N° d'ordre : 2824

THÈSE

présentée par

Vincent LOECHNER

devant

L'UFR DE MATHÉMATIQUE ET D'INFORMATIQUE DE
L'UNIVERSITÉ LOUIS PASTEUR DE STRASBOURG

pour l'obtention du
DIPLÔME DE DOCTORAT
Spécialité : informatique

CONTRIBUTION À L'ÉTUDE DES POLYÈDRES PARAMÉTRÉS ET APPLICATIONS EN PARALLÉLISATION AUTOMATIQUE

Date de soutenance : 11 décembre 1997

Composition du Jury :

Président du Jury et rapporteur interne :	Guy-René Perrin (Professeur, ULP, Strasbourg)
Rapporteurs externes :	Paul Feautrier (Professeur, UVSQ, Versailles)
	Sanjay Rajopadhye (Chargé de Recherches, CNRS, IRISA, Rennes)
Directeur de thèse :	Catherine Mongenet (Maître de Conférences, Habilitée à diriger les recherches, ULP, Strasbourg)
Examinateur :	François Irigoin (Maître de Recherches, ENS Mines de Paris)

Remerciements

Je tiens à exprimer ma gratitude envers :

Catherine Mongenet, pour son travail scientifique, ses idées, son énergie et pour avoir accepté de les partager avec moi durant ces années ;

Doran Wilde et Philippe Clauss, avec qui j'ai eu le plaisir de travailler et qui m'ont communiqué leur *polyèdromanie* ;

Guy-René Perrin, pour l'accueil chaleureux au sein de l'ICPS, et pour avoir accepté de présider le jury de cette thèse ;

Paul Feautrier, pour l'ensemble de ses travaux, source d'idées intarrissable, et pour avoir accepté de rapporter cette thèse ;

Sanjay Rajopadhye, pour avoir accepté de rapporter cette thèse, ainsi que l'ensemble des membres de l'équipe API de l'IRISA, pour leur grande disponibilité, et avec qui les contacts ont été fréquents et fructueux ;

François Irigoin, pour avoir accepté de participer au jury ;

l'ensemble des membres de l'ICPS, une équipe sympathique et soudée ;

ma famille et mes amis de Strasbourg et d'ailleurs, qui m'ont soutenu et supporté durant ces années.

Table des matières

Résumé

Cette thèse est composée de deux parties. La première partie est consacrée à l'étude de la parallélisation de systèmes d'équations récurrentes affines paramétrés. Un système d'équations récurrentes est équivalent à un nid de boucles à assignation unique. Ce travail donne en particulier une méthode de recherche des fonctions de cadencement et d'allocation par variable permettant de minimiser les communications dans le réseau de processeurs cible. Les contraintes de compatibilité entre cadencement et allocation sont également prises en compte. Le modèle géométrique utilisé depuis de nombreuses années dans l'analyse de dépendances de nids de boucles ou de systèmes d'équations récurrentes repose sur l'utilisation de polyèdres convexes. Lorsque le système d'équations récurrentes est paramétré, les polyèdres sont paramétrés. La nécessité de connaître les sommets de polyèdres paramétrés est mise en évidence dans cette première partie.

La deuxième partie est une étude des polyèdres paramétrés et de leur utilisation dans le domaine de la parallélisation automatique. Une méthode originale de recherche des sommets de polyèdres paramétrés, valides par domaines sur les paramètres, y est notamment décrite. Le deuxième chapitre de cette partie présente quelques applications de ces résultats en parallélisation automatique.

Mots-clés

Techniques de parallélisation automatique de nids de boucles, systèmes d'équations récurrentes affines paramétrés, minimisation des communications, polyèdres convexes paramétrés.

Abstract

There are two parts in this work. The first one concerns automatic parallelization of systems of affine recurrence equations, which are formal descriptions of single assignment loop nests. We propose a new algorithm to find efficient allocation and scheduling functions for each variable. The goal is in particular to optimize the communications on the resulting processor array. Compatibility between schedule and allocation is also taken into account. The geometrical model used for many years in order to analyse the dependencies is based on convex polyhedra. When the loop nest or the system of recurrence equations depends on unknown integer variables, the polyhedra are *parameterized*. This first part shows the need for an algorithm finding the vertices, as function of the parameters, of a parameterized polyhedra.

The second part is a study of parameterized polyhedra, and their use in automatic parallelization. We describe an algorithm finding the vertices of such polyhedra, as function of the parameters, defined over parameters validity domains. Some applications of these methods are given in the last chapter.

Keywords

Automatic parallelization techniques, loop nests, systems of parameterized affine recurrence equations, communication minimization, parameterized convex polyhedra.

Introduction

Quel parallélisme?

La nécessité de l'utilisation de machines multi-processeurs est aujourd'hui indéniable. Le parallélisme est utilisé dans de nombreux domaines scientifiques. Citons, par exemple, en mathématiques, le calcul précis de variables telles que π ou les nombres premiers de Mersenne (le plus grand nombre premier connu à ce jour est $2^{2976221}-1$; il contient près de 900000 chiffres), en physique ou en chimie, la simulation réaliste de phénomènes que nous ne savons pas modéliser dans leur intégralité (simulation d'accidents dans des centrales nucléaires, simulation d'écoulement de liquide ou de gaz dans un tube et échauffement, ...), en informatique, le décodage de messages cryptés (la recherche d'une clé de cryptage de 56 bits a été effectuée début 1997 par 80000 machines hétérogènes en 4 mois; elle aurait nécessité plusieurs siècles de calculs sur une machine mono-processeur). Toutes ces réalisations n'auraient été possibles sans machines parallèles.

Cependant, aujourd'hui encore, dans la plupart des programmes parallèles utilisés dans le monde, le parallélisme est décrit explicitement par le programmeur. Cette tâche est fastidieuse et introduit des risques d'erreurs difficilement détectables. Le compilateur idéal, dont nous rêvons, devra être capable de transformer automatiquement une spécification, séquentielle ou fonctionnelle, puisque la manière de penser de l'Homme est intrinsèquement séquentielle, en un programme équivalent s'exécutant de manière efficace sur une machine parallèle. C'est à cette tâche que s'attelle notre communauté scientifique.

Le premier problème rencontré en programmation des machines parallèles est la très grande diversité de classes d'architectures, allant de la machine vectorielle au réseau de machines hétérogènes (reliées par un réseau ethernet par exemple) : machines à mémoire partagée (généralement pour un petit nombre de processeurs) ou à mémoire distribuée (architectures massivement parallèles), machines synchrones ou asynchrones, processeurs reliés par des réseaux statiques de type hypercube ou tore (nCube, Cray T3E, MasPar,...) ou par des réseaux d'interconnexions multi-couches (IBM SP2, ...), ou machines parallèles combinant plusieurs de ces caractéristiques (SGI Origin 2000, ...).

Ce problème d'hétérogénéité des architectures parallèles semble partiellement résolu aujourd'hui par l'émergence de langages de programmation parallèle indépendants de la machine parallèle cible. Ces langages sont cependant toujours à parallélisme explicite : le programmeur doit spécifier explicitement le parallélisme dans le programme. Ces langages correspondent à deux types de programmation :

– la programmation *data-parallèle*, où le parallélisme s'exprime par l'utilisation de données parallèles par nature : les tableaux. Les tableaux sont placés sur différents processeurs par des directives de type ALIGN, et les calculs sont effectués en parallèle grâce à des directives FORALL ou INDEPENDENT, ou grâce à la possibilité de manipuler des tableaux entiers en une seule instruction. Les langages développés dans ce cadre sont essentiellement basés sur Fortran, le langage le plus utilisé par les programmes scientifiques. Plusieurs extensions de Fortran ont été proposées, comme Vienna Fortran [ZBC+92], Fortran D [FHK+91], MPP Fortran [PMM92]. Un standard semble apparaître avec HPF [HPF93]. D'autres langages moins connus se basent par exemple sur C, c'est le cas de PompC [Par92] et d'HyperC [CP93];

– la programmation par communications explicites sur réseau de processeurs virtuels infini, où le parallélisme s'exprime sur chaque processeur virtuel par l'envoi et la réception de données. Dans ce cadre, de nouveaux langages ont été conçus (CSP [Hoa78], Occam [INM88], ADA [ANS93], ...), mais l'émergence de bibliothèques de fonctions dédiées aux communications entre processeurs (PVM [GBD+], MPI [MPI95]) semble s'imposer aujourd'hui.

Le compilateur idéal devra donc produire, à partir d'une spécification, un programme HPF ou MPI par exemple. Il reste à trouver des méthodes automatiques de transformation de programmes séquentiels en programmes parallèles. La partie de programme la plus facile à paralléliser est le nid de boucles. Dans la plupart des cas c'est également la plus importante à paralléliser, car c'est dans les nids de boucles que les programmes passent l'essentiel de leur temps de calcul. Paralléliser le reste du programme s'avère beaucoup plus complexe pour un gain souvent moindre.

Parallélisation automatique de nids de boucles

De nombreuses équipes à travers le monde ont amorcé des projets de compilateur paralléliseur automatique. Voici quelques-uns d'entre eux :
- SUIF [AALT95], Université de Stanford, `http://suif.stanford.edu/`,
- Tiny [Wol91], étendu dans Petit, et Omega, Université du Maryland, `http://www.cs.umd.edu/projects/omega/`,
- Polaris, Université de l'Illinois à Urbana-Champaign, `http://polaris.csrd.uiuc.edu/`,
- LooPo, Université de Passau, `http://brahms.fmi.uni-passau.de/cl/loopo/`,
- PIPS, École Nationale Supérieure des Mines de Paris, `http://www.cri.ensmp.fr/~pips`,
- PAF, Université de Versailles, `http://www.prism.uvsq.fr/english/parallel/paf/autom_us.html`,
- Bouclettes, Université de Versailles,

`http://www.prism.uvsq.fr/public/bop/base/bclt/bouclettes.html`,
- Alpha [LMQ91], IRISA de Rennes,
`http://www.irisa.fr/api/ALPHA/alpha.html`.

Deux approches existent dans ce domaine, mais les différentes méthodes mises en œuvre ont de nombreux points communs. Elles sont basées sur une phase d'analyse de dépendances, et consistent à trouver pour chaque programme une allocation sur un réseau de processeurs (boucles parallèles) et un cadencement dans le temps (boucles séquentielles). La première approche consiste à rechercher des transformations de variables dans un nid de boucles, permettant de remplacer certaines boucles par des boucles indépendantes. Une boucle est indépendante lorsque chacune de ses itérations peut se faire indépendamment des autres. La méthode utilisée consiste à essayer d'appliquer un catalogue de transformations selon certains critères de validité. Une liste exhaustive est donnée dans [KP93]. Le lecteur trouvera une description détaillée de ces méthodes dans [AK84, ZC90, Wol91, Pug92, Ban93] : *loop interchange, loop skewing, loop reversal, statement reordering, loop distribution, loop fusion, loop alignment, loop interleaving, tiling (*ou *blocking), index set splitting, loop coalescing, loop scaling.*

La deuxième méthode se base sur une représentation formelle équivalente à un nid de boucles en assignation unique : le système d'équations récurrentes. Après la transformation du nid de boucles en assignation unique, puis en un système d'équations récurrentes, la parallélisation se fait de la manière suivante. Il faut avant tout extraire la sémantique du programme, dans la phase d'analyse de dépendances, grâce à la construction d'un graphe de dépendance. La parallélisation consiste à trouver ensuite une transformation espace-temps, c'est-à-dire une fonction d'allocation et une fonction de cadencement respectant la sémantique donnée par le graphe de dépendance et optimisant la solution parallèle. Enfin, le code parallèle est donné par l'application de cette transformation au système d'équations récurrentes.

Apports de ce travail

Du fait de l'origine de ces travaux, cette thèse se place dans le contexte des systèmes d'équations récurrentes. Cependant, les résultats théoriques que nous avons obtenus, sur les polyèdres paramétrés notamment, ont des applications dans les deux contextes. En effet, quelle que soit l'approche utilisée, le parcours d'un nid de boucles est équivalent au parcours d'un ensemble de points d'un lattice[1] dont la dimension est le nombre de boucles imbriquées, et contenu dans un polyèdre convexe lorsque les bornes des boucles sont affines (ce qui est généralement le cas). Lorsque ces bornes de boucles dépendent de variables non instanciées au moment de la compilation, les polyèdres sont *paramétrés.*

De nombreux problèmes d'optimisation des solutions parallèles recherchées reposent sur la connaissance des sommets de polyèdres paramétrés. Pour certains d'entre eux, la

1. Un *lattice* est l'ensemble des points d'un espace affine de dimension quelconque, obtenus par combinaison linéaire entière quelconque d'un ensemble de vecteurs, constituant la *base* du lattice.

connaissance du minimum et du maximum lexicographiques suffit. On peut dans ce cas utiliser un outil comme PIP (Parametric Integer Programming [FT89]). D'autres problèmes nécessitent la connaissance de tous les sommets de polyèdres paramétrés. Ce travail fournit une méthode de recherche des sommets de polyèdres paramétrés ; les sommets sont calculés en fonction des paramètres, et en fonction de leurs domaines de validité. C'est la première méthode proposée pour résoudre ce problème à notre connaissance.

Le deuxième apport important de cette thèse est une application de cette méthode, permettant de déterminer une fonction d'allocation optimisant les communications dans la parallélisation d'un système d'équations récurrentes. Elle repose sur une analyse précise des communications résultant de dépendances affines.

Ces résultats théoriques ont été implémentés dans le cadre de cette thèse dans un environnement d'aide à la parallélisation automatique de systèmes d'équations récurrentes affines : OPÉRA (Outil de Parallélisation d'Équations Récurrentes Affines). Ce projet compte en l'état actuel environ 10000 lignes de C, indépendamment des bibliothèques appelées. La recherche de sommets de polyèdres paramétrés a été implémentée séparément dans la bibliothèque de manipulation de polyèdres *PolyLib*, développée à l'IRISA de Rennes [Wil93]. Cette bibliothèque est constituée d'environ 7000 lignes de C, dont la moitié environ concerne les polyèdres paramétrés. Elle est distribuée sous licence GNU : `http://icps.u-strasbg.fr/~loechner/polylib/`

Organisation

La première partie de ce document est consacrée à la parallélisation de systèmes d'équations récurrentes. Après un état de l'art de ce modèle présenté dans le premier chapitre, le deuxime chapitre est consacré au problème de l'optimisation des communications. Il présente en particulier l'une des contributions de ce travail de thèse, à savoir un algorithme de recherche de transformation espace-temps dans ce contexte. Le troisième chapitre présente le logiciel OPÉRA développé dans le cadre de cette thèse. L'utilisation d'OPÉRA et les structures de données manipulées sont dérites en annexe A.

La motivation de l'utilisation de polyèdres paramétrés, et la nécessité d'un algorithme de recherche des sommets d'un polyèdre paramétré, se trouvent dans la première partie.

Le lecteur intéressé par les polyèdres paramétrés pourra se reporter directement à la deuxième partie, sans lire la première. Le premier chapitre de cette deuxième partie donne les bases de la théorie des polyèdres rationnels paramétrés. La méthode de recherche des sommets de tels polyèdres, développée dans le cadre de cette thèse, y est décrite. Le deuxième chapitre présente quelques applications possibles de ces résultats dans le domaine de la parallélisation automatique. L'annexe B décrit les extensions paramétriques de la bibliothèque de manipulation de polyèdres *PolyLib*. Cette annexe a été écrite en anglais, afin d'être utilisée comme document de référence de la *PolyLib*. L'annexe C contient un article écrit en collaboration avec Philippe Clauss et Doran Wilde, décrivant nos recherches dans le domaine des polyèdres paramétrés, et plusieurs applications dans l'analyse et la transformation de nids de boucles.

Première partie

Du système d'équations récurrentes au code parallèle

Introduction

Réseaux systoliques

Dans le début des années 80, les progrès des technologies VLSI ont contribué à l'émergence d'un concept d'architecture parallèle de bas niveau : les réseaux systoliques [KL78, Kun82]. Une architecture systolique est caractérisée par un réseau régulier de processeurs élémentaires fonctionnant en mode synchrone. Ce type d'architecture est *dédié*, c'est-à-dire qu'une architecture spécifique est nécessaire à la résolution de chaque problème. Cette spécificité des architectures systoliques a incité les chercheurs à développer des méthodes de synthèse automatique permettant de déduire un ensemble de solutions adaptées à un problème.

Les premières méthodes sont fondées sur l'utilisation des systèmes d'équations récurrentes uniformes. Cette formalisation mathématique permet de modéliser les problèmes "systolisables" car elle intègre les caractéristiques de régularité et de localité du modèle systolique. Elle fut définie dès 1967, bien avant la conception des premiers réseaux systoliques, par Karp, Miller et Winograd [KMW67].

Un système d'équations récurrentes uniformes (SURE) est un ensemble d'équations définissant une suite. Un produit de deux matrices de taille $N \times N$, $C = A * B$ s'écrit sous forme d'équations récurrentes de la manière suivante :

$$
\begin{array}{ll}
C[i,j,k] = C[i,j,k-1] + a[i,j-1,k] * b[i-1,j,k] & \text{pour } 1 \leq i,j,k \leq N \\
a[i,j,k] = a[i,j-1,k] & \text{pour } 1 \leq i,j,k \leq N \\
b[i,j,k] = b[i-1,j,k] & \text{pour } 1 \leq i,j,k \leq N \\
C[i,j,0] = 0 & \text{pour } 1 \leq i,j \leq N \\
a[i,0,k] = A[i,k] & \text{pour } 1 \leq i,j \leq N \\
b[0,j,k] = B[k,j] & \text{pour } 1 \leq i,j \leq N
\end{array}
$$

Chaque élément de la suite à calculer est identifié par un *vecteur d'indices* contraint par un système d'inégalités par équation, le vecteur (i,j,k) avec $1 \leq i,j,k \leq N$ dans la première équation dans l'exemple ci-dessus. L'ensemble de ces points de calcul est un polyèdre convexe, comme l'a montré Kuck dans le contexte des nids de boucles [Kuc78]. Un tel polyèdre est appelé *domaine d'itération* ou *domaine de calcul*. Dans cet exemple, il s'agit d'un cube de taille N. Les dépendances sont uniformes, ce qui signifie que les références à d'autres éléments de la suite se font de manière régulière et selon des vecteurs constants par rapport au vecteur d'indices. Ainsi, dans l'exemple ci-dessus, la dépendance entre $C[i,j,k]$ et $C[i,j,k-1]$ est caractérisée par le vecteur $(0,0,1)$.

Méthodes de synthèse automatique et extensions du modèle

En 1984, Quinton [Qui84a] propose une méthodologie de synthèse de réseaux systoliques, s'inspirant des travaux de Lamport concernant l'ordonnancement linéaire de boucles imbriquées [Lam74]. Ces idées sont développées parallèlement par Moldovan et Fortes sous forme matricielle [Mol83, FFW85, MF86], et ont donné naissance à la méthode de synthèse dite de *projection*. La solution systolique est obtenue par application d'une transformation espace-temps aux domaines d'itération et aux dépendances du problème. Cette transformation est caractérisée par une *fonction d'allocation* qui répartit les calculs sur les processeurs, et par une *fonction de temps* affine qui cadence ces calculs. La fonction d'allocation est une projection linéaire le long d'un *vecteur d'allocation*, choisi selon des critères d'optimisation des communications et de taille du réseau de processeurs. Pour un problème dont les domaines d'itération sont de dimension n, le réseau de processeurs est de dimension $n-1$. La fonction de temps est définie grâce à la méthode des hyperplans de Lamport, qui partitionne le domaine d'itération par une famille d'hyperplans parallèles, orthogonaux au *vecteur de cadencement*. Chacun de ces hyperplans définit l'ensemble de calculs qui s'effectuent au même instant. On parle dans ce cas de *cadencement monodimensionnel*.

Bien vite cependant ce modèle révéla ses limites. D'abord, parce que de nombreux problèmes ont des dépendances naturellement affines : les références aux autres éléments de la suite utilisés dans les équations récurrentes sont des fonctions affines et non pas uniformes du vecteur d'indices. Ainsi dans l'équation principale de l'algorithme de triangularisation de Gauss il y a trois dépendances affines (seule la première est uniforme) :

$$\begin{aligned} A[i,j,k] = A[i,j,k-1] - A[i,k,k-1] * A[k,j,k-1]/A[k,k,k-1] \\ \text{pour } k+1 \leq i \leq N; k+1 \leq j \leq N+1; 1 \leq k \leq N-1 \end{aligned}$$

Diverses méthodes [VQ88, YC88] ont été proposées pour *uniformiser* les systèmes d'équations récurrentes affines (SARE), mais toutes, par nature, réduisent le degré de parallélisme du problème. En particulier, les communications “multipoints”, entre un processeur et un ensemble de processeurs du réseau, induites par les dépendances affines, sont éliminées par l'uniformisation. Ces dépendances, lorsqu'elles ne sont pas uniformisées, peuvent être implémentées par des communications efficaces bien que non systoliques, par exemple par des diffusions. Mongenet [Mon85, MCP91] propose de déterminer un ensemble de transformations espace-temps affines valides directement pour les SARE, sans les uniformiser préalablement. D'autres travaux ont adressé la synthèse de systèmes d'équations récurrentes affines [DI85, Rao85, RPF86, WD89, RF90].

La deuxième limite importante de la méthode de projection est un ensemble de contraintes sur les transformations espace-temps trop restrictif. En particulier, la dimension du réseau de processeurs est très fortement liée à la dimension du problème traité : pour un problème de dimension n, le réseau obtenu est de dimension $n-1$. Cette contrainte peut être relâchée, en allouant le problème sur un réseau de taille inférieure à $n-1$. Dans ce cas, le cadencement doit être *multidimensionnel* pour garantir la validité de la transformation espace-temps. Le cadencement multidimensionnel [Fea92b] consiste à considérer chaque instant d'exécution du programme parallèle comme un vecteur entier, alors que dans le cas monodimensionnel, c'est un scalaire. Lors de l'exécution, le temps est alors égrené par plusieurs boucles imbriquées. Ceci constitue la première extension du modèle de transformation espace-temps.

La deuxième extension de ce modèle est la suivante. Dans le cas des SARE, les dépendances affines conduisent, sur la solution parallèle, à des communications multipoints. En raison du coût important des communications, il est souhaitable de trouver des transformations espace-temps qui optimisent ces communications. Dans ce but, nous utilisons le concept de *transformation par variable*, permettant d'allouer [Fea94, DR95] et de cadencer [Rao85, MQRS90] différemment chaque variable du problème. Les transformations par variable s'avèrent même indispensables pour paralléliser certains problèmes qui n'ont pas de solution par une transformation unique en raison d'incompatibilité entre les dépendances.

Plan

Cette première partie est organisée de la manière suivante : Le chapitre 1 présente un état de l'art des méthodes de parallélisation de systèmes d'équations récurrentes affines paramétrés. Dans le chapitre 2, nous proposons une méthode originale de détermination des transformations par variable optimisant les communications. L'environnement logiciel, OPÉRA, développé dans le cadre de cette thèse est présenté au chapitre 3.

La plupart des figures présentes dans cette partie sont des copies d'écran de ce logiciel.

Chapitre 1

Parallélisation de systèmes d'équations récurrentes

Les méthodes classiques de synthèse de réseaux de processeurs à partir de systèmes d'équations récurrentes sont présentées dans ce chapitre. Il s'agit d'un état de l'art présentant des résultats connus dans ce domaine, que nous avons parfois reformulé dans un souci d'homogénéité des notations.

La modélisation des dépendances est la première étape du processus de parallélisation d'un système d'équations récurrentes. Plusieurs types de graphes de dépendance peuvent être construits, suivant la classe des problèmes traités. Le paragraphe 1.2 les décrit brièvement. Le modèle que nous utilisons, communément appelé "*look-forwards*", est basé sur la recherche des ensembles d'utilisation et d'émission associés à chaque dépendance.

L'étape suivante est la recherche de transformations espace-temps affines, composées d'une fonction de cadencement et d'une fonction d'allocation. Le cadencement doit vérifier un certain nombre de contraintes, appelées *contraintes causales*, afin de respecter la sémantique du programme. Ces contraintes sont fortement liées aux dépendances : lorsque le calcul de a utilise une donnée b, il faut que le cadencement garantisse que a est calculé après b. Dans le cas des SURE, les contraintes sur le cadencement sont relativement simples, car les vecteurs de dépendance sont constants. Elles sont plus complexes pour les SARE. Deux méthodes sont habituellement utilisées afin de résoudre le système de contraintes causales : la méthode de Farkas [Fea92a], une forme affine du lemme de Farkas [Sch86], et la méthode des sommets [QV89, MQRS90]. Ces méthodes sont brièvement décrites paragraphe 1.3. L'extension au cadencement multidimensionnel se fait très naturellement en remplaçant les inégalités classiques du système de contraintes causales en inégalités lexicographiques.

La recherche de la fonction d'allocation est liée à deux contraintes : la compatibilité avec le cadencement, et l'optimisation du réseau de processeurs virtuels. La contrainte de compatibilité consiste à garantir que deux calculs d'une variable du système d'équation récurrentes ne sont pas effectués au même instant et par le même processeur virtuel.

L'allocation doit ensuite être choisie de manière à optimiser le réseau de processeurs, en particulier de manière à minimiser les communications entre processeurs virtuels. Nous reviendrons sur ce problème dans le chapitre 2.

Dans le modèle que nous utilisons, nous supposons avoir à notre disposition un nombre infini de processeurs. C'est pour cette raison que nous parlons de *processeurs virtuels*, qu'il convient de distinguer des *processeurs physiques* d'une machine parallèle cible. Plusieurs processeurs virtuels peuvent être placés sur le même processeur physique. Ce placement s'effectue dans une phase ultérieure à la phase de compilation présentée ici, au moment précédant immédiatement l'exécution du programme parallèle sur la machine parallèle. Nous supposons que ce placement s'effectue selon un pavage régulier de l'espace afin de conserver les propriétés de localité des communications que nous avons recherchées dans la phase de transformation.

1.1 Définitions

La classe la plus importante de systèmes d'équations récurrentes que nous manipulons sont les systèmes d'équations récurrentes affines paramétrés (PARE). Pour décrire les PARE, nous utilisons le language *Alpha* [Mau89, LMQ91] défini par Mauras.

1.1.1 Systèmes d'équations récurrentes affines paramétrés

Une *équation récurrente affine* est une équation de la forme :

$$X[z] = f(\dots, Y[g(z,p)], \dots) \qquad z \in \mathcal{D}_E(p) \tag{E}$$

où :

- X et Y sont des noms de variables,
- p est un vecteur de $\mathbb{Z}^m$, représentant les m *paramètres* du problème,
- $\mathcal{D}_E(p) \subset \mathbb{Z}^n$ est un polyèdre convexe paramétré appelé *l'espace d'itération* de cette équation. Il est défini par un ensemble de contraintes linéaires de la forme :

  $$\mathcal{D}_E(p) = \{z \in \mathbb{Z}^n \mid Az \geq Bp + b\} \tag{1.1}$$

  où A et B sont des matrices entières, b un vecteur entier,
- $g(z,p)$ est la *fonction d'index.* C'est une fonction affine de la forme :

  $$g(z,p) = Rz + Qp + r \tag{1.2}$$

  où R et Q sont des matrices entières, et r est un vecteur entier.

Un *système d'équations récurrentes affines paramétré* (PARE), ou programme Alpha, contient les informations suivantes :

- un vecteur de paramètres dans un domaine de validité, $p \in \mathcal{D}_p$, avec $\mathcal{D}_p \subset \mathbb{Z}^m$,
- un ensemble de variables typées d'entrée et de sortie et leurs domaines de définition,
- un ensemble de variables typées intermédiaires, utilisées dans les équations récurrentes, et leurs domaines de définition,
- un ensemble d'équations récurrentes affines.

Le domaine de définition d'une variable X est un polyèdre convexe paramétré par p, et est noté $\mathcal{D}_X(p)$. Notons que la dimension de ces domaines n'est pas forcément la même pour toutes les variables du PARE.

1.1.2 Nids de boucles à assignation unique

La classe de nids de boucles équivalents à des systèmes d'équations récurrentes affines paramétrés respecte les contraintes suivantes :

- les indices de boucle sont entiers,
- l'incrément des boucles est de 1,
- les bornes supérieures et inférieures des boucles, et les expressions arithmétiques gauches ou droites des expressions booléennes utilisées dans les tests, sont des expressions affines à coefficients entiers, dépendant de variables entières définies à l'extérieur des boucles (*paramètres*) et des indices de boucles externes à l'instruction considérée,
- l'expression d'un indice de tableau est une fonction affine des paramètres et des indices de boucles externes,
- *(assignation unique)* les variables ne reçoivent qu'une seule valeur pendant toute l'exécution du nid de boucles (sauf les indices de boucles évidemment) ; ainsi, dans un nid de boucles quelconque, un scalaire modifié à chaque itération d'une boucle devra être déclaré comme un tableau à autant de dimensions que le nombre de boucles externes,
- les appels de procédures ou fonctions sont valides s'ils modifient des variables externes de manière prédictible et en respectant les contraintes ci-dessus ; cependant, pour de meilleures performances, toutes les instructions élémentaires considérées doivent avoir un temps d'exécution du même ordre,

– des instructions de branchement structurées telles que l'instruction 'if' peuvent apparaître à tout niveau du nid de boucles,

– les boucles ne sont pas forcément parfaitement imbriquées, c'est-à-dire que deux boucles peuvent se suivre séquentiellement à l'intérieur d'une autre boucle.

Diverses méthodes de transformations de boucles peuvent être appliquées à un nid de boucles quelconque pour satisfaire les contraintes ci-dessus. La mise en assignation unique est la partie la plus délicate, et a été traitée notamment dans [CFR+91, Fea91, PS95].

1.1.3 Autres systèmes d'équations récurrentes

Systèmes d'équations récurrentes uniformes

Un système d'équations récurrentes uniformes est défini de la même manière qu'un système d'équations récurrentes affines, avec la contrainte supplémentaire suivante : toutes les fonctions d'index $g(z, p) = Rz + Qp + r$ sont telles que la matrice R est la matrice identité, et la matrice Q est nulle.

Toutes les équations sont donc de la forme :

$$X[z] = \dots\ Y[z + r]\ \dots$$

où r est un vecteur entier constant. Les dépendances sont caractérisées par les vecteurs de dépendance r.

Systèmes d'équations récurrentes non paramétrés

Un système d'équations récurrentes, affines ou uniformes, non paramétré, est défini de la même manière qu'un système d'équations récurrentes paramétré, en ajoutant la contrainte :

$$p = 0.$$

Les équations sont donc de la forme :

$$X[z] = \dots\ Y[Rz + r]\ \dots$$

où R est une matrice entière et r un vecteur entier.

Le domaines de définition des variables et les domaines d'itération sont des polyèdres convexes non paramétrés.

Systèmes d'équations récurrentes homogènes

On dit qu'un système d'équations récurrentes est *homogène* lorsque toutes les variables, sauf les variables d'entrée et de sortie, sont de même dimension : tous les domaines de définition sont des polyèdres convexes définis dans un espace de dimension n unique.

1.1.4 Exemple : l'élimination de Gauss

L'algorithme de l'élimination de Gauss est souvent utilisé pour résoudre un système de N égalités à N variables de la forme $Bx = b$, où B est une matrice $N \times N$, et b un vecteur de taille N. Cet algorithme consiste à trianguler la matrice `A` $= [B \mid b]$ de taille $N \times (N + 1)$. Le nid de boucles séquentiel correspondant est :

```
do k = 1 TO N-1
   do j = k+1 TO N+1
      do i = k+1 TO N
         A[i,j] = A[i,j] - A[i,k] * A[k,j] / A[k,k]
      done
   done
done
```

Ce nid de boucles est équivalent au nid de boucles à assignation unique suivant, où on a ajouté un indice au tableau `A`. Après avoir initialisé `A[i,j,0]`, et exécuté ce nid de boucles, le résultat se trouve dans `A[i,j,i-1]` :

```
do k = 1 TO N-1
   do j = k+1 TO N+1
      do i = k+1 TO N
         A[i,j,k] = A[i,j,k-1] - A[i,k,k-1] * A[k,j,k-1] / A[k,k,k-1]
      done
   done
done
```

Voici le programme Alpha complet correspondant au PARE de l'élimination de Gauss, prenant une matrice `m` en entrée, et renvoyant une matrice triangulaire `g` en sortie :

```
system GaussianElimination : { N | N>=1 }
               ( m : { i,j | 1<=i<=N; 1<=j<=N+1 } of integer )
       returns ( g : { i,j | 1<=i<=N; i<=j<=N+1 } of real );
var
   A : { i,j,k | k+1<=i<=N; k+1<=j<=N+1; 0<=k<=N-1 } of real;
let
  A[i,j,k] = case
               { | k=0 } : m[i,j];
               { | 1<=k<=N-1 } : A[i,j,k-1] -
                                   A[i,k,k-1] * A[k,j,k-1] / A[k,k,k-1] ;
             esac;
  g[i,j] = A[i,j,i-1];
tel;
```

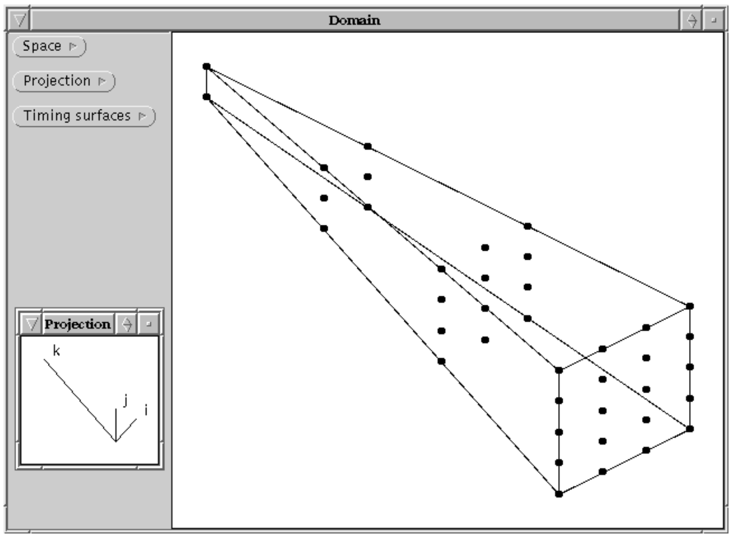


FIG. 1.1 - *Gauss : Domaine de définition de A pour* $N = 4$.

Le domaine de définition de la variable A est représenté figure 1.1, pour la valeur du paramètre $N = 4$.

1.2 Modélisation des dépendances

Dans le domaine de la parallélisation automatique de nids de boucles quelconques, l'analyse des dépendances repose sur la construction d'un graphe orienté, appelé *graphe réduit*, dont chaque noeud représente une variable du nid de boucles, et dont les arcs représentent les dépendances entre ces variables. Quatre types de dépendances apparaissent dans ce modèle : les anti-dépendances, les dépendances d'entrée (*"input-dependences"*), les dépendances de sortie (*"output-dependences"*), et les dépendances de flot (*"flow-dependences"*) [ZC90, BENP93, Dar93, Wol96]. Toute transformation de nid de boucles doit garantir que la sémantique du programme transformé est la même que celle d'origine, c'est-à-dire que toutes les dépendances sont respectées. Les dépendances d'entrée étant liées à l'affectation de la même valeur à plusieurs variables, peuvent être ignorées. C'est pourquoi certains auteurs ne considèrent que trois types de dépendances.

Les seules "vraies" dépendances dans un nid de boucles sont les dépendances de flot. Les anti-dépendances et les dépendances de sortie sont dues à la réutilisation de mémoire. Dans le modèle basé sur les systèmes d'équations récurrentes, toutes ces "fausses" dépendances ont été éliminées par la mise en assignation unique du problème. Le graphe réduit des dépendances est donc simplement le graphe des dépendances de flot.

Lorsque la parallélisation est réalisée sans respecter l'*owner computes rule*, c'est-à-dire que la donnée calculée par une instruction n'est pas forcément placée sur le même processeur virtuel que l'instruction correspondante, la modélisation des dépendances se fait au moyen d'un graphe réduit caractérisé par deux types de noeuds : les noeuds *instructions* et les noeuds *données*. Un arc existe du noeud instruction s vers le noeud donnée a si s calcule a. Un arc existe de a vers s si l'instruction s utilise a dans son calcul. Ce graphe est appelé *graphe de communication* par Darte et Robert [DR94] dans le contexte de la parallélisation des SARE. Dans le but de minimiser les communications, Dion et Robert [DR95] ont étendu ce modèle en associant, à chaque arc du graphe, un poids caractérisant la dépendance. Un tel graphe est appelé *graphe d'accès*.

Une autre manière de représenter les dépendances est de définir le *graphe complet* des dépendances. C'est un graphe orienté dont chaque noeud représente un point d'un domaine d'itération, et dont chaque arc représente une dépendance entre deux points de calcul. En pratique, la taille ce graphe peut être très grande, infinie dans la cas de domaines non bornés. De plus, lorsqu'il synthétise des dépendances affines, ce graphe ne présente aucune régularité, contrairement au cas uniforme.

La solution retenue dans ce travail pour modéliser les dépendances affines, consiste à utiliser les notions d'*ensembles d'utilisation* et d'*ensembles d'émission* des données définis dans le modèle *"look-forward"* [MP87, MCP91, MCP94]. Les noeuds du graphe de dépendance représentent les points entiers de polyèdres convexes (les espaces d'itération). Ce graphe peut donc être projeté dans les espaces géométriques correspondants. Comme de plus les dépendances sont affines, les ensembles d'utilisation et d'émission ont des propriétés géométriques intéressantes dans ces espaces.

Définition 1 (auto-dépendance) *On appelle* auto-dépendance *une dépendance entre deux instances de la même variable :*

$$X[z] = \ldots\ X[g(z,p)]\ \ldots \qquad (E)$$

Définition 2 (dépendance croisée) *On appelle* dépendance croisée *une dépendance de la forme :*

$$X[z] = \ldots\ Y[g(z,p)]\ \ldots \qquad (E)$$

où X et Y sont deux variables différentes du système d'équations récurrentes.

Nous considérons dans la suite de ce paragraphe une dépendance quelconque d'une variable Y vers une variable X de la forme :

$$X[z] = \ldots\ Y[g(z,p)]\ \ldots \qquad (E)$$

Définition 3 (ensemble d'utilisation) *On appelle* ensemble d'utilisation *associé à une dépendance correspondant à une donnée $Y[z_0]$ dans une équation E, l'ensemble :*

$$\mathrm{Util}_{E,Y}(z_0,p) = \{z \in \mathcal{D}_E(p) \mid g(z,p) = z_0\} \qquad (1.3)$$

où $\mathcal{D}_E(p)$ est l'espace d'itération de l'équation E, g est la fonction d'index associée à cette dépendance, z_0 est un point de $\mathcal{D}_Y(p)$, le domaine de définition de la variable Y.

L'ensemble $\mathrm{Util}_{E,Y}(z_0, p)$ correspond à l'ensemble des points de $\mathcal{D}_E(p)$ qui utilisent pour leur calcul la donnée $Y[z_0]$. C'est l'image inverse d'un point par la fonction affine g, d'où la propriété suivante :

Propriété 1 *Un ensemble d'utilisation* $\mathrm{Util}_{E,Y}$ *est l'ensemble des points entiers d'un polyèdre convexe paramétré par z_0 et par p. Il est engendré par le noyau de la matrice R, la partie linéaire de la fonction d'index g.*

Définition 4 (vecteurs d'utilisation) *Soit $g(z,p) = Rz + Qp + r$ la fonction d'index associée à une dépendance. Les vecteurs formant une base de $Ker(R)$ sont appelés les* vecteurs d'utilisation *et sont notés $u_{E,Y,i}$.*

Les vecteurs $u_{E,Y,i}$ forment une base de l'ensemble d'utilisation $\mathrm{Util}_{E,Y}(z_0, p)$.

Définition 5 (ensemble d'émission) *On appelle* ensemble d'émission *associé à une dépendance correspondant à une variable Y dans une équation E, l'ensemble :*

$$\begin{aligned} \mathrm{Emit}_{E,Y}(p) &= \{z_0 \in \mathbb{Z}^n \mid \exists z \in \mathcal{D}_E(p) \textit{ tel que } z_0 = g(z,p)\} \quad &(1.4)\\ &= \{g(z,p) \mid z \in \mathcal{D}_E(p)\} \quad &(1.5)\end{aligned}$$

où g est la fonction d'index associée à cette dépendance.

L'ensemble $\mathrm{Emit}_{E,Y}(p)$ correspond à l'ensemble des points du domaine de définition de Y qui sont utilisés pour un calcul de X dans l'équation E. Sa dimension est égale au rang de la matrice R. C'est l'image des points entiers d'un polyèdre convexe par une fonction affine, appelé LBL selon la définition suivante :

Définition 6 (LBL) *On appelle* LBL (Linearly Bounded Lattice) *l'image des points entiers d'un polyèdre convexe par une fonction affine entière [TT93].*

Définition 7 (Z-polyèdre) *On appelle* Z-polyèdre *l'intersection entre un lattice entier et un polyèdre de même dimension [Anc91, QRR97].*

Propriété 2 *Un LBL est contenu dans un Z-polyèdre.*

Un ensemble d'émission est un LBL paramétré par p. Ce LBL est contenu dans un Z-polyèdre à partir duquel sont définis les vecteurs d'émission :

Définition 8 (vecteurs d'émission) *On appelle* vecteurs d'émission *les vecteurs de base du Z-polyèdre contenant l'ensemble d'émission* $\mathrm{Emit}_{E,Y}(p)$. *On les note $\epsilon_{E,Y,i}$, $1 \leq i \leq Dim(\mathrm{Emit}_{E,Y}(p))$.*

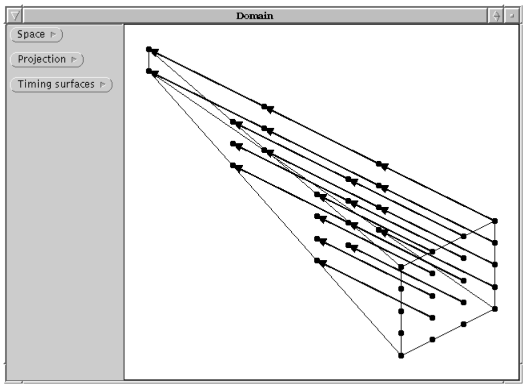


FIG. 1.2 - *Gauss : la première dépendance.*

Les deux définitions suivantes ne sont valides que dans le cas de systèmes d'équations récurrentes homogènes. Pour que la notion de *vecteur* ait un sens, il faut que l'origine et l'extrémité du vecteur soient plongées dans un espace de même dimension, et donc, dans les définitions suivantes, que tous les ensembles d'utilisation et d'émission soient dans un espace de même dimension. Il faut donc que les domaines de définition des variables soient de même dimension. Ce n'est clairement pas le cas lorsque le PARE n'est pas homogène.

Définition 9 (vecteur de dépendance) *On appelle* vecteur de dépendance *un vecteur reliant un point d'émission* $z_0 \in \mathrm{Emit}_{E,Y}(p)$ *à un point de l'ensemble d'utilisation correspondant* $\mathrm{Util}_{E,Y}(z_0, p)$.

Définition 10 (vecteur de dépendance extrémal) *On appelle* vecteur de dépendance extrémal *un vecteur reliant un point d'émission* $z_0 \in \mathrm{Emit}_{E,Y}(p)$ *à un sommet de l'ensemble d'utilisation correspondant* $\mathrm{Util}_{E,Y}(z_0, p)$.

Exemple : algorithme de Gauss (suite)

Ce système d'équations récurrentes est homogène. Les figures ci-dessous représentent les ensembles d'utilisation et les vecteurs de dépendance extrémaux correspondant à chaque dépendance. Numérotons les dépendances de l'équation principale de la manière suivante :

$$A[i, j, k] = A[i, j, k-1]_1 - A[i, k, k-1]_2 * A[k, j, k-1]_3 / A[k, k, k-1]_4$$

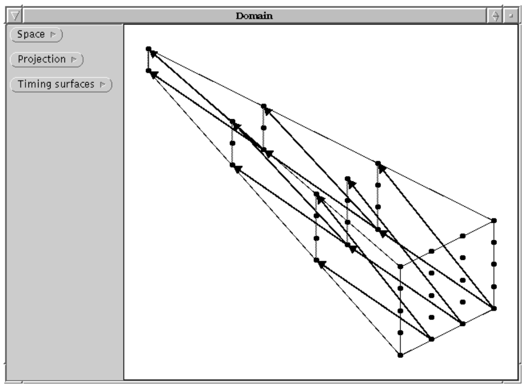


FIG. 1.3 - *Gauss : la deuxième dépendance et les ensembles d'utilisation* Util_2.

– première dépendance (figure 1.2) de $A[i,j,k-1]$ vers $A[i,j,k]$.
L'ensemble d'utilisation $\mathrm{Util}_1(z_0,N)$ est :

$$\mathrm{Util}_1\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \begin{pmatrix} i_0 \\ j_0 \\ k_0+1 \end{pmatrix}$$

et l'ensemble d'émission :

$$\mathrm{Emit}_1(N) = \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathbb{Z}^3 \;\middle|\; \begin{array}{c} k+2 \le i \le N, k+2 \le j \le N+1, \\ 0 \le k \le N-1 \end{array} \right\}$$

Il n'y a pas de vecteur d'utilisation. C'est une dépendance uniforme.

