.1 : Comparaison entre notre méthode et méthode d'interpolation de Clauss et Loechner

Les résultats montrent qu'il y a une différence importante entre les temps d'exécution de deux méthodes.

4.7.3 Comparaison de la version simplifiée de l'algorithme de calcul de polynôme d'Ehrhart et l'algorithme utilisant l'interpolation de Lagrange

Ici, nous faisons une comparaison entre ces deux algorithmes : la version simplifiée de l'algorithme de calcul de polynôme d'Ehrhart et l'algorithme utilisant l'interpolation de Lagrange en donnant des valeurs pour P et R. Les tests ont été faits en prenant quelques valeurs des a_i en variant d de 2 à 5.

Dimension	Nbr chiffres a1	a1	a2	a3	a4	a5	P	R	Temps d'exécution de la version simplifiée de l'algorithme calculant le polynôme d'Ehrhart	Temps d'exécution de l'algorithme utilise d'interpolation de Lagrange
2	5	90000	80000				3	4	0.09000	0.13000
3	3	800	840	640			3	4	0.01000	0.01000
3	5	80000	60000	90000			3	4	0.05000	0.09000
4	5	99999	88888	77777	66666		3	4	0.58000	0.58000
5	5	99999	88888	77777	66666	11111	3	4	17.79000	17.76000
5	5	99999	88888	77777	66666	4444	3	4	17.76000	17.74000
2	5	90000	80000				20	30	0.54000	0.13000
3	3	800	840	640			30	10	0.01000	0.01000
3	5	80000	60000	90000			15	45	0.12000	0.09000
4	5	99999	88888	77777	66666		25	10	0.44000	0.57000
5	5	99999	88888	77777	66666	33333	17	20	17.69000	17.77000

Tab.4.2 : Comparaison des deux algorithmes en donnant des valeurs à p et R

Les résultats obtenus montrent qu'il n'y a pas une grande différence entre les deux méthodes. Le temps de calcul est prèsque le même pour les deux algorithmes.

4.7.4 Cas incalculables avec la méthode d'interpolation de Clauss et Loechner

Nous trouvons que la méthode d'interpolation ne donne pas des résultats dans certains cas, comme les exemples montrés dans le tableau suivant :.

Polytope	**méthode d'interpolation**	**notre algorithme**
$P(p,r) = \{8x_1 + 7x_2 + 9x_3 = 504p + R\}$	pas de réponse	après 1.31 secondes
$P(p,r) = \{12x_1 + 13x_2 + 11x_3 = 1092P + R\}$	pas de réponse	après 16.15 secondes
$P(p,r) = \{18x_1 + 18x_2 + 19x_3 = 342p + R\}$	pas de réponse	après 0.203 secondes

Tab.4.3 : Cas incalculables avec laméthode d'interpolation de Clauss et Loechner

4.8 Exemple d'utilisation de notre algorithme dans un calcul parallèle

Soit le nid de boucle suivant :

pour x_1 *de* 0 *à* p
  pour x_2 *de* 0 *à* $p - x_1$
    pour x_3= *de* 0 *à* $p - x_1 - x_2$
      $S = S + A[x_1, x_2, x_3]$
    fin pour
  fin pour
fin pour

Pour paralléliser ce nid de boucle, on le modéliser par un polytope où chaque pas de calcul est représenté par un point de coordonnées entieres.

$$\begin{cases} 0 \leq x_1 \leq p \\ 0 \leq x_2 \leq p - x_1 \\ 0 \leq x_3 \leq p - x_1 - x_2 \end{cases} \Leftrightarrow \begin{cases} 0 \leq x_1 \\ 0 \leq x_2 \\ 0 \leq x_3 \\ x_1 + x_2 + x_3 = p \end{cases}$$

Le nombre de points entiers dans ce polytope est donné par le plynôme d'Ehrhart en appliquant notre algorithme :

$P(p) = C_{p+d-1}^{d-1} = C_{p+3-1}^{3-1} = C_{p+2}^{2} = \frac{(p+2)(p+1)}{2}$.

Ce nombre présente le nombre total des instructions éxecutées, et on peut l'utiliser pour trouver d'autres informations.