– deuxième dépendance (figure 1.3) de $A[i,k,k-1]$ vers $A[i,j,k]$.
L'ensemble d'utilisation $\mathrm{Util}_2(z_0,N)$ est :

$$\mathrm{Util}_2\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \left\{ \begin{pmatrix} i_0 \\ j \\ k_0+1 \end{pmatrix} \;\middle|\; k_0+1 \le j \le N+1 \right\}$$

et l'ensemble d'émission :

$$\mathrm{Emit}_2(N) = \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathbb{Z}^3 \;\middle|\; k+2 \le i \le N, j = k+1, 0 \le k \le N-1, \right\}$$

C'est une dépendance affine de dimension 1, et le vecteur d'utilisation est $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$.

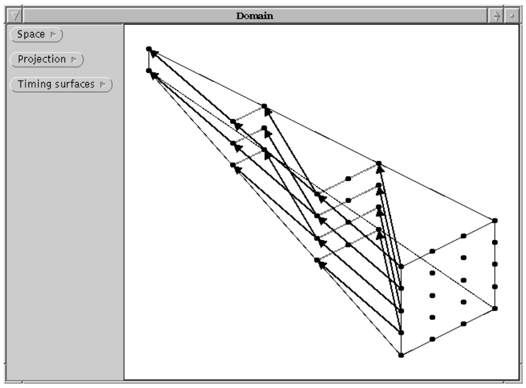


FIG. 1.4 - *Gauss: la troisième dépendance et les ensembles d'utilisation* Util_3.

– troisième dépendance (figure 1.4) de $A[k,j,k-1]$ vers $A[i,j,k]$.
L'ensemble d'utilisation $\mathrm{Util}_3(z_0,N)$ est :

$$\mathrm{Util}_3\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \left\{ \begin{pmatrix} i \\ j_0 \\ k_0+1 \end{pmatrix} \middle| \, k_0+1 \le i \le N+1 \right\}$$

et l'ensemble d'émission :

$$\mathrm{Emit}_3(N) = \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathbb{Z}^3 \middle| \begin{array}{c} i=k+1, k+2 \le j \le N+1, \\ 0 \le k \le N-2 \end{array} \right\}$$

C'est une dépendance affine de dimension 1, et le vecteur d'utilisation est $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$.

– quatrième dépendance (figure 1.5) de $A[k,k,k-1]$ vers $A[i,j,k]$.
L'ensemble d'utilisation $\mathrm{Util}_4(z_0,N)$ est :

$$\mathrm{Util}_4\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \left\{ \begin{pmatrix} i \\ j \\ k_0+1 \end{pmatrix} \middle| \begin{array}{c} k_0+1 \le i \le N+1, \\ k_0+1 \le j \le N+1 \end{array} \right\}$$

et l'ensemble d'émission :

$$\mathrm{Emit}_4(N) = \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathbb{Z}^3 \middle| \, i=k+1, j=k+1, 0 \le k \le N-2, \right\}$$

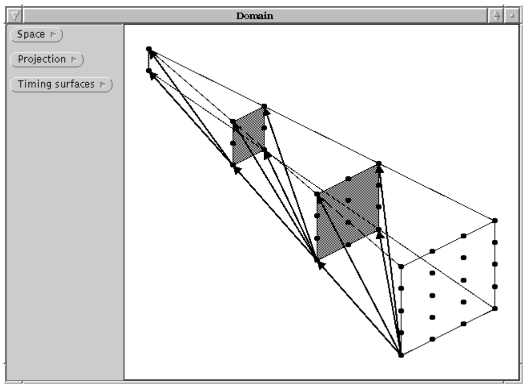


FIG. 1.5 - *Gauss : la quatrième dépendance et les ensembles d'utilisation* Util_4.

C'est une dépendance affine de dimension 2, et les vecteurs d'utilisation sont par exemple $\begin{pmatrix}1\\0\\0\end{pmatrix}$ et $\begin{pmatrix}0\\1\\0\end{pmatrix}$. Notons que ce couple n'est pas unique : tout couple de vecteurs de base du plan d'équation $k = 0$ parallèle aux ensembles d'utilisation convient.

1.3 Cadencement

1.3.1 Calculabilité d'un système d'équations récurrentes

Les contraintes sur le cadencement sont appelées *contraintes causales*. Ces contraintes, liées aux dépendances des données, expriment le fait qu'une donnée doit être calculée avant d'être utilisée par un autre calcul.

Remarquons avant tout que les systèmes d'équations récurrentes ne sont pas toujours calculables : s'il y a un cycle dans le calcul d'une donnée par exemple, il n'y a pas de cadencement possible.

Les premiers travaux sur le cadencement de systèmes d'équations récurrentes uniformes homogènes ont été effectués par Karp, Miller et Winograd [KMW67], dont nous rappelons brièvement les résultats.

Définition 11 (graphe réduit des dépendances) *Le* graphe réduit des dépendances *d'un SURE homogène est défini par* $G = (V_G, E_G)$*, où les sommets* V_G *sont constitués des variables du SURE, et les arcs* E_G *correspondent aux dépendances : il y a un arc entre deux sommets* Y *et* X*, s'il existe une équation dans le SURE où* X *est la partie gauche et* Y *est un membre de la partie droite de l'équation. Le vecteur de dépendance correspondant est associé à cet arc.*

Définition 12 (dépendance) *Soit $\mathcal{I}$ l'ensemble des couples (X, x), où X est une variable, et $x \in \mathcal{D}_X$ un point du domaine de définition de X. On dit que (X, x)* dépend de *(Y, y), $y \in \mathcal{D}_Y$, s'il existe une équation de la forme $X(x) = ...\ Y(y)\ ...$, et on note cette relation $(X, x) \leftarrow (Y, y)$.*

Un élément de l'ensemble $\mathcal{I}$ est appelé une instance.

Définition 13 (graphe étendu des dépendances, ou graphe de dépendance) *Le* graphe de dépendance *est un graphe Γ dont chaque noeud représente un point du domaine d'itération du SURE homogène, et dont les arcs sont constitués des dépendances entre les instances du SURE.*

Définition 14 (cadencement) *On appelle* cadencement par variable *un ensemble de fonctions associées à chaque variable X, de la forme : $t_X : \mathcal{I} \rightarrow \mathbb{Z}$ telles que :*

$$(X, x) \leftarrow (Y, y) \qquad \Rightarrow \qquad t_X(x) > t_Y(y)$$

Définition 15 (cadencement libre) *Un cadencement est dit* libre *si, pour toute variable X, on a :*

- *$t_X(x) = 0$ si (X, x) n'a pas de prédécesseur dans le graphe de dépendance Γ,*
- $t_X(x) = 1 + max \left\{ t_Y(y) \;\middle|\; \begin{array}{l} \textit{pour tout } (Y,y) \textit{ tel qu'il existe un chemin} \\ \textit{de } (X,x) \textit{ vers } (Y,y) \end{array} \right\}.$

Le cadencement libre consiste à associer à chaque point de calcul l'instant de calcul "au plus tôt". S'il existe un cadencement, le cadencement libre est unique et c'est le cadencement optimal. Il correspond au plus court chemin dans le graphe de dépendance.

Définition 16 (cadencement linéaire entier) *Un cadencement est dit* cadencement linéaire entier *si toutes les fonctions de cadencement sont de la forme : $t^l_X : \mathcal{I}_X \rightarrow \mathbb{N}$ tel que :*

$$t^l_X(z) = \lambda_X z$$

avec $\lambda_X \in \mathbb{Z}^n$.

Les deux théorèmes suivants, découverts par Karp, Miller et Winograd, permettent de trouver un cadencement linéaire optimal d'un SURE, dans le cas d'un cadencement unique $t^l(z)$ pour toutes les variables du SURE.

On appelle cycle non positif dans un graphe un cycle dont la somme des vecteurs de dépendances des arcs parcourus est un vecteur dont tous les éléments sont non positifs.

Théorème 1 *Soit G le graphe réduit des dépendances d'un SURE homogène. Une variable X est calculable si et seulement si G ne contient pas de chemin d'une variable Y vers X et où Y est contenu dans un cycle non positif.*

Théorème 2 *Soit une équation d'un SURE, définie sur un domaine $\mathcal{D}$, de la forme :*

$$X(z) = f[X(z-d_1), ..., X(z-d_s)]$$

Les propositions suivantes sont équivalentes :

- $X(z)$ *est calculable,*
- *il existe un vecteur* $(u_1, ..., u_s)$*, avec* $u_i \geq 0$ *pour tout* i*, tel que :*

$$\sum_{j=1}^{s} -u_j d_j \geq 0,$$

- *soit* $\lambda = (\lambda_1, ..., \lambda_n)$ *un vecteur de* $\mathbb{Z}^n$*, où* n *est la dimension du domaine* $\mathcal{D}$*. Le système d'inégalités suivant a une solution :*

$$\begin{cases} d_j\lambda \geq 1, & 1 \leq j \leq s \\ \lambda_i \geq 0, & 1 \leq i \leq n \end{cases}$$

- *pour tout* $z \in \mathcal{D}$*, les deux systèmes suivants ont une valeur optimale* $m(z)$ *commune :*

$$\begin{cases} z - \sum_{j=1}^{s} u_j d_j \geq 0, \\ u_j \geq 0, 1 \leq j \leq s, \\ max\ \sum_{i=1}^{s} u_j \end{cases} \qquad \begin{cases} \lambda_i \geq 0, 1 \leq i \leq n, \\ d_j\lambda \geq 1, 1 \leq j \leq s, \\ min\ z\lambda \end{cases}$$

Cette valeur $m(z)$ est le cadencement linéaire optimal. Darte et al. [DKR91] ont montré que le cadencement linéaire optimal est aussi efficace que le cadencement libre optimal, bien que ce ne soit pas le cas pour certains points se trouvant au bord du domaine d'itération.

En 1984, Quinton [Qui84b] propose de choisir des cadencement affines par variable, dont la partie constante est différente pour chaque variable, mais de partie linéaire unique λ :

$$t_X(z) = \lambda z + \alpha_X.$$

Le modèle est généralisé en 1985 par Rao [Rao85] dans le cas des équations récurrentes uniformes, et par Mauras et al. [MQRS90] dans le cas des équations récurrentes affines, en utilisant une partie linéaire différente pour chaque variable :

$$t_X(z) = \lambda_X z + \alpha_X.$$

Dans le cas paramétré, on utilise un cadencement paramétré :

$$t_X(z, p) = \lambda_X z + \lambda'_X p + \alpha_X.$$

Toutes ces extensions sont décrites en détail dans [DR92].

Enfin, la dernière extension concerne le cadencement multidimensionnel [Fea92b] :

$$t_X : \mathcal{I} \to \mathbb{Z}^d.$$

Dans ce cas, le temps est égrené sur le réseau de processeurs par d boucles imbriquées. Les contraintes causales sont similaires au cas général, en remplaçant les inégalités habituelles par des inégalités lexicographiques.

Le cas général de recherche de cadencement multidimensionnel affine d'un PARE est présenté ci-dessous. Considérons une dépendance de la forme :

$$X[z] = ...\ Y[g(z,p)]\ ... \tag{E}$$

il faut que le cadencement vérifie :

$$t_X(z,p) - t_Y(g(z,p),p) \succ 0 \tag{1.6}$$

où $\succ$ désigne l'opérateur *supérieur* lexicographique :

$$\begin{pmatrix} t_1 \\ \vdots \\ t_{d_\lambda} \end{pmatrix} \succ 0 \Leftrightarrow \left\{ \begin{array}{ll} & t_1 > 0 \\ \text{ou} & t_1 = 0 \text{ et } t_2 > 0 \\ \text{ou} & ... \end{array} \right.$$

Un cadencement respectant l'ensemble de ces contraintes est obtenu grâce à l'une des deux méthodes classiques décrites dans les deux paragraphes suivants : la méthode de Farkas, ou la méthode des sommets.

1.3.2 Méthode de Farkas

Cette méthode repose sur la forme affine du lemme de Farkas [Sch86] rappelé ci-dessous :

Théorème 3 (Forme affine du lemme de Farkas) *Soit $\mathcal{D}$ un polyèdre convexe non vide défini par l inégalités :*

$$a_k z + c_k \geq 0, \quad 1 \leq k \leq l$$

La valeur de $\phi(z)$ est positive ou nulle pour tout point z de $\mathcal{D}$ si et seulement si $\phi(z)$ est une combinaison affine positive des inégalités :

$$\phi(z) = \delta_0 + \sum_{k=1}^{l} \delta_k (a_k z + c_k)$$

où $\delta_0, ..., \delta_l$ sont des constantes positives ou nules.

Le résultat suivant est donné pour un cadencement de dimension 1. Il se généralise facilement au cas multidimensionnel.

Soit Δ la fonction donnant le délai entre deux calculs donnés. C'est une fonction affine positive ou nulle dans le domaine $\mathcal{D}_E(p)$:

$$\Delta(z) = t_X(z,p) - t_Y(g(z,p),p) - 1$$

Le lemme de Farkas garantit l'existence d'un vecteur $\delta \geq 0$ tel que :

$$t_X(z,p) - t_Y(g(z,p),p) - 1 = \delta_0 + \sum_{k=1}^{l} \delta_k(a_k z + b_k p + c_k)$$

où $a_k z + b_k p + c_k \geq 0$ est l'ensemble des l inégalités définissant $\mathcal{D}_E(p)$.

Chaque formule de ce type est une identité stricte, on peut donc sortir chaque coefficient en z et en p et créer un nouveau système de contraintes indépendant de z et de p. Le résultat est un système d'inégalités en δ, relativement grand, qu'on peut résoudre en utilisant par exemple l'algorithme du simplexe.

1.3.3 Méthode des sommets

Chaque contrainte causale (équation (1.6)) peut aussi s'écrire :

$$\begin{aligned} & t_X(z,p) - t_Y(g(z,p),p) \succ 0 && (1.7)\\ \Leftrightarrow\ & t_X(z,p) - t_Y((Rz+Qp+r),p) \succ 0 \\ \Leftrightarrow\ & (\lambda_X - \lambda_Y \cdot R)z - \lambda_Y(Qp+r) + \lambda'_X p - \lambda'_Y p + \alpha_X - \alpha_Y \succ 0 && (1.8) \end{aligned}$$

Quinton et Van Dongen [QV89] proposent de considérer les paramètres comme des indices, en fusionnant le vecteur z et le vecteur p, afin de trouver une solution valide. Il suffit que les contraintes causales soient vérifiées dans chaque équation E pour tout z extrémal des domaines $\mathcal{D}_E(p)$ et du domaine de validité de p.

Cependant cette méthode ne permet pas de trouver le cadencement optimal. Pour trouver le cadencement optimal, il faut connaître les sommets du polyèdre paramétré $\mathcal{D}_E(p)$, en fonction des paramètres. La deuxième partie de ce document donne une méthode de recherche des sommets d'un polyèdre paramétré, en fonction des paramètres.

Le système de contraintes sur les matrices de cadencement résultant de cette première phase est un ensemble de contraintes linéaires en λ_X et λ_Y, multipliées par des inconnues : les paramètres p. Il faut considérer les valeurs extrémales que peut prendre le vecteur p (les sommets du domaine de validité des paramètres $\mathcal{D}_p$) pour simplifier ce système.

Dans la plupart des cas, $\mathcal{D}_p$ n'est pas borné. C'est un polyèdre constitué de sommets, de rayons (vecteurs unidirectionnels) et de lignes (vecteurs bidirectionnels). Le système de contraintes de la forme $Ax \succ b$ doit être vérifié :

– pour tout sommet v :

$$Av \succ b$$

– pour tout rayon r, en supprimant la partie constante des inégalités :

$$Ar \succeq 0$$

– pour toute ligne l, en supprimant la partie constante des inégalités et en les transformant en égalités :

$$Al = 0.$$

1.3.4 Surfaces de temps

Définition 17 (surface de temps) *On appelle* surface de temps *à l'instant t_0, associée à une variable X d'un PARE, l'ensemble des points z du domaine de définition $\mathcal{D}_X$ vérifiant $t_X(z, p) = t_0$.*

Les surfaces de temps contiennent les ensembles de points de calculs isotemporels. Ce sont des polyèdres paramétrés par p et par t_0, qui vérifient de façon évidente la propriété suivante :

Propriété 3 *Les surfaces de temps de la variable X sont orthogonales aux vecteurs lignes de la matrice λ_X, appelés* vecteurs de cadencement *de X.*

1.4 Allocation

La plupart des méthodes de parallélisation respectent la règle de l'*owner computes*, c'est-à-dire que l'instruction calculant une donnée est effectuée par le processeur sur lequel est allouée cette donnée. Dion et Robert [DR95] proposent une méthode d'allocation relâchant cette contrainte, qui peut entraîner une perte d'optimalité du réseau de processeurs en terme de communications.

Löwe, Eisenbiegler et Zimmermann [LEZ96] proposent une méthode de duplication des calculs dans le réseau de processeurs afin d'obtenir une meilleure solution finale.

Ces optimisations amènent de meilleurs résultats dans certains cas, mais l'ordre de grandeur du volume de communications des solutions obtenues reste le même. Dans la suite de ce document, nous nous plaçons dans le cadre de l'"*owner computes rule*", sans duplication des calculs.

La méthode de projection, avec un cadencement monodimensionnel donc, dans le cas des équations récurrentes homogènes, consiste à chercher un *vecteur d'allocation* ξ. Les processeurs virtuels sont déterminés par projection selon ce vecteur. Les communications entre processeurs sont le résultat des projections des vecteurs de dépendance. Le nombre de communications du réseau synthétisé est donc lié au choix du vecteur d'allocation : si ξ est parallèle à un vecteur de dépendance, cette dépendance est projetée sur un seul processeur, et il n'y a pas de communication nécessaire. La méthode de recherche de vecteur d'allocation optimal consiste à *localiser* de cette manière le plus de communications possibles.

La matrice de transformation σ qui correspond à la projection selon le vecteur ξ peut être déterminée en utilisant l'algorithme de Blankinship [Bla63, Mon85], ou de manière équivalente en faisant une décomposition sous forme normale de Hermite [Sch86].

$$\begin{array}{rrcl} \mathrm{alloc}(z): & \mathbb{Z}^n & \rightarrow & \mathbb{Z}^{n-1} \\ & z & \mapsto & \sigma z \end{array}$$

D'autres transformations non linéaires peuvent être appliquées après cette projection afin de minimiser le nombre de processeurs [Cla90, Dar93, WD89] :

- le *groupement* (*"clustering"* ou *"tiling"*) consiste à regrouper sur un seul processeur des processeurs virtuels voisins ayant des périodes d'activité disjointes et cycliques. Ce cas se présente lorsque le produit scalaire du vecteur de cadencement et du vecteur d'allocation est différent de 1 ;
- la sérialisation (*"pipeline"*) consiste à regrouper des processeurs virtuels dont la période d'activité est terminée, avec des processeurs qui n'ont pas encore été actifs. L'ensemble des processeurs actifs à un instant donné est appelé le *cône de dépendance*. C'est un polyèdre, la projection selon ξ d'une coupe de l'espace d'itération, qui se déplace selon un vecteur constant (la projection du vecteur de cadencement).

La solution moins restrictive que la méthode de projection que nous utilisons, avec cadencement multidimensionnel, et qu'on peut appliquer aux systèmes d'équations récurrentes non homogènes, consiste à chercher une fonction d'allocation de la forme suivante pour chaque variable. Pour toute variable X du PARE, on pose :

$$\mathrm{alloc}_X(z) = \sigma_X z + \gamma_X$$

Comme dans la méthode de projection classique, des contraintes de minimisation des communications peuvent être appliquées aux matrices σ_X et aux vecteurs γ_X. Ces contraintes sont présentées au chapitre 2.

1.5 Transformations par variable

Dans la suite de ce document, nous cherchons une solution parallèle à un PARE sous forme d'une transformation espace-temps affine pour chaque variable du PARE. Pour chaque variable X, la transformation espace-temps est composée :

- d'une fonction d'allocation affine de la forme :

$$\begin{array}{rrcl} \mathrm{alloc}_X: & \mathcal{D}_X & \rightarrow & \mathbb{Z}^{d_\sigma} \\ & z & \mapsto & \sigma_X z + \gamma_X \end{array}$$

où σ_X est une matrice entière de d_σ lignes, et γ_X est un vecteur entier de dimension d_σ. Cette fonction définit le processeur virtuel dans un espace de dimension d_σ sur lequel est exécutée l'instruction correspondant au calcul de $X[z]$.

– d'une fonction de cadencement multidimensionnel de la forme :

$$\begin{array}{rcl} t_X : \mathcal{D}_X, \mathcal{D}_E & \rightarrow & \mathbb{Z}^{d_\lambda} \\ (z,p) & \mapsto & \lambda_X z + \lambda'_X p + \alpha_X \end{array}$$

où λ_X et λ'_X sont des matrices entières ayant d_λ lignes, et α_X est un vecteur entier de dimension d_λ. $t_X(z,p)$ définit l'instant d'exécution de l'instruction calculant $X[z]$.

On appelle *matrice de transformation* de la variable X la matrice :

$$T_X = \begin{pmatrix} \lambda_X \\ \sigma_X \end{pmatrix}$$

Cette matrice compte $Dim(D_X(p))$ colonnes, et $d_\lambda + d_\sigma$ lignes.

La solution parallèle est valide si deux instructions correspondant au calcul d'une variable donnée ne sont pas effectuées au même instant et sur le même processeur virtuel. Cela s'exprime par la contrainte sur les matrices de transformation : les lignes de chaque matrice de transformation T_X doivent être linéairement indépendantes, *i.e.* T_X doit être "*full-row-rank*".

Cette contrainte entraîne une *contrainte de compatibilité* entre la matrice d'allocation et matrice de cadencement de chaque variable. L'allocation optimale associée à un critère donné (la minimisation des communications par exemple) n'est pas toujours compatible avec le cadencement optimal.

Exemple : algorithme de Gauss (suite)

Cherchons une solution parallèle à l'algorithme de Gauss sur un réseau de processeurs de dimension 2, avec un cadencement de dimension 1. Il n'y a qu'une seule variable A dans ce PARE. Il est donc inutile d'utiliser un cadencement dépendant de paramètres, car les instants d'exécution de ces calculs peuvent commencer à l'instant 0 par exemple. Le vecteur λ'_A est nul. La fonction de cadencement est donc :

$$\begin{array}{rcl} t_A(z) & = & \lambda_A z + \alpha_A \\ & = & \begin{pmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{pmatrix} z + \alpha_A \end{array}$$

et la fonction d'allocation est :

$$\begin{array}{rcl} \mathrm{alloc}_A(z) & = & \sigma_A z + \gamma_A \\ & = & \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\ \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \end{pmatrix} z + \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \end{array}$$

Les contraintes causales correspondant aux quatre dépendances sont déduites de l'équation (1.8). Ce sont des auto-dépendances, cette équation se réécrit donc :

$$\begin{array}{rrcl} & (\lambda_A - \lambda_A \cdot R)\ z - \lambda_A(Qp + r) + \alpha_A - \alpha_A & \succ & 0 \\ \Leftrightarrow & \lambda_A \cdot (Id - R)\ z - \lambda_A(Qp + r) & \succ & 0 \end{array}$$

Comme les fonctions d'index sont toutes caractérisées par $Q = 0$, cette équation se simplifie en :

$$\lambda_A \cdot (Id - R)\ z - \lambda_A r \ \succ \ 0$$

- la première dépendance uniforme $A[i, j, k] \leftarrow A[i, j, k - 1]$ est caractérisée par $R = Id$, d'où la contrainte :

$$-\lambda_A \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} > 0$$

- la deuxième dépendance $A[i, j, k] \leftarrow A[i, k, k - 1]$, est caractérisée par la matrice $R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$, d'où la contrainte :

$$\lambda_A \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} z - \lambda_A \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} > 0$$

- la troisième dépendance $A[i, j, k] \leftarrow A[k, j, k - 1]$, est caractérisée par la matrice $R = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, d'où la contrainte :

$$\lambda_A \cdot \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\ z - \lambda_A \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} > 0$$

- la quatrième dépendance $A[i, j, k] \leftarrow A[k, k, k - 1]$, est caractérisée par la matrice $R = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$, d'où la contrainte :

$$\lambda_A \cdot \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}\ z - \lambda_A \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} > 0$$

Le point z est contenu dans le polyèdre $\mathcal{D}_A = \{i, j, k | k+1 \leq i \leq N, k+1 \leq j \leq N+1; 1 \leq k \leq N-1\}$. En appliquant la méthode des sommets, on trouve le système de contraintes :

$$\begin{cases} \lambda_1 > 0 \\ \lambda_2 > 0 \\ \lambda_3 > 0 \end{cases}$$

La fonction de temps optimale est :

$$t_a(z) = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} z$$

L'allocation minimisant les communications dans ce problème est la projection selon un des vecteurs d'utilisation : le vecteur $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ par exemple, qui donne un réseau de processeurs de taille légèrement inférieur à celui obtenu par projection selon l'autre vecteur d'utilisation $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$.

La fonction d'allocation est donc par exemple :

$$\mathrm{alloc}_A(z) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} z$$

La matrice de transformation résultante est :

$$T_A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Elle est valide, puisque $Rang(T_A) = 3$.

1.6 Génération de code

Deux types de code peuvent être produits :

- le code SIMD, où les boucles les plus externes égrenent le temps ;
- le code SPMD, où les boucles externes décrivent les processeurs, et les boucles internes le temps.

Le code SIMD est obtenu en appliquant les matrices de transformation $T_X = \begin{pmatrix} \lambda_X \\ \sigma_X \end{pmatrix}$, et le code SPMD par inversion des matrices λ_X et σ_X : $T_X = \begin{pmatrix} \sigma_X \\ \lambda_X \end{pmatrix}$. Les contraintes sur les matrices λ_X et σ_X sont identiques dans les deux cas.

La transformation espace-temps correspond à une transformation des espaces d'itération $z \in \mathcal{D}_X(p)$ vers un espace temps-processeurs (t, a). Les références aux éléments des variables sont également modifiés : puisque les boucles dépendent de t et de a, il faut retrouver les coordonnées en z correspondantes pour pouvoir appliquer la fonction d'index g, et donc calculer les fonctions de transformation inverses. Notons Γ_X la fonction de transformation associée à chaque variable X du PARE. On a la relation suivante pour tout $z \in \mathcal{D}_X(p)$:

$$\begin{pmatrix} t \\ a \end{pmatrix} = \Gamma_X(z) = \begin{pmatrix} \lambda_X \\ \sigma_X \end{pmatrix} z + \begin{pmatrix} \lambda'_X p + \alpha_X \\ \gamma_X \end{pmatrix}$$

Une équation de la forme

$$X[z] = \ldots\ Y[g(z, p)]\ \ldots$$

est donc transformée en :

$$X\left[\begin{pmatrix} t \\ a \end{pmatrix}\right] = \ldots\ Y\left[\Gamma_Y\left(g\left(\Gamma_X^{-1}\begin{pmatrix} t \\ a \end{pmatrix}, p\right)\right)\right]\ \ldots$$

Le programme parallèle consiste à parcourir les domaines de calcul dans l'espace temps-processeurs, avec un parcours séquentiel des instants d'exécution (boucle **DO** t) et un parcours parallèle des processeurs (boucle **DOALL** a). Si les transformations sont unimodulaires, les domaines à parcourir sont des polyèdres paramétrés denses. Les boucles de parcours peuvent être obtenues en utilisant soit l'algorithme de Fourier-Motzkin [MRTT53, AI91, Cha93, LVW94], soit l'algorithme du simplexe [CFR95]. Dans le cas de systèmes définis sur plusieurs domaines, l'algorithme de Fourier-Motzkin est étendu afin de parcourir la superposition de ces domaines [KPR95, WR95, Wil95, QRW95a, QRW95b, WR97, Alb97].

Le choix de matrices de transformation non-unimodulaires conduit à parcourir des Z-polyèdres [Ram92, LP92, BL92]. Cependant, le problème du parcours d'une *superposition* de Z-polyèdres paramétrés reste ouvert, car l'union de deux Z-polyèdres n'est pas forcément un Z-polyèdre. Cette notion de Z-polyèdre doit être étendue afin d'obtenir un ensemble d'objets géométriques stables par l'union, l'intersection, et la différence. Dans la suite de ce travail, nous nous restreignons aux matrices de transformation unimodulaires.

Les langages CSP ou ADA, ou les bibliothèques PVM et MPI, fournissent des primitives explicites de communication. La génération de communications explicites à partir de systèmes d'équations récurrentes est décrite par Reffay [Ref96]. Générer le code des communications consiste à trouver les boucles de parcours des ensembles d'émission, et pour chaque point d'émission, trouver le parcours de l'ensemble d'utilisation correspondant, afin de transmettre chaque donnée aux calculs l'utilisant.

Exemple : algorithme de Gauss (suite)

Ceci est un exemple le code SIMD sans communications explicites pour l'algorithme de Gauss, correspondant à la transformation définie ci-dessus.

```
do t = 2 to 4
   doall a1 = max(t-N-1,1) to min(N,t-1)
      doall a2 = 0
         A[t,a1,a2] = m[a1,t-a1-a2]
      done
   done
done
do t = 5 to 2N+1
   doall a1 = max(t-N-1,1) to min(N,t-1)
      doall a2 = 0
         A[t,a1,a2] = m[a1,t-a1-a2]
      done
   done
   doall a1 = max((t-N)/2, 2) to min(N, t-3)
      doall a2 = max(1,t-a1-N-1) to min(a1-1, (t-a1-1)/2)
         A[t,a1,a2] = A[t-1,a1,a2-1] - A[a1+2a2-1,a1,a2-1] *
                                       A[t-a1+a2-1,a2,a2-1] / A[3a2-1,a2,a2-1]
      done
   done
done
do t = 2N+2 to 3N
   doall a1 = max((t-N)/2, 2) to min(N, t-3)
      doall a2 = max(1,t-a1-N-1) to min(a1-1, (t-a1-1)/2)
         A[t,a1,a2] = A[t-1,a1,a2-1] - A[a1+2a2-1,a1,a2-1] *
                                       A[t-a1+a2-1,a2,a2-1] / A[3a2-1,a2,a2-1]
      done
   done
done
```

Un exemple plus complet, avec transformations par variable et cadencement multidimensionnel, est présenté à la fin du chapitre 2.

Chapitre 2

Optimisation des communications

2.1 Les communications

2.1.1 Introduction

Une part importante du temps d'exécution d'un programme sur une machine parallèle est consacré aux communications. Si le programmeur est inattentif à ce problème, dans le cas d'une machine à mémoire distribuée ou d'une architecture NUMA [1], l'exécution sur la machine parallèle peut même devenir plus lente que sur une machine séquentielle.

De nombreuses recherches ont été effectuées dans ce domaine. Les langages data-parallèles incluent des directives d'*alignement des données*. Les compilateurs paralléliseurs proposent diverses techniques utilisant ces directives, permettant de diminuer le nombre de communications. Huang et Sadayappan [HS91], et Ramanujam et Sadayappan [RS91], ainsi que Li et Chen [LC91] sont à l'origine de ces travaux, formalisés à l'aide de l'algèbre linéaire par Bau et al. [BKK+95] et Diderich et Gengler [DG96]. De nombreuses équipes à travers le monde ont mené des recherches complémentaires à celles-ci : [WL91, AL93, LL94, Lam94], [KLS90], [KM92], [LP92], [AI92, AICK93], [Fea94, DR94, DR95, Pla95], [CGO+96].

Ce chapitre présente une méthode originale de recherche de matrices d'allocation minimisant les communications de la solution parallèle d'un système d'équations récurrentes affines. Cette méthode, développée en collaboration avec Catherine Mongenet, a fait l'objet d'une communication lors de la conférence Euro-Par'97 [LM97].

Nous faisons la distinction entre deux types de communications :

– les communications *locales* : communications entre processeurs situés à une distance constante,

1. Non Uniform Memory Acces.

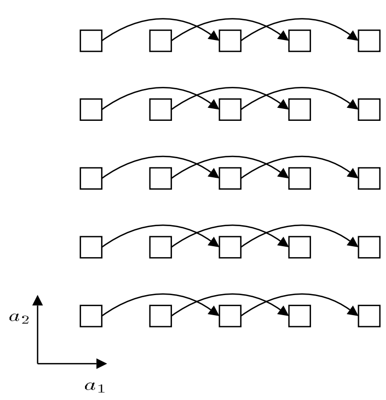


FIG. 2.1 - *Communications locales :* $(a_1, a_2) \rightarrow (a_1 + 2, a_2)$.

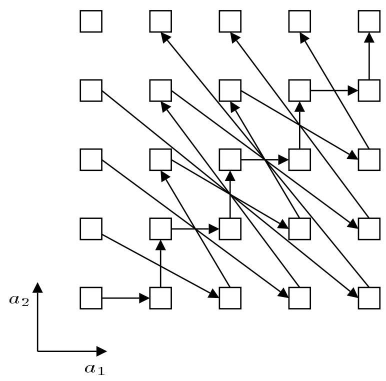


FIG. 2.2 - *Communications distantes :* $(a_1, a_2) \rightarrow (a_2 + 1, a_1)$.

– les communications *distantes* : communications de vecteur de communication [2] variable.

Les figures 2.1 et 2.2 donnent respectivement un exemple de communications locales et de communications distantes : considérons une architecture parallèle composée d'un réseau de processeurs virtuels de dimension 2, dans un repère (a_1, a_2). Les communications locales entre un ensemble de processeurs virtuels (a_1, a_2) et leurs voisins de coordonnées $(a_1 + 2, a_2)$ à un instant donné, selon le vecteur constant $(2, 0)$, est représenté figure 2.1. La figure 2.2 représente les communications entre (a_1, a_2) et $(a_2 + 1, a_1)$. Il s'agit d'une communication distante selon le vecteur $(a_2 - a_1 + 1, a_1 - a_2)$. A chaque instant, tout

2. On appelle *vecteur de communication* un vecteur ayant pour origine un processeur virtuel envoyant une donnée, et pour extrémité le processeur virtuel qui la reçoit.

processeur virtuel de coordonnées (a_1, a_2) envoie une donnée à un autre processeur virtuel selon ce vecteur.

Nous faisons cette distinction, en considérant que les communications locales sont effectuées de manière plus efficace que les communications distantes par certaines machines parallèles, notamment les architectures massivement parallèles à mémoire distribuée de type MasPar par exemple. Sur ces machines parallèles, le temps d'exécution d'un ensemble de communications locales est du même ordre que le temps nécessaire à l'exécution d'une seule de ces communications.

Sur des machines parallèles dont le nombre de processeurs est plus petit, comme le Cray T3D/E ou le nCUBE, l'architecture du réseau de type hypercube implique que les communications distantes se font à des distances assez courtes, et ne sont donc pas très coûteuses. Cependant, lorsque de nombreuses communications distantes ont lieu au même instant, la charge globale du réseau est plus importante, et risquent de provoquer des goulots d'étranglement sur le réseau.

Une autre solution qui peut être efficace sur certaines de ces architectures est de faire de la *diffusion*: envoyer une donnée calculée à un *ensemble* de processeurs cibles, en une seule instruction. La donnée doit dans ce cas être reçue par les processeurs cibles et stoquée en mémoire jusqu'à son utilisation. Cette méthode peut être très coûteuse en mémoire.

D'autre part, la minimisation des communications distantes est une priorité si l'architecture cible est un circuit électronique spécialisé. En effet, dans ces circuits, la régularité et la localité des liens entre processeurs élémentaires sont très importantes pour une question de facilité de conception et de coût du circuit.

2.1.2 Modélisation des communications

Les communications sur le réseau de processeurs virtuels résultent des dépendances entre les variables du système d'équations récurrentes. Considérons l'équation :

$$X[z] = \dots\ Y[g(z,p)]\ \dots \tag{E}$$

Le vecteur de communication résultant de cette dépendance de Y vers X est :

$$\begin{aligned}
& \mathrm{alloc}_X(z) - \mathrm{alloc}_Y(g(z,p)) \\
= \;& \mathrm{alloc}_X(z) - \mathrm{alloc}_Y(Rz + Qp + r) \\
= \;& (\sigma_X - \sigma_Y \cdot R)z - \sigma_Y(Qp + r) + \gamma_X - \gamma_Y \\
= \;& (\sigma_X - \sigma_Y \cdot R)z + \rho(p) \qquad (2.1)
\end{aligned}$$

où $\rho(p) = -\sigma_Y(Qp + r) + \gamma_X - \gamma_Y$ est une fonction affine de p ne dépendant pas de z.

Si le vecteur de communication est nul, il n'y a pas de communication : la donnée peut alors être stockée dans une variable temporaire locale au processeur correspondant.

2.2 Optimisation des communications

2.2.1 Condition forte

A partir de l'équation 2.1, on trouve une condition *suffisante mais non nécessaire* pour éliminer les communications distantes [Fea94, DR95, Mon97]. Nous l'appelons *condition forte*:

Théorème 4 (condition forte) *La dépendance de Y vers X, de la forme:*

$$X[z] = \dots\ Y[Rz + Qp + r]\ \dots$$

ne génère pas de communications distantes si les matrices d'allocation de X et Y vérifient:

$$\sigma_X = \sigma_Y \cdot R$$

Preuve du théorème 4.
Lorsque la condition $\sigma_X = \sigma_Y \cdot R$ est vérifiée, l'équation (2.1) se réécrit:

$$\mathrm{alloc}_X(z) - \mathrm{alloc}_Y(g(z,p)) = \rho(p).$$

Les vecteurs de communication ne dépendent donc pas de z. Les communications sont locales.
□

Propriété 4 *Soit $z_0 \in \mathrm{Emit}_{E,Y}(p)$ un point d'un ensemble d'émission. Lorsque la condition forte est vérifiée, tous les points de l'ensemble d'utilisation $\mathrm{Util}_{E,Y}(z_0,p)$ associé à z_0 sont projetés sur le même processeur virtuel.*

Preuve.
Soit z_0 un point de $\mathrm{Emit}_{E,Y}(p)$, $z_0 = g(z,p)\ \forall z \in \mathrm{Util}_{E,Y}(z_0,p)$. L'équation (2.1) s'écrit:

$$\begin{aligned} & \mathrm{alloc}_X(z) - \mathrm{alloc}_Y(g(z,p)) = \rho(p) \qquad \forall z \in \mathrm{Util}_{E,Y}(z_0,p) \\ \Leftrightarrow\ & \mathrm{alloc}_X(z) = \rho(p) + \mathrm{alloc}_Y(z_0) \qquad \forall z \in \mathrm{Util}_{E,Y}(z_0,p) \end{aligned}$$

On en déduit que $\mathrm{alloc}_X(z)$ ne dépend pas de z quel que soit $z \in \mathrm{Util}_{E,Y}(z_0,p)$. Tous ces points z sont donc projetés sur le même processeur virtuel.
□

Il n'y a alors qu'une seule communication, du processeur correspondant au point d'émission vers le processeur correspondant à l'ensemble d'utilisation. Il y a cependant une contrainte importante: les calculs des points de l'ensemble d'utilisation ne peuvent être effectués au même instant, car un processeur ne doit pas exécuter plusieurs calculs au même instant.

Propriété 5 *La condition forte $\sigma_X = \sigma_Y \cdot R$ ne peut être satisfaite que si le rang de la matrice R est supérieur ou égal à d_σ, le nombre de lignes des matrices d'allocation.*

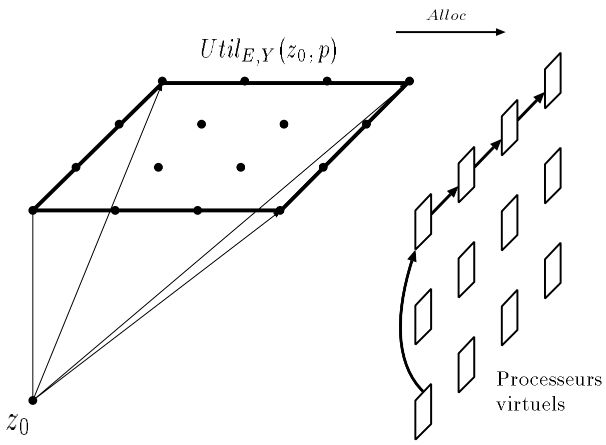


FIG. 2.3 - *Modèle de propagation.*

Preuve.

Les lignes des matrices d'allocation σ_X et σ_Y doivent être linéairement indépendantes (condition de validité des matrices de transformation). Lorsque le rang de la matrice R est inférieur au nombre de lignes des matrices σ_X et σ_Y, il n'existe pas de solution du système $\sigma_X = \sigma_Y \cdot R$ vérifiant cette condition de validité.

□

La condition forte est dans de nombreux cas trop restrictive pour être satisfaite par toutes les dépendances d'un PARE. Il existe une condition moins forte qui élimine les communications distantes dues à une dépendance, sans nécessairement projeter chaque ensemble d'utilisation sur un seul processeur. Cette condition est liée à un mécanisme de communications appelé modèle de propagation et décrit dans le paragraphe suivant.

2.2.2 Modèle de propagation

Considérons une dépendance de la forme :

$$X[z] = ...\ Y[g(z,p)]\ ... \qquad (E)$$

Soit z_0 un point d'émission, et $\mathrm{Util}_{E,Y}(z_0,p)$ l'ensemble d'utilisation correspondant. Supposons que le cadencement soit tel que le premier point actif de l'ensemble d'utilisation est unique. Dans le modèle de propagation, les communications générées par cette dépendance se font de la manière suivante (figure 2.3) :

- une première communication de la donnée ayant pour origine l'image par alloc_Y du point d'émission z_0, et pour destination l'image par alloc_X du premier point actif de l'ensemble d'utilisation,

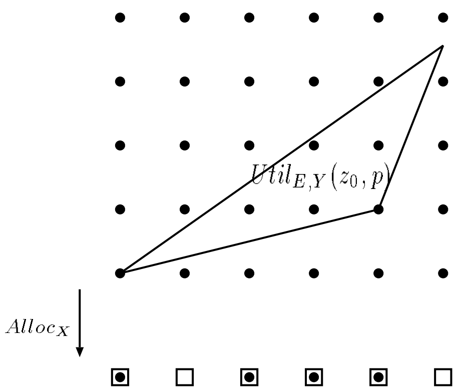


FIG. 2.4 - *Projection des points entiers d'un polyèdre.*

– cette donnée est ensuite transmise de voisin à voisin dans l'image par la fonction alloc_X de l'ensemble d'utilisation. Ces communications sont locales, car l'ensemble d'utilisation est l'ensemble des points entiers d'un polyèdre convexe : l'image de ces points entiers selon la fonction affine alloc_X est une partie d'un lattice régulier contenu dans un polyèdre convexe.

L'image des points entiers d'un ensemble d'utilisation rationnel $\mathrm{Util}_{E,Y}(z_0,p)$ n'est pas toujours exactement égale aux points entiers de l'image de $\mathrm{Util}_{E,Y}(z_0,p)$ (voir par exemple la figure 2.4). Cependant, dans notre modèle synchrone, la transmission de ces données à des processeurs ne les utilisant pas n'affecte pas le temps d'exécution total. La donnée est donc transmise à l'ensemble des processeurs correspondant aux points entiers de l'image de $\mathrm{Util}_{E,Y}(z_0,p)$. De plus, ce cas se présente rarement, sur les bords des ensembles d'utilisation (voir [Ref96] pour une analyse détaillée des projections de points entiers dans des convexes).

Propriété 6 *Lorsque le cadencement est affine, le premier point actif d'un ensemble d'utilisation, s'il est unique, est un des sommets de l'enveloppe convexe de ses points entiers.*

Preuve.

Un cadencement affine est défini par un ensemble de vecteurs de cadencement. Ces vecteurs, projetés sur l'espace affine portant l'ensemble d'utilisation, donnent l'ordre d'exécution des points entiers de cet espace. Il est évident que le premier point actif est un des sommets de l'enveloppe convexe des points entiers de cet ensemble d'utilisation.

□

Si le premier point actif n'était pas unique, l'ensemble des premiers points actifs serait une des faces de l'ensemble d'utilisation, définie par ses sommets entiers ayant l'instant d'exécution minimal. Il faudrait dans ce cas faire une diffusion vers les processeurs correspondants.

La condition faible, présentée dans le paragraphe ci-dessous, garantit l'absence de communications distantes. Dans le modèle de propagation que nous venons de décrire, les seules communications distantes éventuelles sont dues à la communication vers le premier point actif de l'ensemble d'utilisation uniquement. Après avoir appliqué la condition faible, on peut de plus éliminer certaines communications locales résiduelles en cherchant une allocation qui annule l'image de certains vecteurs d'utilisation, ou qui annule l'image du vecteur de communication de la première communication. La solution qui consiste à localiser la première communication, puis à annuler tous les vecteurs d'utilisation, est strictement équivalente à celle obtenue par la condition forte.