Par exemple :

- Si on propose une fonction d'allocation $alloc(x_1, x_2, x_3) = x_1$, alors le nombre de processeurs est $p + 1$ $(0 \leq x_1 \leq p)$. Pour faire l'équilibrage de la charge, le nombre d'instructions par processeur est donné par :

$$\text{Nombre d'instructions par processeur} = \frac{Nombre\ total\ des\ instructions}{nombre\ de\ processeurs}$$

$$\text{Nombre d'instructions par processeur} = \frac{\frac{(p+2)(p+1)}{2}}{p+1} = \frac{(p+2)}{2}$$

- Nous pouvons calculer aussi le temps d'exécution du programme en connaissant le temps d'éxécution d'une instruction.

Remarque

Si on prend par exemple d'autres nids de boucles telque

$$
\begin{array}{l}
pour \;\; x_1 \; de \; 0 \; à \; \frac{6000\ p+R}{2000} \\
\quad pour \; x_2 \; de \; 0 \; à \; \frac{6000\ p+R-2000x_1}{3000} \\
\quad\quad pour \; x_3 = \; de \; 0 \; à \; \frac{6000\ p+R+2000x_1-3000x_2}{6000} \\
\quad\quad\quad S = S + A[x_1, x_2, x_3] \\
\quad\quad fin \; pour \\
\quad fin \; pour \\
fin \; pour
\end{array}
$$

En appliquant la même démarche que celle de l'exemple précédent, et d'après le tableau de comparaison entre le temps d'exécution de notre algorithme et l'implémentation de *Polylib*, nous trouvons le nombre d'instructions exécutées après 0.312 *secoondes* en utilisant notre algorithme et après plus de 2 *heures* en utilisant *Polylib*.

4.9 Conclusion

Le travail réalisé ici, même s'il est pour un ensemble de polytope restreint, il donne des résultats intéressants en terme de temps de calcul et la taille des données en entrée. L'idée était de partitionner le problème général de calcul du nombre de points dans un polytope en sous problèmes afin de trouver des résultats améliorés que de calculer ce nombre de points dans le cas général.

Conclusion Générale et Perspactives

Dans ce mémoire, et dans le contexte de dénombrement de points entiers dans un polytope qui est utilisé dans plusieurs stades, surtout dans la parallélisation automatique, on a choisi à travailler sur un sous ensemble de polytope. La partie la plus importante de notre travail, consiste en le développement d'un algorithme qui permet - expérimentalement - d'améliorer le temps d'exécution par rapport à la méthode d'interpolation de Clauss et Loechner implémentée dans la bibliothèque Polylib, et améliorer les input par rapport à la méthode d'interpolation de Clauss et Loechner et de Barvinok paramétré. On a trouvé aussi, que dans ce sous-ensemble de polytope, l'implémentation de la méthode d'interpolation de Clauss et Loechner ne donne pas des résultats pour des cas comme illustré dans le $4^{ième}$ chapitre.

Malgré la restriction de contexte choisi, le résultat trouvé nous encouragent à développer des algorithmes pour des sous-ensembles de polytopes afin d'améliorer un côté de calcul (temps d'exécution, input...) par rapport à une méthode existe en bénéficiant des contraintes mathématiques particuliers de ces ensembles des polytopes.

Comme perspectifs, nous proposons d'utiliser des algorithmes adaptatifs (un algorithme adaptatif est un algorithme qui s'adapte aux ressources disponibles pendant l'exécution du programme parallèle, par exemple, partage la charge de calcul entre les processeurs libres ou moins chargés pendant l'exécution du programme parallèle) de façon à améliorer le temps de calcul, pour :

- Développer des versions parallèles de notre algorithme.

- Ainsi, développer des versions parallèles des bibliothèques de dénombrement des points entiers tel que Polylib et Barvinok en utilisant des algorithmes adaptatifs.

Noter que le développement des algorithmes adaptatifs est une approche intéressante de parallélisation qui est utilisée surtout à l'arrivée des processeurs multicores.

Bibliographie

[1] V. Baldoni, N.Berline and M. Vergne, Sum over lattice points of a polygon with iterated Laurent series. User's guide, March 2008.