2.2.3 Condition faible

Tout sommet entier $v_i(z_0, p)$ de l'ensemble d'utilisation $\mathrm{Util}_{E,Y}(z_0, p)$ est une fonction affine de z_0 et p, sur un certain *domaine de validité* de z_0 et de p :

$$v_i(z_0, p) = A_i z_0 + B_i p + c_i \qquad \text{pour } z_0 \in Val_{z_0}, p \in Val_p$$

où Val_{z_0} est le domaine de validité, convexe, de z_0, et Val_p est le domaine de validité, convexe, de p. Le premier point actif de l'ensemble d'utilisation est un de ces points $v_i(z_0, p)$, suivant la fonction de cadencement. Appelons-le $v_0(z_0, p)$.

Dans ce cas, la première communication dans le modèle de propagation est :

$$\begin{aligned}
\mathrm{alloc}_X(v_0(z_0,p)) - \mathrm{alloc}_Y(z_0) &= \mathrm{alloc}_X(A_0 z_0 + B_0 p + c_0) - \mathrm{alloc}_Y(z_0) \\
&= \sigma_X \cdot A_0 z_0 + \sigma_X \cdot B_0 p + \sigma_X c_0 + \gamma_X - \sigma_Y z_0 - \gamma_Y \\
&= (\sigma_X \cdot A_0 - \sigma_Y) z_0 + \sigma_X \cdot B_0 p + \sigma_X c_0 + \gamma_X - \gamma_Y \\
\mathrm{alloc}_X(v_0(z_0,p)) - \mathrm{alloc}_Y(z_0) &= (\sigma_X \cdot A_0 - \sigma_Y) z_0 + \rho'(p) \qquad (2.2)
\end{aligned}$$

où $\rho'(p) = \sigma_X \cdot B_0 p + \sigma_X c_0 + \gamma_X - \gamma_Y$ est une fonction qui ne dépend pas de z_0.

Théorème 5 (Condition faible) *Une dépendance d'une variable Y vers une variable X dans une équation (E) d'un PARE ne génère aucune communication distante si et seulement si :*

$$(\sigma_X \cdot A_0 - \sigma_Y)\epsilon_{E,Y,j} = 0 \quad \forall j \in [1, dim(\mathrm{Emit}_{E,Y}(p))]$$

où A_0 est la partie linéaire en z_0 du premier point actif $v_0(z_0, p)$ de l'ensemble d'utilisation, σ_X et σ_Y sont les matrices d'allocation de X et Y respectivement, et $\epsilon_{E,Y,j}$, $1 \leq j \leq dim(\mathrm{Emit}_{E,Y}(p))$ sont les vecteurs d'émission de $\mathrm{Emit}_{E,Y}(p)$.

Preuve du théorème 5.

D'après l'équation (2.2), il n'y a pas de communication distante si et seulement si $(\sigma_X \cdot A_0 - \sigma_Y)z_0$ est constant quel que soit z_0 dans l'ensemble d'émission.

Soient z_0 et z_1 deux points de l'ensemble d'émission $\mathrm{Emit}_{E,Y}(p)$. Notons d_E la dimension de l'ensemble d'émission $\mathrm{Emit}_{E,Y}(p)$, $d_E = dim(\mathrm{Emit}_{E,Y}(p))$. Par définition, il existe un vecteur $\alpha \in \mathbb{Z}^{d_E}$ tel que :

$$z_0 = z_1 + \sum_{j=1}^{d_E}(\alpha_j \epsilon_{E,Y,j})$$

La relation suivante doit être vérifiée :

$$\begin{array}{lll} & (\sigma_X \cdot A_0 - \sigma_Y)z_0 = (\sigma_X \cdot A_0 - \sigma_Y)z_1 & \forall z_0, z_1 \in \mathrm{Emit}_{E,Y}(p) \\ \Leftrightarrow & (\sigma_X \cdot A_0 - \sigma_Y)(z_0 - z_1) = 0 & \forall z_0, z_1 \in \mathrm{Emit}_{E,Y}(p) \\ \Leftrightarrow & (\sigma_X \cdot A_0 - \sigma_Y)\sum_{j=1}^{d_E}(\alpha_j \epsilon_{E,Y,j}) = 0 & \forall \alpha_j \in \mathbb{Z} \\ \Leftrightarrow & (\sigma_X \cdot A_0 - \sigma_Y)\epsilon_{E,Y,j} = 0 & \forall j \in [1, d_E] \end{array}$$

car les vecteurs d'émission sont linéairement indépendants.

□

Ce théorème donne la condition nécessaire et suffisante pour qu'il n'y ait pas de communication distante liée à une dépendance, lorsque le premier point actif des ensembles d'utilisation est connu. On peut en déduire une méthode de recherche d'allocation éliminant les communications distantes, en considérant chacun des sommets des ensembles d'utilisation, et en cherchant des conditions sur le cadencement afin que chacun de ces sommets soit le premier point actif.

2.2.4 Contraintes sur le cadencement

Le théorème suivant donne la condition nécessaire pour qu'un sommet donné de l'ensemble d'utilisation soit le premier point actif :

Théorème 6 *Soit $v_i(z_0, p)$, un sommet entier de l'ensemble $\mathrm{Util}_{E,Y}(z_0, p)$. Soient $u_{i,j}$ les vecteurs de base du plus petit cône contenant l'ensemble d'utilisation, ayant pour sommet $v_i(z_0, p)$. On note U_i la matrice dont les colonnes sont constituées des vecteurs $u_{i,j}$.*

$v_i(z_0, p)$ est le premier point actif de l'ensemble d'utilisation si et seulement si :

$$\lambda_X \cdot u_{i,j} \succ 0 \qquad \forall j$$

où λ_X est la partie linéaire de la fonction de cadencement de la variable X : $t_X(z, p) = \lambda_X z + \lambda'_X p + \alpha_X$.

Preuve du théorème 6.

Les vecteurs $u_{i,j}$ sont les vecteurs directeurs des arêtes de l'ensemble d'utilisation $\mathrm{Util}_{E,Y}(z_0, p)$ issus du sommet $v_i(z_0, p)$. Dans la preuve du théorème 7 (page 96) nous démontrons que les vecteurs directeurs des facettes d'un polyèdre paramétré ne dépendent pas des paramètres. Les arêtes sont des intersections de facettes, elles sont donc portées par des vecteurs constants. Les vecteurs $u_{i,j}$ sont donc constants par rapport aux paramètres z_0 et p.

Tout point z du cône généré par $v_i(z_0, p)$ et les vecteurs $u_{i,j}$ s'écrit sous la forme :

$$z = v_i(z_0, p) + U_i\beta$$

où U_i est une matrice constante et β est un vecteur entier positif.

Cherchons une condition nécessaire et suffisante pour que tout point de ce cône soit exécuté après $v_i(z_0, p)$:

$$\begin{array}{rcl} t_X(z,p) \succ t_X(v_i(z_0,p),p) & \Leftrightarrow & t_X((v_i(z_0,p) + U_i\beta), p) \succ t_X(v_i(z_0,p),p) \quad \forall \beta \\ & \Leftrightarrow & \lambda_X U_i \beta \succ 0 \qquad \forall \beta \\ & \Leftrightarrow & \lambda_X \cdot u_{i,j} \succ 0 \qquad \forall j \end{array}$$

Cette condition est donc nécessaire et suffisante pour que $v_i(z_0, p)$ soit le premier point actif de ce cône minimal contenant l'ensemble d'utilisation, et donc qu'il soit le premier point actif de l'ensemble d'utilisation.
□

Discussion

Le nombre et la valeur des vecteurs $u_{i,j}$ sont parfois liés aux vecteurs d'utilisation. Lorsque l'ensemble d'utilisation est de dimension 0, il n'y a pas de vecteur $u_{i,j}$, ni de condition supplémentaire sur le cadencement. La condition faible est équivalente à la condition forte dans ce cas.

Les ensembles d'utilisation de dimension 1 ont un vecteur d'utilisation, et le vecteur $u_{i,1}$ est alors égal au signe près à ce vecteur d'utilisation : $u_{i,1} = \pm u_{E,Y,1}$.

Un ensemble d'utilisation de dimension 2 a deux vecteurs d'utilisation. Il a également deux vecteurs $u_{i,1}$ et $u_{i,2}$, car tout cône de dimension 2 est défini par deux vecteurs. Les vecteurs $u_{i,1}$ et $u_{i,2}$ sont coplanaires à $u_{E,Y,1}$ et $u_{E,Y,2}$.

Un ensemble d'utilisation de dimension supérieure ou égale à 3, $dim(\mathrm{Util}_{E,Y}) \geq 3$, a $dim(\mathrm{Util}_{E,Y})$ vecteurs d'utilisation, et *au moins* autant de vecteurs $u_{i,j}$: un cône de dimension n, $n \geq 3$, est défini par *au moins* n vecteurs.

2.2.5 Cas particuliers

Matrice R de rang plein

Soit une équation de la forme $X[z] = \ldots Y[Rz + Qp + r] \ldots$. Lorsque la matrice R est de rang plein, la dimension de l'ensemble d'utilisation est nulle. Il n'y a qu'un point d'utilisation par point d'émission, et il existe une bijection entre chaque point d'émission z_0 et le point d'utilisation correspondant[3] v_1 :

$$\begin{array}{rll} & z_0 = Rv_1 + Qp + r & \\ \Leftrightarrow & v_1 = R^{-1}z_0 - R^{-1} \cdot Qp - R^{-1}r & \text{car } R \text{ est inversible.} \end{array}$$

Or, d'après l'algorithme de recherche des sommets, on a aussi :

$$v_1 = A_1 z_0 + B_1 p + c_1$$

On déduit de ces deux égalités la relation : $A_1 = R^{-1}$.

D'autre part, v_1 est le seul sommet de l'ensemble d'utilisation, on a donc $v_0 = v_1$. La condition d'élimination des communications distantes donnée par le théorème 5 se réécrit donc :

$$\begin{array}{rll} & (\sigma_X \cdot R^{-1} - \sigma_Y)\epsilon_{E,Y,j} = 0 & \forall j \in [1, dim(\text{Emit}_{E,Y}(p))] \\ \Leftrightarrow & (\sigma_X - \sigma_Y \cdot R)\epsilon_{E,Y,j} = 0 & \forall j \in [1, dim(\text{Emit}_{E,Y}(p))] \\ \Leftrightarrow & \sigma_X - \sigma_Y \cdot R = 0 & \end{array}$$

car $dim(\text{Emit}_{E,Y}(p)) = dim(\mathcal{D}_E)$ puisque R est de rang plein, et donc les vecteurs d'émission génèrent le domaine d'itération complet. Cette égalité est la même que celle donnée par la condition forte.

Il n'y a pas de condition suplémentaire sur le cadencement (théorème 6), car la dimension de l'ensemble d'utilisation est nulle.

La condition faible est donc strictement équivalente à la condition forte dans ce cas [Mon97].

Dépendance uniforme

La dépendance uniforme est un cas particulier du cas précédent : la matrice R est égale à la matrice identité, et est donc de rang plein. La condition faible ou forte s'écrit :

$$\begin{array}{rl} & \sigma_X - \sigma_Y \cdot R = 0 \\ \Leftrightarrow & \sigma_X = \sigma_Y \end{array}$$

3. Pour des raisons de lisibilité, $v_1(z_0, p)$ et simplement noté v_1.

Auto-dépendance

Soit une équation de la forme $X[z] = ... X[Rz + Qp + r]$ .... Les communications dues à cette auto-dépendance sont locales si (condition forte) :

$$\begin{aligned} & \sigma_X - \sigma_X \cdot R = 0 \\ \Leftrightarrow \quad & \sigma_X (Id - R) = 0 \end{aligned}$$

Cette condition peut être vérifiée si et seulement si le rang de la matrice $(Id - R)$ est inférieur ou égal à $dim(\mathcal{D}_E) - d_\sigma$.

La condition faible, suffisante pour éliminer les communications distantes, s'écrit :

$$\begin{aligned} & (\sigma_X \cdot A_0 - \sigma_X)\epsilon_{E,Y,j} = 0 \quad \forall j \in [1, dim(\mathrm{Emit}_{E,Y}(p))] \\ \Leftrightarrow \quad & \sigma_X \cdot (A_0 - Id)\epsilon_{E,Y,j} = 0 \quad \forall j \in [1, dim(\mathrm{Emit}_{E,Y}(p))] \end{aligned}$$

2.3 Une méthode de recherche de transformations optimales

Un système d'équations récurrentes est en général caractérisé par plusieurs dépendances, qui sont souvent incompatibles pour ce qui concerne l'élimination des communications : la combinaison des contraintes éliminant toutes les communications conduit souvent à une solution sur un seul processeur. C'est pourquoi nous avons élaboré une stratégie qui permet de minimiser les communications distantes, puis de minimiser les communications locales résiduelles, en tenant compte de leurs poids. Feautrier propose de classer les dépendances selon le volume de communications engendré [Fea94]. Nous utilisons une méthode d'estimation du volume de communication similaire, selon la dimension des images par les fonctions d'allocation des ensembles d'utilisation et des ensembles d'émission. Ces informations donnent les dimensions des ensembles de processeurs virtuels faisant des communications distantes et locales, que nous comparons pour rechercher la meilleure solution selon ces critères.

Les dépendances du PARE sont traitées successivement, en commençant par celles qui risquent de générer le plus grand nombre de communications distantes puis locales. Le nombre de communications distantes est au pire égal au nombre de points de l'ensemble d'émission : il y a une communication qui peut être distante le long de chaque premier vecteur de dépendance. Sa dimension est donc $dim(\mathrm{Emit}_{E,Y}(p)))$. Le nombre de communications locales est le nombre de points de tous les ensembles d'utilisation relatifs à cette dépendance. Ce nombre de points est approximé par la dimension de l'ensemble d'émission plus celle des ensembles d'utilisation : $dim(\mathrm{Emit}_{E,Y}) + dim(\mathrm{Util}_{E,Y})$.

L'algorithme consiste à construire un arbre de solutions. Chaque niveau de l'arbre correspond à une dépendance : la hauteur de l'arbre est le nombre de dépendances du

PARE. Un ensemble de contraintes sur les fonctions de cadencement et d'allocation est associé à chaque noeud de l'arbre. La racine de l'arbre contient les contraintes causales et les contraintes de validité. Les successeurs d'un noeud sont construits en appliquant successivement les stratégies de minimisation des communications données ci-dessous. Les nouvelles contraintes sont l'intersection de celles du noeud parent et de celles correspondant à la stratégie appliquée. A chaque création de noeud, l'algorithme vérifie que l'ensemble de contraintes a une solution. S'il n'en a pas, le noeud n'est pas créé. Les feuilles de l'arbre correspondent aux différentes solutions trouvées. Les volumes de communication sont calculés à chaque noeud de l'arbre, sous forme de tableaux comptant le nombre de communications de chaque dimension. Il y a deux tableaux de ce type associés à chaque noeud : le premier compte les communications distantes, et le second les communications locales.

Les stratégies appliquées à chaque noeud pour créer ses descendants sont les suivantes :

1a. **Condition forte et** $\rho(p) = 0$ (théorème 4 et équation (2.1)).

Création d'un noeud avec les contraintes supplémentaires $\sigma_X = \sigma_Y \cdot R$ et $\rho(p) = 0$, si elles peuvent être satisfaites. Les fonctions d'allocation projettent sur le même processeur tout l'ensemble d'utilisation, ainsi que les points d'émission correspondants. Il n'y a aucune nouvelle communication dans ce cas.

1b. **Condition forte** (théorème 4).

Création d'un noeud avec la contrainte supplémentaire $\sigma_X = \sigma_Y \cdot R$, si elle peut être satisfaite. Les fonctions d'allocation projettent sur le même processeur tout l'ensemble d'utilisation. Les seules communications restantes sont locales, entre chaque processeur d'émission et le processeur de l'ensemble d'utilisation correspondant. La dimension de ces communications locales est la dimension de l'ensemble d'émission.

2a. **Condition faible et élimination de communications locales résiduelles** (théorème 5, théorème 6 et équation (2.2)).

Parcours des sommets v_i de l'ensemble d'utilisation. Pour chaque sommet v_i, les nouvelles contraintes sur l'allocation sont données par le théorème 5, et les contraintes sur cadencement par le théorème 6. De plus, la communication résiduelle est éliminée, en imposant $(\sigma_X \cdot A_0 - \sigma_Y)z_0 + \rho'(p) = 0$ (équation (2.2)). Si toutes ces contraintes peuvent être satisfaites, un nouveau noeud est créé.

Le volume des nouvelles communications locales est la dimension de l'ensemble d'utilisation plus celle de l'ensemble d'émission (communication de chaque donnée dans les ensembles d'utilisation). Certaines de ces communications peuvent être éliminées en choisissant une fonction d'allocation projetant l'ensemble d'utilisation sur un polyèdre de dimension inférieure (en annulant un des vecteurs d'utilisation par exemple). Il n'y a pas de nouvelles communications distantes.

2b. **Condition faible** (théorème 5 et théorème 6).

Parcours des sommets v_i de l'ensemble d'utilisation. Pour chaque sommet v_i, les nouvelles contraintes sur l'allocation sont données par le théorème 5, et les contraintes sur cadencement par le théorème 6. Si elles peuvent être satisfaites, un nouveau noeud est créé.

Le volume des nouvelles communications locales est la dimension de l'ensemble d'émission (communications dues à la communication le long du premier vecteur de dépendance), plus la dimension de l'ensemble d'utilisation plus celle de l'ensemble d'émission (communication de chaque donnée dans les ensembles d'utilisation). Certaines de ces dernières communications dans l'ensemble d'utilisation peuvent être éliminées en choisissant une fonction d'allocation projetant l'ensemble d'utilisation sur un polyèdre de dimension inférieure. Il n'y a pas de nouvelles communications distantes.

3. **Pas de contraintes**.

Création d'un nouveau noeud, sans contrainte supplémentaire. Ceci garantit l'existence d'au moins une solution, et permet parfois de trouver de meilleures solutions lorsqu'il y a des incompatibilités entre les optimisations de dépendances plus coûteuses que d'autres, mais moins nombreuses.

Le volume des nouvelles communications distantes est la dimension de l'ensemble d'émission (communications le long du premier vecteur de dépendance). Le volume des nouvelles communications locales, est la dimension de l'ensemble d'utilisation plus celle de l'ensemble d'émission (communication de chaque donnée dans les ensembles d'utilisation). Dans ce dernier cas, il y a autant d'ensembles d'utilisation que de points d'émission : le volume de communication est donc la somme des dimensions des ensembles d'utilisation et d'émission.

Voici l'algorithme complet, écrit en pseudo-code. La fonction AddNode effectue les opérations nécessaires à la création du noeud : elle prend en paramètre l'ensemble de contraintes sur le cadencement et l'allocation et le volume des nouvelles communications.

Création des fils d'un noeud

Input: index function, utilization and emission sets,
basis vectors related to the emission set $\varepsilon_{E,Y,j}$.

Output: the set of descendants of the node. Each node is characterized
by a set of constraints and an estimation of the volume of communication.

procedure GenDescendants (*node* : current node,
R, Q, r : matrices defining the index function $g(z,p)$,
Util : parameterized polyhedron $\mathrm{Util}_{E,Y}(z_0,p)$,
v_i : vertices of $\mathrm{Util}_{E,Y}(z_0,p)$,

```
                         εj : emission vectors ε_{E,Y,j}
                        )
begin
  node n;                    /* temporary node */
  constraints C, C', Ci;     /* temporary sets of constraints */

  /*******************************************/
  /*** CASE 1a. Strong condition and ρ(p) = 0 ***/
  /*******************************************/
  C = node→constraints ∩ {σY · R = σX}
  if C ≠ ∅ then
    if C ∩ {ρ(p) = 0} ≠ ∅ then
      AddNode ( C ∩ {ρ(p) = 0}, 0, 0 )            /* no new communications */
    endif

    /********************************/
    /*** CASE 1b. Strong condition ***/
    /********************************/
    AddNode ( C, 0, dim(Emit) )        /* dim(Emit) = Rank(R) new local comms */
  endif

  /**********************************************/
  /*** CASE 2a. Weak condition and no first comm. ***/
  /*** scan the vertices vi                       ***/
  /**********************************************/
  for all integer vi do
                          /* Compute the stronger constraints on the schedule */
    Ui = EdgeVectors ( vi, Util )       /* Ui: matrix of generating vectors u_{i,j} */
    C = node→constraints ∩ {∩k ∩j λk · u_{i,j} ≥ 0}
    if C ≠ ∅ then
      Ci = C ∩ {∩j (σX · (vi→A) − σY) εj = 0}             /* weak condition */
      if Ci ≠ ∅ then
        C' = Ci ∩ {(σX · vi→A − σY) z0 + ρ'(p) = 0}        /* residual comm. */
        if C' ≠ ∅ then
              /* no new distant but dim(Emit) + dim(Util) new local comms. */
          AddNode ( C', 0, dim(Emit) + dim(Util) )
        endif
      endif
    endif
  done

  /**************************************************/
  /*** CASE 2b. Weak condition: scan the vertices vi ***/
  /**************************************************/
  for all Ci ≠ ∅ do
```

```
                              /* the constraints Ci are computed in case 2a. */
    AddNode ( Ci, 0, (dim(Emit) + dim(Util)) + dim(Emit) )   /* new local comms */
  done

  /*****************************/
  /*** CASE 3. No constraints ***/
  /*****************************/
                                         /* new distant and local communications */
  AddNode( node→constraints, dim(Emit), (dim(Emit) + dim(Util)) )
end
```

2.4 Exemple

Considérons le PARE suivant, dépendant d'un paramètre N :

$$\begin{array}{rlll} X[i,j,k] = & Y[j,i-1,0]+2 & D_1 = \{i,j,k \mid 0 \le j,k \le N, 1 \le i \le N\} & (1) \\ Y[i,j,k] = & X[0,0,k-1]/5 & D_2 = \{i,j,k \mid 0 \le i,j \le N, 1 \le k \le N\} & (2) \end{array}$$

Nous allons chercher une solution parallèle à ce PARE dans deux cas de figures : dans un espace de processeurs virtuels de dimension 2 d'abord, puis dans un espace de dimension 1 avec un cadencement de dimension 2 donc.

2.4.1 Espace de processeurs de dimension 2

Soient $\sigma_X = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}$ et $\sigma_Y = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$ les matrices d'allocation de chaque variable, où chaque élément x_* ou y_* est un vecteur colonne de dimension 2.

Contraintes causales.

Puisque l'espace de processeurs virtuels de dimension 2, les fonctions de cadencement sont de dimension 1. λ_X et λ_Y sont les matrices de cadencement monodimensionnel (ces matrices comportent donc une seule ligne) des variables X et Y. Les contraintes causales sont :

$$\left\{ \begin{array}{lll} \lambda_X \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \alpha_X > \lambda_Y \begin{pmatrix} j \\ i-1 \\ 0 \end{pmatrix} + \alpha_Y & & \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in D_1 \\ \lambda_Y \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \alpha_Y > \lambda_X \begin{pmatrix} 0 \\ 0 \\ k-1 \end{pmatrix} + \alpha_X & & \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in D_2 \end{array} \right. \qquad (2.3)$$

Afin de simplifier les calculs, et sans incidence sur le résultat obtenu dans le cas traité ici, simplifions ce système en choisissant une fonction de cadencement unique :

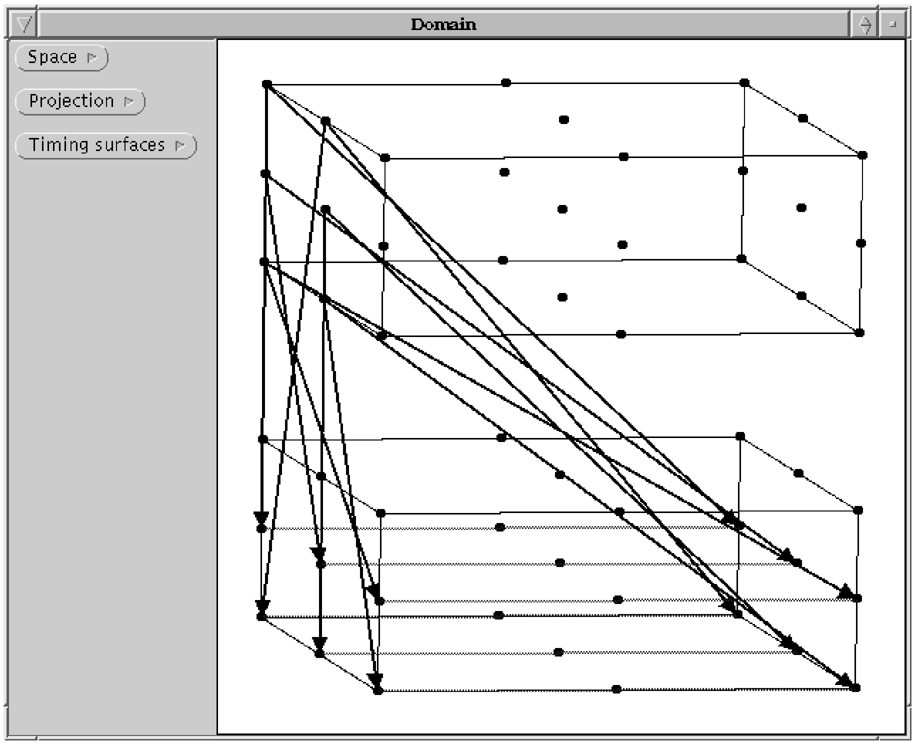


FIG. 2.5 - *La première dépendance, de $Y[j, i-1, 0]$ vers $X[i, j, k]$.*

$\lambda_X = \lambda_Y = \lambda = \begin{pmatrix}\lambda_1 & \lambda_2 & \lambda_3\end{pmatrix}$ et $\alpha_X = \alpha_Y$. Nous obtenons alors le système de contraintes causales suivant :

$$\begin{cases} \lambda_1 = \lambda_2 > 0 \\ \lambda_3 > 0 \end{cases} \tag{2.4}$$

Ensembles d'utilisation.

Les ensembles d'utilisation sont :

$$\mathrm{Util}_{1,Y}\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \left\{ \begin{pmatrix} j_0+1 \\ i_0 \\ k \end{pmatrix} \middle| \, 0 \le k \le N \right\}$$

$$\mathrm{Util}_{2,X}\left(\begin{pmatrix} i_0 \\ j_0 \\ k_0 \end{pmatrix}, N\right) = \left\{ \begin{pmatrix} i \\ j \\ k_0+1 \end{pmatrix} \middle| \, 0 \le i, j \le N \right\}$$

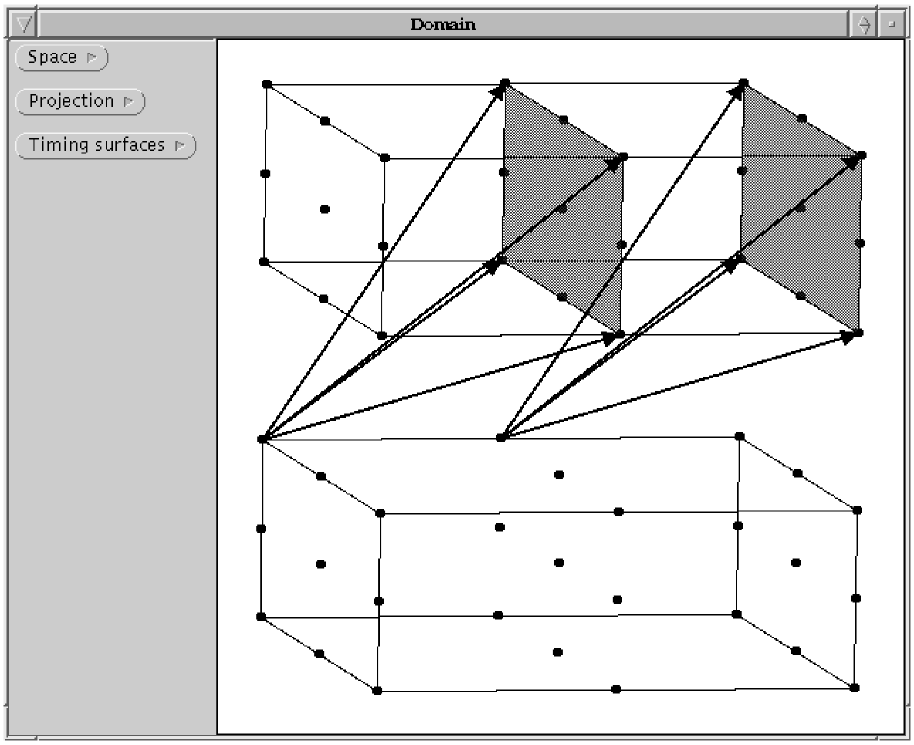


FIG. 2.6 - *La deuxième dépendance, de $X[0, 0, k-1]$ vers $Y[i, j, k]$.*

$\mathrm{Util}_{1,Y}$ est de dimension 1, et a deux sommets, alors que $\mathrm{Util}_{2,X}$ est de dimension 2, et a quatre sommets. Les figures 2.5 et 2.6 montrent les dépendances et les ensembles d'utilisation pour la valeur du paramètre $p = 2$. Dans chacune de ces fenêtres, le polyèdre du haut correspond à la variable Y, et celui du bas à la variable X.

Les sommets des ensembles d'utilisation $\mathrm{Util}_{1,Y}$ et $\mathrm{Util}_{2,X}$, et les nouvelles contraintes sur le cadencement imposées par le choix de chacun d'entre eux comme premier vecteur de communication sont présentés dans le tableau suivant. La matrice U_i contient les vecteurs $u_{i,j}$ portant les arêtes de l'ensemble d'utilisation ayant pour origine le sommet v_i. Les nouvelles contraintes sur le cadencement sont $\lambda_X U_i > 0$ (théorème 6).

dep.	v_i	coord.	U_i	nouvelles contraintes sur le cadencement : $\lambda \cdot U_i > 0$
$(1), Y$	v_1	$\begin{pmatrix} j_0+1 \\ i_0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\lambda_3 > 0$
	v_2	$\begin{pmatrix} j_0+1 \\ i_0 \\ N \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$	$\lambda_3 < 0$ incompatible avec (2.4)
$(2), X$	v_1	$\begin{pmatrix} 0 \\ 0 \\ k_0+1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{cases} \lambda_1 > 0 \\ \lambda_2 > 0 \end{cases}$
	v_2	$\begin{pmatrix} 0 \\ N \\ k_0+1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$	$\begin{cases} \lambda_1 > 0 \\ \lambda_2 < 0 \end{cases}$ incompatible avec (2.4)
	v_3	$\begin{pmatrix} N \\ 0 \\ k_0+1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{cases} \lambda_1 < 0 \\ \lambda_2 > 0 \end{cases}$ incompatible avec (2.4)
	v_4	$\begin{pmatrix} N \\ N \\ k_0+1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}$	$\begin{cases} \lambda_1 < 0 \\ \lambda_2 < 0 \end{cases}$ incompatible avec (2.4)

Seul le choix des deux premiers sommets de chaque ensemble d'utilisation impose des contraintes sur le cadencement compatibles avec les contraintes causales. Pour $\mathrm{Util}_{1,Y}$, la propagation est possible le long du sommet $v_1 = (j_0+1, i_0, 0)$ et la nouvelle contrainte sur le cadencement est $\lambda_3 > 0$. Pour $\mathrm{Util}_{2,X}$, la propagation est possible le long du sommet $v_1 = (0, 0, k_0 + 1)$ et les nouvelles contraintes sur le cadencement sont $\lambda_1 > 0$ et $\lambda_2 > 0$. Remarquons que ces contraintes sont automatiquement vérifiées par tout cadencement valide, car les contraintes causales imposent : $\lambda_1 = \lambda_2 > 0$ et $\lambda_3 > 0$. Ces contraintes ne sont donc pas reportées dans le tableau suivant.

Nouvelles contraintes.

Le tableau suivant représente les contraintes obtenues par application de chaque stratégie de minimisation des communications.

La colonne "nouvelles contraintes (cas b.)" donne les contraintes à satisfaire par l'allocation correspondant à chaque condition (forte ou faible) et à chaque dépendance (cas 1b. ou 2b. dans l'algorithme de la page 57).

La colonne de droite "contraintes supplémentaires éliminant les communications résiduelles (cas a.)" correspond à l'élimination de toutes les communications dues à cette dépendance dans la condition forte (cas 1a. : $\rho(p) = 0$), et à l'élimination des communications résiduelles dues à la première dépendance dans la condition faible (cas 2a.).

dép.	stratégie	nouvelles contraintes (cas b.)	contraintes supplémentaires éliminant les communications résiduelles (cas a.)
$(1), Y$	cond. forte (cas 1.)	$x_1 = y_2$ $x_2 = y_1$ $x_3 = 0$	$\gamma_Y = \gamma_X + y_2$
	cond. faible (v_1) (cas 2.)	$x_1 = y_2$ $x_2 = y_1$	$\gamma_Y = \gamma_X + x_1$
$(2), X$	cond. forte (cas 1.)	$y_1 = 0$ $y_2 = 0$ $y_3 = x_3$ $\Rightarrow \nexists\, \sigma_Y$ *full row rank* $\Rightarrow$	la condition forte ne peut être satisfaite
	cond. faible (v_1) (cas 2.)	$x_3 = y_3$	$\gamma_X = \gamma_Y + y_3$

Remarquons que la seule incompatibilité des contraintes de ce tableau est celle entre l'élimination des communications résiduelles dans le cas 1a. de la première dépendance, et l'élimination des communications résiduelles dans le cas 2a. de la seconde dépendance : $\gamma_Y = \gamma_X + y_2$ et $\gamma_X = \gamma_Y + y_3$ implique l'égalité $y_2 = y_3$. De plus, les autres contraintes imposent $y_3 = x_3 = 0$. On a donc $y_2 = 0$ et $y_3 = 0$, et il n'y a pas de matrice d'allocation σ_Y ayant deux lignes indépendantes respectant ces conditions.

Arbre des solutions.

L'arbre des solutions est représenté figure 2.7. Le volume de communication est représenté à chaque noeud par le nombre de communications de dimensions 1, 2, et 3, dans deux tableaux : celui de gauche correspond aux communications distantes, et celui de droite aux communications locales.

La feuille de l'arbre notée "pas de contraintes" est uniquement informative. Elle montre que la plus mauvaise solution implique deux ensembles de communications distantes de dimensions 1 et 2, plus deux ensembles de communications locales de dimension 3.

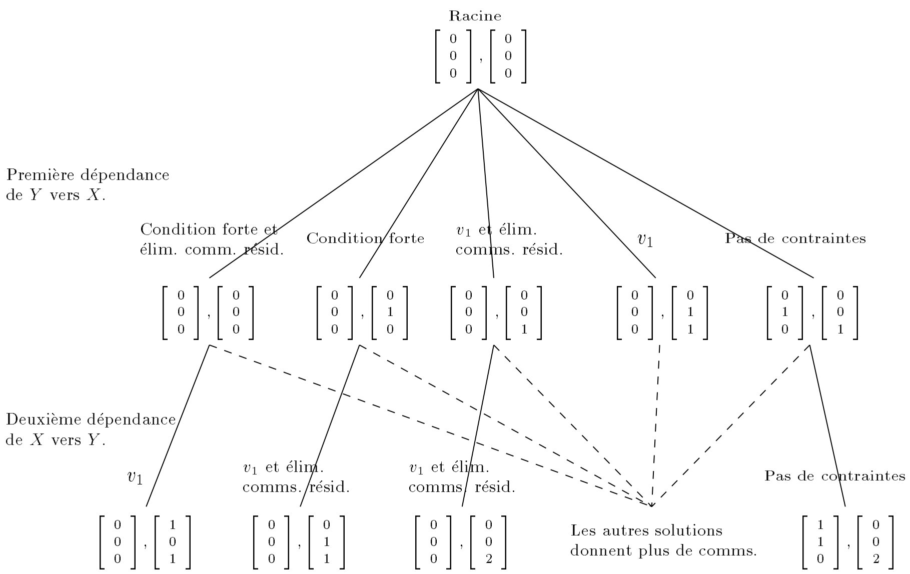


FIG. 2.7 - *L'arbre des solutions.*

La feuille la plus à gauche de l'arbre est la meilleure solution : il y a deux ensembles de communications locales, de dimensions 3 et 1. Les contraintes correspondant à cette solution sont :

$$\begin{cases} \sigma_X = (x_1, x_2, 0) \\ \sigma_Y = (x_2, x_1, 0) \\ \gamma_Y = \gamma_X + x_1 \\ \lambda_1 = \lambda_2 > 0 \\ \lambda_3 > 0 \end{cases}$$

où x_1 et x_2 sont deux vecteurs entiers indépendants. Il n'y a pas d'autre réduction des communications possible. En effet, il est impossible d'annuler le produit de σ_X et des vecteurs d'utilisation de $\mathrm{Util}_{2,X}$, $(1,0,0)$ et $(0,1,0)$, pour projeter ces ensembles d'utilisation sur un espace de dimension inférieure.

Une solution possible est : $\sigma_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $\gamma_X = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\sigma_Y = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$, $\gamma_Y = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\lambda = (1,1,1)$ et $\alpha = 0$.

Le réseau de processeurs bidimensionnel correspondant à cette solution est représenté figure 2.8. Toutes les communications sont locales et de distance 1.

Cet exemple montre bien que notre méthode permet de trouver de meilleures solutions qu'en appliquant la condition forte uniquement. En effet, les communications dues à la deuxième dépendance ne peuvent être optimisées grâce à la condition forte dans ce problème.

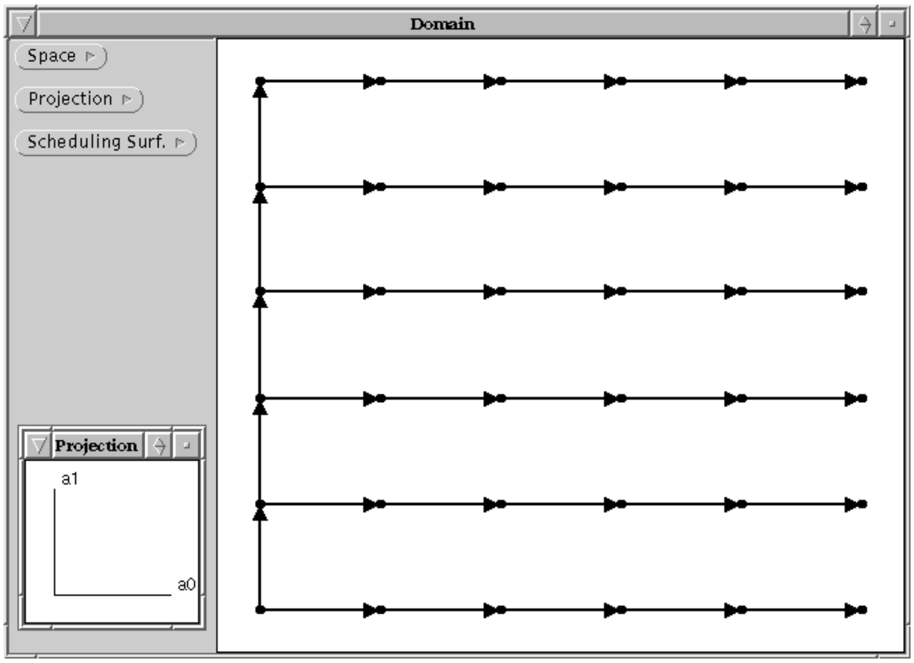


FIG. 2.8 - *Le réseau de processeurs et les communications.*

2.4.2 Espace de processeurs de dimension 1

Cherchons une solution au même problème, dans un espace de processeurs monodimensionnel.

Soient $\sigma_X = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}$ et $\sigma_Y = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$ les matrices d'allocation de chaque variable, où chaque élément x_* ou y_* est un entier.

Contraintes causales.

Puisque l'espace de processeurs virtuels de dimension 1, les fonctions de cadencement sont de dimension 2. λ_X et λ_Y sont les matrices de cadencement des variables X et Y. Posons $\lambda_X = \begin{pmatrix} \lambda_{X1} & \lambda_{X2} & \lambda_{X3} \end{pmatrix}$ et $\lambda_Y = \begin{pmatrix} \lambda_{Y1} & \lambda_{Y2} & \lambda_{Y3} \end{pmatrix}$, où chaque λ_{X*} et λ_{Y*} est un vecteur colonne de dimension 2. Les contraintes causales sont :

$$\left\{ \begin{array}{ll} \lambda_X \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \alpha_X > \lambda_Y \begin{pmatrix} j \\ i-1 \\ 0 \end{pmatrix} + \alpha_Y & \forall \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in D_1 \\ \lambda_Y \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \alpha_Y > \lambda_X \begin{pmatrix} 0 \\ 0 \\ k-1 \end{pmatrix} + \alpha_X & \forall \begin{pmatrix} i \\ j \\ k \end{pmatrix} \in D_2 \end{array} \right.$$

$$\Leftrightarrow \left\{ \begin{array}{l} \lambda_{X1} \succeq \lambda_{Y2} \\ \lambda_{X2} \succeq \lambda_{Y1} \\ \lambda_{X3} \succeq 0 \\ \alpha_X \succ \alpha_Y - \lambda_{Y2} \\ \lambda_{Y1} \succeq 0 \\ \lambda_{Y2} \succeq 0 \\ \lambda_{Y3} \succeq \lambda_{X3} \\ \alpha_Y \succ \alpha_X - \lambda_{X3} \end{array} \right.$$

Simplifions ces équations, en posant pour les besoins de cet exemple :

$$\left\{ \begin{array}{l} \alpha_X = \alpha_Y = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \lambda_{X1} \succeq \lambda_{Y2} \succ 0 \\ \lambda_{X2} \succeq \lambda_{Y1} \succ 0 \\ \lambda_{Y3} \succeq \lambda_{X3} \succ 0 \end{array} \right. \tag{2.5}$$

Les sommets des ensembles d'utilisation $\mathrm{Util}_{1,Y}$ et $\mathrm{Util}_{2,X}$, et les nouvelles contraintes sur le cadencement imposées par le choix de chacun d'entre eux comme premier vecteur de communication sont les mêmes que dans le paragraphe précédent. Seul le choix des deux premiers sommets de chaque ensemble d'utilisation impose des conditions sur le cadencement compatibles avec les contraintes causales. Pour $\mathrm{Util}_{1,Y}$, la propagation est possible le long du sommet $v_1 = (j_0 + 1, i_0, 0)$ et la nouvelle contrainte sur le cadencement est $\lambda_{X3} \succ 0$. Pour $\mathrm{Util}_{2,X}$, la propagation est possible le long du sommet $v_1 = (0, 0, k_0+1)$ et les nouvelles contraintes sur le cadencement sont $\lambda_{Y1} \succ 0$ et $\lambda_{Y2} \succ 0$. Remarquons que ces contraintes sont automatiquement vérifiées par tout cadencement vérifiant les équations (2.5).

Nouvelles contraintes.

Ce tableau représente les contraintes obtenues par application de chaque stratégie de minimisation des communications :

dép.	stratégie	nouvelles contraintes (cas b.)	contraintes supplémentaires éliminant les communications résiduelles (cas a.)
$(1), Y$	cond. forte (cas 1.)	$x_1 = y_2$ $x_2 = y_1$ $x_3 = 0$	$\gamma_Y = \gamma_X + y_2$
	cond. faible (v_1) (cas 2.)	$x_1 = y_2$ $x_2 = y_1$	$\gamma_Y = \gamma_X + x_1$
$(2), X$	cond. forte (cas 1.)	$y_1 = 0$ $y_2 = 0$ $y_3 = x_3$	$\gamma_X = \gamma_Y + x_3$
	cond. faible (v_1) (cas 2.)	$x_3 = y_3$	$\gamma_X = \gamma_Y + y_3$

Remarquons une incompatibilité entre les deux conditions fortes : $x_3 = 0$ dans les contraintes de la condition forte de la première dépendance, et $y_1 = y_2 = 0$, et $y_3 = x_3$, donc $y_3 = 0$ pour la deuxième. Or la matrice σ_Y ne peut être nulle (contrainte de validité : la matrice de transformation doit être *full row rank*).

Arbre des solutions.

L'arbre des solutions est représenté figure 2.9. En appliquant la condition forte avec élimination des communications résiduelles à la première dépendance, on peut encore appliquer la condition faible avec élimination des communications résiduelles à la seconde dépendance. Cette solution ne génère que les communications de la propagation dans l'ensemble d'utilisation de la deuxième dépendance. Si l'on veut appliquer la condition forte à la deuxième dépendance, il faut appliquer la condition faible à la première. Cette solution est moins bonne que la précédente, d'une part parce que le coût des communications de la première dépendance est plus important que celui de la seconde, et d'autre part parce que l'élimination des communications résiduelles de ces deux contraintes sont incompatibles.

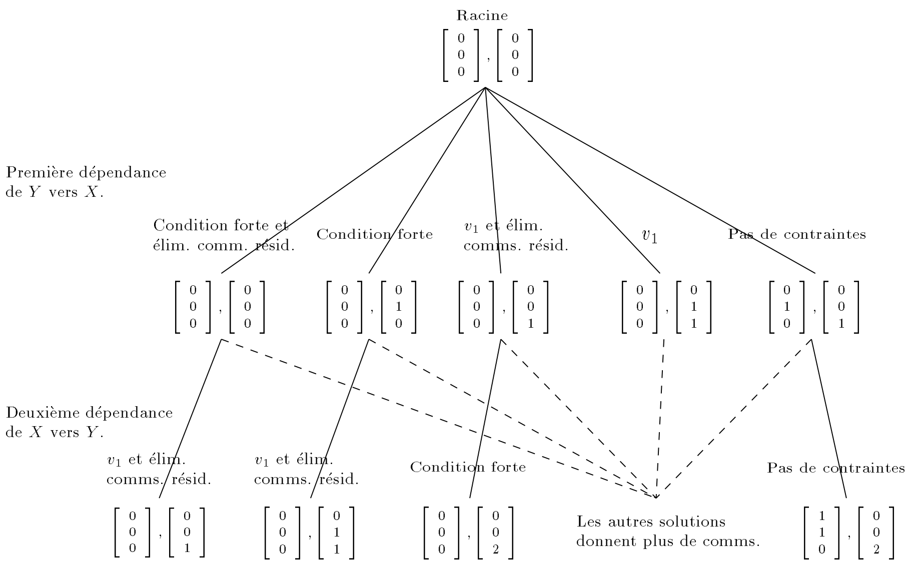


FIG. 2.9 - *L'arbre des solutions.*

La meilleure solution est donc la feuille la plus à gauche de l'arbre. Il ne reste qu'un ensemble de communications locales de dimension 3. Les contraintes associées à cette solution sont :

$$\begin{cases} \lambda_{X1} \succeq \lambda_{Y2} \succ 0 \\ \lambda_{X2} \succeq \lambda_{Y1} \succ 0 \\ \lambda_{Y3} \succeq \lambda_{X3} \succ 0 \\ \alpha_X = \alpha_Y = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ x_1 = y_2 = 0 \\ x_2 = y_1 \\ x_3 = y_3 = 0 \\ \gamma_Y = \gamma_X \end{cases}$$

Les communications résiduelles sont dues aux images des ensembles d'utilisation $\mathrm{Util}_{2,X}$. Ces ensembles d'utilisation sont contenus dans des plans parallèles aux vecteurs d'utilisation $(1,0,0)$ et $(0,1,0)$. Le produit de la matrice σ_X et du vecteur $(1,0,0)$ est nul, car $x_1 = 0$. Sans ajouter de contrainte supplémentaire, cette communication résiduelle n'est donc que de dimension 2 et non pas 3. La contrainte nécessaire à l'annulation du deuxième vecteur d'utilisation, $x_2 = 0$, ne peut être satisfaite (comme on pouvait s'y attendre puisque satisfaire cette condition supplémentaire serait équivalent à la condition forte, qui n'est pas compatible avec la première).

Une solution est par exemple : $\lambda_X = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$, $\lambda_Y = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$, $\alpha_X = \alpha_Y = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ et

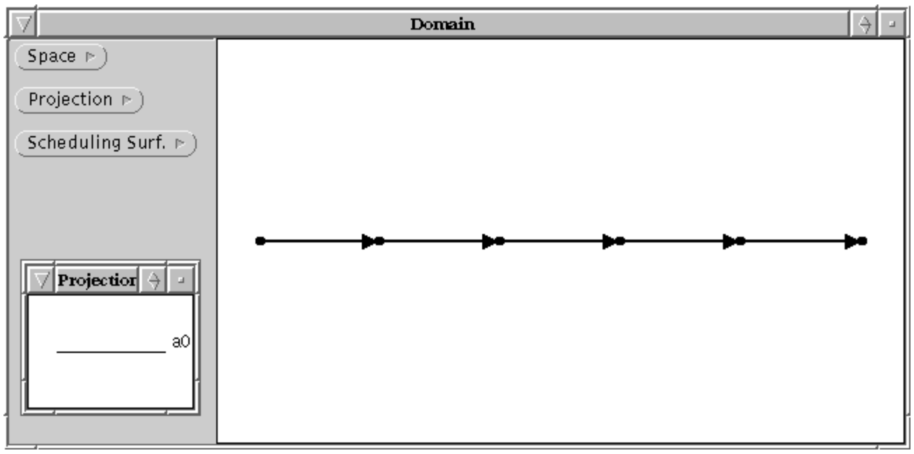


FIG. 2.10 - *Le réseau de processeurs et les communications.*

$\sigma_X = \begin{pmatrix}0 & 1 & 0\end{pmatrix}$, $\sigma_Y = \begin{pmatrix}1 & 0 & 0\end{pmatrix}$, et $\gamma_X = \gamma_Y = 0$.

Le réseau de processeurs monodimensionnel correspondant à cette solution est représenté figure 2.10. Toutes les communications sont locales et de distance 1.

2.5 Conclusion

La méthode originale présentée ici, reposant sur la condition faible, permet de trouver de meilleures solutions parallèles à certains problèmes, comme le montre clairement l'exemple ci-dessus. Elle devra encore être étendue afin de traiter le cas d'ensembles d'utilisation dont les sommets paramétrés sont rationnels, ou reposer sur un algorithme de recherche de sommets *entiers* de polyèdres paramétrés rationnels. Ceci est une voie de recherche ouverte.

Le procédé mis en œuvre dans le modèle de propagation, reposant sur le choix de certains vecteurs de dépendance extrémaux, est une généralisation de la procédure d'uniformisation des PARE. En effet, le choix d'un des vecteurs de dépendance comme vecteur correspondant à la première activation de l'ensemble d'utilisation, transforme cette dépendance multidimensionnelle en une dépendance monodimensionnelle, suivie de communications locales de voisin à voisin dans l'ensemble d'utilisation. La principale différence par rapport aux méthodes d'uniformisation, est que notre méthode n'est pas aussi restrictive, car elle s'applique au niveau de la recherche de solution parallèle, et non pas au niveau du système d'équations récurrentes, et ne génère donc pas de nouvelles contraintes non nécessaires.

D'autre part, nous avons vu que l'allocation et le cadencement étaient fortement liés.

La plupart des travaux antérieurs considérent ces deux problèmes séparément, alors qu'ils ne sont pas indépendants. La première relation entre allocation et cadencement est la suivante : il peut y avoir des problèmes d'incompatibilité entre les "bonnes" solutions sur le cadencement et les "bonnes" solutions sur l'allocation. Ce problème était contourné par l'argument suivant : si l'on cherche à optimiser le temps de calcul, il faut considérer le cadencement avant l'allocation ; si l'on veut réduire les communications ou optimiser le nombre de processeurs, il faut chercher l'allocation en priorité.

Le deuxième problème liant cadencement et allocation est plus important : comme nous l'avons démontré dans les paragraphes 2.2.3 et 2.2.4, certaines contraintes sur le cadencement permettent de déterminer le vecteur de communications entre les processeurs virtuels correspondant à un point d'émission et correspondant au premier point actif de l'ensemble d'utilisation. Lorsque ces contraintes sont combinées à la condition faible sur l'allocation, on peut minimiser ce vecteur de communication : les communications générées par cette dépendance peuvent être rendues locales, ou même être complétement éliminées.

Notre méthode permet donc de construire une allocation *et* un cadencement optimaux selon les critères que nous avons choisis. Les contraintes de validité du cadencement sont obtenues selon les méthodes classiques utilisées par le passé. La construction de l'arbre des solutions peut sembler relativement complexe, mais cet arbre a une taille maximale connue, et pas trop grande dans les cas traités en général, puisqu'il a, au plus, autant de niveaux qu'il y a de dépendances dans le PARE. De plus, de nombreuses branches peuvent être éliminées assez rapidement lorsqu'on a trouvé une solution relativement efficace : en parcourant l'arbre en profondeur, et en calculant les fils de gauche d'abord, on obtient une solution qui est souvent la meilleure, et dont le nombre de communications peut servir de critère minimal à respecter pendant le parcours des autres branches. Ainsi, pour construire les feuilles du dernier niveau de l'arbre, c'est toujours le fils le plus à gauche de chaque noeud du niveau précedent qui donne la solution la plus efficace pour ce noeud : dès qu'une feuille est trouvée, il est inutile de chercher les autres feuilles issues du même noeud.

Chapitre 3

Réalisation : le logiciel OPÉRA

3.1 Objectifs

Le but du logiciel OPÉRA[1] est de fournir à l'utilisateur un environnement convivial lui permettant de transformer un système d'équations récurrentes affines paramétré en un programme parallèle efficace et correct.

L'analyse lexicale et syntaxique est effectuée par l'analyseur de programmes Alpha de l'IRISA, utilisant les programmes GNU flex et bison. Les structures de données sous forme d'arbre sont ensuite transformées, car OPÉRA utilise ses propres structures internes, décrites en annexe A. Les fonctions de manipulation de polyèdres, paramétrés ou non, et de manipulation de systèmes d'inégalités linéaires, utilisent la bibliothèque PolyLib étendue, écrite en C. Le programme OPÉRA est également entièrement écrit en C, en utilisant le compilateur GNU gcc. L'interface graphique fait appel à la bibliothèque XView, dans l'environnement X11R5. Ce logiciel complet a été conçu sous SunOS4 et Solaris2, et a été porté sans difficulté majeure sous Linux et HPUX.

3.2 Langage Alpha

Le système d'équations récurrentes affines paramétré est donné sous forme de programme Alpha. On trouvera une description complète de ce langage dans [Mau89, Wil94]. La syntaxe Alpha que nous utilisons est brièvement décrite ci-dessous. Un programme Alpha est de la forme :

```
system <nom du système> : <déclaration des paramètres>
         ( <déclaration de variables d'entrée> )
  returns ( <déclaration de variables de sortie> );
var
```

1. OPÉRA est un acronyme signifiant “Outil pour la Parallélisation d'Équations Récurrentes Affines”.

```
  <déclaration de variables intermédiaires>;
let
  <équations>
tel;
```

Alpha est un langage fortement typé. Toutes les variables utilisées dans les équations sont déclarées comme variable d'entrée, de sortie, ou intermédiaire, sous une des formes :

```
<var-list> : <domain> of real ;

<var-list> : <domain> of integer ;

<var-list> : <domain> of boolean ;
```

Les domaines de définition `<domain>` des variables et des paramètres sont de la forme :

```
{ <liste d'indices> | <liste de contraintes> }
```

Une `<liste de contraintes>` est une liste d'expressions booléennes, séparées par des points-virgules, de la forme :

```
<exp_a><opérateur logique><exp_a>[ <opérateur logique><exp_a>]
```

`<exp_a>` est une liste séparée par des virgules, d'expressions de la forme : une fonction linéaire des indices + une fonction linéaire des paramètres + une constante.

Exemples :

```
{ i,j,k | 0<=i<=10 ; 0<j<10 ; i-j<=2k<=j+5}

{ i,j | 0 <= i,j <= 10 }      --carré de côté 10

{ i,j | 5i = 3j }      --droite dans le plan
```

Les équations sont de la forme :

```
X[<indices>] = <expression>;
```

Une expression peut contenir un test sous une des deux formes suivantes :

```
if <liste de contraintes> then <expression1> else <expression2>

case
    { | <liste de contraintes> }: <expression1>;
    { | <liste de contraintes> }: <expression2>;
    ...
esac
```

3.3 Analyse

Après l'analyse lexicale et syntaxique, OPÉRA effectue automatiquement l'analyse de dépendances du PARE. Les ensembles d'utilisation et les ensembles d'émission sont calculés, en fonction de leurs paramètres.

Calcul des ensembles d'utilisation

Soit une dépendance de la forme :

$$X[z] = ...Y[g(z,p)]... \qquad z \in \mathcal{D}_E(p) \tag{E}$$

avec p le paramètre, défini dans le contexte du PARE : $p \in \mathcal{D}_p$.

Le calcul de l'ensemble d'utilisation $\mathrm{Util}_{E,Y}(z_0, p) = \{z \in \mathcal{D}_E(p) \mid g(z,p) = z_0\}$ se fait de la manière suivante. Cet ensemble d'utilisation est un polyèdre paramétré, défini dans le domaine d'itération de l'équation $\mathcal{D}_E(p)$, de dimension[2] $dim(\mathcal{D}_E(p))$. Il dépend de deux paramètres, z_0 et p. Le vecteur z_0 est un point d'émission, défini dans $\mathcal{D}_Y(p)$, l'espace de définition de la variable Y. Le vecteur p est le paramètre du PARE, défini dans le contexte $\mathcal{D}_p$.

L'ensemble d'utilisation est représenté dans l'espace combiné des données et des paramètres, comme décrit dans la deuxième partie de ce manuscrit : les paramètres sont considérés comme des indices supplémentaires du polyèdre. Ce polyèdre est donc de dimension $dim(z) + dim(z_0) + dim(p) = dim(\mathcal{D}_E(p)) + dim(\mathcal{D}_Y(p)) + dim(\mathcal{D}_p)$. Il est obtenu par intersection des quatre polyèdres suivants :

- $\left\{ \begin{pmatrix} z \\ z_0 \\ p \end{pmatrix} \middle| \; z \in \mathcal{D}_E(p) \right\}$, z est un point du domaine d'itération ;
- $\left\{ \begin{pmatrix} z \\ z_0 \\ p \end{pmatrix} \middle| \; g(z,p) = z_0 \right\}$, le calcul de $X[z]$ utilise la donnée $Y[z_0]$, donc $g(z,p) = z_0$;
- $\left\{ \begin{pmatrix} z \\ z_0 \\ p \end{pmatrix} \middle| \; z_0 \in \mathcal{D}_Y(p) \right\}$, z_0 est un point du domaine de définition de Y ;
- $\left\{ \begin{pmatrix} z \\ z_0 \\ p \end{pmatrix} \middle| \; p \in \mathcal{D}_p \right\}$, p est un paramètre valide.

2. Il ne s'agit pas de la dimension géométrique de l'ensemble d'utilisation, mais de la dimension de l'espace dans lequel il est défini.

L'algorithme est de la forme suivante :

Calcul des $\mathrm{Util}_{E,Y}$
begin
  /* Parcours des dépendances */
  for all E, Y **do**
    /* Calcul des polyèdres étendus dans l'espace de dimension adéquate */
    $d = dim(\mathcal{D}_E(p)) + dim(\mathcal{D}_Y(p)) + dim(\mathcal{D}_p)$
    /* Les fonctions suivantes ajoutent des colonnes dans les matrices de contraintes */
    /* définissant chacun de ces polyèdres */
    ExtConstrDe = ExtendConstrIterationSpace ($\mathcal{D}_E(p), d$)
    ExtConstrDy = ExtendConstrEmissionSpace ($\mathcal{D}_Y(p), d$)
    ExtConstrDp = ExtendConstrParameterSpace ($\mathcal{D}_p, d$)
    /* créer le polyèdre vérifiant : $g(z,p) = z_0$ */
    /* (cette égalité est sous forme d'une matrice de contraintes ci-dessous) */
    ExtEq = Constraints2Polyhedron($(g(z,p) = z_0)$)
    $\mathrm{Util}_{E,Y}$ = DomainAddConstraints(DomainAddConstraints(
        DomainAddConstraints(ExtEq, ExtConstrDp), ExtConstrDy), ExtConstrDe)
  done
end

Calcul des vecteurs de dépendance extrémaux

Les vecteurs de dépendance extrémaux, dépendant de z_0 et p, sont les sommets paramétrés du polyèdre $\mathrm{Util}_{E,Y}(z_0, p)$. L'algorithme de recherche des sommets de polyèdres paramétrés est décrit dans la deuxième partie de cette thèse.

Calcul des vecteurs d'utilisation

Les vecteurs d'utilisation sont obtenus en cherchant une base de $Ker(R)$, le noyau de la matrice d'index de la fonction g : $g(z,p) = Rz + Qp + r$.

Calcul des ensembles d'émission

On calcule aisément une base de l'espace vectoriel contenant un polyèdre paramétré. Les vecteurs d'émission de chaque dépendance sont obtenus en calculant l'image des vecteurs d'une base du domaine d'itération $\mathcal{D}_E(p)$ par la fonction d'index g correspondante.

Le contour des ensembles d'émission est obtenu en calculant l'image rationnelle du domaine d'itération $\mathcal{D}_E(p)$ par la fonction d'index g, grâce à la fonction DomainImage de la PolyLib.

Contextes

Il peut y avoir plusieurs domaines de validité des paramètres, séparant le problème original en plusieurs sous-problèmes : lorsque les sommets paramétrés des domaines des équations sont définis dans plusieurs domaines de validité distincts, il faut traiter le PARE séparément dans chacun de ces domaines de validité. En effet, la forme des domaines de définition ainsi que celle des ensembles d'utilisation et d'émission n'est pas la même, dans chacun de ces domaines.

3.4 Transformations

La solution parallèle repose sur le choix d'une transformation espace-temps pour chaque variable du PARE. Le cadencement peut être multidimensionnel : l'utilisateur choisit la dimension de l'espace des processeurs ou du cadencement. Ces dimensions sont liées puisque leur somme est égale à la dimension maximale des variables du PARE. OPÉRA calcule ensuite automatiquement les contraintes sur les matrices de cadencement, en utilisant la méthode des sommets. Cet algorithme est présentée paragraphe 1.3.3. La bibliothèque PolyLib est utilisée pour résoudre les systèmes d'inégalités linéaires. L'algorithme est de la forme suivante :

Calcul des contraintes causales
begin
  ContraintesCausales = UniversePolyhedron()
  /* Parcours des valeurs extrémales des paramètres */
  for all $p \in \mathcal{D}_p - > Rays$ **do**
    /* Parcours des extrema (sommets et rayons) de $\mathcal{D}_E(p)$ */
    for all $r \in \mathcal{D}_E(p) - > Rays$ **do**
      /* Parcours des dépendances */
      for all E, Y **do**
        /* Calcul de la nouvelle contrainte, avec les valeurs r et p */
        /* dans la dépendance E, Y */
        newC = CalcContrainte(p, r, E, Y)
        AddConstraint(ContraintesCausales, newC)
      done
    done
  done
end

La méthode de recherche d'allocation et de cadencement présentée dans le chapitre précédent n'est pas encore implémentée, mais le sera à très court terme. Dans l'état actuel, l'utilisateur donne manuellement les matrices de transformation de chaque variable.

OPÉRA vérifie automatiquement la validité de ces matrices : respect des contraintes causales, et des contraintes de validité (matrices "*full row rank*", incluant la contrainte de compatibilité entre cadencement et allocation).

3.5 Génération de code

Deux projets de génération de code sont greffés à OPÉRA. Le premier d'entre eux concerne la génération du code des communications. Il a été écrit par Christophe Reffay dans le cadre de sa thèse [Ref96]. Cet ensemble de routines permet de calculer automatiquement les boucles et les instructions nécessaires à la communication des données correspondant à chaque dépendance, dans le cas d'une transformation unique du PARE et avec un cadencement monodimensionnel. Afin d'être entièrement intégré dans la version actuelle d'OPÉRA, ce projet doit être étendu et traiter les transformations par variable et le cadencement multidimensionnel.

Le second projet, implémenté par Olivier Albiez lors de son stage de DEA [Alb97], concerne la génération de code data-parallèle du PARE complet, dans le contexte des transformations par variable et du cadencement de dimension quelconque. Le problème majeur de la génération de ce type de code est d'obtenir un parcours valide et efficace du temps et des processeurs : avec les transformations par variable, le domaine espace-temps parcouru n'est pas un simple polyèdre convexe, mais c'est une superposition de polyèdres convexes. L'algorithme de Fourier-Motzkin permet de parcourir *un* polyèdre convexe, mais il doit être étendu pour parcourir plusieurs polyèdres dans un seul nid de boucles. Afin de trouver un parcours efficace, il faut éviter les tests coûteux à l'intérieur du nid de boucles. C'est pour cette raison que la superposition des domaines est découpée en convexes élémentaires disjoints, chacun d'entre eux correspondant à un seul ensemble convexe de domaines à parcourir. Les paramètres sont ici considérés comme des boucles externes : les "boucles" externes sont simplement remplacées par des tests dans ce cas.

3.6 Visualisation

La figure 3.1 présente une vue globale d'OPÉRA dans son état actuel. L'aspect de visualisation est particulièrement important : il permet à l'utilisateur expérimenté de comprendre son problème, lorsque le nombre de boucles n'est pas trop grand évidemment. Bien que le logiciel ne soit pas limité en nombre de boucles, l'esprit humain perçoit difficilement la géométrie de dimension 5 et plus. Les problèmes les plus fréquents sont cependant de dimension inférieure ou égale à 3, ou parfois 4.

Pour utiliser les possibilités de visualisation d'OPÉRA -mais seulement pour cela- il faut évidemment donner des valeurs aux paramètres. Ces valeurs sont modifiables dynamiquement, et le choix du domaine de validité des paramètres correspondant à ces valeurs

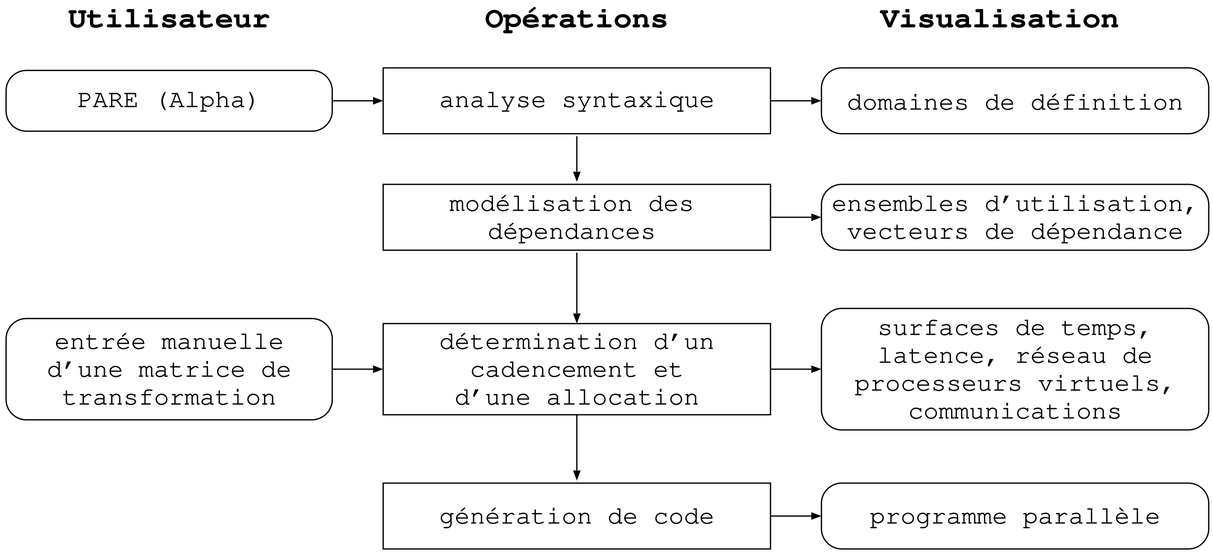


FIG. 3.1 - *Schéma d'OPÉRA*

est automatiquement vérifié. L'utilisateur doit prendre soin de traiter le problème dans chacun des domaines de validité des paramètres.

OPÉRA est utilisé par de nombreuses personnes afin de mieux comprendre les problèmes traités. La visualisation permet d'observer :

- les domaines de définition des variables, les domaines d'itération des équations,
- pour chaque dépendance : les vecteurs de dépendance, les ensembles d'utilisation, les vecteurs d'utilisation, les ensembles d'émission,
- lorsque les transformations espaces-temps sont déterminées : les surfaces de temps, le réseau de processeurs virtuels, l'activité du réseau de processeurs (images des fonctions de temps), les communications (images des dépendances).

Le paragraphe suivant décrit les différentes fenêtres intéressantes d'OPÉRA.

Deuxième partie

Polyèdres paramétrés : modèle théorique et applications en parallélisation automatique

Introduction

La nécessité de manipuler des polyèdres paramétrés a été mise en évidence dans la première partie de ce manuscrit. Nos travaux dans le domaine des polyèdres paramétrés est dérite dans cette deuxième partie. Elle composée de deux chapitres, le premier consacré à la théorie des polyèdres paramétrés, et le second aux applications de ces résultats en parallélisation automatique.

Polyèdres paramétrés

Ce premier chapitre contient une introduction à la théorie des polyèdres convexes. Nous décrivons ensuite notre représentation des polyèdres paramétrés : nous appelons polyèdre paramétré, un polyèdre défini par un ensemble d'inégalités dont la partie constante est une fonction affine d'un vecteur de paramètres.

Nous démontrons que les lignes et les rayons de polyèdres paramétrés ne dépendent pas du paramètre. Seuls les sommets s'expriment sous forme de fonction affine des paramètres, par domaine de validité sur les paramètres. Une méthode de recherche des sommets et des domaines de validité de polyèdres paramétrés, développée en collaboration avec Doran Wilde, est proposée. Ceci est la première méthode permettant de résoudre ce problème à notre connaissance. Cet algorithme a donné lieu à la publication d'un article dans International Journal of Parallel Programming [LW97].

La fin de ce chapitre présente une méthode de comptage symbolique des points entiers d'un polyèdre paramétré, découverte par Philippe Clauss et reposant sur les travaux d'Eugène Ehrhart. Cette méthode repose sur notre algorithme de recherche des sommets et domaines de validités d'un polyèdre paramétré.

Une implémentation de ces deux algorithmes a été réalisée par Doran Wilde et moi-même, sous forme d'une bibliothèque de fonctions C indépendante, la bibliothèque *PolyLib* [Wil93] (voir annexe B).

Applications en parallélisation de nids de boucles

Les premières applications de la méthode de recherche des sommets de polyèdres paramétrés ont été décrites dans la première partie de ce document : au paragraphe 1.3.3

(page 36 et suivantes) pour la recherche de cadencement avec la méthode des sommets, et au chapitre 2 (page 45 et suivantes) pour le calcul des sommets des ensembles d'utilisation, et la méthode d'optimisation des communications qui utilise ces résultats.

Le deuxième chapitre de cette partie présente quelques autres applications en analyse de performances d'une exécution parallèle d'un nid de boucles : le calcul de la latence, du parallélisme potentiel, et du nombre de communications du programme parallèle.

Les travaux présentés dans ce chapitre font partie d'un travail collectif, mené en collaboration avec Philippe Clauss et Doran Wilde. D'autres résultats sont présentés dans notre rapport de recherche [CLW97], en annexe C de ce manuscrit (en anglais).

Chapitre 1

Polyèdres paramétrés

Notes

Une matrice nulle, dont tous les coefficients sont nuls, est notée 0.

Nous appelons *matrice entière* une matrice à coefficients entiers.

Un k-vecteur est un vecteur de dimension k.

Soit v un vecteur. La notation $v \geq 0$ signifie que chaque élément du vecteur est supérieur ou égal à 0. La notation $\sum v$ représente la somme de tous les coefficients du vecteur.

Dans ce chapitre, nous considérons les polyèdres convexes, non bornés, dans des espaces rationnels de dimension quelconque. Les résultats que nous allons démontrer sont également valides dans des espaces réels. Dans ce cas, il suffit de remplacer les scalaires entiers ou rationnels manipulés par des réels.

1.1 Polyèdres convexes

Les théorèmes et propriétés donnés dans ce chapitre sont démontrés dans [Sch86] et dans [Zie95].

1.1.1 Représentation des polyèdres

Les définitions suivantes donnent les deux représentations possibles de polyèdres convexes non bornés.

Définition 18 (représentation implicite) *Un polyèdre $\mathcal{P}$ est un sous-espace de $\mathbb{Q}^n$ borné par un nombre fini d'hyperplans. $\mathcal{P}$ s'écrit sous forme d'intersection d'un nombre fini de demi-espaces affines.*

On appelle représentation implicite *la définition du polyèdre par un système d'inégalités et d'égalités, ou contraintes :*

$$\mathcal{P} = \{x \in \mathbb{Q}^n \mid Ax = a, \quad Bx \geq b\} \tag{1.1}$$

où : A et B sont des matrices entières, a et b sont des vecteurs entiers, $n \in \mathbb{N}$ est la dimension de l'espace dans lequel est défini le polyèdre. Cette forme est l'intersection des demi-espaces affines définis par les inégalités suivantes : $Ax \geq a, Ax \leq a, Bx \geq b$.

Soit $lin(\mathcal{P})$ le plus grand espace vectoriel contenu dans le polyèdre $\mathcal{P}$:

$$lin(\mathcal{P}) = \{y \in \mathbb{Q}^n \mid Ay = 0, \quad By = 0\}$$

Un polyèdre $\mathcal{P}$ est dit *pointé* lorsque $lin(\mathcal{P}) = \emptyset$.

Tout polyèdre $\mathcal{P}$ admet une représentation unique comme combinaison d'un espace vectoriel $lin(\mathcal{P})$ et d'un polyèdre pointé $\overline{\mathcal{P}}$:

$$x \in \mathcal{P} \Leftrightarrow \exists (x_v \in lin(\mathcal{P}), x_p \in \overline{\mathcal{P}}) \text{ tel que } x = x_v + x_p$$

Définition 19 (polytope) *On appelle* polytope *ou* polytope convexe *un polyèdre généré par les combinaisons convexes d'un nombre fini de points :*

$$\{x = V\nu, \nu \geq 0, \sum \nu = 1\}$$

où V est une matrice rationnelle dont chaque colonne est un sommet du polytope, ν est un vecteur libre.

Un polytope est un polyèdre pointé borné.

On appelle *cône convexe* ou simplement *cône*, un polyèdre pointé n'ayant qu'un seul sommet : l'origine O.

$$\mathcal{C} = \{y \in \mathbb{Q}^n \mid y = Y\mu \ , \ \mu \geq 0\}$$

où μ est un vecteur, et Y la matrice dont les colonnes sont constituées des *vecteurs générateurs* du cône.

Tout polyèdre pointé $\overline{\mathcal{P}}$ peut s'écrire de manière unique comme combinaison d'un polytope $\overline{\mathcal{P}_P}$ et d'un cône $\overline{\mathcal{P}_C}$:

$$x \in \overline{\mathcal{P}} \Leftrightarrow \exists (x_c \in \overline{\mathcal{P}_C}, x_q \in \overline{\mathcal{P}_P}) \text{ tel que } x = x_c + x_q$$

La décomposition d'un polyèdre quelconque en un espace vectoriel, un cône et un polytope est donnée dans la définition ci-dessous.

Définition 20 (représentation de Minkowski) *La* représentation de Minkowski *[Min96] (aussi appelée représentation paramétrique dans [Sch86]), définit le polyèdre $\mathcal{P}$ par l'ensemble de ses sommets, rayons et lignes :*

L'ensemble des lignes est le plus petit ensemble de vecteurs entiers générant l'espace vectoriel entièrement contenu dans $\mathcal{P}$ appelé $lin(\mathcal{P})$. Soit L la matrice dont les colonnes sont constituées de ces vecteurs. Soit $\overline{\mathcal{P}}$ le polyèdre pointé associé à $\mathcal{P}$, tel que $\mathcal{P} = lin(\mathcal{P}) + \overline{\mathcal{P}}$.

L'ensemble des rayons est constitué des vecteurs définissant le plus grand cône $\overline{\mathcal{P}_C}$ de $\overline{\mathcal{P}}$. Soit R la matrice dont les colonnes sont les vecteurs entiers correspondant aux rayons.

Soit V la matrice dont les colonnes sont les coordonnées rationnelles des sommets du polyèdre pointé $\overline{\mathcal{P}}$.

Le polyèdre $\mathcal{P}$ peut alors s'écrire comme étant la composition d'une combinaison linéaire des lignes, d'une combinaison linéaire positive des rayons et d'une combinaison convexe des sommets :

$$\mathcal{P} = \left\{ x \in \mathbb{Q}^n \mid x = L\lambda + R\mu + V\nu, \quad \forall\lambda,\ \forall\mu \geq 0,\ \forall\nu \geq 0,\ \sum \nu = 1 \right\} \tag{1.2}$$

Les deux représentations données dans les définitions 18 et 20 sont duales. Deux méthodes permettent de passer d'une représentation à l'autre [MR80] : la méthode du simplexe, et la méthode de la double description de Motzkin [MRTT53], améliorée successivement par Chernikova [Che65], Rubin [Rub75], et Fernandez, Quinton et Le Verge [FQ88, Le 92]. C'est cette seconde méthode qui est utilisée dans la bibliothèque *PolyLib*. Elle est basée sur la construction récursive de l'ensemble des lignes et rayons d'un *cône* par ajout itératif de chaque contrainte.

Cet algorithme ne peut être appliqué que sur les cônes. Il faut donc pouvoir associer un cône unique à tout polyèdre afin de l'utiliser dans le cas général. L'équivalence entre un polyèdre de dimension n et un cône de dimension $(n+1)$, découverte par Minkowski au début de ce siècle [Min96, Min11] est détaillée ci-dessous.

Définition 21 (représentation homogène) *Tout polyèdre défini par la représentation implicite* (1.1) *a une représentation équivalente, dite* homogène, *obtenue en appliquant la* *transformation* $x \to \begin{pmatrix}\xi x\\ \xi\end{pmatrix}$, *avec* $\xi \geq 0$:

$$\widehat{\mathcal{P}} = \left\{ \begin{pmatrix}\xi x\\ \xi\end{pmatrix} \in \mathbb{Q}^{n+1} \,\middle|\, \widehat{A}\begin{pmatrix}\xi x\\ \xi\end{pmatrix} = 0, \quad \widehat{B}\begin{pmatrix}\xi x\\ \xi\end{pmatrix} \geq 0 \right\} \tag{1.3}$$

avec $\widehat{A} = \left[\begin{array}{c|c} A & -a\end{array}\right]$ *et* $\widehat{B} = \left[\begin{array}{c|c} B & -b \\ \hline 0\cdots 0 & 1\end{array}\right]$.

Le polyèdre $\mathcal{P}$ original est obtenu en faisant l'intersection de $\widehat{\mathcal{P}}$ avec l'hyperplan d'équation $\xi = 1$. En effet, en ajoutant la contrainte $\xi = 1$ au système de contraintes donné équation (1.3), on trouve :

$$\begin{array}{l} \left[\begin{array}{c|c} A & -a\end{array}\right]\begin{pmatrix}x\\1\end{pmatrix} = 0, \quad \left[\begin{array}{c|c} B & -b \\ \hline 0\cdots 0 & 1\end{array}\right]\begin{pmatrix}x\\1\end{pmatrix} \geq 0 \\ \Leftrightarrow \quad Ax = a, \quad Bx \geq b \end{array}$$

La représentation de Minkowski homogène est la forme duale de la représentation implicite homogène. C'est également la forme homogène de la représentation de Minkowski (équation 1.3) :

$$\widehat{\mathcal{P}} \;=\; \left\{ \begin{pmatrix}\xi x\\ \xi\end{pmatrix} \in \mathbb{Q}^{n+1} \mid \begin{pmatrix}\xi x\\ \xi\end{pmatrix} = \widehat{L}\lambda' + \widehat{R}\mu', \ \forall\lambda', \ \forall\mu' \geq 0 \right\} \tag{1.4}$$

avec $\widehat{L} = \left[\begin{array}{c} L \\ \hline 0\cdots 0\end{array}\right]$ et $\widehat{R} = \left[\begin{array}{c|c} R & V \\ \hline 0\cdots 0 & 1\cdots 1\end{array}\right]$.

Les sommets de $\mathcal{P}$ sont transformés dans $\widehat{\mathcal{P}}$ par des rayons rationnels, que l'on peut multiplier par un dénominateur commun sans modifier la forme du polyèdre. $\widehat{L}$ et $\widehat{R}$ sont donc des matrices entières.

Cette représentation simplifie considérablement la manipulation des polyèdres, qui sont maintenant des cônes : ils n'ont plus qu'un seul sommet, l'origine. Le coût de cette transformation est l'ajout d'une dimension suplémentaire. Goldmann a démontré que cette transformation est unique et que la propriété d'inclusion géométrique des polyèdres est conservée [Gol56].

La table 1.1 résume les différentes représentations des polyèdres.

Exemple 1

Soit $\mathcal{P}$ le polyèdre représenté figure 1.1 :

$$\mathcal{P} \;=\; \left\{ \begin{pmatrix} i\\ j\\ k\end{pmatrix} \in \mathbb{Q}^3 \mid -j \leq i \leq j, j \geq 1 \right\}$$

$$\mathcal{P} \;=\; \left\{ \begin{pmatrix} i\\ j\\ k\end{pmatrix} \in \mathbb{Q}^3 \,\middle|\, \begin{pmatrix} 1 & 1 & 0\\ -1 & 1 & 0\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix} i\\ j\\ k\end{pmatrix} \geq \begin{pmatrix} 0\\ 0\\ 1\end{pmatrix} \right\}$$

	Système inhomogène	Système homogène
Structure	Polyèdre $\mathcal{P}$, dimension n	Cône $\widehat{\mathcal{P}}$, dimension $n+1$
Représentation implicite	$\mathcal{P} = \{x \mid A.x = a, B.x \geq b\}$	$\widehat{\mathcal{P}} = \{\begin{pmatrix}\xi x \\ \xi\end{pmatrix} \mid \widehat{A}.\begin{pmatrix}\xi x \\ \xi\end{pmatrix} = 0, \widehat{B}.\begin{pmatrix}\xi x \\ \xi\end{pmatrix} \geq 0\}$ $\widehat{A} = \left[\begin{array}{c\|c} A & -a \end{array}\right]$ $\widehat{B} = \left[\begin{array}{c\|c} B & -b \\ \hline 0\cdots 0 & 1 \end{array}\right]$
Représentation de Minkowski	$\mathcal{P} = \{x \mid x = L\lambda + R\mu + V\nu,$ $\forall\lambda, \forall\mu \geq 0, \forall\nu \geq 0, \sum \nu = 1\}$	$\widehat{\mathcal{P}} = \{\begin{pmatrix}\xi x \\ \xi\end{pmatrix} = \widehat{L}\lambda' + \widehat{R}\mu', \forall\lambda', \forall\mu' \geq 0\}$ $\widehat{L} = \left[\begin{array}{c} L \\ \hline 0\cdots 0 \end{array}\right]$ $\widehat{R} = \left[\begin{array}{c\|c} R & V \\ \hline 0\cdots 0 & 1\cdots 1 \end{array}\right]$
Sommets	$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}, v \in V$ rationnel	$\widehat{r_v} = \begin{pmatrix} \xi_0 v_1 \\ \xi_0 v_2 \\ \vdots \\ \xi_0 v_n \\ \xi_0 \end{pmatrix}, \xi_0 > 0, \widehat{r_v} \in \widehat{R}$
Rayons	$r = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix}, r \in R$	$\widehat{r_r} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \\ 0 \end{pmatrix}, \widehat{r_r} \in \widehat{R}$
Lignes	$l = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{pmatrix}, l \in L$	$\widehat{l} = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \\ 0 \end{pmatrix}, \widehat{l} \in \widehat{L}$

TAB. 1.1 - *Dualité entre polyèdre et cône.*

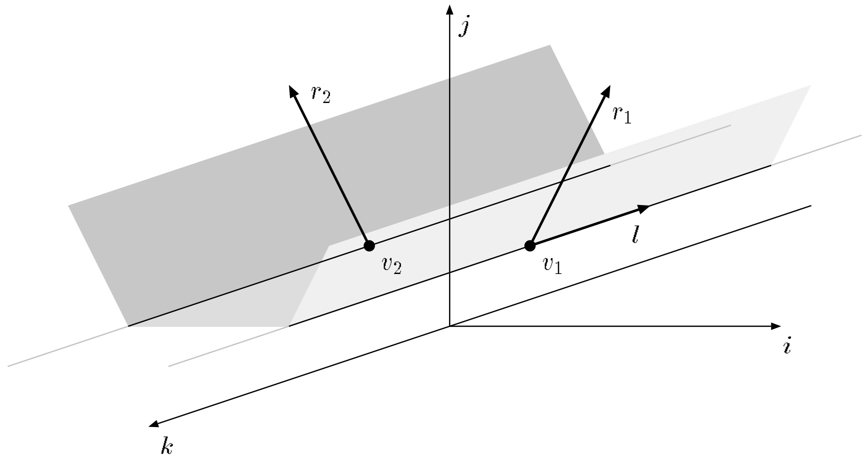


FIG. 1.1 - *Le polyèdre* $\mathcal{P} = \{(i,j,k) \mid -j \leq i \leq j, j \geq 1\}$, *ses sommets* v_1 *et* v_2, *rayons* r_1 *et* r_2 *et sa ligne* l.

Ce polyèdre a une ligne $L = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, deux rayons $R = \begin{pmatrix} -1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}$, et deux sommets $V = \begin{pmatrix} -1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}$. Cette représentation n'est pas unique.

$$\begin{aligned}\mathcal{P} \;=\; \{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \lambda + \begin{pmatrix} -1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \end{pmatrix}, \\ \mu_1, \mu_2, \nu_1, \nu_2 \geq 0, \nu_1 + \nu_2 = 1 \}\end{aligned}$$

Dans l'espace homogène ce polyèdre s'écrit :

$$\widehat{\mathcal{P}} \;=\; \left\{ \begin{pmatrix} \xi i \\ \xi j \\ \xi k \\ \xi \end{pmatrix} \in \mathbb{Q}^4 \;\middle|\; \begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \xi i \\ \xi j \\ \xi k \\ \xi \end{pmatrix} \geq 0 \right\}$$

et sous la forme de Minkowski :

$$\widehat{\mathcal{P}} \;=\; \left\{ \begin{pmatrix} \xi i \\ \xi j \\ \xi k \\ \xi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \lambda + \begin{pmatrix} -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \nu_1 \\ \nu_2 \end{pmatrix}, \quad \mu_1, \mu_2, \nu_1, \nu_2 \geq 0, \xi \geq 0 \right\}$$

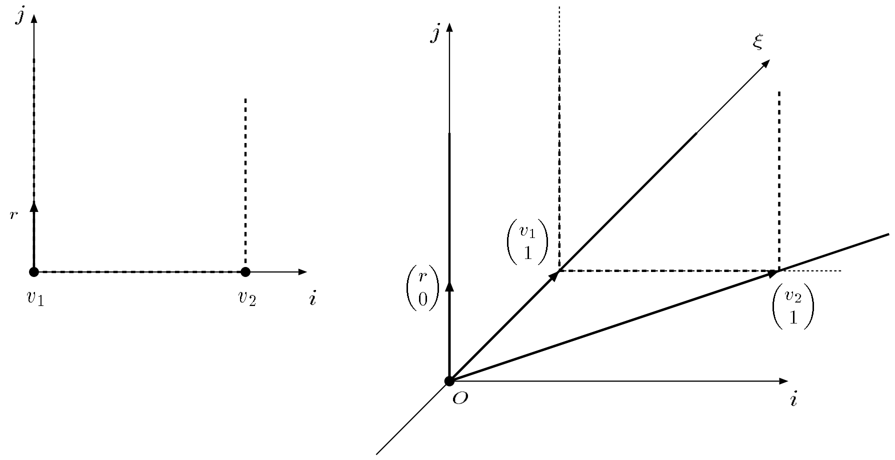


FIG. 1.2 - *Le polyèdre* $\mathcal{P} = \{(i,j) \in \mathbb{Q}^2 \mid 0 \leq i \leq 5, j \geq 0\}$ *et sa représentation homogène.*

Exemple 2

Soit $\mathcal{P}$ le polyèdre représenté figure 1.2 :

$$\mathcal{P} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \middle| \, 0 \leq i \leq 5, j \geq 0 \right\}$$

Ce polyèdre a un rayon $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ et deux sommets $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ et $\begin{pmatrix} 5 \\ 0 \end{pmatrix}$. La forme de Minkowski est donc :

$$\mathcal{P} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mu + \begin{pmatrix} 0 & 5 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \end{pmatrix}, \quad \mu, \nu_1, \nu_2 \geq 0, \quad \nu_1 + \nu_2 = 1 \right\}$$

La représentation homogène implicite est :

$$\widehat{\mathcal{P}} = \left\{ \begin{pmatrix} \xi i \\ \xi j \\ \xi \end{pmatrix} \middle| \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \xi i \\ \xi j \\ \xi \end{pmatrix} \geq 0 \right\}$$

La forme de Minkowski homogène est :

$$\widehat{\mathcal{P}} = \left\{ \begin{pmatrix} \xi i \\ \xi j \\ \xi \end{pmatrix} = \begin{pmatrix} 0 & 0 & 5 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix}, \quad \mu_1, \mu_2, \mu_3 \geq 0, \quad \xi \geq 0 \right\}$$

1.1.2 Propriétés

Définition 22 (dimension géométrique) *On appelle* dimension géométrique *d'un polyèdre la dimension du plus petit espace affine pouvant contenir le polyèdre :*

$$\begin{aligned} d(\mathcal{P}) &= n - \textit{nombre de lignes non redondantes de } A \\ &= n - \textit{nombre d'égalités non redondantes} \end{aligned} \tag{1.5}$$

On appelle d-polyèdre *un polyèdre de dimension géometrique* d.