[2] V. Baldoni, N. Berline , & M. Vergne, Summing a Polynomial Function Over Integral Points Of a Polygone. User's Guide, disponible à http ://www.math.polytechnique.fr/~berline/maple.html, Mai 2009.

[3] C. Bartzis ,T. Bultan, Efficient symbolic representations for arithmetic constraints in verification. Int J Found Comput Sci 14(4) :605-624, 2003.

[4] A. Barvinok, A Polynomial Time Algorithm for Counting Integral Points in Polyhedra when the Dimension Is Fixed, 0272-542W93 IEEE, 1993.

[5] A. Barvinok, Lattice Points, Polyhedra, and Complexity, IAS/Park City Mathematics Series,Volume, Department of Mathematics, University of Michigan, Ann Arbor, MI 48109-1109, 2004.

[6] A. Barvinok, Computing the Ehrhart Quasi-Polynomial Of a Rational Simplex, Mathematics Of Computation, Volume 75, Number 255, July 2006, Pages 1449–1466, S 0025-5718(06)01836-9. Article electronically published on March 10, 2006.

[7] A. Barvinok, Lattice Points, Polyhedra, and Complexity, Geometric Combinatorics, IAS/Park City Mathematics Series, 19-62, 2007.

[8] M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, A Compiler Framework for Optimization of Affine Loop Nests for General Purpose Computations on GPUs, in Proc. 22nd ACM International Conference on Supercomputing, Island of Kos, Greece, June 2008.

[9] M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan, Automatic Data Movement and Computation Mapping for Multi-level Parallel Architectures with Explicitly Managed Memories, in Proc. 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, (PPoPP 2008)„ February 2008.

[10] C. Bastoul, Code Generation in the Polyhedral Model Is Easier Than You Think, Proceedings of the 13th International Conference on Parallel Architectures and Com-

pilation Techniques, pages : 7 - 16, ISBN ~ISSN :1089-795X 0-7695-2229-7, IEEE Computer Society, 2004.

[11] M. Beck. The partial-fractions method for counting solutions to integral linear systems. Discrete Comp. Geom., 32 :437–446, 2004.

[12] B. Boigelot. Number-Set Representations for Infinite-State Verification. Proc. 1st NATO Advanced Research Workshop "Verification of Infinite State Systems with Applications to Security" (VISSAS 2005), NATO Security through Science Series D.

[13] U. Bondhugula, Effective Automatic Parallelization and Locality Optimization Using The Polyhedral Model, Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University, 2008. Information and Communication Security 1, IOS Press, pages 1-16, Timisoara, March 2005.

[14] U. Bondhugula, M. Baskaran1, S.Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model, L. Hendren (Ed.) : CC 2008, LNCS 4959, pp. 132–146. Springer-Verlag Berlin Heidelberg, 2008.

[15] P. Boulet, A. Darte, G-A. Silber & F. vivien, Loop Parallelization algorithms : from parallelism extraction to code generation, Parallel Computing, 24(3-4) :421–444, May 1998.

[16] Ph. Clauss, Sur les extensions et les utilisations en Informatique d'un résultat Mathématique : Les polynômes d'Ehrhart, article de vulgarisation, Bulletin de l'APMEP, numéros 418 et 419, septembre et octobre 1998.

[17] Ph. Clauss, Méthodes polyédriques pour la parallélisation et l'optimisation de programmes, Université Louis Pasteur, Habilitation à diriger des recherches, 2000.

[18] Ph. Clauss and V. Loechner, Parametric Analysis of Polyhedral Iteration Spaces. Journal of VLSI Signal Processing, Vol. 19, No. 2, p. 179-194, Kluwer Academic Pub, July 1998.

[19] F. Dupont de Dinechin, Systèmes structurés d'équations récurrentes : mise en oeuvre dans le langage Alpha et applications,Thèse de doctorat, Institut de Formation Supérieur en Informatique et Communication,Université de Rennes 1, 1997.

[20] P. Feautrier, Automatic Parallelization in the Polytope Model, In Guy-René Perrin and Alain Darte, editors, The Data-Parallel Programming Model, volume LNCS 1132, pages 79–103 , 1996.

[21] P. Feautrier, Parallélisation automatique Bases théoriques, slide de cours, ENS de Lyon, 2008.