Définition 23 (saturation) *On dit qu'un rayon ou une ligne* r *d'un polyèdre homogène* sature *une inégalité* $c \cdot x \geq 0$ *(*c *une ligne de la matrice* $\widehat{B}$*) ou une égalité* $c \cdot x = 0$ *(*c *une ligne de la matrice* $\widehat{A}$*) du système homogène définissant le polyèdre, lorsque l'égalité* $c \cdot r = 0$ *est vérifiée.*

Définition 24 (matrice d'incidence) *La* matrice d'incidence S *est une matrice booléenne dont les lignes correspondent aux contraintes (lignes des matrices* $\widehat{A}$ *et* $\widehat{B}$*, équation* (1.3)*) et les colonnes correspondent aux rayons et lignes (colonnes des matrices* $\widehat{L}$ *et* $\widehat{R}$*, équation* (1.4)*). Chaque élément* $s_{i,j}$ *de* S *est défini de la manière suivante :*

$$s_{i,j} = \begin{cases} 1, \ \textit{si le rayon ou la ligne } r_j \textit{ sature la contrainte } c_i, \\ \qquad\qquad\qquad\qquad \textit{c'est-à-dire si } c_i \cdot r_j = 0 \\ 0, \ \textit{sinon, i.e. si } c_i \cdot r_j > 0 \end{cases}$$

En substituant $\begin{pmatrix} \xi x \\ \xi \end{pmatrix}$ par sa valeur donnée par l'équation (1.4) dans l'équation (1.3), on obtient :

$$\forall (\mu \geq 0, \ \lambda) \ : \ \begin{cases} \widehat{A}\widehat{L}\lambda + \widehat{A}\widehat{R}\mu = 0 \\ \widehat{B}\widehat{L}\lambda + \widehat{B}\widehat{R}\mu \geq 0 \end{cases} \Longrightarrow \begin{cases} \widehat{A}\widehat{L} = 0, \ \widehat{A}\widehat{R} = 0 \\ \widehat{B}\widehat{L} = 0, \ \widehat{B}\widehat{R} \geq 0 \end{cases} \tag{1.6}$$

où les lignes de $\widehat{A}$ et $\widehat{B}$ sont respectivement les égalités et les inégalités homogènes, et les colonnes de $\widehat{L}$ et $\widehat{R}$ sont respectivement les lignes et les rayons unidirectionnels du polyèdre homogène.

On déduit de l'équation (1.6) que les lignes de la matrice S associées aux équations ($\widehat{A}$) sont saturées, et les colonnes de S associées aux lignes ($\widehat{L}$) sont saturées. Seuls les éléments de S associés aux inégalités $\widehat{B}$ et aux rayons $\widehat{R}$ peuvent prendre la valeur 0. La forme générale de la matrice d'incidence S est donc :

S	$\widehat{L}$	$\widehat{R}$
$\widehat{A}$	(1)	(1)
$\widehat{B}$	(1)	(0 ou 1)

Définition 25 (matrice d'incidence réduite) *On appelle* matrice d'incidence réduite, *la matrice d'incidence construite à partir des inégalités et des rayons seulement.*

Définition 26 (k-face) *On appelle* face *d'un polyèdre* $\mathcal{P} = \{Ax = a, Bx \geq b\}$ *un sous ensemble* $\mathcal{F}$ *de* $\mathcal{P}$ *défini par :*

$$\mathcal{F} = \mathcal{P} \cap \left(\bigcap_{i \in E} (B_i x = b_i) \right)$$

où E *est un ensemble de valeurs valides de* i, B_i *est une ligne de la matrice* B, *et* b_i *est l'élément correspondant du vecteur* b.

Toute face $\mathcal{F}$ *d'un polyèdre* $\mathcal{P}$ *est elle-même un polyèdre, de dimension inférieure ou égale. On dit que* $\mathcal{F}$ *est une* k*-face si* $\mathcal{F}$ *est de dimension géométrique* k.

Toute face $\mathcal{F}$ de $\mathcal{P}$ vérifie l'une de ces 2 conditions :

- $\mathcal{F}$ est l'intersection de $\mathcal{P}$ avec un hyperplan tangent à $\mathcal{P}$.
- $\mathcal{F} = \mathcal{P}$

Les 0-faces d'un d-polyèdre pointé sont ses sommets, les 1-faces ses arêtes, les $(d-1)$-faces ses facettes.

On note f_k le nombre de k-faces d'un polyèdre.

Chaque ligne de la matrice d'incidence réduite correspond à une inégalité, donc à une facette du polyèdre. Chacune de ses colonnes correspond à un rayon ou un sommet. La matrice d'incidence réduite est la représentation combinatoire du polyèdre par ses facettes par rapport à ses rayons ou lignes : elle décrit l'inclusion de chaque rayon ou ligne à chacune des facettes du polyèdre.

Propriété 7 *Soient* f *et* g *des polyèdres. La relation* $f \vdash g$, *"f est une face de g" est transitive et anti-symétrique. La relation* $\vdash$ *définit donc un ordre partiel sur les faces d'un polyèdre* $\overline{\mathcal{P}}$.

Définition 27 (face-lattice) *Le* graphe des faces *d'un polyèdre* $\mathcal{P}$ *est un graphe acyclique dont les sommets sont les faces de* $\mathcal{P}$, *et les arêtes sont construites avec la relation* $\vdash$ *ci-dessus.*

L'unique sommet du graphe des faces n'ayant pas d'arcs entrants est le polyèdre $\mathcal{P}$ lui-même, l'unique sommet n'ayant pas d'arcs sortants est la face vide $\emptyset$ appelée la (-1)-face.

La figure 1.3 donne la forme générale du graphe des faces. Toutes les faces, sauf la (-1)-face, contiennent $lin(\mathcal{P})$, et vérifient les égalités $Ax = a$.

La propriété d'inclusion est conservée dans la représentation homogène. Le graphe des faces du polyèdre homogène est donc le même que le graphe des faces du polyèdre non homogène correspondant, à la dimension supplémentaire près.

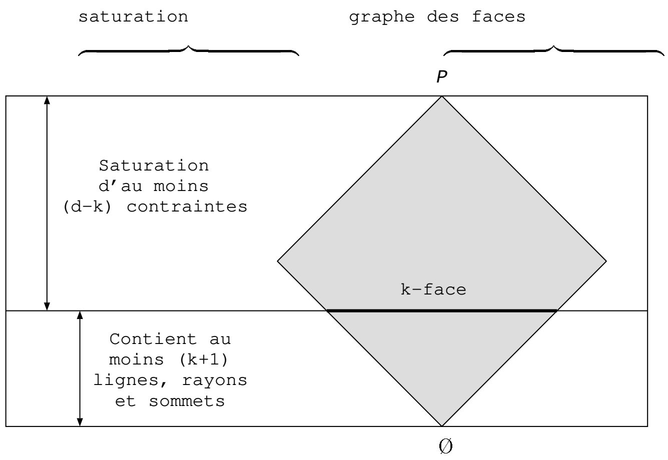


FIG. 1.3 - *Une k-face dans le graphe des faces.*

1.1.3 Algorithme de parcours des faces d'un polyèdre

L'algorithme décrit ci-dessous est purement combinatoire. Il consiste à faire les intersections d'un ensemble de lignes de la matrice d'incidence afin d'obtenir la liste des lignes, rayons et sommets saturant l'ensemble des contraintes correspondantes. Les lignes, rayons et sommets, ainsi que la matrice d'incidence sont calculés par l'algorithme de Chernikova implémenté dans la PolyLib [Le 92]. L'algorithme de parcours des faces d'un polyèdre a été implémenté en C dans la bibliothèque *PolyLib* dans le cadre de cette thèse. Cet algorithme repose sur la propriété suivante :

Propriété 8 *$\mathcal{F}$ est une k-face d'un polyèdre $\mathcal{P}$ si et seulement si les deux conditions suivantes sont vérifiées :*

- *$\mathcal{F}$ sature au moins $(d-k)$ égalités et inégalités, dont toutes les égalités de $\mathcal{P}$.*
- *$\mathcal{F}$ contient $k+1$ lignes, rayons et sommets indépendants.*

Cette seconde condition est relativement coûteuse à vérifier. Cependant, dans le contexte de recherche des sommets paramétrés, il faut trouver $k+1$ lignes et rayons homogènes indépendants (section 1.3). Nous allons donc construire la matrice contenant toutes les lignes et rayons, puis faire une triangularisation. Si tous les éléments diagonaux de la matrice résultante sont non nuls, la face est valide.

Afin d'éviter de parcourir plusieurs fois la même face, l'algorithme ci-dessous utilise l'ordre des contraintes pour ne sélectionner que la première occurrence parcourue d'une k-face. Soit C un ensemble de contraintes correspondant à une nouvelle face. Soit i_M l'indice de contrainte le plus grand de C. L'algorithme vérifie que toutes les contraintes d'indice inférieur à i_M et non contenues dans C ne saturent pas cette face. Si l'une d'entre elles sature la face, elle a déjà été reportée.

Algorithme de parcours des k-faces d'un d-polyèdre $\mathcal{P}$

Entrée: polyèdre $\mathcal{P}$.
matrice S : matrice d'incidence réduite (booléenne).
entier k : dimension des faces à parcourir.

Variables: C : contraintes (vecteur booléen).
c : numéro de contrainte.
E : ensemble des rayons saturant les contraintes C (vecteur booléen).
r : numéro de rayon.
$ReportF$: booléen. vrai si $\mathcal{F}$ est une face non redondante.

```
begin
  forall C = combinaison de (d − k) contraintes dont toutes les égalités de P do
    E = 0
    forall c ∈ C do              /* Calcul de E */
      E = E ∩ S[c]
    done
    ReportF = true               /* Test de redondance */
    for i = 0 to last_constraint(E) do
      if i ∉ C and S[i] ∩ E = E then
        ReportF = false
        break
      endif
    done
    if ReportF then
      /* Suite de l'algo */
    endif
  done
end
```

Discussion

Cet algorithme combinatoire trouve les k-faces en générant toutes les combinaisons de k' inégalités parmi les q inégalités du système de contraintes (avec $k' = k - dim(lin(\mathcal{P}))$), puis, pour chacune de ces $C_q^{k'}$ combinaisons, fait l'intersection de k' r-vecteurs binaires (où r est le nombre de rayons et sommets du polyèdre), en $O(k'r)$. L'élimination de faces redondantes est faite en $O((q-k')r)$. Cet algorithme est donc de complexité $O(C_q^{k'}qr)$, sans compter l'élimination de Gauss effectuée par la suite.

Fukuda et Rosta, dans [FR94], utilisent un algorithme qui calcule le graphe des faces de polytopes jusqu'au rang k, en partant soit des facettes, soit des sommets. Ils démontrent qu'il est de complexité $O({f_k}^2\ d\ \min\{f_{d-1}, f_0\})$, où d est la dimension du polyèdre, f_k le nombre de k-faces, f_0 le nombre de sommets et f_{d-1} le nombre de facettes. L'algorithme de Fukuda et Rosta est plus coûteux en mémoire que le notre, et sa complexité, si une élimination de Gauss est nécessaire ensuite, n'est pas toujours meilleure.

1.2 Polyèdres paramétrés

1.2.1 Représentation

Définition 28 (polyèdre paramétré) *On appelle* polyèdre paramétré *un polyèdre défini par un système de contraintes dont la partie constante dépend linéairement de paramètres* $p = \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} \in \mathbb{Q}^m$ *:*

$$\mathcal{D}(p) = \{x \in \mathbb{Q}^n \mid Ax = A'p + a, \quad Bx \geq B'p + b\} \tag{1.7}$$

où : A, A', B et B' sont des matrices entières, a et b des vecteurs entiers.

Ce polyèdre paramétré peut être représenté dans l'*espace combiné* des indices et des paramètres de la manière suivante :

$$\mathcal{D}' = \left\{ \begin{pmatrix} x \\ p \end{pmatrix} \in \mathbb{Q}^{n+m} \middle| [A \mid -A'] \begin{pmatrix} x \\ p \end{pmatrix} = a, \quad [B \mid -B'] \begin{pmatrix} x \\ p \end{pmatrix} \geq b \right\} \tag{1.8}$$

$\mathcal{D}'$ est un polyèdre non paramétré, défini dans l'espace de dimension $n + m$.

La propriété suivante donne la relation entre un polyèdre paramétré $\mathcal{D}(p)$ et le *polyèdre combiné* $\mathcal{D}'$ associé :

Propriété 9 *Soit $\mathcal{S}(p)$ le sous-espace affine fonction de $p \in \mathbb{Q}^m$ défini par :*

$$\mathcal{S}(p) = \left\{ \begin{pmatrix} x \\ p \end{pmatrix} \in \mathbb{Q}^{n+m} \right\}$$

Soit $\mathcal{D}'$ le polyèdre combiné associé au polyèdre paramétré $\mathcal{D}(p)$. On a alors :

$$\begin{pmatrix} \mathcal{D}(p) \\ p \end{pmatrix} = \mathcal{D}' \cap \mathcal{S}(p) \tag{1.9}$$

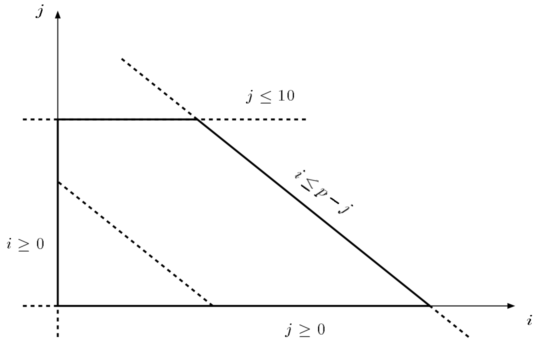


FIG. 1.4 - *Le polyèdre paramétré* $\mathcal{D}_1(p)$.

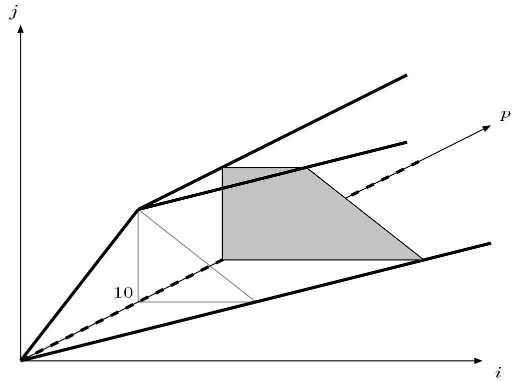


FIG. 1.5 - *Le polyèdre combiné* $\mathcal{D}_1'$.

Exemple 1

Soit $\mathcal{D}_1(p)$ le polyèdre à un paramètre p:

$$\begin{aligned}
\mathcal{D}_1(p) &= \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid i \geq 0,\ i \leq p - j,\ j \geq 0,\ j \leq 10 \right\} \\
&= \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid \begin{pmatrix} 1 & 0 \\ -1 & -1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \end{pmatrix} p + \begin{pmatrix} 0 \\ 0 \\ 0 \\ -10 \end{pmatrix} \right\}
\end{aligned}$$

Le polyèdre combiné associé à $\mathcal{D}_1(p)$ est :

$$\mathcal{D}_1' = \left\{ \begin{pmatrix} i \\ j \\ p \end{pmatrix} \in \mathbb{Q}^3 \middle| \begin{pmatrix} 1 & 0 & 0 \\ -1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \\ p \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ -10 \end{pmatrix} \right\}$$

$\mathcal{S}_1(p)$ est un plan parallèle aux axes i et j. Dans la figure 1.5, le plan grisé correspond à une instance du polyèdre $\begin{pmatrix} \mathcal{D}_1(p) \\ p \end{pmatrix} = \mathcal{D}_1' \cap \mathcal{S}_1(p)$. On voit sur cette figure que lorsque p est inférieur ou égal à 10, $\mathcal{D}_1(p)$ possède trois sommets, alors qu'il y en a quatre quand p est supérieur à 10.

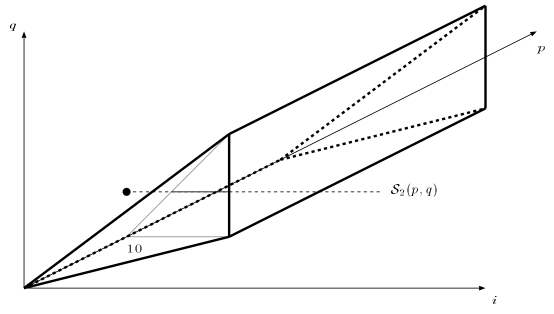


FIG. 1.6 - *Le polyèdre combiné $\mathcal{D}_2'$.*

Exemple 2

Soit $\mathcal{D}_2$ le polyèdre à deux paramètres $p \geq 0$ et $q \geq 0$:

$$\mathcal{D}_2(p,q) = \left\{ (i) \in \mathbb{Q} \mid i \leq p, i \leq 10, 2i \geq q, i \geq p - 25 \right\}$$

Le polyèdre combiné $\mathcal{D}_2'$ avec le contexte $\{p \geq 0, q \geq 0\}$ est représenté figure 1.6 :

$$\begin{aligned}
\mathcal{D}_2' &= \left\{ \begin{pmatrix} i \\ p \\ q \end{pmatrix} \in \mathbb{Q}^3 \middle| i \leq p, i \leq 10, 2i \geq q, i \geq p-25, p \geq 0, q \geq 0 \right\} \\
&= \left\{ \begin{pmatrix} i \\ p \\ q \end{pmatrix} \in \mathbb{Q}^3 \middle| \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 2 & 0 & -1 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ p \\ q \end{pmatrix} \geq \begin{pmatrix} 0 \\ -10 \\ 0 \\ -25 \\ 0 \\ 0 \end{pmatrix} \right\}
\end{aligned}$$

Une instance du polyèdre $\mathcal{D}_2(p,q)$ est obtenue en faisant l'intersection de $\mathcal{D}_2'$ avec la droite $\mathcal{S}_2(p,q)$ parallèle à l'axe i.

1.2.2 Théorèmes fondamentaux

Théorème 7 *La représentation de Minkowski, sous forme de lignes, rayons et sommets d'un polyèdre paramétré est de la forme :*

$$\mathcal{D}(p) = \left\{ x \in \mathbb{Q}^n \mid x = L\lambda + R\mu + V(p)\nu,\ \forall\lambda,\ \forall\mu \geq 0,\ \forall\nu \geq 0,\ \sum \nu = 1 \right\}$$

où L et R, les matrices contenant les lignes et les rayons du polyèdre paramétré, sont des matrices entières constantes (ne dépendant pas du paramètre p), et $V(p)$, la matrice *contenant les sommets du polyèdre, est une matrice rationnelle dont chaque coefficient est une fonction affine du paramètre p définie sur un certain domaine de validité de p.*

Preuve du théorème 7, première partie.

Démontrons d'abord que les lignes et rayons du polyèdre paramétré $\mathcal{D}(p)$ sont indépendants du paramètre.

Il suffit pour cela de prouver que les vecteurs normaux aux facettes de $\mathcal{D}(p)$ sont constants. En effet, les lignes et rayons L et R sont portés par des intersections de facettes de $\mathcal{D}(p)$. Si ces facettes sont de directions indépendantes de p, alors L et R seront indépendants de p.

Chaque facette $\mathcal{F}$ de $\mathcal{D}(p)$ est portée par un hyperplan $H(p)$ défini par :

$$H(p) = \{x \in \mathbb{Q}^n \mid ax = bp + c\}$$

où a et b sont des vecteurs entiers constants, et c est une constante entière.

Dans $\mathbb{Q}^n$, cette facette est de manière évidente orthogonale au vecteur a pout tout p donné. Les facettes changent de position suivant la valeur du paramètre p, mais ne changent jamais de direction. Les lignes et rayons de $\mathcal{D}(p)$ sont donc constants.

□

Il reste à démontrer que la matrice $V(p)$ est une matrice rationnelle dont chaque coefficient est une fonction affine du paramètre p définie sur un certain domaine de validité de p. Quelques résultats liant le polyèdre paramétré $\mathcal{D}(p)$ au polyèdre combiné associé $\mathcal{D}'$ seront utilsés dans cette démonstration :

Lemme 1 *Soient $v(p)$ un sommet paramétré du polyèdre $\mathcal{D}(p) \in \mathbb{Q}^n$ dépendant d'un vecteur quelconque $p \in \mathbb{Q}^m$, $\mathcal{D}'$ le polyèdre combiné associé à $\mathcal{D}(p)$, $\mathcal{S}(p)$ le sous-espace affine défini par : $\mathcal{S}(p) = \left\{ \begin{pmatrix} x \\ p \end{pmatrix} \in \mathbb{Q}^{n+m} \right\}$.*

Il existe une m-face $\mathcal{F}(\mathcal{D}')$ telle que :

$$\begin{pmatrix} v(p) \\ p \end{pmatrix} = \mathcal{F}(\mathcal{D}') \cap \mathcal{S}(p) \tag{1.10}$$

Preuve du lemme 1.

La figure 1.7 illustre les relations employées dans cette démonstration.

Supposons le polyèdre $\mathcal{D}(p)$ pointé. Si ce n'est pas le cas, on construit facilement le polyèdre pointé $\overline{\mathcal{D}}(p)$ associé à $\mathcal{D}(p)$. Les sommets paramétrés de $\overline{\mathcal{D}}(p)$ sont identiques à ceux de $\mathcal{D}(p)$. Considérons ce polyèdre pointé dans la suite de cette démonstration.

Soit $v(p)$ un sommet paramétré du polyèdre pointé $\mathcal{D}(p)$. Par définition d'une 0-face d'un polyèdre pointé, il existe un hyperplan $H(p)$ tangent à $\mathcal{D}(p)$ tel que $H(p) \cap \mathcal{D}(p) = v(p)$ et $H(p) \cap \mathcal{D}(p)$ est de dimension 0.

$$\begin{tikzcd}[column sep=huge, row sep=large]
\text{Espace combiné } \mathbb{Q}^{n+m} & \text{Espace } \mathbb{Q}^{n} \\
H' \arrow[r, "{\binom{H(p)}{p} = H' \cap \mathcal{S}(p)}"'] \arrow[d, "\text{tangent à}"] & H(p) \arrow[d, "\text{tangent à}"] \\
\mathcal{D}' \arrow[r, "{\binom{\mathcal{D}(p)}{p} = \mathcal{D}' \cap \mathcal{S}(p)}"'] \arrow[d, "\text{en}"] & \mathcal{D}(p) \arrow[d, "\text{en}"] \\
\mathcal{F}(\mathcal{D}') \arrow[r, "{\binom{v(p)}{p} = \mathcal{F}(\mathcal{D}') \cap \mathcal{S}(p)}"'] & v(p)
\end{tikzcd}$$

FIG. 1.7 - *Relations utilisées dans la preuve du lemme 1.*

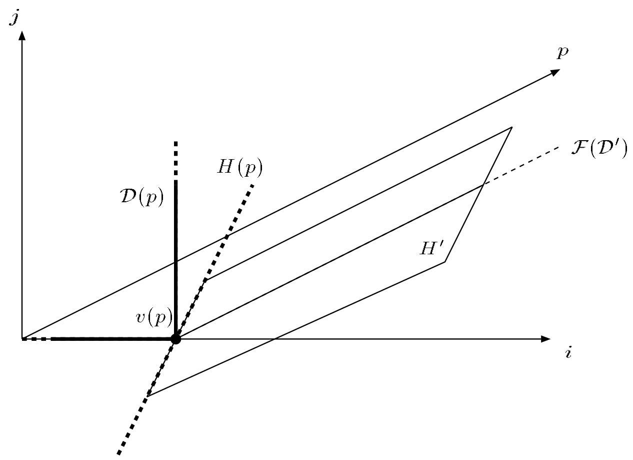


FIG. 1.8 - *L'hyperplan $H(p)$ dans le plan $\mathbb{Q}^n$ et H' portant $\mathcal{F}(\mathcal{D}')$ dans l'espace combiné.*

Soit $H(p)$ tel que $\begin{pmatrix} H(p) \\ p \end{pmatrix} = H' \cap \mathcal{S}(p)$. H' est un hyperplan de l'espace combiné $\mathbb{Q}^{n+m}$ qui définit l'hyperplan paramétré $H(p)$ dans l'espace $\mathbb{Q}^n$. H' est un hyperplan tangent à $\mathcal{D}'$ (figure 1.8). On a alors :

$$\begin{aligned} v(p) &= H(p) \cap \mathcal{D}(p) \\ \begin{pmatrix} v(p) \\ p \end{pmatrix} &= (H' \cap \mathcal{S}(p)) \cap (\mathcal{D}' \cap \mathcal{S}(p)) \\ \begin{pmatrix} v(p) \\ p \end{pmatrix} &= H' \cap \mathcal{D}' \cap \mathcal{S}(p) \end{aligned} \tag{1.11}$$

Le point $\begin{pmatrix} v(p) \\ p \end{pmatrix}$ est obtenu en faisant l'intersection $H' \cap \mathcal{D}' \cap \mathcal{S}(p)$. Le polyèdre $H' \cap \mathcal{D}'$ est au moins de dimension m, car le vecteur p contient m éléments indépendants. D'autre part, $H' \cap \mathcal{D}'$ est au plus de dimension m car l'intersection avec $\mathcal{S}(p)$ (de dimension n) est de dimension 0 (c'est le sommet $\begin{pmatrix} v(p) \\ p \end{pmatrix}$). Donc le polyèdre $H' \cap \mathcal{D}'$ est exactement de dimension m. Comme H' est un plan tangent à $\mathcal{D}'$, $H' \cap \mathcal{D}'$ est une m-face de $\mathcal{D}'$. L'équation (1.11) peut être réécrite :

$$\begin{pmatrix} v(p) \\ p \end{pmatrix} = \mathcal{F}(\mathcal{D}') \cap \mathcal{S}(p)$$

où $\mathcal{F}(\mathcal{D}')$ est une m-face de $\mathcal{D}'$.
□

Corollaire 1 *Soit $v(p)$ le sommet paramétré correspondant à la face $\mathcal{F}(\mathcal{D}')$. Le domaine de validité sur p pour lequel $v(p)$ est défini est la projection π de la face $\mathcal{F}(\mathcal{D}')$ dans l'espace des paramètres $\mathbb{Q}^m$:*

$$v(p) \text{ est défini ssi } p \in \pi(\mathcal{F}(\mathcal{D}')) \tag{1.12}$$

Preuve du corollaire 1.

Le domaine des valeurs de p tel que $\mathcal{F}(\mathcal{D}') \cap \mathcal{S}(p) \neq \emptyset$ est évidemment $\pi(\mathcal{F}(\mathcal{D}'))$, la projection de la face $\mathcal{F}(\mathcal{D}')$ sur l'espace des paramètres $\mathbb{Q}^m$:

$$p \in \pi(\mathcal{F}(\mathcal{D}')) \Rightarrow \mathcal{F}(\mathcal{D}') \cap \mathcal{S}(p) \neq \emptyset$$

□

Lemme 2 *Soient $\mathcal{F}(\mathcal{D}')$ une m-face du polyèdre $\mathcal{D}'$, le polyèdre combiné associé au polyèdre paramétré $\mathcal{D}(p)$, $\begin{pmatrix} x \\ p \end{pmatrix}$ un point de cette face : $\begin{pmatrix} x \\ p \end{pmatrix} \in \mathcal{F}(\mathcal{D}')$.*

L'une de ces deux relations est vraie :

i.) x est une fonction affine de p, i.e. $x = v(p)$ où $v(p)$ est un sommet paramétré de $\mathcal{D}(p)$.

ii.) x est libre, i.e. pour une valeur donnée de p, il existe plusieurs valeurs de x telles que $\begin{pmatrix} x \\ p \end{pmatrix} \in \mathcal{F}(\mathcal{D}')$.

Preuve du lemme 2.

Soit $\begin{pmatrix} x \\ p \end{pmatrix}$ un point de la face $\mathcal{F}(\mathcal{D}')$. Une m-face est portée par un sous-espace affine de dimension m défini par n égalités. On a donc une relation de n égalités :

$$Ux = U'p + u \tag{1.13}$$

où U est une matrice $n \times n$, U' une matrice $n \times m$, et u un vecteur de taille n.

Si la matrice U est inversible, on a :

$$x = U^{-1} \cdot U'p + U^{-1}u = v(p) \tag{1.14}$$

Ceci démontre le cas *(i.)* de ce lemme.

Si U est non-inversible, x peut prendre un ensemble de valeurs dans un polyèdre de dimension $\delta > 0$, contenu dans la face $\mathcal{F}(\mathcal{D}')$. Dans ce cas cette face ne correspond pas à un sommet paramétré x de $\mathcal{D}(p)$, mais à une face de dimension δ de $\mathcal{D}(p)$ (cas *ii.*).

□

Grâce au lemme 2 et au corollaire 1, nous allons pouvoir démontrer la deuxième partie du théorème 7, donnant la forme générale de la matrice $V(p)$:

Preuve du théorème 7, deuxième partie.

Construisons la matrice $V(p)$, contenant les sommets paramétrés du polyèdre $\mathcal{D}(p)$. Cette matrice est obtenue par concaténation des sommets $v(p)$ obtenus de la manière suivante : pour toute m-face $\mathcal{F}$ du polyèdre combiné $\mathcal{D}'$ telle que $\mathcal{F}$ est portée par le sous-espace $Ux = U'p + u$ (équation (1.13)) :

$$v(p) = \begin{cases} U^{-1} \cdot U'p + U^{-1}u & \text{si } U \text{ est inversible et pour } p \in \pi(\mathcal{F}) \\ \text{non défini} & \text{sinon.} \end{cases} \tag{1.15}$$

On a donc démontré que $V(p)$ est une matrice rationnelle dont chaque coefficient est une fonction affine du paramètre p définie sur un certain domaine de validité de p.

□

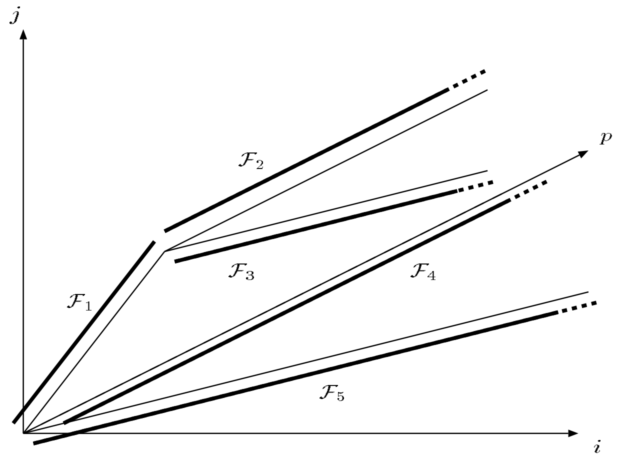


FIG. 1.9 - *Les 1-faces dans le polyèdre combiné $\mathcal{D}'_1$.*

Exemple 1 (suite)

$\mathcal{D}_1(p)$ est le polyèdre à un paramètre (figure 1.4) :

$$\mathcal{D}_1(p) = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid i \geq 0, i \leq p - j, j \geq 0, j \leq 10 \right\}$$

Les 1-faces du polyèdre combiné $\mathcal{D}'_1$ sont représentées figure 1.9. Le tableau ci-dessous donne leurs contraintes respectives, ansi que les sommets paramétrés et domaines de validité déduits de ces contraintes. Toutes les matrices U (preuve du lemme 2, cas i.)) sont inversibles :

1-face	contraintes	sommet paramétré	domaine de validité
$\mathcal{F}_1$	$i = 0, \quad j = p, \quad 0 \leq p \leq 10$	$\begin{pmatrix} 0 \\ p \end{pmatrix}$	$0 \leq p \leq 10$
$\mathcal{F}_2$	$i = 0, \quad j = 10, \quad p \geq 10$	$\begin{pmatrix} 0 \\ 10 \end{pmatrix}$	$p \geq 10$
$\mathcal{F}_3$	$i = p - 10, \quad j = 10, \quad p \geq 10$	$\begin{pmatrix} p-10 \\ 10 \end{pmatrix}$	$p \geq 10$
$\mathcal{F}_4$	$i = 0, \quad j = 0, \quad p \geq 0$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$p \geq 0$
$\mathcal{F}_5$	$i = p, \quad j = 0, \quad p \geq 0$	$\begin{pmatrix} p \\ 0 \end{pmatrix}$	$p \geq 0$

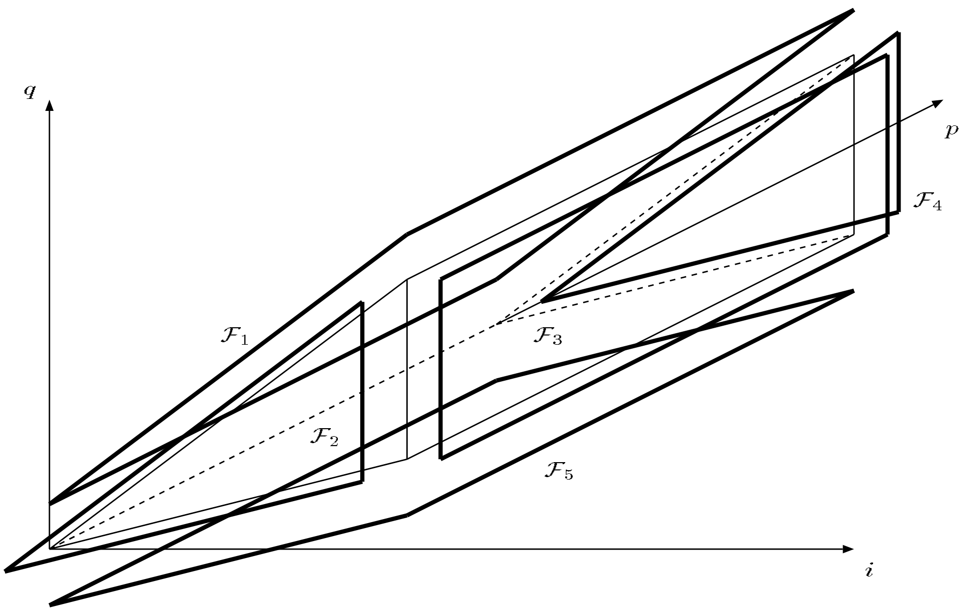


FIG. 1.10 - *Les 2-faces dans le polyèdre combiné $\mathcal{D}_2'$.*

Exemple 2 (suite)

$$\mathcal{D}_2(p,q) = \left\{ \left(i\right) \in \mathbb{Q} \mid i \leq p, i \leq 10, 2i \geq q, i \geq p - 25 \right\}$$

Les 2-faces du polyèdre combiné $\mathcal{D}_2'$ (figure 1.6) sont représentées figure 1.10. Les tableaux ci-dessous donnent leurs contraintes respectives, ansi que les sommets paramétrés et domaines de validité déduits de ces contraintes. La matrice U (preuve du lemme 2) correspondant à la face $\mathcal{F}_5$ est non inversible. En effet, pour une valeur valide donnée de (p,q), l'intersection de $\mathcal{S}_2(p,q)$ avec $\mathcal{F}_5$ est le polyèdre $\{i \geq 0, i \geq p - 35, i \leq p, i \leq 10\}$ de dimension 1. Il correspond à une arête du polyèdre $\mathcal{D}_2(p,q)$ (il correspond exactement au polyèdre $\mathcal{D}_2(p,q)$ puisqu'ils sont tous deux de dimension 1).

1-face	contraintes		
$\mathcal{F}_1$	$2i = q,$	$q/2 \leq p \leq q/2 + 25,$	$0 \leq q \leq 20$
$\mathcal{F}_2$	$i = p,$	$q/2 \leq p \leq 10,$	$q \geq 0$
$\mathcal{F}_3$	$i = 10,$	$10 \leq p \leq 35,$	$0 \leq q \leq 20$
$\mathcal{F}_4$	$i = p - 25,$	$q/2 + 25 \leq p \leq 35,$	$q \geq 0$
$\mathcal{F}_5$	$0 \leq i \leq 10,$	$i \leq p \leq i + 25,$	$q = 0$

1-face	sommet paramétré	domaine de validité
$\mathcal{F}_1$	$\left(q/2\right)$	$q/2 \le p \le q/2+25, \quad 0 \le q \le 20$
$\mathcal{F}_2$	$\left(p\right)$	$q/2 \le p \le 10, \quad q \ge 0$
$\mathcal{F}_3$	$\left(10\right)$	$10 \le p \le 35, \quad 0 \le q \le 20$
$\mathcal{F}_4$	$\left(p-25\right)$	$q/2+25 \le p \le 35, \quad q \ge 0$
$\mathcal{F}_5$	U non inversible	

1.3 Recherche des sommets d'un polyèdre paramétré

Ce paragraphe décrit l'algorithme de recherche des sommets et des domaines de validité d'un polyèdre paramétré $\mathcal{D}(p)$. Cet algorithme a été implémenté dans la bibliothèque PolyLib.

1.3.1 Réécriture de l'algorithme

Afin de diminuer la complexité de l'algorithme décrit en 1.2.2, nous le transformons de la manière suivante :

- nous utilisons les représentations homogènes, décrites dans le paragraphe 1.1.1, des polyèdres et des matrices de transformation,
- nous calculons une base de chaque m-face : cette base permet de calculer facilement les matrices de transformation correspondant aux projections dans l'espace des indices et dans l'espace des paramètres.

Théorème 8 *Soient $\mathcal{F}$ une m-face du polyèdre combiné $\mathcal{D}'$ associé au polyèdre paramétré $\mathcal{D}(p)$, $\mathcal{B}$ la matrice correspondant à la représentation homogène d'une base de la face $\mathcal{F}$. $\mathcal{B}$ est construite en prenant $m+1$ colonnes indépendantes de l'ensemble des sommets, rayons et lignes homogènes du polyèdre $\mathcal{F}$ (il existe exactement $m+1$ colonnes indépendantes car $\mathcal{F}$ est de dimension m). $\mathcal{B}$ est une matrice entière $(m+n+1)\times(m+1)$.*

Soient $\mathcal{B}_P$ et $\mathcal{B}_D$ les restrictions de la matrice $\mathcal{B}$ à l'espace homogène des paramètres $\mathbb{Q}^{m+1}$ et des indices $\mathbb{Q}^{n+1}$ respectivement : $\mathcal{B}_P$ est constituée des $m+1$ dernières lignes de $\mathcal{B}$, et $\mathcal{B}_D$ est construite en prenant les n premières lignes et la dernière ligne de $\mathcal{B}$.

Sous ces conditions, la relation suivante est vérifiée :

$$\begin{pmatrix} x \\ p \end{pmatrix} \in \mathcal{F} \Leftrightarrow \exists \begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix} \in \mathbb{Q}^{m+1} \text{ tel que } \left\{ \begin{array}{l} \begin{pmatrix} \xi_0 x \\ \xi_0 \end{pmatrix} = \mathcal{B}_D \begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix} \\ \begin{pmatrix} \xi_1 p \\ \xi_1 \end{pmatrix} = \mathcal{B}_P \begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix} \end{array} \right. \tag{1.16}$$

où ξ_0, ξ_1 et ξ_2 sont des entiers positifs.

Preuve du théorème 8.

Il suffit de prendre le vecteur $X \in \mathbb{Q}^{m+1}$, les coordonnées du point $\begin{pmatrix} x \\ p \end{pmatrix}$ de la m-face $\mathcal{F}$, dans la base homogène correspondant à la matrice $\mathcal{B}$. Le théorème est vérifié de manière évidente.

□

Le théorème suivant donne la nouvelle procédure de construction de $V(p)$ d'après les démonstrations du lemme 2 et du théorème 7.

Théorème 9 *La matrice $V(p)$ contenant les sommets paramétrés du polyèdre $\mathcal{D}(p)$ est définie par la concaténation des sommets $v(p)$ obtenus de la manière suivante : pour toute m-face $\mathcal{F}$ du polyèdre combiné $\mathcal{D}'$, avec $\mathcal{B}$ la matrice correspondant à la représentation homogène d'une base de $\mathcal{F}$, $\mathcal{B}_P$ et $\mathcal{B}_D$ les restrictions de la matrice $\mathcal{B}$ à l'espace homogène des paramètres $\mathbb{Q}^{m+1}$ et des indices $\mathbb{Q}^{n+1}$ respectivement :*

$$\begin{pmatrix} \xi_0 v(p) \\ \xi_0 \end{pmatrix} = \begin{cases} \mathcal{B}_D \cdot \mathcal{B}_P^{-1} \begin{pmatrix} \xi_1 p \\ \xi_1 \end{pmatrix} & \text{si } \mathcal{B}_P \text{ est inversible et pour } p \in \pi(\mathcal{F}) \\ \text{non défini} & \text{sinon.} \end{cases} \tag{1.17}$$

Preuve du théorème 9.

La condition $\mathcal{B}_P$ *est inversible* est équivalente à la condition U *est inversible* utilisée dans la preuve du lemme 2. En effet, pour qu'une valeur donnée de p corresponde à une et une seule valeur de x, il faut que la transformation qui projette le point $\begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix}$ de $\mathcal{F}$ sur l'espace homogène des paramètres $\mathbb{Q}^{m+1}$ soit inversible, et donc que la matrice $\mathcal{B}_P$ soit inversible.

D'après le théorème 8, équation (1.16), on a alors :

$$\begin{aligned} & \begin{pmatrix} \xi_1 p \\ \xi_1 \end{pmatrix} = \mathcal{B}_P \begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix} \\ \Leftrightarrow \quad & \begin{pmatrix} \xi_2 X \\ \xi_2 \end{pmatrix} = \mathcal{B}_P^{-1} \begin{pmatrix} \xi_1 p \\ \xi_1 \end{pmatrix} \\ \Leftrightarrow \quad & \begin{pmatrix} \xi_0 x \\ \xi_0 \end{pmatrix} = \mathcal{B}_D \cdot \mathcal{B}_P^{-1} \begin{pmatrix} \xi_1 p \\ \xi_1 \end{pmatrix} \end{aligned} \tag{1.18}$$

□

Ce nouvel algorithme est utilisé pour calculer les sommets paramétrés des exemples 1 et 2, présentés dans les tableaux suivants.

1-face	$\mathcal{B}$	$\mathcal{B}_D$	$\mathcal{B}_P$	$\mathcal{B}_P^{-1}$	$\mathcal{B}_D \cdot \mathcal{B}_P^{-1}$	sommet
$\mathcal{F}_1$	$\begin{pmatrix} 0 & 0 \\ 0 & 10 \\ 0 & 10 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 10 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{10} & 1 \\ \frac{1}{10} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ p \end{pmatrix}$
$\mathcal{F}_2$	$\begin{pmatrix} 0 & 0 \\ 0 & 10 \\ 1 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 10 \end{pmatrix}$
$\mathcal{F}_3$	$\begin{pmatrix} 1 & 0 \\ 0 & 10 \\ 1 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -10 \\ 0 & 10 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} p-10 \\ 10 \end{pmatrix}$
$\mathcal{F}_4$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$\mathcal{F}_5$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} p \\ 0 \end{pmatrix}$

TAB. 1.2 - **Exemple 1 (suite)**

Exemple 2 (suite)

1-face	$\mathcal{B}$	$\mathcal{B}_D$	$\mathcal{B}_P$	$\mathcal{B}_P^{-1}$	$\mathcal{B}_D \cdot \mathcal{B}_P^{-1}$	sommet
$\mathcal{F}_1$	$\begin{pmatrix} 0 & 10 & 0 \\ 0 & 10 & 25 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 & 25 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{25} & -\frac{3}{100} & 1 \\ 0 & \frac{1}{20} & 0 \\ \frac{1}{25} & -\frac{1}{50} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$(q/2)$
$\mathcal{F}_2$	$\begin{pmatrix} 0 & 10 & 10 \\ 0 & 10 & 10 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 & 10 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 & 10 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{10} & 0 & 1 \\ 0 & \frac{1}{20} & 0 \\ \frac{1}{10} & -\frac{1}{20} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	(p)
$\mathcal{F}_3$	$\begin{pmatrix} 10 & 10 & 10 \\ 10 & 10 & 35 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 10 & 10 & 10 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 10 & 10 & 35 \\ 0 & 20 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{25} & -\frac{1}{20} & \frac{7}{5} \\ 0 & \frac{1}{20} & 0 \\ \frac{1}{25} & 0 & -\frac{2}{5} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 10 \\ 0 & 0 & 1 \end{pmatrix}$	(10)
$\mathcal{F}_4$	$\begin{pmatrix} 0 & 10 & 10 \\ 25 & 35 & 35 \\ 0 & 0 & 20 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 10 & 10 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 25 & 35 & 35 \\ 0 & 0 & 20 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -\frac{1}{10} & 0 & \frac{7}{2} \\ \frac{1}{10} & -\frac{1}{20} & -\frac{5}{2} \\ 0 & \frac{1}{20} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & -25 \\ 0 & 0 & 1 \end{pmatrix}$	$(p-25)$
$\mathcal{F}_5$	$\begin{pmatrix} 0 & 0 & 10 \\ 0 & 25 & 10 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 10 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 25 & 10 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\mathcal{B}_P$ non inversible		

1.3.2 Algorithme de recherche des sommets et des domaines de validité d'un polyèdre paramétré

Cet algorithme se greffe sur l'algorithme de parcours des k-faces d'un polyèdre donné en 1.1.3, et qui a fait appel à l'algorithme de Chernikova pour trouver les lignes, rayons et sommets d'un polyèdre. La boucle externe consiste à parcourir les m-faces du polyèdre combiné $\mathcal{D}'$, puis à construire une matrice $\mathcal{B}$ contenant $m+1$ colonnes indépendantes correspondant à une base de chaque face.

Les matrices $\mathcal{B}_P$ et $\mathcal{B}_D$ sont ensuite calculées. Si $\mathcal{B}_P$ est inversible, l'algorithme calcule l'expression paramétrée du sommet (la matrice $\mathcal{B}_D \cdot \mathcal{B}_P^{-1}$) et le domaine de validité de ce sommet paramétré (la projection de la face sur l'espace des paramètres).

Algorithme de recherche des sommets et domaines d'un polyèdre paramétré

Entrée: Polyèdre $\mathcal{D}'$: polyèdre combiné.
int m : nombre de paramètres.

Variables: $\mathcal{B}$: base homogène de la face (matrice).
$\mathcal{B}_P$: restriction de $\mathcal{B}$ à l'espace des paramètres.
$\mathcal{B}_D$: restriction de $\mathcal{B}$ à l'espace des indices.
T : matrice $B_D \cdot \mathcal{B}_P^{-1}$.
Val : domaine de validité.

Sortie: Liste L de couples (sommet paramétré,domaine de validité).

```
begin
  L = ∅
  /* parcours des m-faces de D' */
  forall F = m-face de D' de base homogène B do
    B_P = copie des m + 1 dernières lignes de B
    B_D = copie des n premières et de la dernière lignes de B
    if MatInversible(B_P) then
      T = MatProduit( B_D, MatInverse( B_P ) )
      Val = PolyProjection( F, m )
      L = L + (T, Val)
    endif
  done
end
```

La sortie de cet algorithme est une liste de couples (sommet paramétré, domaine de validité). Les domaines de validité sont des m-polyèdres convexes, correspondant aux projections des m-faces de $\mathcal{D}'$ dans l'espace des paramètres $\mathbb{Q}^m$. Leur union constitue la projection $\pi(\mathcal{D}')$ dans $\mathbb{Q}^m$, c'est-à-dire l'ensemble des valeurs valides du paramètre.

1.4 Découpage de l'espace de validité des paramètres

Ce paragraphe décrit la méthode de recherche d'une représentation des sommets d'un polyèdre paramétré plus facile à exploiter. Cette représentation est constituée d'une liste de domaines de validité dans l'espace des paramètres, et dans chacun d'entre eux, la liste complète des sommets paramétrés.

1.4.1 Algorithme

La représentation recherchée des sommets d'un polyèdre paramétré est de la forme : une liste de couples (domaine de validité, liste de sommets paramétrés), où les domaines de validité sont disjoints deux à deux au sens large (leurs intersections sont de dimension inférieure à m).

Il faut pour cela trouver une *partition* de $\pi(\mathcal{D}')$ à partir des domaines de validité donnés ci-dessus.

L'algorithme proposé ci-dessous ajoute récursivement chaque sous-domaine de l'ensemble de départ, pour créer une liste de domaines toujours disjoints. Les fonctions de la bibliothèque *PolyLib* utilisées sont l'intersection et la différence au sens large. Ces fonctions s'appliquent non pas à des polyèdres convexes, mais à des domaines constitués d'unions de convexes : la différence entre deux convexes n'est pas toujours convexe, c'est une union de convexes. La différence au sens large, notée $\setminus\!\setminus$, entre deux domaines de dimension m, consiste à faire la différence entre les deux domaines sans éliminer les faces de dimension 0 à $m-1$ du bord du résultat (cf. figure 1.11).

$$V_1 \setminus\!\setminus V_2 = (V_1 \setminus V_2) \cup (V_1 \cap \mathrm{bord}(V_2))$$

Si l'intersection de V_1 et V_2 est de dimension $m-1$ au moins, alors :

$$V_1 \setminus\!\setminus V_2 = V_3 \;\Rightarrow\; \mathrm{Dim}(V_3 \cap V_2) = m-1$$

Algorithme de découpage des domaines

Entrée: Liste L : couples (sommet T, domaine de validité Val)
$L[i]$ désigne le i-ième élément de la liste.
int m : nombre de paramètres.

Variables: P' : Liste de couples (domaine Val, liste de sommets S) avec domaines de validité disjoints.
$P'[j]$ désigne le j-ième élément de la liste.
int i : numéro de couple dans L (parcours).

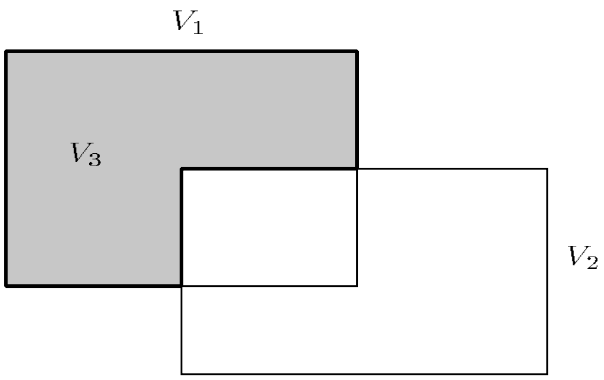


FIG. 1.11 - *Différence large entre les deux rectangles V_1 et V_2.*

int j : numéro de couple dans P (parcours).
V : domaine de validité à ajouter à P'.

Sortie: P : Liste de couples (domaine Val, liste de sommets S) avec domaines de validité disjoints.

begin
  /* initialisation de P avec le premier élément de L */
  $P = (L[1].Val, (L[1].T))$
  /* parcours des autres éléments de L */
  for $i = 2$ **to** Size(L) **do**
    $P' = ()$
    $V = L[i].Val$
    for $j = 1$ **to** Size(P) **do**
      if Dim($P[j].Val \cap V$) $= m$ **then**
        /* le domaine V et $P[j].Val$ se recouvrent : il faut les séparer */
        /* intersection de $P[j].Val$ et V */
        $P' = P' + (\ P[j].Val \cap V,\ P[j].S + (L[i].T)\ )$
        /* différence entre $P[j].Val$ et V */
        $P' = P' + (\ P[j].Val \setminus\setminus V,\ P[j].S\ )$
        /* reste de V */
        $V = V \setminus\setminus P[j].Val$
      else
        $P' = P' + (P[j])$
      endif
    done
    /* Si le reste est de dimension m, c'est un nouvel élément de P' */

```
    if Dim(V) = m then
      P' = P' + (V, (L[i].T))
    endif
    P = P'
  done
end
```

1.4.2 Preuve

Il est évident, par construction, qu'à tout élément $P[j]$ correspond un et un seul ensemble de domaines $P[j].Val$. Il reste à démontrer que l'union des domaines $P[j].Val$ de P est égale à l'union des domaines de L, et que les $P[j].Val$ sont disjoints au sens large.

Nous allons procéder par récurrence sur les domaines $L[i].Val$, et démontrer à chaque étape que :

(a) l'union des domaines de P est égale à l'union des domaines de L.

(b) les différents domaines de P sont disjoints au sens large.

Initialisation de la récurrence : $P = \{L[1]\}$.
Les deux conditions (a) et (b) ci-dessus sont vérifiées de manière évidente.

Récurrence : Supposons que $P = \{P_1, ..., P_v\}$ correspondant à $L[1], ..., L[i-1]$ vérifie les conditions (a) et (b). Montrons que P' construit à partir de P et de $L[i]$ vérifie aussi ces deux conditions.
Soit $(\delta_i)_{1\leq i\leq v+1}$ la suite définie de la manière suivante :

$$\delta_1 = \mathrm{L[i]}$$

$$\delta_{j+1} = \delta_j \setminus\!\!\setminus P_j, \quad 1 \leq j \leq v$$

On peut aussi écrire :

$$\delta_j = \mathrm{L[i]} \setminus\!\!\setminus (P_1 \cup P_2 \cup ... \cup P_{j-1})$$

D'après les lois fondamentales de la théorie des ensembles, on a les deux relations suivantes :

$$P_k.Val = (P_k.Val \cap \delta_k) \cup (P_k.Val \setminus\!\!\setminus \delta_k) \tag{1.19}$$

$$(P_k.Val \cap \delta_k) \text{ et } (P_k.Val \setminus\!\!\setminus \delta_k) \text{ sont disjoints.} \tag{1.20}$$

$P'.Val$ est défini de la manière suivante :

$$P'.Val = \{(P_1.Val \cap \delta_1), (P_1.Val \setminus\!\!\setminus \delta_1), ..., (P_v.Val \cap \delta_v), (P_v.Val \setminus\!\!\setminus \delta_v), \delta_{v+1}\}$$

- Vérification de la condition (a).
  $P_k.Val$ est égal à $(P_k.Val \cap \delta_k) \cup (P_k.Val \setminus\setminus \delta_k)$, donc $P'.Val$ contient tous les éléments de $P_k.Val$ contenus dans P. De plus, P' contient entièrement L[i] car P' contient δ_{v+1} et L[i] $= (P_1 \cap \delta_1) \cup ... \cup (P_v \cap \delta_v) \cup \delta_{v+1}$.
  La première condition est vérifiée.

- Vérification de la condition (b).
  Montrons que les $P'_i.Val$ sont disjoints au sens large. Pout tout j et k, les polyèdres $P_j.Val$ et $P_k.Val$ sont disjoints, on a donc :

  - $(P_j.Val \cap \delta_j)$ et $(P_k.Val \setminus\setminus \delta_k)$ sont disjoints.
  - $(P_j.Val \cap \delta_j)$ et $(P_k.Val \cap \delta_k)$ sont disjoints.
  - $(P_j.Val \setminus\setminus \delta_j)$ et $(P_k.Val \setminus\setminus \delta_k)$ sont disjoints.
  - $(P_j.Val \setminus\setminus \delta_j)$ et $(P_k.Val \cap \delta_k)$ sont disjoints.

  D'autre part, δ_{v+1} et $P_j.Val$ sont disjoints quel que soit j, $1 \leq j \leq v$, car $\delta_{v+1} =$ L[i] $\setminus\setminus (P_1.Val \cup ... \cup P_v.Val)$. Comme $P_j.Val$ est égal à $(P_j.Val \cap \delta_j) \cup (P_j.Val \setminus\setminus \delta_j)$, les polyèdres δ_{v+1} et $(P_j.Val \cap \delta_j)$ sont disjoints, et donc δ_{v+1} et $(P_j.Val \setminus\setminus \delta_j)$ sont disjoints. Tous les P'_j sont donc disjoints. La condition (b) est vérifiée.

□

1.4.3 Exemples

Exemple 1 (suite)

Le découpage de l'espace des paramètres donne deux domaines de validité :

- $Val_1 : 0 \leq p \leq 10$. Les sommets sont : $\begin{pmatrix} 0 \\ p \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} p \\ 0 \end{pmatrix}$.

- $Val_2 : p \geq 10$. Les sommets sont : $\begin{pmatrix} 0 \\ 10 \end{pmatrix}, \begin{pmatrix} p-10 \\ 10 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} p \\ 0 \end{pmatrix}$.

Exemple 2 (suite)

Dans cet exemple, il y a trois domaines de validité :

- $Val_1 : q/2 \leq p \leq 10, q \geq 0$. Les sommets sont : $\begin{pmatrix} q/2 \end{pmatrix}, \begin{pmatrix} p \end{pmatrix}$.

- $Val_2 : 10 \leq p \leq q/2 + 25, 0 \leq q \leq 20$. Les sommets sont : $\begin{pmatrix} q/2 \end{pmatrix}, \begin{pmatrix} 10 \end{pmatrix}$.

- $Val_3 : q/2 + 25 \leq p \leq 35, q \geq 0$. Les sommets sont : $\begin{pmatrix} 10 \end{pmatrix}, \begin{pmatrix} p-25 \end{pmatrix}$.

1.5 Comptage de points dans un polyèdre paramétré

Ce paragraphe présente une méthode de comptage des points d'un lattice contenu dans un polyèdre paramétré, exprimé de manière symbolique en fonction des paramètres. Elle repose sur les travaux du mathématicien Eugène Ehrhart [Ehr62, Ehr67a, Ehr67b, Ehr77], étendus et corrigés par I. G. Macdonald [Mac63, Mac71] et par R. P. Stanley [Sta83, Sta86], ainsi que sur les travaux récents de Philippe Clauss [Cla95, Cla96] en collaboration avec moi-même [CL96] et avec Doran Wilde [CLW97].

1.5.1 Lattice-polytopes paramétrés

Définition 29 (lattice) *On appelle* lattice $\mathcal{L}^n$ *l'ensemble des points*

$$\mathcal{L}^n = \{x \in \mathbb{Q}^n \mid x = \alpha_1 g_1 + ... + \alpha_n g_n, \quad \alpha_1 ... \alpha_n \in \mathbb{Z}\} \tag{1.21}$$

où $g_1, ..., g_n$ *sont des vecteurs rationnels linéairement indépendants appelés* base du lattice.

La matrice G *dont les colonnes sont constituées des vecteurs* $g_1, ..., g_n$ *est appelée la* matrice de dilatation.

Le *lattice standard*, noté $\mathbb{Z}^n$, contient tous les points à coordonnées entières de l'espace.

Tout lattice $\mathcal{L}^n$ de matrice de dilatation G est constitué des points Gx avec x un point du lattice standard, $x \in \mathbb{Z}^n$.

$$\mathcal{L}^n = G \cdot \mathbb{Z}^n \tag{1.22}$$

Gruber et Lekkerkerker, dans [GL87], démontrent que tout lattice $\mathcal{L}^d$ est obtenu en appliquant la transformation inversible par la matrice G au lattice standard[1]. Dans la suite de ce chapitre, nous ne considérons que le lattice standard $\mathbb{Z}^d$. Le cas général d'un lattice quelconque est obtenu par application d'une transformation G.

Définition 30 (lattice-polytope) *On appelle* lattice-polytope *l'intersection du lattice* $\mathbb{Z}^n$ *et d'un* n*-polytope paramétré* $\mathcal{D}(p)$*, avec* p *un vecteur de paramètres* entiers.

Définition 31 (polytope entier) *On dit qu'un polytope paramétré est* entier *si tous ses sommets sont à coordonnées entières, pour toute valeur entière des paramètres.*

Définition 32 (polytope rationnel) *On dit qu'un polytope paramétré est* rationnel *si l'un de ses sommets au moins est à coordonnées rationnelles pour toute valeur entière des paramètres.*

Définition 33 (dénominateur d'un polytope) *On appelle* dénominateur *d'un polytope paramétré le plus petit commun multiple des dénominateurs des coordonnées de ses sommets pour toute valeur entière des paramètres.*

1. la matrice G est inversible car les colonnes de G sont linéairement indépendantes.

Le dénominateur d'un polytope entier est 1.

Définition 34 (polyèdre homothétique-bordé) *On appelle* polyèdre homothétique-bordé *un polyèdre paramétré dont les sommets sont des fonctions affines (strictes) des paramètres.*

Propriété 10 *Tout polyèdre paramétré est homothétique-bordé par domaine de validité (calculé au paragraphe 1.4).*

1.5.2 Pseudo-polynômes

Définition 35 (nombre périodique unidimensionnel) *Soit une suite* (u_i)*, définie pour tout entier naturel* i*, de période* q*. On appelle* nombre périodique unidimensionnel *la fonction* $u(i) = u_{i \ mod \ q}$*, notée :*

$$u_i = [u_0, u_1, ..., u_{q-1}]_i \tag{1.23}$$

Définition 36 (nombre périodique multidimensionnel) *Soit* I *un vecteur entier de dimension* γ*,* $I = (I_j)_{j\in[1,\gamma]}$*. Un* nombre γ-périodique $u(I)$ *est défini par un tableau* u *de* γ *dimensions et de taille* $q_1 \times q_2 \times ... \times q_\gamma$ *tel que :*

$$u(I) = u_{N_1,N_2,...,N_\gamma}, \quad avec\ N_j = I_j\ mod\ q_j, \forall j \in [1, \gamma] \tag{1.24}$$

Le vecteur $q = (q_1, ..., q_\gamma])$ *est appelé la* multi-période *de* $u(I)$*. Le plus petit commun multiple des coefficients de ce vecteur est appelé la* période *de* $u(I)$*.*

Exemples

- $(-1)^i = [1, -1]_i$
- $(-1)^{i+j} = \left[\begin{array}{rr} 1 & -1 \\ -1 & 1 \end{array}\right]_{(i,j)} = [[1, -1]_j, [-1, 1]_j]_i$

La notation matricielle ci-dessus peut être utilisée pour représenter les nombres 2-périodiques uniquement. La notation sous forme de vecteur de vecteurs est utilisable dans tous les cas (nombres périodiques de dimension quelconque).

Définition 37 (pseudo-polynôme) *Soit* N *un vecteur entier de dimension* γ*,* $N = (N_1, ..., N_\gamma)$*. Un pseudo-polynôme de degré* k *est défini par :*

$$f(N) = \sum_{i_1=0}^{k} \sum_{i_2=0}^{k} ... \sum_{i_\gamma=0}^{k} c_{i_1,...,i_\gamma} N_1^{i_1} N_2^{i_2} ... N_\gamma^{i_\gamma} \tag{1.25}$$

où chaque $c_{i_1,...,i_\gamma}$ *est un nombre périodique.*

On appelle période *du pseudo-polynôme le plus petit commun multiple des périodes des* $c_{i_1,...,i_\gamma}$.

Exemples

- Pour tout entier $i \geq 2$:

$$\sum_{1}^{\frac{i}{2}} 1 = \frac{1}{2}i + \left[0, -\frac{1}{2}\right]_i$$

- Pour tout couple d'entiers i et j positifs :

$$\begin{aligned}\sum_{0}^{i/2}\sum_{0}^{j/2} 1 &= \left(\frac{1}{4}i + \left[\frac{1}{2}, \frac{1}{4}\right]_i\right) j + \left[\left(\frac{1}{2}i + \left[1, \frac{1}{2}\right]_i\right), \left(\frac{1}{4}i + \left[\frac{1}{2}, \frac{1}{4}\right]_i\right)\right]_j \\ &= \frac{1}{4}ji + \left[\frac{1}{2}, \frac{1}{4}\right]_i j + \left[\frac{1}{2}, \frac{1}{4}\right]_j i + \left[\begin{array}{cc} 1 & 1/2 \\ 1/2 & 1/4 \end{array}\right]_{j,i}\end{aligned}$$

1.5.3 Polynômes d'Ehrhart

La preuve du théorème ci-dessous est donnée dans le cas d'un seul paramètre entier par Ehrhart dans [Ehr62], et dans le cas général (m paramètres entiers) par Clauss dans [CL96].

Théorème 10 (Théorème d'Ehrhart-Clauss)
Soit $\mathcal{P}(p)$ un polytope paramétré homothétique-bordé de dimension d, ayant m paramètres $p = (p_1, ..., p_m)$.

Le nombre de points entiers contenus dans $\mathcal{P}(p)$, appelé polynôme d'Ehrhart, *est un pseudo-polynôme en p de degré d et dont la période est le dénominateur de $\mathcal{P}(p)$. Tous les termes dont la somme des degrés des variables est supérieure à d sont nuls.*

La forme générale du polynôme d'Ehrhart de $\mathcal{P}(p)$ est donc :

$$E(p) = \sum_{i_1=0}^{d} \sum_{i_2=0}^{d-i_1} \ldots \sum_{i_m=0}^{d-(i_1+\ldots+i_{m-1})} c_{i_1,i_2,\ldots,i_m}\; p_1^{i_1} p_2^{i_2} \ldots p_m^{i_m} \tag{1.26}$$

avec $c_{i_1,i_2,\ldots,i_m}$ des nombres m-périodiques au plus, et dont la période est inférieure ou égale au dénominateur de $\mathcal{P}(p)$.

Le nombre de points entiers contenus dans un polyèdre paramétré quelconque est donné par sous-domaine des paramètres, qui sont les domaines de validité des sommets du polyèdre paramétré, calculés au paragraphe 1.4. Un polynôme d'Ehrhart est calculé pour chacun des domaines de validité.

Connaissant la forme générale du polynôme d'Ehrhart dans chaque domaine de validité, il suffit de résoudre un système d'égalités linéaires déduit d'un certain nombre de valeurs initiales du polynôme. Ces valeurs initiales sont obtenues en comptant le nombre de points entiers dans les polyèdres non-paramétrés correspondant à des instanciations des paramètres dans le domaine de validité concerné.

1.5.4 Exemples

Exemple 1 (suite)

Calcul du polynôme d'Ehrhart dans chacun des deux domaines de validité du polytope paramétré $\mathcal{D}_1(p)$:

- Val_1 : $0 \leq p \leq 10$. Les sommets sont : $\begin{pmatrix}0\\p\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}p\\0\end{pmatrix}$. Le polytope est entier. La forme générale du polynôme d'Ehrhart est : $E_{1,1}(p) = c_{1,1}p^2 + c_{2,1}p + c_{3,1}$.

  Avec trois valeurs initiales $E_{1,1}(0) = 1, E_{1,1}(1) = 3$, et $E_{1,1}(2) = 6$, on obtient un système d'égalités linéaires en c_1, c_2, c_3 qui permet de trouver finalement :

$$E_{1,1}(p) = \frac{1}{2}p^2 + \frac{3}{2}p + 1$$

- $Val_2 : p \geq 10$. Les sommets sont : $\begin{pmatrix}0\\10\end{pmatrix}, \begin{pmatrix}p-10\\10\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}p\\0\end{pmatrix}$. Le polytope est entier. La forme générale du polynôme d'Ehrhart est : $E_{1,2}(p) = c_{1,2}p^2 + c_{2,2}p + c_{3,2}$.

  Les valeurs initiales utilisées sont : $E_{1,2}(10) = 66, E_{1,2}(11) = 77$, et $E_{1,2}(12) = 88$. On obtient :

$$E_{1,2}(p) = 11p - 44$$

Finalement, l'expression du nombre de points entiers du polytope paramétré $\mathcal{D}_1(p)$ est :

$$Card(\mathcal{D}_1(p) \cap \mathbb{Z}^2) = \begin{cases} \frac{1}{2}p^2 + \frac{3}{2}p + 1 & \text{Si } 0 \leq p \leq 10 \\ 11p - 44 & \text{Si } p \geq 10 \\ 0 & \text{Sinon.} \end{cases}$$

Exemple 2 (suite)

Dans les trois domaines de validité du polytope paramétré $\mathcal{D}_2(p, q)$:

- Val_1 : $q/2 \leq p \leq 10, q \geq 0$. Les sommets sont : $\begin{pmatrix}q/2\end{pmatrix}, \begin{pmatrix}p\end{pmatrix}$. Le dénominateur du polytope est 2. Sa forme générale est : $E_{2,1}(p, q) = c_{1,1}q^2 + (c_{2,1}p + c_{3,1})q + (c_{4,1}p^2 + c_{5,1}p + c_{6,1})$ où chaque coefficient c est un nombre 2-périodique de multi-période $(2, 2)$.

  Finalement, avec suffisemment de valeurs initiales, et après simplification, on obtient :

$$E_{2,1}(p, q) = p - \frac{1}{2}q + \left[1, \frac{1}{2}\right]_q$$

– Val_2 : $10 \leq p \leq q/2 + 25, 0 \leq q \leq 20$. Les sommets sont : $\big(q/2\big)$, $\big(10\big)$. La forme générale du polynôme d'Ehrhart est la même que dans le cas Val_1. On obtient :

$$E_{2,2}(p,q) = -\frac{1}{2}q + \left[11, \frac{21}{2}\right]_q$$

– Val_3 : $q/2 + 25 \leq p \leq 35, q \geq 0$. Les sommets sont : $\big(10\big)$, $\big(p-25\big)$. Ce polytope est entier. La forme générale du polynôme d'Ehrhart est la même, mais les coefficients sont des rationnels. On obtient :

$$E_{2,3}(p,q) = -p + 36$$

L'expression du nombre de points entiers du polytope paramétré $\mathcal{D}_2(p,q)$ est :

$$Card(\mathcal{D}_2(p,q) \cap \mathbb{Z}^1) = \begin{cases} p - \frac{1}{2}q + \left[1, \frac{1}{2}\right]_q & \text{Si } \frac{q}{2} \leq p \leq 10 \text{ et } q \geq 0 \\ -\frac{1}{2}q + \left[11, \frac{21}{2}\right]_q & \text{Si } 10 \leq p \leq \frac{q}{2} + 25 \text{ et } 0 \leq q \leq 20 \\ -p + 36 & \text{Si } \frac{q}{2} + 25 \leq p \leq 35 \text{ et } q \geq 0 \\ 0 & \text{Sinon} \end{cases}$$

On voit sur ces deux exemples que l'expression symbolique du nombre de points entiers d'un polytope paramétré est très variable d'un domaine de validité à un autre. Ce polynôme peut être de degrés, de périodes, et même être fonction de paramètres différents dans chaque domaine de validité. Dans l'exemple 2, c'est un polynôme de degré 1 en p, q dans le premier domaine, en q dans le second, et en p dans le troisième.

La méthode qui a été présentée dans ce paragraphe a été implémentée dans la *PolyLib* par Doran Wilde et moi-même.

L'annexe B décrit en détail les nouvelles fonctions et structures que nous avons ajoutées à la *PolyLib*. Cette annexe est également le document de référence de ce programme, et est pour cette raison écrite en anglais.

Chapitre 2

Parallélisation de nids de boucles paramétrés

Les algorithmes décrits dans le chapitre précédent, permettant de calculer les sommets et le nombre de points entiers de polyèdres paramétrés, ont de nombreuses applications dans le domaine de la parallélisation automatique. Lorsque les problèmes traités dépendent de paramètres, dans le cas des systèmes d'équations récurrentes affines paramétrés (PARE), et même quand il n'y a pas de paramètres (pour le calcul du parallélisme potentiel par exemple), ces valeurs symboliques d'un polyèdre paramétré sont d'une grande utilité.

Ce chapitre présente quelques unes de ces applications, correspondant à l'efficacité d'une solution parallèle d'un PARE :

– le calcul de la latence d'un programme parallèle,

– le calcul du parallélisme potentiel,

– le calcul du nombre de communications.

Il y a bien d'autres applications de ces résultats, tant dans le domaine de la parallélisation de PARE, que dans l'analyse et la transformation de nids de boucles, la mise en assignation unique d'un nid de boucles, ou l'optimisation de la mémoire [Cla96, CL96, Cla97]. Ce chapitre présente quelques applications. Une partie de ce travail a été mené en étroite collaboration avec Philippe Clauss et Doran Wilde, et a donné lieu a une publication présentée en annexe C comme complément à ce chapitre.

Transformation affine d'un polyèdre paramétré

A plusieurs reprises dans ce chapitre, nous allons faire une transformation affine avec paramètres d'un polyèdre paramétré rationnel $\mathcal{D}(p)$, de la forme :

$$d(p) = T\,\mathcal{D}(p) + T'p + t$$

où T est une matrice $k \times n$, T' une matrice $k \times m$, et t un k-vecteur, le polyèdre $\mathcal{D}(p)$ est de dimension n, le paramètre p est un vecteur de dimension m, et le polyèdre $d(p)$ est de dimension k.

Soit $\mathcal{D}'$ le polyèdre combiné associé à $\mathcal{D}(p)$, et d' le polyèdre combiné associé à $d(p)$. Dans ce cas, on a :

$$d' = \left(\begin{array}{c|c} T & T' \\ \hline 0 & Id \end{array}\right) D' + t$$

Cette extension des fonctions affines à l'espace combiné est utilisée afin de calculer facilement l'image d'un polyèdre paramétré, à l'aide de la bibliothèque PolyLib.

2.1 Latence

Nous mesurons la latence d'un programme parallèle par le nombre de périodes d'activité des processeurs virtuels : chaque période d'activité correspond à l'ensemble des instructions exécutées à chaque itération du nid de boucles. La latence est le nombre d'exécution de ces groupes d'instructions.

Les instants d'exécution de chaque point de calcul des variables d'un PARE sont donnés par l'application de leurs fonctions de cadencement respectives aux domaines de définition correspondants. La latence est la différence entre les instants d'exécution minimal et maximal de l'ensemble de ces variables.

Remarquons que ce ne sont pas forcément le sommet le plus proche et le sommet le plus éloigné de l'origine du repère, dans chaque domaine de définition, qui correspondent aux instants d'exécution mimimun et maximum dans ces domaines. Ceci n'est vrai que si le vecteur de cadencement est positif, et si les points du domaine de définition sont à coordonnées positives, lorsque le cadencement est monodimensionnel.

Une méthode exacte de recherche de la latence d'un programme consiste à compter les points entiers contenus dans l'enveloppe convexe de l'union des images des domaines de définition par chaque fonction de cadencement. Cette méthode est également valide dans le cas multidimensionnel. Elle donne le nombre exact de pas des boucles sur le temps.

L'image d'un domaine de définition paramétré $\mathcal{D}(p)$ est calculée grâce à l'image du polyèdre combiné associé $\mathcal{D}'$ par la fonction de cadencement étendue à l'espace paramétré.

L'algorithme décrit ci-dessous utilise les fonctions suivantes :

- Image ($\mathcal{P}$, t) : renvoie l'image du polyèdre paramétré $\mathcal{P}$ par la fonction de cadencement t,
- ConvexUnion ($\mathcal{P}_1$, $\mathcal{P}_2$) : calcule l'enveloppe convexe de l'union des polyèdres $\mathcal{P}_1$ et $\mathcal{P}_2$,
- EP_Enumerate ($\mathcal{P}$) : calcule les polynômes d'Ehrhart par domaines de validité du polyèdre paramétré $\mathcal{P}$.

Algorithme de calcul de la latence

Input: PARE,
timing functions.
Output: Ehrhart polynomials over validity domains.

```
function Latency ( X : set of variables of the PARE,
                   D_{X_p} : set of parameterized definition domains,
                   t_X : set of timing functions
                 )
begin
  Polyhedron U_p;      /* temporary convex union of parameterized polyhedra */

  U_p = ∅
  forall A ∈ X do
    U_p = ConvexUnion ( U_p, Image ( D_{A_p}, t_A ) )
  done
  return ( EP_Enumerate ( U_p ) )
end
```

Exemple

Reprenons l'exemple 1 du chapitre précédent. Soit un PARE à une variable A définie sur le domaine

$$\mathcal{D}_A(p) = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid 0 \leq i \leq p - j, 0 \leq j \leq 10 \right\}$$

Le polyèdre combiné associé est $\mathcal{D}'_A$. Il contient deux sommets $v_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ et $v_2 = \begin{pmatrix} 0 \\ 10 \\ 10 \end{pmatrix}$, et deux rayons $r_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ et $r_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$.

Soit la fonction de cadencement monodimensionnel :

$$t_A\left(\begin{pmatrix} i \\ j \end{pmatrix}\right) = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

Les surfaces de temps correspondantes sont représentées figure 2.1 pour $p = 15$.

La fonction de cadencement étendue à l'espace paramétré transforme les indices comme indiqué par la fonction de temps, et conserve les paramètres :

$$t'_A\left(\begin{pmatrix} i \\ j \\ p \end{pmatrix}\right) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ p \end{pmatrix}$$

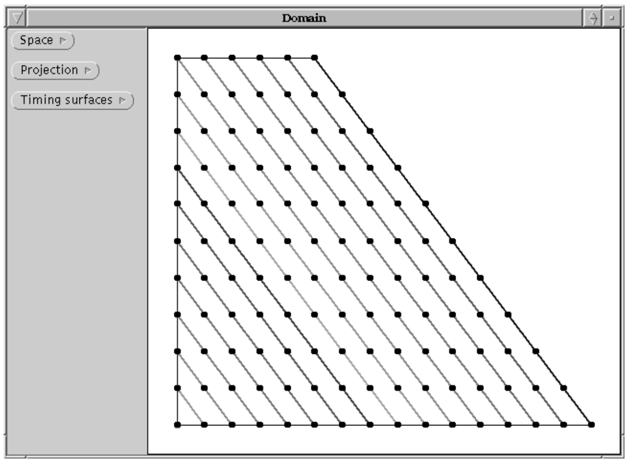


FIG. 2.1 - *Surfaces de temps pour* $\lambda = (1\ 1)$.

L'image du polyèdre $\mathcal{D}'_A$ par la fonction t'_A est calculée grâce à la bibliothèque PolyLib. $t'_A(\mathcal{D}'_A)$ est le polyèdre combiné correspondant à l'image du polyèdre paramétré $\mathcal{D}_A(p)$ par la fonction de temps. Il est composé du sommet $t'_A(v_1) = \begin{pmatrix}0\\0\end{pmatrix}$, et des rayons $t'_A(r_1) = \begin{pmatrix}1\\1\end{pmatrix}$ et $t'_A(r_2) = \begin{pmatrix}0\\1\end{pmatrix}$ (l'image du deuxième sommet v_2 est contenue dans ce polyèdre, et n'est donc pas reportée) :

$$t'_A(\mathcal{D}'_A) = \left\{ \begin{pmatrix}t\\p\end{pmatrix} \in \mathbb{Q}^2 \mid p \geq 0, 0 \leq t \leq p \right\}$$

Ce polyèdre combiné correspond à un polyèdre paramétré $d(p)$. Le polynôme d'Ehrhart associé à $d(p)$, caclulé par la méthode présentée au chapitre précédent, est :

$$E(d(p)) = p + 1 \qquad \text{pour } p \geq 0$$

La latence correspondant à ce cadencement est donc :

$$\boxed{l(p) = p + 1 \qquad \forall p \geq 0}$$

Exemple (variante)

Prenons une autre fonction de cadencement :

$$t_A\left(\begin{pmatrix}i\\j\end{pmatrix}\right) = \begin{pmatrix}1 & -1\end{pmatrix}\begin{pmatrix}i\\j\end{pmatrix}$$

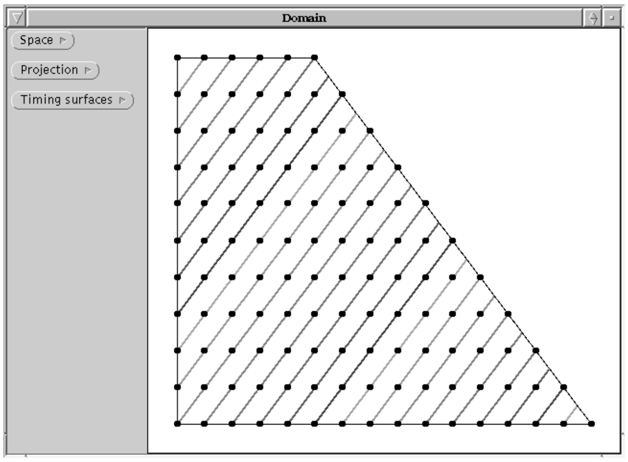


FIG. 2.2 - *Surfaces de temps pour* $\lambda = (1 \ \ -1)$.

Les surfaces de temps correspondantes sont représentées figure 2.2 pour $p = 15$.

La fonction de cadencement étendue est :

$$t'_A\left(\begin{pmatrix} i \\ j \\ p \end{pmatrix}\right) = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ p \end{pmatrix}$$

$t'_A(\mathcal{D}'_A)$ est le polyèdre combiné composé des sommets $t'_A(v_1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $t'_A(v_2) = \begin{pmatrix} -10 \\ 10 \end{pmatrix}$, et des rayons $t'_A(r_1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ et $t'_A(r_2) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

$$t'_A(\mathcal{D}'_A) = \left\{ \begin{pmatrix} t \\ p \end{pmatrix} \in \mathbb{Q}^2 \mid -p \leq t \leq p, t \geq -10 \right\}$$

La latence est le nombre de points entiers du polyèdre paramétré associé à ce polyèdre combiné :

Si $0 \leq p \leq 10$,	$l(p) = 2p + 1$
Si $p \geq 10$,	$l(p) = p + 11$

2.2 Parallélisme potentiel

Soit un PARE et un ensemble de fonctions de cadencement et d'allocation associées à chacune de ses variables. Soient $\mathcal{A}_X(p, t_0)$ les polyèdres contenant l'ensemble des processeurs calculant la variable X du PARE à l'instant t_0 :

$$\mathcal{A}_X(p, t_0) = \text{alloc}_X(\{z \mid z \in \mathcal{D}_X(p), t_X(z, p) = t_0\})$$

Chaque polyèdre $\mathcal{A}_X(p, t_0)$ est l'image par la fonction d'allocation alloc_X de la surface de temps correspondant à l'instant t_0 pour la variable X.

L'union des polyèdres $\mathcal{A}_X(p, t_0)$ pour toute variable X du PARE, à un instant t_0 donné, représente l'ensemble des processeurs virtuels actifs à l'instant t_0, et est appelé le *cône d'activité*. Le nombre de points entiers du cône d'activité à l'instant t_0 est noté $a(p, t_0)$. Le nombre maximal de points entiers du cône d'activité à tout instant t_0 est appelé le *parallélisme potentiel* :

$$\text{pp}(p) = \underset{t_0}{\text{Max}}(a(p, t_0))$$

Il s'agit du nombre maximal de processeurs actifs au même instant. Notons cependant que les processeurs contenus dans le cône d'activité peuvent être actifs cycliquement, si la matrice d'allocation d'une variable du PARE n'est pas unimodulaire.

Le calcul de $a(p, t_0)$ est effectué de la manière suivante. L'image du polyèdre $\mathcal{A}_X(p, t_0)$ par la fonction d'allocation alloc_X se fait de la même manière que dans le calcul de la latence dans le paragraphe précédent, en étendant la fonction d'allocation à l'espace combiné, et en calculant ainsi l'image du polyèdre combiné $\mathcal{A}'_X$. Ces polyèdres $\mathcal{A}_X(p, t_0)$ sont des polyèdres paramétrés à la fois par p et par t_0. L'espace combiné contient donc le vecteur d'indice z et les vecteurs p et t_0. Le résultat est le polyèdre combiné associé au polyèdre paramétré $\text{alloc}_X(\mathcal{A}_X(p, t_0))$.

Ces polyèdres paramétrés, un pour chaque variable du PARE, ne sont en général pas disjoints. Il faut trouver une partition disjointe de l'espace combiné à partir des polyèdres combinés associés, afin d'éviter de compter plusieurs fois l'activité du même processeur. Ceci se fait par intersections et différences successives des polyèdres combinés. On obtient ainsi un ensemble de polyèdres paramétrés disjoints. Finalement, l'expression de $a(p, t_0)$ est obtenue en faisant la somme des polynômes d'Ehrhart associés à chacun de ces polyèdres. On cherche ensuite le maximum sur t_0 de ces valeurs $a(p, t_0)$ pour obtenir le parallélisme potentiel en fonction de p.

Exemple

Ce problème se simplifie évidemment lorsqu'il n'y a qu'une seule variable dans le PARE. Traitons dans cet exemple le problème suivant : soit X la seule variable intermédiaire d'un PARE, définie sur le domaine :

$$\mathcal{D}_X = \{(i, j, k) \mid 0 \le i \le p,\ 0 \le j \le i,\ 0 \le k \le i - j\}$$

Soient la fonction de cadencement $t_X(i, j, k, p) = i + j + k$ et la fonction d'allocation $\text{alloc}_X(i, j, k) = (i, j)$.

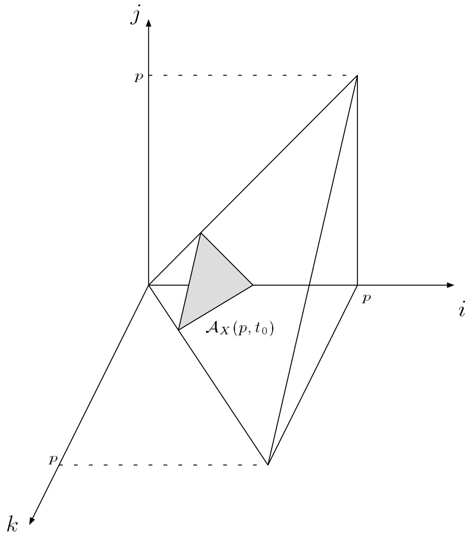


FIG. 2.3 - $\mathcal{A}_X(p,t_0)$, *premier domaine de validité*

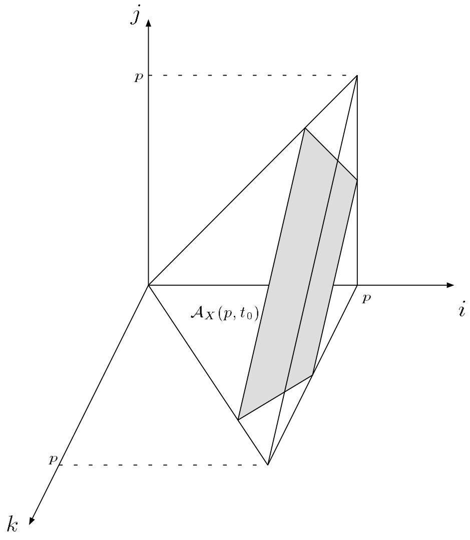


FIG. 2.4 - *Deuxième domaine de validité*

Le polyèdre paramétré $\mathcal{A}_X(p,t_0)$ est représenté figures 2.3 et 2.4. Ce polyèdre paramétré est défini par l'équation :

$$\mathcal{A}_X(p,t_0) = \{(i,j,k) \mid 0 \le i \le p,\ 0 \le j \le i,\ 0 \le k \le i-j,\ i+j+k = t_0\}$$

Le polyèdre combiné associé à $\mathcal{A}'_X$ a quatre rayons $r_1 = \begin{pmatrix}1\\0\\1\\1\\2\end{pmatrix}$, $r_2 = \begin{pmatrix}1\\1\\0\\1\\2\end{pmatrix}$, $r_3 = \begin{pmatrix}0\\0\\0\\1\\0\end{pmatrix}$, $r_4 = \begin{pmatrix}1\\0\\0\\1\\1\end{pmatrix}$ et un sommet $v_1 = \begin{pmatrix}0\\0\\0\\0\\0\end{pmatrix}$.

La fonction d'allocation étendue à l'espace combiné est :

$$\mathrm{alloc}'_X\left(\begin{pmatrix} i \\ j \\ k \\ p \\ t_0 \end{pmatrix}\right) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ k \\ p \\ t_0 \end{pmatrix}$$

L'image du polyèdre combiné $\mathcal{A}'_X$ par la fonction d'allocation étendue alloc'_X est composé des quatre rayons et du sommet : $\mathrm{alloc}'_X(r_1) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 2 \end{pmatrix}$, $\mathrm{alloc}'_X(r_2) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix}$, $\mathrm{alloc}'_X(r_3) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$, $\mathrm{alloc}'_X(r_4) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$, et $\mathrm{alloc}'_X(v_1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$. Voici la représentation implicite de ce polyèdre :

$$\mathrm{alloc}'_X(\mathcal{A}'_X) = \{(a_1, a_2, p, t_0) \mid a_1 + a_2 \leq t_0,\ 2a_1 \geq t_0,\ a_1 \leq n\}$$

Ce polyèdre est la représentation combinée du polyèdre paramétré $\mathrm{alloc}_X(\mathcal{A}_X(p, t_0))$. Le parallélisme potentiel est le maximum du nombre de points entiers de ce polyèdre paramétré. Les polynômes d'Ehrhart associés à chaque domaine de validité de ce polyèdre sont :

– si $0 \leq t_0 \leq p$:

$$E(\mathrm{alloc}_X(\mathcal{A}_X(p, t_0))) = \frac{1}{8}{t_0}^2 + \left[\frac{3}{4}, \frac{1}{2}\right]_{t_0} t_0 + \left[1, \frac{3}{8}\right]_{t_0}$$

– si $p \leq t_0 \leq 2p$:

$$E(\mathrm{alloc}_X(\mathcal{A}_X(p, t_0))) = -\frac{1}{2}p^2 + pt_0 - \frac{3}{8}{t_0}^2 + \frac{1}{2}p + \left[\frac{1}{4}, 0\right]_{t_0} t_0 + \left[1, \frac{3}{8}\right]_{t_0}$$

Puisque ce PARE contient une variable unique, il n'y a pas de problème de recouvrement des activités des processeurs par plusieurs variables. Il suffit de calculer le maximum sur t_0 de ces deux polynômes pour obtenir le parallélisme potentiel :

– si $0 \leq t_0 \leq p$:

$$\begin{aligned}
&\underset{t_0\in[0,p]}{\mathrm{Max}}\left(\frac{1}{8}{t_0}^2+\left[\frac{3}{4},\frac{1}{2}\right]_{t_0}t_0+\left[1,\frac{3}{8}\right]_{t_0}\right)\\
=\ &\mathrm{Max}\left(\underset{\substack{t_0\in[0,p]\\ t_0 \text{ pair}}}{\mathrm{Max}}\left(\frac{1}{8}{t_0}^2+\frac{3}{4}t_0+1\right),\ \underset{\substack{t_0\in[0,p]\\ t_0 \text{ impair}}}{\mathrm{Max}}\left(\frac{1}{8}{t_0}^2+\frac{1}{2}t_0+\frac{3}{8}\right)\right)\\
=\ &\frac{1}{8}p^2+\left[\frac{3}{4},\frac{1}{2}\right]_p p+\left[1,\frac{3}{8}\right]_p
\end{aligned}$$

– si $p \leq t_0 \leq 2p$:

$$\begin{aligned}
&\underset{t_0\in[p,2p]}{\mathrm{Max}}\left(-\frac{1}{2}p^2+pt_0-\frac{3}{8}{t_0}^2+\frac{1}{2}p+\left[\frac{1}{4},0\right]_{t_0}t_0+\left[1,\frac{3}{8}\right]_{t_0}\right)\\
&=\mathrm{Max}\left(\underset{\substack{t_0\in[p,2p]\\ t_0 \text{ pair}}}{\mathrm{Max}}\left(-\frac{1}{2}p^2+pt_0-\frac{3}{8}{t_0}^2+\frac{1}{2}p+\frac{1}{4}t_0+1\right),\right.\\
&\qquad\left.\underset{\substack{t_0\in[p,2p]\\ t_0 \text{ impair}}}{\mathrm{Max}}\left(-\frac{1}{2}p^2+pt_0-\frac{3}{8}{t_0}^2+\frac{1}{2}p+\frac{3}{8}\right)\right)\\
&=\mathrm{Max}\left(\underset{\substack{t_0\in[p,2p]\\ t_0 \text{pair}}}{\mathrm{Max}}\left(a_1(p,t_0)\right),\ \underset{\substack{t_0\in[p,2p]\\ t_0 \text{ impair}}}{\mathrm{Max}}\left(a_2(p,t_0)\right)\right)
\end{aligned}$$

Pour trouver cette dernière valeur, une étude de cas est nécessaire. Les dérivées en t_0 des deux fonctions $a_1(p,t_0)$ $a_2(p,t_0)$ s'annulent en $t_0 = \frac{4p+1}{3}$ et en $t_0 = \frac{4p}{3}$ respectivement. C'est pour cette raison que les différents cas correspondent aux valeurs possibles de p modulo 3 :

– si $p \; mod \; 3 = 0$, le maximum est atteint pour $t_0 = \frac{4p}{3}$, et il vaut : $\frac{1}{6}p^2 + \frac{5}{6}p + 1$,

- si $p \; mod \; 3 = 1$, le maximum est atteint pour $t_0 = \frac{4p+2}{3}$, il vaut : $\frac{1}{6}p^2 + \frac{5}{6}p + 1$,
- si $p \; mod \; 3 = 2$, le maximum est atteint pour $t_0 = \frac{4p-2}{3}$, il vaut : $\frac{1}{6}p^2 + \frac{5}{6}p + \frac{2}{3}$.