[22] P. Feautrier, D. Trystram, Y. Slimani, and M. Jemni, Parallélisation automatique, editors, Informatique répartie, Hors serie TSI, chapter 6, pages 161-182. Hermes, 2005.

[23] A. Gröblinger, M. Griebl & C. Lengauer, Quantifier elimination in automatic loop parallelization, Journal of Symbolic Computation, 41(11) :1206–1221, November 2006.

[24] V. Loechner, D.K. Wilde, Parameterized Polyhedra and their Vertices, International Journal of Parallel Programming, volume 25(6), December 1997.

[25] B. Meister, Approximations of Polytope Enumerators using Linear Expansions, Technical report, Universite Louis Pasteur, May 2007. Revised edition of a paper submitted in 2005.

[26] B. Meister, Polynomial approximations in the polytope model : Bringing the power of quasi-polynomials to the masses, J. Sankaran, T. Vander Aa (eds.) Digest of the 6th Workshop on Optimization for DSP and Embedded Systems, ODES-6 (2008).

[27] C. Michaux, R. Villemaire, Presburger arithmetic and recognizability of sets of natural numbers by automata : New proofs of Cobham's and Semenov's theorems, Ann. Pure Appl. Logic 77 (3) (1996) 251-277.

[28] E. Parker, S. Chatterjee, An Automata-Theoretic Algorithm for Counting Solutions to Presburger Formulas, Compiler Construction 2004, volume 2985 of Lecture Notes in Computer Science, pages 104–119, Apr. 2004.

[29] S. Phani, K. Nookala and T. Risset, Alibrary for Z-polyhedral Operations, IRISA, Publication interne n°1330−Mai 2000-29 pages.

[30] W. Pugh, The Omega Test : a fast and practical integer programming algorithm for dependence analysis, In Proceedings of the 1991 ACM/IEEE conference on Supercomputing, pages 4–13. ACM Press, 1991.

[31] T. Rabl,Volume Calculation and Estimation of Parameterized Integer Polytopes, Diploma Thesis, Universität Passau Fakultät für Mathematik und Informatik,30. Januar 2006.

[32] E. Rijpkema, Modeling Task Level Parallelism in Piece-wise Regular Programs Proefschrift, PhD thesis, Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands, Sept. 2002.

[33] R. Seghir, Méthodes de Dénombrement de Points Entiers De Polyèdres et Applications a L'optimisation de Programmes,Thèse de doctorat, Université Louis Pasteur-Strasbourg, Spécialité : Informatique, Decembre 2006.

[34] R. Seghir, S. Verdoolaege, K. Beyls and V. Loechner, Analytical Computation of Ehrhart Polynomials and its Application in Compile-Time Generated Cache Hints, ICPS Research report, February 2004.

[35] S. Verdoolaege, Incremental Loop Transformations and Enumeration of Parametric Sets, K. U. LEUVEN, Departement Computerwetenschappen, Ph.D. thesis, ISBN : 90-5682-594-1, 2005.

[36] S. Verdoolaege, Barvinok : User Guide. Version : barvinok-0.30-41-g008f3f8,April 24, 2010. Peut trouvé à : http ://www.kotnet.org/~skimo/barvinok/barvinok.pdf.

[37] S. Verdoolaege, M. Bruynooghe, Algorithms for Weighted Counting over Parametric Polytopes : A Survey and a Practical Comparison, InM. Beck and T. Stoll, eds., The 2008 International Conference on Information Theory and Statistical Learning, July 2008.

[38] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner and M. Bruynooghe, Analytical Computation of Ehrhart Polynomials : Enabling more Compiler Analyses and Optimizations, Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, pages 248–258, Washington D.C., September 2004.

[39] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner & M. Bruynooghe, Counting Integer Points in Parametric Polytopes using Barvinok's Rational Functions, Algorithmica, volume 48(1), pages 37–66, Springer New York, May 2007.

[40] S. Verdoolaege, K. Woods , M. Bruynooghe ,R. Cools,Computation and Manipulation of Enumerators of Integer Projections of Parametric Polytopes, Department of Computer Science, K.U.Leuven, CW Reports vol : CW392 pages :104, Mar-2005.