La solution dans le deuxième cas est donc : $\frac{1}{6}p^2 + \frac{5}{6}p + \left[1, 1, \frac{2}{3}\right]_p$.

Finalement, le parallélisme potentiel est le maximum de ces deux valeurs obtenues pour $0 \leq t_0 \leq p$ et pour $p \leq t_0 \leq 2p$:

$$\mathrm{pp}(p) = \mathrm{Max}\left(\left(\frac{1}{8}p^2 + \left[\frac{3}{4}, \frac{1}{2}\right]_p p + \left[1, \frac{3}{8}\right]_p\right), \left(\frac{1}{6}p^2 + \frac{5}{6}p + \left[1, 1, \frac{2}{3}\right]_p\right)\right)$$

$$\boxed{\mathrm{pp}(p) = \frac{1}{6}p^2 + \frac{5}{6}p + \left[1, 1, \frac{2}{3}\right]_p}$$

2.3 Nombre de communications

Dans le chapitre 2 de la première partie, nous estimons le nombre de communications dans un réseau de processeurs virtuels par leur volume. Le volume de communications est la dimension de l'ensemble de processeurs qui transmettent une donnée plus la dimension du temps (*i.e.* le nombre de boucles sur le temps) durant lequel ils transmettent cette donnée. Si la meilleure solution est unique, c'est la solution optimale pour des valeurs de paramètres assez grandes. Lorsque plusieurs solutions équivalentes en volume de communications existent, il est utile de connaître une approximation plus précise du nombre de communications.

En effet, si la complexité en dimension est équivalente, le gain peut cependant être relativement important, en choisissant une solution comportant de l'ordre de p^3 communications, plutôt qu'une solution avec $8p^3$ communications. La charge du réseau entre processeurs sera moindre, et la vitesse globale des communications sera donc meilleure. La complexité de cette analyse est cependant assez grande, et il vaut mieux l'éviter si possible, afin de ne pas allonger excessivement le temps de compilation de gros programmes.

Le nombre de communications dépend du nombre de points de l'ensemble d'émission et du nombre de points des ensembles d'utilisation. Considérons une équation de la forme :

$$X[z] = \ldots\ Y[g(z,p)]\ \ldots \qquad z \in \mathcal{D}_E(p) \tag{E}$$

Les points $z \in \mathcal{D}_E(p)$ constituent les points de tous les ensembles d'utilisation associés à cette dépendance. Le nombre exact de points appartenant aux ensembles d'utilisation est donc le nombre de points du polyèdre paramétré $\mathcal{D}_E(p)$.

L'ensemble d'émission correspondant à cette dépendance est :

$$\mathrm{Emit}_{E,Y} = \{g(z,p) \mid z \in \mathcal{D}_E(p)\}$$

C'est l'image par la fonction d'index g du polyèdre $\mathcal{D}_E(p)$. La valeur exacte du nombre de points de l'image par une fonction affine des points entiers d'un polyèdre paramétré est plus délicate à calculer. Elle n'est pas égale au nombre de points entiers de l'image du polyèdre paramétré (rationnel).

Clauss [Cla97] propose une méthode de recherche de ce nombre exact. Cette méthode est cependant de complexité plus grande que la méthode présentée ici : elle est multipliée par un facteur c, de l'ordre de la période du polynôme d'Ehrhart pour chaque domaine de validité des paramètres, correspondant au polyèdre avant transformation (si le polynôme est multi-périodique, il faut multiplier les périodes entre elles).

Les coefficients du polynôme correspondant au nombre de points entiers de l'image du polyèdre paramétré sont du même ordre de grandeur que les coefficients du polynôme exact. Nous utilisons cette valeur afin d'estimer de manière plus précise le nombre de processeurs d'émission. Cette valeur n'est cependant pas forcément la valeur exacte.

Conclusion

Contexte

Les travaux menés dans cette thèse se placent dans le contexte de la parallélisation automatique de nids de boucles. Les nids de boucles considérés sont à assignation unique, et dépendent de paramètres non instanciés au moment de la compilation. L'analyse de dépendance exacte est effectuée grâce aux concepts d'ensemble d'utilisation et d'ensemble d'émission, dépendant des paramètres. Dans le modèle géométrique que nous utilisons, les ensembles d'utilisation sont l'ensemble des points entiers contenus dans des polyèdres convexes paramétrés, et les ensembles d'émission sont les images de tels polyèdres par des fonctions affines lorsque les dépendances du nid de boucles sont affines. Les polyèdres paramétrés ont des formes géométriques variables, mais constantes par domaine de validité des paramètres. Afin de traiter les nids de boucles paramétrés, il est utile de connaître les sommets paramétrés de polyèdres paramétrés, définis par domaine de validité des paramètres.

D'autres outils abordent ce problème :

– PIP [FT89] permet de trouver les minimum et maximum lexicographiques entiers d'un polyèdre paramétré. Ces deux valeurs correspondent à deux sommets entiers du polyèdre paramétré. Le résultat de cet algorithme est une forme différente de celle présentée ici, par domaines de validité, mais le concept de QUAST (Quasi-Affine Syntax Tree) intégrant des tests sur les paramètres, rejoint l'idée de domaines de validité.

  La connaissance du minimum et du maximum lexicographiques est suffisante pour résoudre certains des problèmes concernant la parallélisation de nids de boucles, le calcul de la latence en fonction des paramètres par exemple. Par contre, dans d'autres problèmes, liés au comptage des points de polyèdres paramétrés, ou à la méthode d'optimisation des communications présentée au chapitre 2 par exemple, il est nécessaire de connaître *tous* les sommets de polyèdres paramétrés.

  L'avantage de ce logiciel réside dans le fait que PIP donne le minimum et le maximum *entiers* d'un polyèdre paramétré, alors que nos travaux sur les polyèdres paramétrés permettent pour l'instant de ne trouver que ses sommets *rationnels* en fonction des paramètres.

- L'Omega test [Pug92] est un outil permettant de manipuler de manière symbolique des formules de Presburger, qui sont des systèmes d'inégalités linéaires incluant les quantificateurs $\forall$ et $\exists$. Une méthode de test d'existence d'une solution et de simplification de formules de Presburger a été implémentée. Cependant, l'Omega test ne permet pas, en l'état actuel, de trouver de manière symbolique l'ensemble des sommets d'un polyèdre paramétré quelconque.

Principales contributions

L'objectif de ce travail n'était évidemment pas de créer le compilateur paralléliseur complet de demain, mais d'étudier des modèles théoriques qui seront peut-être intégrés un jour à ce compilateur. Ces modèles sont validés par une implémentation, prouvant leur faisabilité.

Les contributions théoriques de ce travail concernent deux domaines. D'une part, en parallélisation automatique de systèmes d'équations récurrentes affines paramétrés, une nouvelle méthode de recherche de cadencement et d'allocation a été proposée, dans le cadre des transformations par variable. L'optimisation des communications est le principal critère de sélection retenu, mais la compatibilité entre les fonctions d'allocation et les fonctions de temps est également prise en compte. Cette méthode utilise les résultats théoriques du deuxième thème de recherche de cette thèse : les polyèdres paramétrés.

Le travail sur les polyèdres paramétrés a été mené en étroite collaboration avec Doran Wilde. Nous avons élaboré une méthode de recherche de sommets de polyèdres paramétrés, par domaines de validité des paramètres. Philippe Clauss utilise ces résultats dans sa méthode de comptage du nombre de points entiers d'un polytope paramétré, reposant sur les travaux d'Eugène Ehrhart dans les années 50 et 60 sur le nombre de points entiers de polytopes à un seul paramètre. La connaissance des domaines de validité et des dénominateurs des sommets des polytopes à plusieurs paramètres empêchait Ehrhart de généraliser ses travaux.

Ces travaux théoriques ont été validés par des implémentations : la première, en parallélisation de systèmes d'équations récurrentes, par le logiciel OPÉRA. Si, à terme, ce logiciel se veut être entièrement automatique, il est, dans l'état actuel, un outil de visualisation fort utile pour l'utilisateur qui cherche à comprendre la géométrie des domaines de calculs et des dépendances dans un système d'équations récurrentes. OPÉRA est utilisé dans ce but par de nombreuses équipes de recherche, notamment à l'IRISA de Rennes, l'ENS de Lyon, l'ENSMP à Paris, en Allemagne par des équipes des universités de Paderborn et de Passau, aux Etats-Unis à l'université de New York à Stony Brook et à l'université Brigham Young à Provo, Utah, et évidemment à l'ICPS à Strasbourg.

Le travail concernant les polyèdres paramétrés est écrit sous forme d'une extension à la bibliothèque PolyLib. La première version de cette bibliothèque a été écrite à l'IRISA de Rennes et comprenait des fonctions de manipulation de vecteurs et de matrices, de polyèdres non paramétrés, et l'algorithme de la double représentation de Motzkin. Mon apport concerne le calcul et la représentation interne des sommets de polyèdres paramétrés et de leurs domaines de validité. Ces fonctions ont été optimisées et testées en collaboration avec Doran Wilde. Le calcul de polynômes d'Ehrhart a été écrit par Doran Wilde, et nous l'avons également optimisé et testé ensemble.

Perspectives

A court terme, diverses extensions de ces deux logiciels sont envisagées. Dans OPÉRA, les prochains développements seront axés sur la génération de code : la fin de l'intégration du générateur de code des communications de Christophe Reffay, et du générateur de code data-parallèle de parcours des boucles transformées d'Olivier Albiez sont à effectuer. Nous avons d'autre part le projet de générer du code parallèle effectif, spécifique à certaines architectures, destiné à être utilisé sur les machines parallèles SGI Origin 2000 et MasPar MP2 notamment.

La bibliothèque PolyLib sera également dans un très proche avenir améliorée de manière à résister efficacement aux débordements de nombres entiers. Un premier portage en entiers longs est actuellement effectué au CRI à l'ENSMP. Une extension plus générale, permettant de manipuler des entiers de taille quelconque, doit être réalisé. D'autre part, l'extension de la méthode d'Ehrhart-Clauss au comptage du nombre de points entiers d'*images affines* de polyèdres paramétrés est en prévision.

A moyen terme, différentes voies de recherche théorique sont ouvertes. Dans le domaine de la parallélisation de nids de boucles, il faudra raffiner l'algorithme de recherche de transformations, de manière à tenir compte d'un coût des communications plus précis, en tenant compte des caractéristiques de l'architecture cible. Il faut notamment permettre les diffusions sur les architectures implémentant ces communications de manière efficace, et permettre la diffusion par anticipation si le gain de performance le justifie : sur certaines architectures, il peut être intéressant d'envoyer les données calculées vers tous les processeurs les utilisant, avant qu'elles ne soient effectivement utilisées par les processeurs cibles. D'autre part, un travail théorique est encore nécessaire afin de générer du code data-parallèle lorsque les transformations utilisées sont non unimodulaires. L'algorithme écrit par Olivier Albiez permet de parcourir les superpositions des *points entiers* de polyèdres convexes. Lorsque les transformations ne sont pas unimodulaires, les images des domaines de calcul sont inclues dans des lattices non standard. La superpostition de ces ensembles de points n'est alors plus contenue dans un lattice, car l'union de deux lattices n'est pas toujours un lattice.

Le calcul de sommets *entiers* de polyèdres paramétrés sera le prochain thème de recherche dans le domaine des polyèdres paramétrés. Les sommets entiers d'un polyèdre paramétré sont des vecteurs périodiques, c'est-à-dire des vecteurs dont chaque coefficient est un nombre périodique. Cette idée intuitive, validée sur quelques exemples, nécessite une étude théorique sérieuse dans le cas général.

L'objectif à long terme est bien évidemment d'intégrer toutes ces méthodes au compilateur paralléliseur de demain. La plupart des paralléliseurs actuels sont basés sur l'application expérimentale d'une collection de transformations simples de nids de boucles. Des transformations de boucles plus efficaces peuvent être réalisées après l'analyse de dépendance exacte. Les exemples traités dans ce manuscrit en sont une illustration. Grâce aux nombreux travaux de recherche réalisés dans ce domaine, le compilateur paralléliseur idéal existera certainement un jour...

Annexe A

OPÉRA : utilisation et structures internes

Cette annexe décrit l'utilisation d'OPÉRA et les structures de données utilisées dans le logiciel. Les algorithmes mis en oeuvre sont présentées dans le chapitre 3 de la première partie de ce manuscrit.

A.1 Utilisation

La fenêtre principale (figure A.1) contient les boutons suivants :

- "File→Load Alpha" : permet de lire un nouveau PARE dans un fichier source Alpha,
- "File→Quit" : quitte OPÉRA,
- "Code generator II" : ouvre/ferme la fenêtre génération de code (version data-parallèle). Les matrices de transformations doivent être valides pour activer ce bouton,
- "Analyser" : ouvre/ferme la fenêtre d'analyse du PARE,
- "Trans. matrices" : ouvre/ferme la fenêtre de saisie des matrices de transformation,
- "Param. Domain" : permet de sélectionner un domaine de validité des paramètres,
- "SIMD/SPMD" : choisit le mode de génération de code : SIMD pour avoir les boucles externes égrenant le temps, ou SPMD pour les boucles externes parcourant les processeurs virtuels.

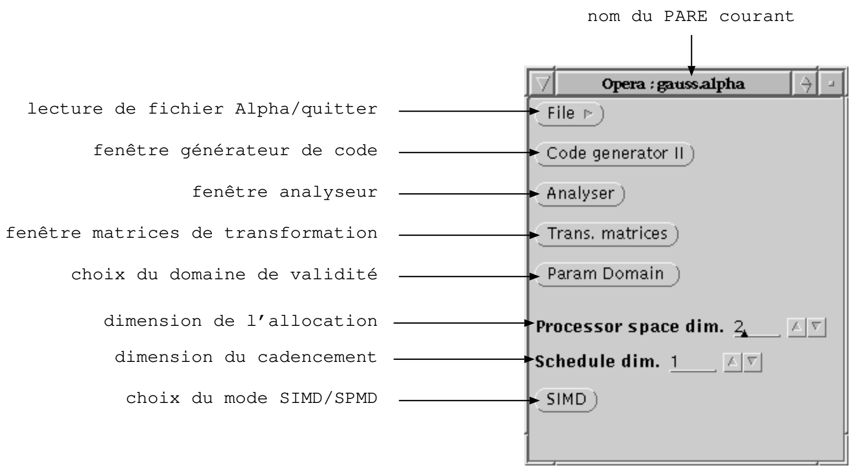


FIG. A.1 - *Fenêtre principale d'OPÉRA*

La fenêtre "Analyser" (figure A.2) s'active à partir de la fenêtre principale. Elle contient le PARE, et divers boutons :

– les boutons de la première ligne permettent de choisir le mode de visualisation des dépendances, dans la fenêtre représent les domaines d'itération :

  – "Hide/View DepExtr" : affiche les vecteurs de dépendance extrémaux de chaque dépendance,
  – "Hide/View DepFirst" : affiche les vecteurs reliant chaque point d'émission au(x) premier(s) point(s) actif(s) de l'ensemble d'utilisation correspondant, *i.e.* le(s) vecteur(s) de dépendance de composante temporelle minimale,
  – "Util Set : Fill/Vectors/None" : mode de visualisation des ensembles d'utilisation : remplissage, affichage des vecteurs d'utilisation, ou aucun affichage,
  – "Next Util Set" : sélectionne l'ensemble d'utilisation suivant (voir ci-dessous),
  – "All/One Util Set" : affiche un seul ensemble d'utilisation, ou tous les ensembles d'utilisation relatifs à la dépendance sélectionnée,
  – "Hide/View Emit Set" : affiche l'ensemble d'émission.

– les lignes suivantes contiennent les noms et les valeurs des paramètres. Les valeurs des paramètres sont modifiables dynamiquement. Lorsque tous les paramètres sont valides, la fenêtre "Domain" est activée.

– les lignes suivantes contiennent toutes les équations du PARE, et leurs domaines d'itération. Chaque bouton permet de visualiser les domaines de définition et les

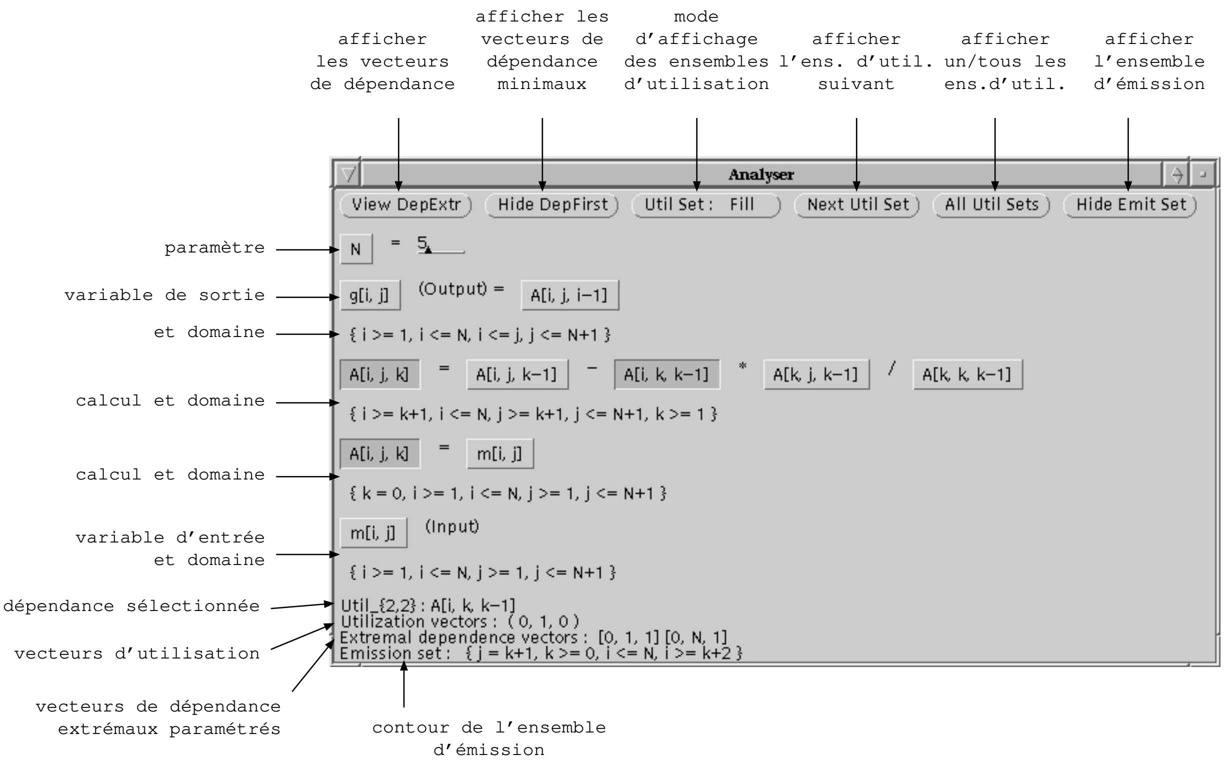


FIG. A.2 - *Fenêtre "Analyser"*

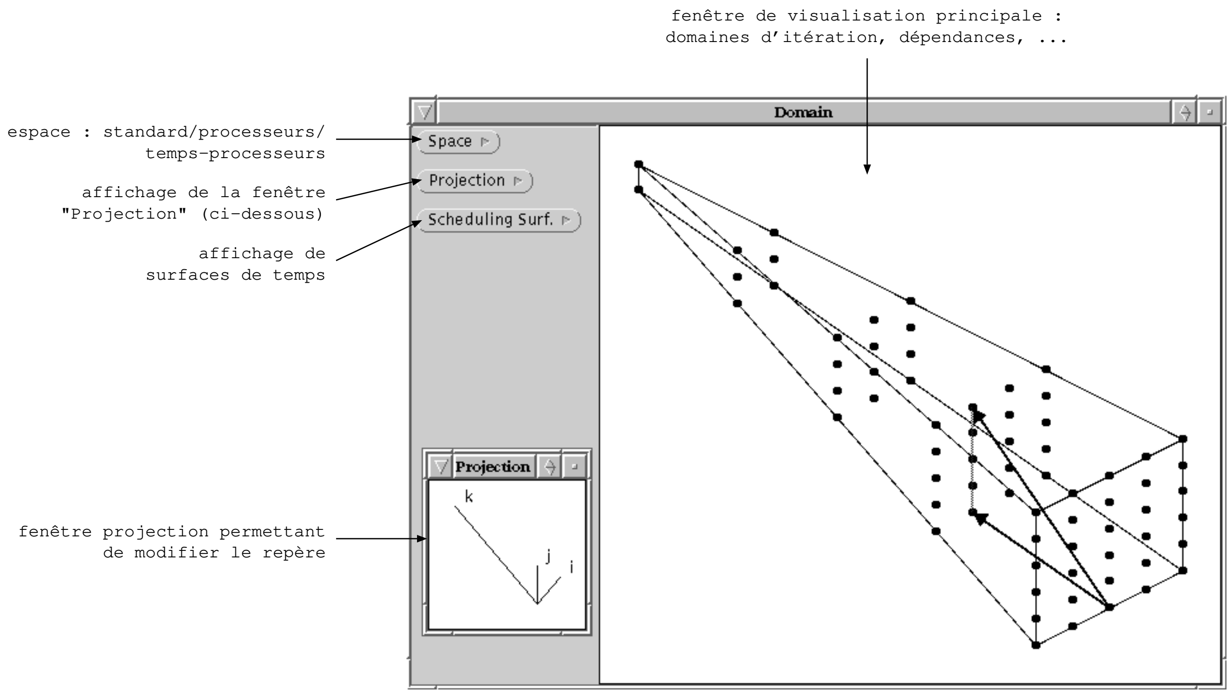


FIG. A.3 - *Fenêtre "Domain"*

dépendances dans le PARE. La dépendance sélectionnée est affichée suivant la configuration des boutons de la première ligne de cette fenêtre.

- les quatre dernières lignes contiennent des informations sur la dépendance sélectionnée :
  - la dépendance sélectionnée,
  - les vecteurs d'utilisation s'il y en a,
  - les vecteurs de dépendance extrémaux paramétrés, utilisés pour calculer les contraintes causales,
  - le polyèdre convexe contenant l'ensemble d'émission,

La fenêtre "Domain" est représentée figure A.3. Cette fenêtre devient active lorsque les paramètres sont tous validés. Elle contient les boutons suivants :

- "Space" : choix du repère :
  - "Standard" : repère $(i, j, k, ...)$,
  - "Processors" : repère $(a1, a2, ...)$ de l'espace des processeurs,

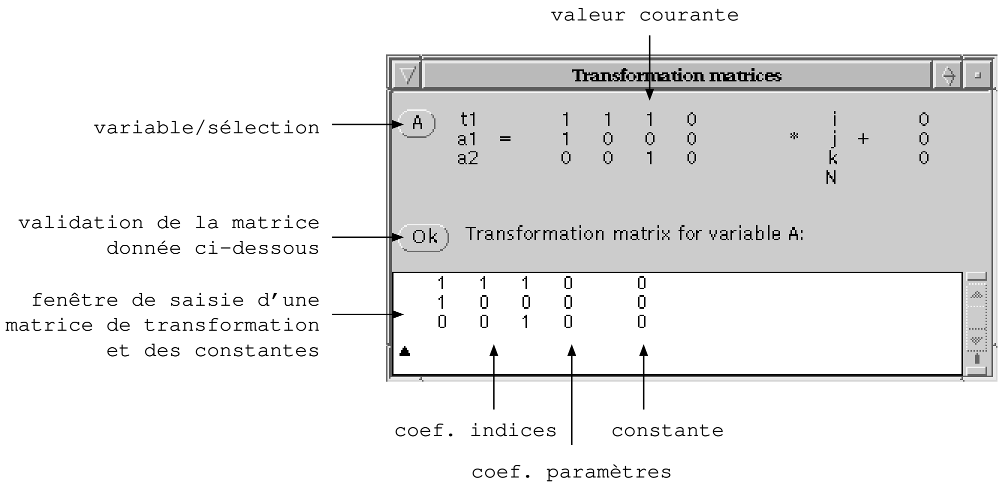


Fig. A.4 - *Fenêtre "Transformation matrices"*

  - "Time-Processors" : repère $(t1, ..., a1, a2, ...)$ du temps et des processeurs.

Ces deux derniers modes ne sont accessibles que lorsque les matrices de transformation nécessaires à l'affichage des variables et dépendances sélectionnées sont valides.

- "Projection" : "View" (affiche)/ "Standard" (réinitialise) la fenêtre "Projection". Cette fenêtre permet de changer l'angle de vue en cliquant sur les axes et en déplaçant la souris. Suivant le mode choisi, les axes sont représentés par $(i, j, k, ...)$, $(a1, a2, ...)$, ou $(t1, ..., a1, a2, ...)$.

- "Scheduling Surf." : affichage des surfaces de temps :

  - "Next step" : affiche la surface de temps suivante,
  - "Activation at step #" : sélectionne un instant précis et affiche la ou les surfaces de temps correspondantes (suivant les variables sélectionnées),
  - "None" : efface toutes les surfaces de temps,
  - "All" : affiche toutes les surfaces de temps.

La fenêtre "Transformation matrices" (figure A.4) permet de choisir une matrice de transformation pour chaque variable du PARE.

A.2 Structures de données

A.2.1 Polyèdres

Les structures permettant de représenter les polyèdres non paramétrés et les matrices sont celles de la PolyLib. On en trouvera une description détaillée dans la documentation de la PolyLib [Wil93].

Voici la structure utilisée pour représenter les matrices :

```
typedef struct matrix
{
  unsigned NbRows;                                /* nombre de lignes */
  unsigned NbColumns;                           /* nombre de colonnes */
  int **p;                          /* tableau de pointeurs permettant
                                   d'accéder aux éléments de la matrice */
}
Matrix;
```

On accède à l'élément `[i,j]` de la matrice déclarée par `Matrix *m`, de la manière suivante : `m->p[i][j]`, avec $0 \leq$ `i` $<$ `m->NbRows` et $0 \leq$ `j` $<$ `m->NbColumns`.

Voici la structure utilisée pour représenter les polyèdres :

```
typedef struct polyhedron
{
  struct polyhedron *next;
              /* ce pointeur est utilisé pour chaîner les polyèdres,
                       et ainsi représenter les unions de polyèdres */
  unsigned Dimension;                        /* dimension du polyèdre */
  unsigned NbConstraints;
               /* nombre de contraintes dans la matrice Constraint */
  unsigned NbRays;            /* nombre de rayons dans la matrice Ray */
  unsigned NbEq;                                 /* nombre d'égalités */
  unsigned NbBid;        /* nombre de rayons bidirectionnels (lignes) */
  int **Constraint;                         /* matrice de contraintes */
  int **Ray;                           /* matrice de rayons et sommets */
}
Polyhedron;
```

Soit un polyèdre déclaré par : `Polyhedron *P`. Les matrices `P->Constraint` et `P->Ray` ont (`P->Dimension`+2) colonnes, et respectivement `P->NbConstraints` et `P->NbRays` lignes. Dans la matrice `P->Constraint`, la première colonne indique pour chaque ligne si c'est une égalité (0) ou une inégalité (1). La dernière colonne est la constante. Dans la

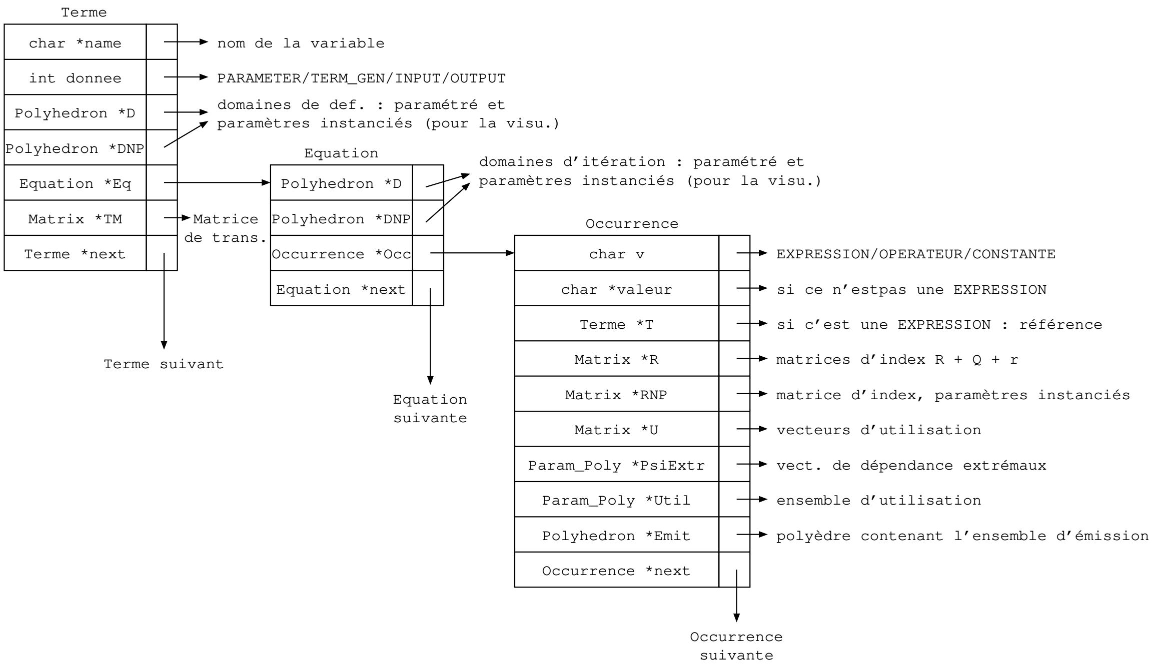


FIG. A.5 - *structure utilisée pour manipuler les PARE.*

matrice `P->Ray`, la première colonne indique si chaque ligne correspond à un rayon bidirectionnel (0) ou à un rayon unidirectionnel ou sommet (1). La dernière colonne distingue les rayons unidirectionnels (0) des sommets (dénominateur commun des coordonnées rationnelles du sommet, strictement supérieur à 0).

La structure utilisée pour décrire les sommets des polyèdres paramétrés est décrite dans l'annexe B.

A.2.2 Systèmes d'équations récurrentes

La structure utilisée pour manipuler les systèmes d'équations récurrentes affines paramétrés est représentée figure A.5.

Chaque variable (d'entrée, de sortie, ou intermédiaire) et chaque paramètre est associé à une structure "`Terme`". Une liste d'équations "`Equation`" définissant les équations récurrentes est associée à chaque variable intermédiaire. A chaque équation est associée la liste de termes droits de l'équation : une liste d'"`Occurrence`".

A.2.3 Organisation des fichiers sources

Voici une liste des fichiers contenus dans le répertoire principale d'OPÉRA et une brève description de leurs contenus :

```
Alpha/              (répertoire) analyseur lexical et syntaxique Alpha
Christophe_CodeGen/ (répertoire) générateur de code des communications
Olivier_CodeGen/    (répertoire) générateur de code data-parallèle
Polylib/            (répertoire) repertoire de base de la PolyLib
examples/           (répertoire) examples de programmes Alpha
CHANGES             historique des modifications
FILES               ce fichier
Makefile            makefile
TODO                développements futurs
alloc.c             algorithme de Blankinship : vecteur
alloc.h                            d'allocation -> matrice
base.c              calcul d'une base d'un espace vectoriel contenant un
base.h                             polyèdre
config.h            divers #define relatifs à l'interface
dependance.c        calcul des ensembles d'utilisation, vecteurs de
dependance.h                       dépendance, ...
domains.c           manipulation des domaines de calculs
domains.h
free.c              libération de la mémoire allouée par un PARE
free.h
geom.c              fonctions géométriques diverses (ext. de la PolyLib)
geom.h
load_alpha.c        transformation de la structure Alpha en structure
load_alpha.h                       interne OPÉRA
node.h              lien vers Alpha/node.h
nodetypes.h         lien vers Alpha/nodetypes.h
opera.README        fichier d'aide
opera.c             routine principale (main)
opera.h
pol_init.c          routines d'initialisation après lecture d'un fichier
pol_init.h
pulsar.icon         icône
pulsar.xbm
terme.h             structures de données (Terme, Equation, Occurrence)
timing.c            fonctions relatives au cadencement
timing.h
trans.c             fonctions relatives aux transformations espace-temps
```

```
trans.h
x_analyse.c         interface X : fenêtre analyse
x_analyse.h
x_colors.c          X : couleurs
x_colors.h
x_ddraw.c           X : routines de dessin dans la fenêtre Domain
x_ddraw.h
x_domain.c          X : fenêtre Domain
x_domain.h
x_file.c            X : lecture d'un fichier
x_file.h
x_param.c           X : valeurs des paramètres et sélection d'un domaine
x_param.h                          de validité
x_repere.c          X : fenêtre Projection
x_repere.h
x_trans.c           X : fenêtre Transformations
x_trans.h
```

Annexe B

PolyLib : A Library for Parameterized Polyhedra

B.1 Introduction

The polyhedral library (or *PolyLib* for short) operates on objects made up of unions of polyhedra of any dimension. It was developed by Doran Wilde at IRISA, in Rennes, France, in connection with the ALPHA project. This document is a continuation of the technical report "A library for doing polyhedral operations" by Doran Wilde, IRISA report no 785, Dec. 1993, describing the previous release of the PolyLib, version 1.1. Version 1.1 only manipulates non parameterized unions of polyhedra through the following operations : intersection, difference, union, convex hull, simplify, image and preimage, plus some input and output functions. The polyhedra are manipulated in their dual implicit and Minkowski representations, in homogeneous spaces. Each polyhedron is represented by two matrices : a matrix of lines and rays, and a matrix of equalities and inequalities. The first column of these matrices distinguishes lines from rays and equalities from inequalities respectively.

This document describes the functions and data structures that have been added to PolyLib1, in order to manipulate *parameterized polyhedra*. Release 2 of the PolyLib includes parameterized vertices computation. The version including Ehrhart polynomials computation is the last release, called PolyLib3.

Parameterized polyhedra representation

PolyLib manipulates *rational* polyhedra. There are two dual representations of polyhedra : the implicit representation, as a set of constraints, and the Minkowski representation, as a set of lines, rays and vertices. A *parameterized polyhedron* is defined in the implicit form by a finite number of inequalities and equalities, which constant part depends linearly on a parameter vector p:

$$\mathcal{D}(p) = \{x \in \mathbb{Q}^n \mid Ax = A'p + a, \quad Bx \geq B'p + b\} \qquad \text{with } p \in \mathbb{Q}^m$$

where A is a $k \times n$ integer matrix, A' a $k \times m$ integer matrix, a is an integer k-vector, B is a $k' \times n$ integer matrix, B' a $k' \times m$ integer matrix and b is an integer k'-vector.

The Minkowski representation, as a set of lines, rays, and vertices, of a parameterized polyhedron is:

$$\mathcal{D}(p) = \left\{x \in \mathbb{Q}^n \mid x = L\lambda + R\mu + V(p)\nu,\ \forall\lambda,\ \forall\mu \geq 0,\ \forall\nu \geq 0,\ \sum \nu = 1\right\}$$

where L is the matrix containing the lines, R the matrix containing the rays, and $V(p)$ the matrix depending on the parameters p containing the vertices of the polyhedron.

PolyLib2 includes an algorithm, described below, computing the vertices $V(p)$ of a parameterized polyhedron.

Parameterized vertices representation

Each vertex of a parameterized polyhedron is an affine function of the parameters p, defined over a *validity domain*: each vertex exists only if p is included into the validity domain associated to this vertex. There are two ways of representing such a set of parameterized vertices and validity domains:

- as a list of pairs, each one containing a vertex and its validity domain,
- as a list of distinct validity domains, and the complete matrix $V(p)$ associated to each validity domain.

There are two functions in PolyLib2 computing the parameterized vertices in these two representations: `Polyhedron2Param_Vertices` and `Polyhedron2Param_Domain` respectively.

Ehrhart polynomials representation

The *Ehrhart polynomials* associated to each of the distinct validity domains correspond to the number of integer points contained in a parameterized polytope, when the parameters are integers. They are *pseudo-polynomials*: polynomials whose coefficients are *periodic numbers*, i.e. numbers taking different values depending on the rest of the division of the parameters by the *period* of this periodic number. The function `Polyhedron_Enumerate`, in PolyLib3, returns a list of validity domains and each corresponding Ehrhart polynomial.

Summary

The next section describes the structures used for parameterized vertices and pseudo-polynomials internal representations. Section B.3 is an overview of the different algorithms. For a complete formal description of these algorithms refer to chapter II.1. Section B.4 is a description of the standalone executables, and some examples. Some comments on distribution and authors are given in the last section.

B.2 Data stuctures

Parameterized vertices

There are two ways of representing the vertices of parameterized polyhedra. Both use the same data structure, `Param_Polyhedron`, described below.

```
typedef struct _Param_Poly
{
   int nbV;
   Param_Vertices *V;
   Param_Domain   *D;
}
Param_Polyhedron;
```

The fields of the `Param_Polyhedron` structure are as follows:

`->nbV` number of parameterized vertices.

`->V` linked list of parameterized vertices.

`->D` linked list of validity domains.

```
typedef struct _Param_V
{
   struct \_Param\_V *next;
   Matrix *Vertex;
   Matrix *Domain;
}
Param_Vertices;
```

The fields of the `Param_Vertices` structure are as follows:

`->next` pointer to the next vertex. The list is NULL-terminated.

`->Vertex` this matrix contains the coordinates of the vertex, as function of the parameters. Each line is a coordinate of the vertex. The m first columns are the coefficients of the parameters, the $(m+1)$th value is the constant, the $(m+2)$th value is the common denominator. It is an $n \times (m+2)$ matrix.

`->Domain` the constraints on parameters, `Polyhedron` format.

```
typedef struct _Param_Domain
{
   struct _Param_Domain *next;
   Polyhedron *Domain;
   int *F;
} Param_Domain;
```

The fields of the `Param_Domain` structure are as follows :

`->next` pointer to the next domain. The list is NULL-terminated.

`->Domain` the validity domain : constraints on the parameters, `Polyhedron` format.

`->F` a bit array of the corresponding vertices in the `Param_Vertices->Vertex` ordered list.

Figure B.1 shows these structures dependencies. In the first representation, as a list of vertices and a validity domain associated to each vertex, the `Param_Domain *D` field of a `Param_Polyhedron` is not used. In the second representation, as distinct validity domains and a set of vertices associated to each of them, the `Matrix *Domain` of each vertex of the list `Param_Vertices` is not used.

There is a convenient macro to scan all the vertices of a parameterized polyhedron, that can be used as follows :

```
{
   /* declarations */
   Param_Polyhedron *P;
   Param_Domain     *D;
   Param_Vertices   *V;

      [...]

   /* Computation of P  */
   P = Polyhedron2Param_Domain(  [...]  );

   /* scan the validity domains */
```

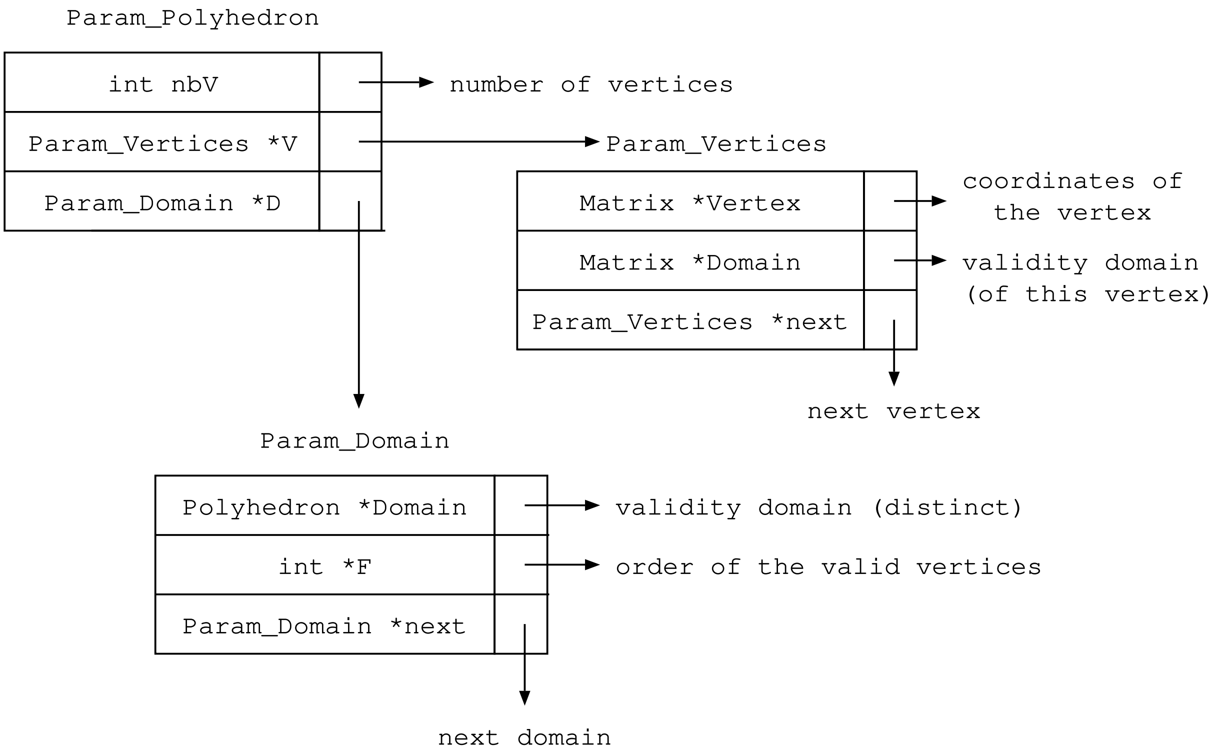


FIG. B.1 - *The* `Param_Polyhedron` *structure.*

```
  for( D=PA->D ; D ; D=D->next )
  {
     /* print the current validity domain */
     printf( "---------------------------------------\n" );
     printf( "Domain :\n");
     Print_Domain( stdout, P->Domain );

     /* scan the vertices */
     printf( "Vertices :\n");
     FORALL_PVertex_in_ParamPolyhedron(V, D, P)
     {
        /* print each vertex */
        Print_Vertex( stdout, V->Vertex );
        printf( "\n" );
     }
     END_FORALL_PVertex_in_ParamPolyhedron;
  }
}
```

Ehrhart polynomials

The structure Enumeration described below is used to represent Ehrhart polynomials over validity domains. It is a linked list of distinct validity domains and Ehrhart polynomials :

```
typedef struct _enumeration
{
  struct _enumeration *next;  /* next in the list, NULL-terminated */
  Polyhedron *ValidityDomain; /* contraints on the parameters      */
  evalue EP;                  /* corresponding Ehrhart Polynomial  */
} Enumeration;

typedef struct _evalue
{
   int d;
   union
   {
      int n;
      struct _enode *p;
   } x;
} evalue;
```

An `evalue` is either a constant rational, or a pointer to an `enode`. The fields of the `evalue` structure are as follows:

`->d` the denominator of the constant rational if not equal to 0. If d is equal to 0, the pointer to an enode is used.

`->x.n` the numerator of the constant rational.

`->x.p` a pointer to an `enode`.

```
typedef enum { polynomial, periodic, evector } enode_type;

typedef struct _enode
{
   enode_type type;
   int size;
   int pos;
   evalue arr [1];
} enode;
```

An `enode` is either a `polynomial`, or a `periodic` (the `evector` type is not used). The fields of the `enode` structure are as follows:

`->type` the type of the enode: `polynomial` or `periodic`.

`->size` the number of attached pointers in the array `arr`. If the enode is of type `periodic`, `size` is the period of the periodic number. If the enode is of type `polynomial`, `size` is the degree plus 1 of the polynomial.

`->pos` the number of the parameter used in this enode. Parameters are numbered from 1 to m.