[41] F.Vivien, Détection de parallélisme dans les boucles imbriquées, Thèse de doctorat de l'école normale supérieure de Lyon, Spécialité Informatique, No : 97 ENSL 0077, 1997.

[42] K. Wilde, A library for doing polyhedral operations, Rapports de recherche- INRIA, ISSN 0249-6399, N° : INRIA-RR - 2157, 1993.

[43] B. R. K. YOUTA, Vers un profiling adaptatif : Etat de l'art des méthodes actuelles d'analyse de programmes et perspectives, Master thesis, Université de Yaoundé I, Cameroun, November 2002.

[44] G. Zumbush, , Data Dependance Analysis for the Parallelization of Numerical Tree Codes, Lecture Notes In Computer Science, Numb 4699, pages 890-899, ISSN : 0302-9743, 2007.

Annexe

Complexité de l'Algorithme Calcul_N_p3

Résumé de l'algorithme Calcul_N_p3
- Lire : Lecture des a_i. - **Etape1** : *Ppcm_p* (calcul du plus petit commun multiple des a_i). - **Etape 2** : *N_point* (calcul initial du nombre de points). - **Etapes 3 et 4** : *System_equ* (construction de systèmes d'équations des coeffcients de polynôme d'Ehrhart, pour $0 \leq R < ppcm(a_i)$).

Complexité

Etape 1 : Ppcm_p (calcul du plus petit commun multiple des a_i) :
- Temps : $0(d \times log^3(Max\{a_i\})$.
- Espace : $0(d \times log(Max\{a, a'\}))$.

♦ Calcul du pgcd de deux entiers positifs a et a' par l'algorithme d'Euclide : temps $0(\log^3(Max\{a, a'\}))$,espace $0(log(Max\{a, a'\}))$.

preuve **:** l'algorithme d'Euclide utilise $0(log(Max\{a; a'\}))$ étapes comportant chacune une division euclidienne. Le calcul par récurrence ne nécessite pas de stockage d'une file.

♦ Calcul du ppcm de deux entiers positifs a et a' : temps $0(log^3(Max\{a, a'\})$, espace $0(log(Max\{a; a'\})$.

preuve : on utilise $ppcm(a, a') = \frac{a \times a'}{p\gcd(a,a')}$; l'opération utilise une multiplication et une division supplémentaire ; les variables utilisées sont majorées par $a \times a'$.

♦ Calcul du ppcm de d entiers positifs $a_1, ..., a_d$: temps $0(d \times log^3(Max\{a_i\})$, espace $0(d \times log(Max\{a, a'\}))$.

preuve : on utilise récursivement le calcul du ppcm de deux entiers positifs ; le résultat est $ppcm(a_i) \leq Max^d\{a_i\}$.

Etape 2 : *N_point* (calcul initial du nombre de points)

♦ Instruction *tester si* $a_1x_1 + ...a_dx_d = p \times ppcm(ai) + R = n$: on prend comme entrées n et les a_i :

- Temps $0((d-1) \times log(n) \times log(Max\{a_i\}))$,

- Espace $0(log(n) + log(Max\,\{a_i\})$

preuve : on utilise $x_i \leq p \times ppcm(ai) + R = n$, le calcul de la somme est fait par récurrence : son temps est donc $0((d-1) \times log(n) \times log(Max\,\{a_i\}))$ et comme on ne stocke pas de le, son espace est $0(log(n) + log(Max\,\{a_i\})$.

♦ Boucles : temps $0(n^d(d-1) \times log(n) \times log(Max\,\{a_i\})$, espace $0((d+1)log(n) + log(Max\,\{a_i\})$

preuve : on utilise d boucles de longueurs $1 + maxxi \leq n$ chacune ; le temps de calcul est donc $0(n^d \times (d-1) \times log(n) \times log(Max\,\{a_i\})$; le calcul par récurrence ne stocke pas de le, et le contenu du compteur est inférieur à n^d, son espace est celui utilisée par chaque instruction augmenté de l'espace consacré au compteur.

Etapes 3 et 4 : *System_ equ* (construction de systèmes d'équations des coefficients de polynôme d'Ehrhart, pour $0 \leq R < ppcm(a_i)$) : temps $0(d^2 \times n^d \times (d-1) \times log(n) \times log(Max\,\{a_i\}))$, espace $0(d^2 \times log(Max\,\{a_i\})$.

♦ Chaque système d'équations a d inconnues et d équations : le temps de constitution du tableau des coefficients est d^2 calculs précédents ; le temps total pour obtenir un système s'obtient donc en multipliant le temps de l'étape *Nbr_ point* par d^2 ; l'espace utilisé est $0(d^2 \times log(Max\,\{a_i\})$.

♦ La résolution de chaque système par la méthode de Gauss est en temps cubique par rapport à la taille des entrées, ce qui conserve la complexité issue des étapes précédentes ; en revanche, elle n'utilise pas d'espace supplémentaire.

♦ Le nombre de systèmes à résoudre est $ppcm(a_i) \leq n$; les résolutions se faisant successivement, on n'utilise pas d'espace supplémentaire en résolvant plusieurs systèmes.

Conclusion :

La complexité en temps de l'algorithme est $0(d^2 \times n^{d+1} \times (d-1) \times log(n) \times log(Max\,\{a_i\}))$.

La complexité en espace de l'algorithme est $0(Max\{d^2 \times log(Max\,\{a_i\}\ , (d+1)log(n) + log(Max\,\{a_i\})\})$.

Cette étude montre que l'algorithme est LINSPACE (en espace linéaire), mais la complexité théorique en temps est exponentielle. On voit que la plus grande partie (de loin) du temps est consommé dans le calcul exhaustif de nombres de solutions dans l'étape 2. Cependant, la taille de l'espace étant faible, cet algorithme est fiable pour des grands nombres sans adaptation.

Résumé

La parallélisation automatique permet de convertir automatiquement un programme séquentiel à une version qui peut être directement exécutée en parallèle sans toucher la sémantique du programme séquentiel. Les parties de ce dernier qui portent généralement plus de parallélisme sont les nids de boucles. Par nature, la plupart des nids de boucles qui apparaissent dans des programmes sont affines. Le domaine d'itération d'un nid de boucles affines peut être décrit par les points entiers appartenant à un polytope. Les problèmes d'analyse et d'optimisation de nids de boucles - nécessaires dans la parallélisation automatique - se réduisent souvent au comptage de points entiers dans des polytopes. Le résultat du comptage peut être donné par un ou plusieurs polynômes ayant des coefficients périodiques. Ces polynômes sont connus sous le nom de quasi-polynômes d'Ehrhart.

Le travail réalisé dans ce mémoire consiste en l'élaboration d'un algorithme de dénombrement de points entiers pour une famille de polyèdres (enveloppes convexes de n points à coordonnées entières affinement indépendants dans un espace affine de dimension n) basée sur la méthode d'interpolation pour le calcul de quasi-polynômes d'Ehrhart. L'évaluation de son temps de calcul et la présentation des données en entrée sont comparés à ceux d'algorithme d'interpolation de Clauss qui est implémenté dans la librairie Polylib. Notre algorithme est plus rapide pour la résolution du problème restreint à la classe de polyèdres considérés.

Mots clés : Dénombrement, Points entiers, Polytope, Quasi-polynômes d'Ehrhart, Parallélisation automatique.

Abstract

Automatic parallelization can automatically convert a sequential program to a version that can be executed directly in parallel without changing the semantics of sequential program. Parts of this latter which are usually more parallelism are loop nests. By nature, most loop nests that appear in programs are affine. The iteration domain of loop nests can be described by integer points in a polytope. The problems of analysis and optimization of nested loops, - necessary in the automatic parallelization - are often reduced to counting integer points in polytopes. The result of counting can be given by one or several polynomials with periodic coefficients. These polynomials are known as the Ehrhart quasi-polynomials.

The work in this memory is the development of an algorithm for counting integer points for a family of polyhedra (convex hulls of n points with integer coordinates Independent refinement in an affine space of dimension n) based on the method of interpolation for the calculation of quasi-Ehrhart polynomial. The evaluation of the computing time and the presentation of input data are compared with those of Clauss interpolation algorithm that is implemented in the library Polylib. Our algorithm is faster in solving the

problem restricted to the class of polyhedra considered.

Keywords : Enumeration, Integer points, Polytope, Ehrhart quasi-polynomials, Automatic parallelization.