`->arr` an array of `size` evalues. Do not trust the declaration, the number of evalues is *not* 1: when an `enode` structure is allocated, depending on the number of necessary evalues, the necessary amount of memory is allocated [Doran's optimisation].

The `enode` structure corresponding to the Ehrhart polynomial $[1,2]p^2+3p+\frac{5}{2}$ is given figure B.2.

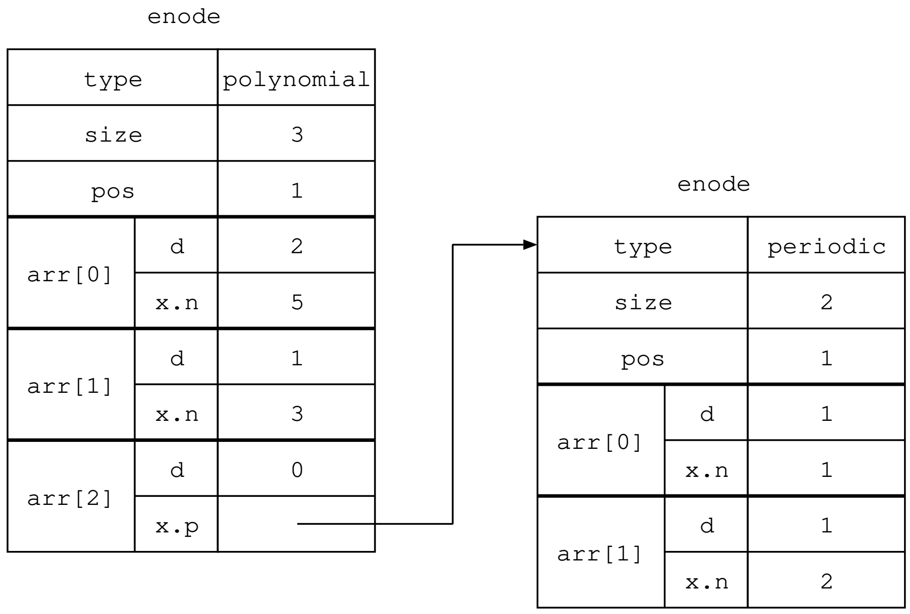


FIG. B.2 - *The Ehrhart polynomial* $[1,2]p^2 + 3p + \frac{5}{2}$.

B.3 Description of operations

Parameterized vertices computation

As mentioned section B.1, the vertices of a parameterized polyhedron can be represented in two ways: the first form as a set of vertices and their validity domains, and the second one as a set of distinct validity domains and all the corresponding vertices. These two forms are respectively computed by the following functions:

```
Param_Polyhedron *Polyhedron2Param_Vertices ( Polyhedron *D,
                                               Polyhedron *C,
                                               int MaxRays );

Param_Polyhedron *Polyhedron2Param_Domain ( Polyhedron *D,
                                             Polyhedron *C,
                                             int MaxRays );
```

Both functions take as input:

- `D`: the combined polyhedron of dimension $n + m$ where n is the size of the index space, and m the number of parameters.

- `C` : the context polyhedron, of size m : this polyhedron contains the constraints on the parameters. If there are no constraints, `C` must be equal to the universe polyhedron of dimension m.

- `MaxRays` : an integer, which gives the maximum number of rays in a (non parameterized) polyhedron. This value is used by the functions of the PolyLib1.1.

The output is the same `Param_Polyhedron` structure, but with different contents as described above.

`Polyhedron2Param_Domain` makes a first call to `Polyhedron2Param_Vertices`, and then finds a partition of the parameter space given the validity domains of the vertices, using a function named `Compute_PDomains`.

The other functions used in this context are the following :

```
void Param_Polyhedron_Free ( Param_Polyhedron *P );
```

This function frees all memory allocated by a `Param_Polyhedron` structure.

```
void Print_Domain ( FILE *DST, Polyhedron *D );

void Print_Vertex ( FILE *DST, Matrix *V );
```

These two functions make pretty printouts on the file DST, of a validity domain and of a parameterized vertex respectively.

All these functions are defined in the C source files `polyparam.[ch]`.

Ehrhart polynomials computation

The main function computing a set of validity domains and Ehrhart polynomials is :

```
Enumeration *Polyhedron_Enumerate ( Polyhedron *P, Polyhedron *C,
                                                     int MaxRays );
```

It takes as input :

- `P` : the combined polyhedron of dimension $n+m$. It must of course be a parameterized *polytope*.

- `C` : the context polyhedron, of size m. This polyhedron contains the constraints on the parameters. If there are no supplementary constraints, you have to provide the universe polyhedron of dimension m.

- `MaxRays` : an integer, which gives the maximum number of rays in a (non parameterized) polyhedron. This value is used by the functions of the PolyLib1.1.

The output is an `Enumeration` structure as described above.

`Polyhedron_Enumerate` makes a call to `Polyhedron2Param_Domain` to compute the validity domains and the vertices of the parameterized polyhedron. It then computes the denominator of the Ehrhart polynomials, and then, for each of the validity domains, makes a call to the function `P_Enum` which computes the Ehrhart polynomial corresponding to this validity domain. For a description of this algorithm, see chapter II.2, section 1.5. The C source files are `ehrhart.[ch]`.

The other important function defined in this context is :

```
int compute_poly ( Enumeration *E, int *p );
```

This function computes the value of an Ehrhart polynomial given the parameters. It first looks for the right validity domain in `E`. If there is no validity domain corresponding to the value of p, the polyhedron is empty, and the function returns 0. If a validity domain is found, it computes the corresponding Ehrhart polynomial and returns its integer value. It is defined in the C source files `eval_ehrhart.[ch]`.

B.4 Usage

The PolyLib can be used as a linked C library, or can also be used as one of the standalone executables described below. Three executables manipulating parameterized polyhedra are created by the makefile :

- `findv` : computes the list of vertices of a parameterized polyhedron and their validity domains, and prints them on the standard output.
- `pp` : computes the list of distinct validity domains and their associated sets of parameterized vertices, and prints them on the standard output.
- `ehrhart` : computes the list of distinct validity domains, their associated sets of parameterized vertices, and their associated Ehrhart polynomials, and prints them on the standard output.

By default, the parameter names are P, Q, R, ... The inputs of these programs are given as two matrices, the combined and the context polyhedra, in PolyLib format.

Example

The following example is taken from chapter II.2, section 2.2. Consider the polyhedron :

$$\mathcal{A}(P,Q) = \{(i,j,k) \mid 0 \leq i \leq P, \quad 0 \leq j \leq i, \quad 0 \leq k \leq i-j, \quad i+j+k = Q\}$$

The correponding input is :

```
#------------------------------
# Dimension of the matrix :
7 7
# Constraints :
#     i  j  k    P  Q    cte
1     1  0  0    0  0     0    # 0 <= i
1    -1  0  0    1  0     0    #      i <= P
1     0  1  0    0  0     0    # 0 <= j
1     1 -1  0    0  0     0    #      j <= i
1     0  0  1    0  0     0    # 0 <= k
1     1 -1 -1    0  0     0    #      k <= i-j
0     1  1  1    0 -1     0    # Q = i + j + k

# 2 parameters, no constraints.
0 4
```

And this is the output of the Ehrhart program :

```
---------------------------------------
Domain :
         2P - Q  >= 0
         - P + Q  >= 0
          1 >= 0

Vertices :
[ Q/2,  Q/2,  0 ]
[ Q/2,  0,  Q/2 ]
[ P,  -P+Q,  0 ]
[ P,  0,  -P+Q ]

Ehrhart Polynomial:
( -1/2 * P^2 + ( 1 * Q + 1/2 )
 * P + ( -3/8 * Q^2 + [ 1/4, 0 ]_Q * Q + [ 1, 3/8 ]_Q )
 )

---------------------------------------
Domain :
         P - Q  >= 0
         Q  >= 0
          1 >= 0
```

```
Vertices :
[ Q/2,  Q/2,  0 ]
[ Q/2,  0,  Q/2 ]
[ Q,  0,  0 ]

Ehrhart Polynomial:
( 1/8 * Q^2 + [ 3/4, 1/2 ]_Q * Q + [ 1, 3/8 ]_Q )
```

There are two validity domains. The mathematical form of this output is :

$$\begin{cases} -\frac{1}{2}P^2 + QP - \frac{3}{8}Q^2 + \frac{1}{2}P + \left[\frac{1}{4}, 0\right]_Q Q + \left[1, \frac{3}{8}\right]_Q & \text{if } P \leq Q \leq 2P \\ \frac{1}{8}Q^2 + \left[\frac{3}{4}, \frac{1}{2}\right]_Q Q + \left[1, \frac{3}{8}\right]_Q & \text{if } 0 \leq Q \leq P \end{cases}$$

The evaluation function is then called, if the input of the program is the standard input (and not a file).

B.5 Conclusion

Authors

Various authors have contributed to the PolyLib. Version 1.1 has been designed and programmed by Doran Wilde and Hervé Le Verge in the API group at IRISA (France). Since version 1.1, the main contributors are Doran Wilde (`wilde@ee.buy.edu`, Brigham Young University, Provo, Utah), and Vincent Loechner (`loechner@icps.u-strasbg.fr`, ICPS, Université Louis Pasteur, Strasbourg, France). The parameterized polyhedra and validity domains computation functions were first written by Vincent Loechner. Doran Wilde wrote the Ehrhart polynomials computation. Both sets of functions were tested, debugged and optimized by Doran Wilde and Vincent Loechner. The function evaluating an Ehrhart polynomial was written by Emmanuel Jeannot (`Emmanuel.Jeannot@ens-lyon.fr`, ENS, Lyon, France), and Xavier Redon (`Xavier.Redon@eudil.univ-lille1.fr`, LIFL, Université de Lille, France) also contributed to some bug corrections.

Distribution

The version 1.1 of the PolyLib and its documentation are copyrighted 1993 by IRISA, Université de Rennes I, France, all rights reserved. It is distributed under the terms of the GNU General Public license.

The last release of the PolyLib, version 3, is freely distributed on :

`http://icps.u-strasbg.fr/~loechner/polylib/`

Permission is granted to copy, use, and distribute for any commercial or noncommercial purpose under the terms of the GNU General Public license, version 2, June 1991.

Annexe C

Deriving Formulae to Count Solutions to Parameterized Linear Systems using Ehrhart Polynomials: Applications to the Analysis of Nested-Loop Programs

Deriving Formulae to Count Solutions to Parameterized Linear Systems using Ehrhart Polynomials: Applications to the Analysis of Nested-Loop Programs

Philippe Clauss †, Vincent Loechner †, Doran Wilde ‡

† ICPS, Université Louis Pasteur de Strasbourg,
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France
e-mail : {clauss,loechner}@icps.u-strasbg.fr

‡ Brigham Young University
Department of Electrical and Computer Engineering
459 Clyde Building, Provo, Utah 84602
e-mail : wilde@ee.byu.edu

Keywords

Polyhedron, Enumeration, Parametric Analysis, Ehrhart Polynomial, Symbolic Analysis, Nested-Loop Programs, Lattice, Parallelizing Compiler, Volume

Abstract

Optimizing parallel compilers need to be able to analyze nested loop programs with parametric affine loop bounds, in order to derive efficient parallel programs. The iteration spaces of nested loop programs can be modeled by polyhedra and systems of linear constraints. Using this model, important program analyses such as computing the number of flops executed by a loop, computing the number of memory locations or cache lines touched by a loop, and computing the amount of processor to processor communication needed during the execution of a loop—all reduce to the same mathematical problem: finding the formula for number of integer solutions to a system of parameterized linear constraints, as a function of the parameters.

In this paper, we present a method for deriving a closed-form symbolic formula for the number of integer points contained in a polyhedral region in terms of size parameters. If the set of integer points to be counted lies inside a union of rational convex polytopes, then the number of points can be formulated by a special kind of polynomial called an *Ehrhart pseudo-polynomial.* The procedure consists of first, computing the *parametric vertices* of a polytope defined by a set of parametric linear constraints, and then solving for the pseudo-polynomial which gives the exact formula for the number of integer points in the polytope. An algorithm to derive this formula is presented. The method is then illustrated using many examples from the literature and in the area of parallel program optimization.

1 Introduction

Counting the number of integer solutions to a system of linear constraints, and counting the number of integer points in a polytope are duals of the same problem. If the constraints are expressed in terms of one or more parameters (the polytope has vertices expressed in terms of one or more parameters), then the count can be expressed by a formula which is a function of those parameters. This paper shows how such a formula may be derived.

Kuck [15] showed that the iteration domain of a nested-loop program with affine lower and upper bounds can be described in terms of a polyhedron. The *polyhedral model* has since been used to model and analyze static nested-loop programs. As a result, fundamental mathematical problems in the areas of discrete and convex geometry, enumerative combinatorics, geometry of numbers and linear programming have been posed in connection with the analysis of nested-loop programs. One of these fundamental problems is finding a formula for the number of integer points contained in a parameterized polytope. Moreover, the formulation needs to be parameterized, since scientific algorithms (and hence the polyhedra that model them) often depend on size parameters.

Such an enumeration formula is helpful when analyzing parallel algorithms. In order to produce efficient parallel programs, a compiler may need to know the number of operations performed by a loop, the number of memory locations or cache lines touched by a loop, or the number of array elements that need to be transmitted from a processor to another during the execution of a loop. Such analyses pose the same general problem: counting the number of integer solutions of selected free variables in a set of linear constraints. The resulting information can be used to estimate the execution time of a code segment, compare the memory bandwidth requirements versus the flop count of a code segment [21], determine which loops will flush the cache, and then calculate the cache miss rate [23, 11], evaluate message traffic and allocate message buffers, determine whether a parallel loop is load balanced (i.e., performs the same amount of computation in each iteration [27]), or if not balanced, determine number of iterations to assign to each processor in order to balance the work load (balanced chunk-scheduling) [13], exploit maximum parallelism, optimize the number of needed processors, and optimize loop and data partitioning.

A *parameterized* linear system is specified by a set of parametric linear constraints which linearly depend on one or more positive integral size parameters $n_1, n_2, \cdots$. A parameterized linear constraint can be written as:

$$\begin{aligned} \sum_i a_i x_i &\leq \sum_j b_j n_j + c \qquad \text{or,} \\ \sum_i a_i x_i &= \sum_j b_j n_j + c \end{aligned}$$

where the a_i's, b_i's and c are rational constants, the n_j's are the size parameters and the x_i's are free variables. A system of such constraints defines a lattice of integral points $x \in \mathcal{Z}^k$ contained in a convex polytope. We want to know the number of lattice points contained in a parameterized polytope, or in other words, the number of integer solutions to the parameterized system of constraints.

This number can be expressed as a symbolic sum, which is not generally an easy formulation. Some symbolic mathematical packages such as Maple or Mathematica can generate such symbolic sums, however they assume that the lower bound is never greater than the upper bound. If this assumption is violated, then their answer is wrong. For example, Mathematica computes that

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

This answer is valid only if $n \leq m$. A different formula, $\frac{m(m+1)}{2}$, is required for the case $n > m$. In general, the symbolic number of integer solutions to a system of constraints may differ for different subdomains of the size parameters. This fact was pointed out by the French mathematician Eugène Ehrhart in 1977 [10, p. 139] based on work he did in the late 1950's:

Conjecture 1 (Ehrhart's conjecture) *For any significant diophantine linear system of any dimension, linearly dependent on several positive integral parameters, the symbolic number of integer solutions as a function of those parameters is expressed at different subdomains of the parameter space by different pseudo-polynomials.*

This conjecture is proved later in this paper (theorem 2). A *pseudo-polynomial* is a polynomial where the values of the coefficients rotate cyclically through an array of constants according to the modulos of the parameters. A more formal definition is given in the next section. Ehrhart showed that the values of these coefficients are closely related to the vertices of the parameterized polytope. The validity domains associated with the pseudo-polynomials are also associated with the parametric coordinates of the vertices.

Earlier methods to compute the coefficients of a pseudo-polynomial were proposed using several approaches. For example, Barvinok[1] [2] proposed a complexity study of the computation of these coefficients in terms of computation of the volumes of faces, since

[1]Barvinok also proposed in [1] a polynomial-time algorithm for counting integral points in non-parametric polyhedra, i.e. when the values of the parameters are fixed.

any coefficient can be related to volumes or surface areas of an associated polyhedron. But since the problem of volume computation is NP-hard [6], results given by Barvinok relied on the strong assumption of a *Volume Computation Oracle*.

In this paper, we present a new algorithmic method for computing the formula for the number of integer points contained in a parameterized polytope, that is, a polytope defined by a set of linear constraints depending linearly on a set of parameters. The first step of this method consists of computing the parametric vertices of the polytope. To find the vertices of parameterized polyhedra, a new algorithm developed by Loechner and Wilde [17] is used. This algorithm partitions the parameter domain into subdomains where the vertices of a parametric polyhedron are stable, and it provides the information about the vertices that is needed to apply Ehrhart's results. We have extended Ehrhart's results that were based on a single positive integral parameter, to any number of positive integral parameters. The implementation of this extension was made possible by the use of the Loechner-Wilde algorithm. Using this information, a pseudo-polynomial is formulated with unknown coefficients that meets Ehrhart's criteria for being an enumerator for the polytope. Then, a certain number of instances of the polytope are scanned for a range of small fixed values of the parameters and the number of interior points in each instance is counted. Enough information is gathered to build a system of linear equations to solve for the unknown coefficients. Finally, the pseudo-polynomial is symbolically simplified according a set of rewrite rules. This procedure is repeated for each of the parameter subdomains identified by the Loechner-Wilde algorithm. The final result is a piecewise pseudo-polynomial enumerator formula for the polytope.

This paper will be presented as follows. In section 2, we begin by presenting the *polyhedral model* and by giving some geometric and arithmetic definitions and some of Ehrhart's contributions. Then, in section 3 we present the definition and properties of *pseudo-polynomials* and show how they are used to formulate the number of lattice points inside a union of convex polytopes. In section 4, we describe the algorithm used to derive these formula. In section 5, the method presented here is illustrated with many examples and compared with related works [11, 27, 13, 14, 26, 21]. Many applications can make good use of a symbolic formula for the number of integral solutions to system of parameterized constraints. In section 6, we will show example applications in which the results in this paper can be effectively used for the analysis and transformation of scientific programs. In section 7 we briefly suggest to how our ideas may be applied to certain non-linear systems. And finally, in section 8, we summarize what we have presented.

2 The polyhedral model

We first recall some basic notions dealing with geometry of numbers [12, 22, 3] and enumerative combinatorics [24, 25]. Then some more specific concepts, closely related to the content of the paper, are introduced.

2.1 Polyhedra and polytopes

Let $\mathcal{Q}$ denote the set of rational numbers and $\mathcal{Z}$ the set of integers. A *(convex) polyhedron* is defined by a finite system of linear inequalities $P = \{x \in \mathcal{Q}^d \mid Ax \leq b\}$ where A is a rational matrix and b a rational vector. A bounded convex polyhedron is called a convex *polytope*, and represents the convex hull of a finite number of points $x_1, \ldots, x_{m+1}$ in $\mathcal{Q}^d$. If $x_1, \ldots, x_{m+1}$ are affinely independent points of $\mathcal{Q}^d$ (which also means that $x_2 - x_1$, $x_3 - x_1, \ldots, x_{m+1} - x_1$ are linearly independent) then the convex hull of $x_1, \ldots, x_{m+1}$ is called an *m-dimensional simplex* of $\mathcal{Q}^d$ with vertices $x_1, \ldots, x_{m+1}$.

Since our goal is to count the number of integer points contained in a polyhedron, it must be bounded, and thus only polytopes are considered in the rest of the paper.

The *affine span* of P is the smallest affine subspace of $\mathcal{Q}^d$ entirely containing P. The *dimension* of P is the dimension of its affine span, and a k-dimensional polytope is called a k-polytope for brevity.

If $\vec{c}$ is a nonzero vector and γ is a rational constant defined by $\gamma = MAX\{\vec{c}x \mid x \in P\}$, the affine hyperplane $\{x \mid \vec{c}x = \gamma\}$ is called a *supporting hyperplane* of P. The *faces* of P are $\emptyset$, P, and the intersections of P with its supporting hyperplanes. Each face of P is itself a polyhedron and is called a k-face if it is k-dimensional. If P is a k-polyhedron, faces of P of dimension 0, 1 and $k-1$ are called *vertices, edges* and *facets*, respectively.

2.2 Parameterized lattice polytopes

We consider parameterized polytopes of the form:

$$P_N = \{x \in \mathcal{Q}^d \mid Ax \leq BN + b; Cx = DN + d\}$$

where N is the symbolic p-vector of the parameters, A, B, C, and D are constant rational matrices, and b and d are constant rational vectors. P_N can also be thought of as a family of polytopes where each valid assignment of values to the vector N gives one member of the set.

Parameterized polytopes may also be represented in the *combined* data and parameter space as a polyhedron P' defined as:

$$P' = \left\{ \begin{pmatrix} x \\ N \end{pmatrix} \in \mathcal{Q}^{d+p} \mid [A \mid -B] \begin{pmatrix} x \\ N \end{pmatrix} \leq c; [C \mid -D] \begin{pmatrix} x \\ N \end{pmatrix} = d \right\}$$

2.3 Lattices

A *lattice* is the set of points $\mathcal{L}^d = \{\alpha_1\vec{g}_1 + \ldots + \alpha_d\vec{g}_d \mid \alpha_1, \cdots, \alpha_d \in \mathcal{Z}\}$ where $\vec{g}_1, \ldots, \vec{g}_d$ are linearly independent vectors of $\mathcal{Q}^d$, called the *basis* of the lattice. The set of all the integral points is called *standard lattice* denoted by $\mathcal{Z}^d$. Let G be the matrix whose columns are the vectors $\vec{g}_1, \ldots, \vec{g}_d$, called the *dilatation matrix*. Any lattice $\mathcal{L}^d$ consists of the points $\mathcal{L}^d = \left\{x \mid x = Gz, z \in \mathcal{Z}^d\right\}$. A *lattice polytope* or a *lattice simplex* is then defined by a set of linear inequalities of the form $\{z \in \mathcal{L}^d \mid Az \leq b\}$ where A is a $m \times d$ rational matrix and

b a m-dimensional rational vector. The intersection between a lattice $\mathcal{L}^d$ and a polytope is called a *lattice polytope* P of $\mathcal{L}^d$. A lattice polytope P of $\mathcal{L}^d$ is a $\mathcal{L}^d$-polytope if all of its vertices are in $\mathcal{L}^d$. If $\mathcal{L}^d = \mathcal{Z}^d$, P is an *integral polytope*. If some vertices of P do not belong to $\mathcal{L}^d$ and have rational coordinates, P is a *rational* polytope. The *enumeration* of any set E is the number of points in $E \cap \mathcal{L}^d$. An *enumerator* is a formula to give the number of points in $E \cap \mathcal{L}^d$.

Since by definition, any lattice $\mathcal{L}^d$ can be derived from $\mathcal{Z}^d$ by a non-singular transformation G [12], any $\mathcal{L}^d$-polytope is affinely equivalent to some integral polytope. Hence, results given in the rest of this paper concerning integral polytopes can be directly extended to $\mathcal{L}^d$-polytopes. From now on and without loss of generality, the generic term of integral polytopes will be used.

2.4 Ehrhart polyhedra

Many scientific algorithms are defined parametrically in order to apply to any size of problem. Implementations of these algorithms yield programs with loops involving parameterized affine loop bounds. When modeling such loops with convex polytopes, parameterized polytopes have to be considered. Eugène Ehrhart defined a type of parameterized polytope which he called *homothetic* and *bordered systems* [8, 9] that are similar to what was defined above in section 2.2. However, his definitions were limited to homothetic and bordered systems of only a single parameter n. We will extend his results to the general case of any number of parameters.

In this work, we also consider polytopes which may be non-convex, called *normal polyhedra*, according to the following definition. This definition corresponds to the one given by Ehrhart in [8] and [10, p. 47] and includes unions of convex polytopes.

Definition 1 (normal polyhedron)
*A **normal polyhedron** P is a closed, not necessarily convex polyhedron which admits a finite rectilinear simplicial subdivision. It is defined to be the union of a finite number of simplices $S_1, S_2, ..., S_n$ of equal dimension such that,*

- *The intersection between any two simplices is either empty or is a common face,*
- *Each facet of one of these simplices belongs to at most one other simplex. Faces belonging to only one simplex form the border of the polyhedron,*
- *Assuming $P_h = S_1 \cup S_2 \cup ... \cup S_h$, these simplices can be ordered such that for any $h < n$, the intersection $P_h \cap S_{h+1}$ is either empty, or is the union of facets common to S_{h+1} and the border of P_h.*

Our method of counting the number of integer points is based on the decomposition of a parametric polyhedron into several *homothetic and bordered* systems, associated with *validity domains*, according to the following definition which is an extension of Ehrhart's definition [8] to any number of parameters.

Definition 2 (homothetic system)
*A system H_N, $N = (n_1, \cdots, n_q)$ consists of a set of **homogeneous** constraints of the form $\sum a_i x_i = \sum b_j n_j$, $\sum a_i x_i < \sum b_j n_j$, $\sum a_i x_i \leq \sum b_j n_j$, where the a_i's and the b_j's are integer constants, the x_i's are free variables and the n_j's are positive integral parameters. H_N defines a (convex) polyhedron P_N.*

*Such a system is **significant** if and only if $H_{(1,1,\ldots,1)}$ defines a (convex) polytope $P_{(1,1,\ldots,1)}$.*

*H_N is **homothetic** if all the q one-parameter polytopes $P_{0,0,\ldots,0,n_j,0,\ldots,0}$, $1 \leq j \leq q$, each correspond to $P_{0,0,\ldots,0,1,0,\ldots,0}$ by a n_j-homothetic transformation* [2] *of the origin O. All the $P_{0,0,\ldots,0,1,0,\ldots,0}$'s are called the **primitive domains** of H_N.*

The properties of such homothetic polyhedra yield the main mathematical result given later in the paper. This definition can be extended to *bordered polytopes* in order to handle polytopes modeling loops with *affine* loop bounds. The following definition is also an extension of Ehrhart's [8].

Definition 3 (bordered system)
*A system B_N, $N = (n_1, \cdots, n_q)$ consists of a set of **affine** constraints of the form $\sum a_i x_i = \sum b_j n_j + c$, $\sum a_i x_i < \sum b_j n_j + c$, $\sum a_i x_i \leq \sum b_j n_j + c$, where the a_i's, b_j's, and c are integer constants, the x_i's are free variables and the n_j's are positive integral parameters. B_N defines a (convex) polyhedron P'_N.*

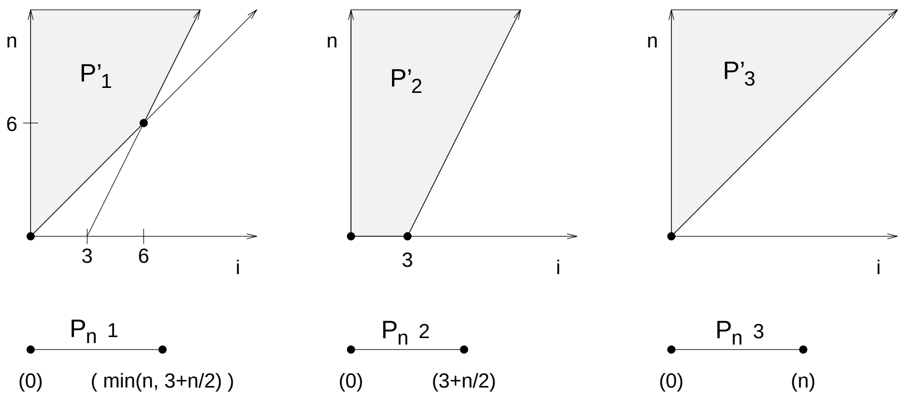
*The system B_N is **bordered** if removing all the c's results in a homothetic system H_N which defines a polyhedron P_N. P'_N may be obtained by intersecting hyperplanes parallel to each facet of P_N at a rational distance from the facet independent of the parameters N. P_N is called the **kernel** of P'_N.*

Observe that the borders can cut the kernel while the parameters are less than certain minimal values. Hence, P'_N is said to be *complete* if it exists, i.e. when all the borders are involved as bounds. Observe also that a translation, independent of the parameters, transforms an homothetic polytope P_N into a bordered polytope, complete for all values of the parameters.

In general, a parameterized polyhedron is a set of homothetic or bordered polyhedra qualified by subdomains of the parameter space called *validity domains*. These subdomains are computed by the algorithm that finds the parametric vertices described in a later section.

[2] Let O be a point in a vector-space $\mathcal{R}^d$, P be a set of points in $\mathcal{R}^d$, and k be a real number not equal to zero. Then, the *k-homothetic transformation* of P about a center point O is defined as $\{\ N \in \mathcal{R}^d \ \mid\ M \in P;\ \overrightarrow{ON} = k\ \overrightarrow{OM}\ \}$.

Example 1



In the above figure, we have three parameterized, one dimensional polyhedra,

$$\begin{aligned} P_n1 &= \{i \mid i \geq 0; i \leq n; 2i \leq n+6\} \\ P_n2 &= \{i \mid i \geq 0; n \geq 0; 2i \leq n+6\} \\ P_n3 &= \{i \mid 0 \leq i \leq n;\} \end{aligned}$$

P_n3 is a homothetic system, and P_n2 is a bordered system. P_n1 is neither homothetic nor bordered, but can be made as a union of homothetic and bordered systems qualified by certain subdomains of the parameter (n) space,

$$P_n1 = \cup \begin{cases} P_n2 & \text{when} \quad 0 \leq n \leq 6 \\ P_n3 & \text{when} \quad 6 \leq n \end{cases}$$

□

The results of this paper apply to any k-polyhedron P_N that is a union of homothetic or bordered systems of the form $\{z \in \mathcal{Q}^d \mid Az \leq BN + b; Cz = DN + d\}$, where $N = (n_1, \ldots, n_q)$.

2.5 Pseudo-polynomials

The enumerator of a parameterized polyhedron can be formulated using *polynomials* and *pseudo-polynomials* according to the following definitions. Before we can define pseudo-polynomials, we must first introduce the concept of periodic numbers. We extend Ehrhart's definition of 1-dimensional periodic numbers [10] to any dimension.

Definition 4 (periodic number) [3]
A one-dimensional periodic number $u(n) = [u_0, u_1, \ldots, u_{p-1}]_n$ *is equal to the item whose*

[3] Ehrhart originally defined a periodic number to be
$$u(n) = [u_1, u_2, \ldots, u_p]_n = \begin{cases} u_1 & \text{if } (n \bmod p) = 1 \\ u_2 & \text{if } (n \bmod p) = 2 \\ \ldots & \\ u_p & \text{if } (n \bmod p) = 0 \end{cases}$$
The definition used in this paper has been modernized to index the array u starting at 0, instead of 1.

index is equal to $(n \bmod p)$*. p is called the* ***period*** *of* $u(n)$*.*

$$u(n) = [u_0, u_1, \ldots, u_{p-1}]_n = \left\{ \begin{array}{ll} u_0 & \text{if } (n \bmod p) = 0 \\ u_1 & \text{if } (n \bmod p) = 1 \\ \vdots & \vdots \\ u_{p-1} & \text{if } (n \bmod p) = p-1 \end{array} \right\} = u_{(n \bmod p)}$$

Let $N = (n_1, n_2, \cdots, n_q)$ *be a q-dimensional parameter vector. A q-dimensional periodic number* $u(N)$ *is defined by a q-dimensional array U of size* $p_1 \times p_2 \times \ldots \times p_q$ *such that*

$$u(N) = u_{n_1, n_2, \cdots, n_q} = U_{(n_1 \bmod p_1, n_2 \bmod p_2, \cdots, n_q \bmod p_q)}$$

q is the ***dimension*** *of* $u(N)$*. The vector* $p = (p_1, p_2, \cdots, p_q)$ *is called the* ***multi-period*** *of* $u(N)$ *and the least common multiple of the* p_i*'s is called the* ***period*** *of* $u(N)$*.*

Example 2

$(-1)^n = [1, -1]_n$

$(-1)^{n-m} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{(n,m)} = [\,[1, -1]_m, [-1, 1]_m]_n$

Note that the matrix array notation works well for 2-dimensional periodic numbers, but the vector of vectors notation works for all dimensions.

□

Ehrhart [10] gave the basic properties of one-dimensional periodic numbers which can easily be extended to the multi-dimensional case:

Let $u(N)$ and $u'(N)$ be q-periodic numbers of multi-period p and p' respectively. They can be reduced to the same multi-period $m = (m_i)$, where $m_i = p_i\alpha = p'_i\beta$ is the least common multiple of p_i and p'_i, by repeating α times (respectively β times) the sequence of items in the i^{th} dimension of $u(N)$ (respectively $u'(N)$). Since $u(N)$ and $u'(N)$ are reduced to the same period, $u(N) + u'(N) = (U_I + U'_I)_N$ and $u(N)u'(N) = (U_I U'_I)_N$, where $I = (n_1 \bmod p_1, n_2 \bmod p_2, \cdots, n_q \bmod p_q)$.

Ehrhart defined pseudo-polynomials for a single variable n and 1-dimensional periodic coefficients.

Definition 5 (pseudo-polynomial)

A polynomial

$$f(n) = c_k n^k + c_{k-1} n^{k-1} + \cdots + c_0$$

is called a (1-dimensional) ***pseudo-polynomial*** *of degree k if its coefficients* c_i *are 1-dimensional periodic numbers, and* $f(n)$ *is integral for all n. We have extended this definition to cover multi-dimensional polynomials.*

Let $N = (n_1, n_2, \cdots, n_q)$ *be a q-dimensional parameter vector. A q-dimensional pseudo-polynomial* $f(N)$ *of degree k is defined as*

$$f(N) = \sum_{i_1=0}^{k} \sum_{i_2=0}^{k} \cdots \sum_{i_q=0}^{k} c_{i_1, i_2, \ldots, i_q} n_1^{i_1} n_2^{i_2} \ldots n_q^{i_q}$$

where the coefficients $c_{i_1,i_2,\ldots,i_q}$'s are periodic numbers whose dimensions are at most q and $f(N)$ is integral for all N.

A pseudo-polynomial has three characteristics: its dimension, degree, and its pseudo-period. The ***dimension*** *of the polynomial $F(N)$ is the number of parameter variables q, the* ***degree*** *of the $F(N)$ is the greatest exponent k, and the* ***pseudo-period*** *of $F(N)$ is the least common multiple of the periods of its coefficients c_i.*

Example 3

The summation $\sum_{i=0}^{n/2}\sum_{j=0}^{m/2}(1)$ can be formulated as a pseudo-polynomial of 2 variables, n and m, of degree 2, and period 2.

$$\begin{aligned}\sum_{i=0}^{n/2}\sum_{j=0}^{m/2}(1) &= \left(\tfrac{1}{4}m + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m\right) n + \left[\left(\tfrac{1}{2}m + \left[1, \tfrac{1}{2}\right]_m\right), \left(\tfrac{1}{4}m + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m\right)\right]_n \\ &= \tfrac{1}{4}nm + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m n + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_n m + \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/4 \end{bmatrix}_{n,m}\end{aligned}$$

□

3 Parametric enumerators with pseudo-polynomials

In this section, we recall and extend some results due to Ehrhart who made a study of the enumeration of lattice points in polyhedra by means of polynomials along the same lines as Euler. All the initial proofs can be found in [7, 8, 9, 10]. Some aspects of Ehrhart's work were corrected, streamlined, and expanded by I. G. Macdonald [18, 19] and R. P. Stanley [24, 25].

The **enumerator** of a normal polyhedral domain is the number of integer lattice points contained in the domain. We denote the enumerator of a parameterized polyhedron P_N as E_N.

Definition 6 (denominator of a polyhedron)
The ***denominator of a rational point*** *is the least common multiple of the denominators of its coordinates.*
The ***denominator of a rational polyhedron*** *is the least common multiple of the denominators of its vertices.*

Example 4

The denominator of a point $\left(\frac{1}{2}, \frac{5}{3}\right)$ is $lcm(2,3) = 6$.
The denominator of a polyhedron with vertices: $\left(\frac{1}{2}, \frac{5}{3}\right)$, $\left(\frac{3}{2}, \frac{9}{4}\right)$, and $\left(-\frac{1}{2}, \frac{1}{6}\right)$ is $lcm(6,4,6) = 12$
□

Ehrhart's main result consists in the following theorem dealing with parametric polyhedra which depend on a single parameter n.

Theorem 1 (Ehrhart's fundamental theorem)
The enumerator E_n of any homothetic or bordered k-polytope P_n is a polynomial in n of degree k if P_n is integral, and it is a pseudo-polynomial in n of degree k whose pseudo-period is the denominator of P_n if P_n is rational.

We extend this theorem to any number of parameters $N = (n_1, \ldots, n_p)$.

Theorem 2 (extended fundamental theorem)
Given any homothetic or bordered d-polytope $\mathcal{P}_N$ defined in terms of p linearly-independent parameters and having a denominator q, the enumerator E_N of such a polytope can be expressed as a p-variable pseudo-polynomial:

$$E(N) = \sum_{i_1=0}^{d} \sum_{i_2=0}^{d-i_1} \cdots \sum_{i_p=0}^{d-i_1-i_2-\cdots-i_{p-1}} c_{i_1,i_2,\ldots,i_p} n_1^{i_1} n_2^{i_2} \ldots n_p^{i_p}$$

whose terms have degree at most d, and whose the coefficients $c_{i_1,i_2,\ldots,i_p}$'s are periodic numbers with periods at most q.

Inasmuch as any polytope is composed of different homothetic or bordered polytopes over different subdomains of the parameters, the enumerator of any polytope can be expressed as different piecewise pseudo-polynomials defined over those same parameter subdomains.

The proof uses the following lemma stating that if a parametric polytope is defined by affine constraints, then the coordinates of its vertices are also affine functions of the parameters.

Lemma 1 (parametric vertices)
If P_N is a parametric polytope defined by affine constraints of the form $ax \leq bN + c$, then the parametric coordinates of its vertices are also affine functions of the parameters of the form $dN + e$, where e is a rational constant p-vector and e is a rational scalar constant.

Proof (Lemma 1)
Without loss of generality, let P_N be a k-polytope defined by a non-redundant system of constraints. Each vertex is a k-vector that can be defined as the intersection of at least k constraints which are saturated at the vertex. An inequality $Ax \leq BN + c$ is *saturated* when the equality $Ax = BN + c$ is true. The resolution of such a system of equations derived from k saturated constraints leads to a solution where each element of the x vector is expressed as an affine function of the parameters. We note that if the constant c is zero for all constraints, that all constraints are linear functions of N and all coordinates of vertices are likewise linear functions of N. Such a system is *homothetic*. Systems with non-zero c's are *bordered* and have affine constraint systems and affine coordinate vertices.
□

We note with interest that the reverse of lemma 1 is *not* always true. The convex hull of a set of parameterized vertices in a data space does not always generate a corresponding polyhedron in the combined data and parameter space. Here is a counter-example.

Example 5

Consider a line segment in 2-space with end points at $[1,1]$ and $[2,n]$, where n is a positive parameter. Then a point on the closed segment connecting $[1,1]$ and $[2,n]$ can be written as $[i,j] = [1+k(2-1), 1+k(n-1)]$, for $0 \le k \le 1$. Solving for k: $k = i-1 = \frac{j-1}{n-1}$ leading to $(j-1) = (i-1)(n-1)$ which shows that i, j, and n are not *linearly* related, and thus the set of points in i,j,n-space corresponding to points on the parametric closed segment between $[1,1]$ and $[2,n]$, is not a polyhedron. [Example due to Sanjay Rajopadhye.]
□

Proof (Theorem 2)
Part 1

We use an inductive proof to show that the enumerator of P_N is a pseudo-polynomial. We use as the induction variable, p, the number of parameters.

Base Case: Consider a bordered polytope P_n depending on one parameter n, $(p=1)$. The enumerator of P_n is a pseudo-polynomial by application of theorem 1.

Inductive Step: Assume that the enumerator of a bordered polytope $P_{n_1,\ldots,n_{p-1}}$ depending on $p-1$ parameters is a pseudo-polynomial.

Now, consider a parametric polytope $P_N = P_{n_1,\ldots,n_p}$ depending on p parameters. By fixing parameters $n_1,\ldots,n_{p-1}$ to constants, and leaving only n_p a free parameter, we obtain a bordered polytope $P_{n_p}(n_1,\ldots,n_{p-1})$. By application of theorem 1, the enumerator of $P_{n_p}(n_1,\ldots,n_{p-1})$ can be written:

$$E_{n_p}(n_1,\ldots,n_{p-1}) = c_d(n_1,\ldots,n_{p-1})n_p^d + c_{d-1}(n_1,\ldots,n_{p-1})n_p^{d-1} + \ldots + c_0(n_1,\ldots,n_{p-1})$$

By substituting n_p in the above equation with some initial values 1, 2, ...we get a system of linear equations of the form:

$$\left\{\begin{array}{lcl} c_d(n_1,\ldots,n_{p-1})_1 1^d + c_{d-1}(n_1,\ldots,n_{p-1})_1 1^{d-1} + \ldots + c_0(n_1,\ldots,n_{p-1})_1 & = & E_{n_1,\ldots,n_{p-1},1} \\ c_d(n_1,\ldots,n_{p-1})_2 2^d + c_{d-1}(n_1,\ldots,n_{p-1})_2 2^{d-1} + \ldots + c_0(n_1,\ldots,n_{p-1})_2 & = & E_{n_1,\ldots,n_{p-1},2} \\ \vdots & & \vdots \end{array}\right.$$

The enumerators $E_{n_1,\ldots,n_{p-1},1}, E_{n_1,\ldots,n_{p-1},2},\ldots$ are pseudo-polynomial by the inductive assumption. Enough initial values for n_p can be chosen so that a solution for the coefficients $c_i(n_1,\ldots,n_{p-1}), d \ge i \ge 0$ can be derived. Each coefficient $c_i(n_1,\ldots,n_{p-1})$ will be in fact a linear combination of the pseudo-polynomials $E_{n_1,\ldots,n_{p-1},1}, E_{n_1,\ldots,n_{p-1},2},\ldots$ and therefore each coefficient $c_i(n_1,\ldots,n_{p-1})$ will itself be a pseudo-polynomial. And since a pseudo-polynomial with pseudo-polynomial coefficients is itself a pseudo-polynomial, E_N is also a pseudo-polynomial. This completes part 1 of the proof.

Part 2

In this part, we show that no term of E_N has a degree greater than d. $\mathcal{P}_N$ has a finite number of vertices $v_k, 1 \le k \le r$, whose indices $v_k[i], 1 \le i \le d$ are affine functions of the parameters, $n_1, n_2, \cdots, n_p$ (by Lemma 1). The rectangular bounding box of $\mathcal{P}$ has an enumeration of

$$\begin{array}{c} (\max_{(1\le k\le r)} v_k[1] - \min_{(1\le k\le r)} v_k[1] + 1)\times \\ (\max_{(1\le k\le r)} v_k[2] - \min_{(1\le k\le r)} v_k[2] + 1)\times \\ \vdots \\ (\max_{(1\le k\le r)} v_k[d] - \min_{(1\le k\le r)} v_k[d] + 1) \end{array}$$

Parameterized Polyhedron	Pseudo-Polynomial Enumerator
number of parameters	number of variables (dimension)
dimension	degree
denominator	period of coefficients
parameter subdomain for stable vertices	parameter subdomain for formula

Figure 1: Relationship between a parametric polynomial and its enumerator

Since each term $v_k[i]$ is an affine function of the parameters, the term $\max_{(1\leq k\leq r)} v_k[d] - \min_{(1\leq k\leq r)} v_k[d] + 1$. will also be an affine function of the parameters. The product of d affine functions of the parameters

$$\begin{array}{c}(a_{10} + a_{11}n_1 + a_{12}n_2 + \cdots + a_{1p}n_p)\times \\ (a_{20} + a_{21}n_1 + a_{22}n_2 + \cdots + a_{2p}n_p)\times \\ \vdots \\ (a_{d0} + a_{d1}n_1 + a_{d2}n_2 + \cdots + a_{dp}n_p)\end{array}$$

is a polynomial

$$\sum_{i_1=0}^{d}\sum_{i_2=0}^{d-i_1}\cdots\sum_{i_p=0}^{d-i_1-i_2-\cdots-i_{p-1}} c_{i_1,i_2,\ldots,i_p} n_1^{i_1} n_2^{i_2}\ldots n_p^{i_p}$$

The actual enumeration of $\mathcal{P}$ is always bounded by the enumeration of the bounding box of $\mathcal{P}$. If the pseudo-polynomial $E(N)$ that was the actual enumerator of $\mathcal{P}$ had a term whose order of complexity exceeded any term in the enumerator $E_{BB}(N)$ of the bounding box of $\mathcal{P}$, that is, if there existed some term $c_{i_1,i_2,\ldots,i_p} n_1^{i_1} n_2^{i_2}\ldots n_p^{i_p}$ in $E(N)$ where $i_1 + i_2 + \cdots + i_p > d$, then $O(E(N)) > O(E_{BB}(N))$ and there would exist some value of $N = (n_1, n_2, \cdots, n_p)$ such that $E(N) > E_{BB}(N)$. This leads to a contradiction, and the theorem follows.
$\square$

Due to these theorems, the form of the parametric enumeration formula is known. The relationship between a parameterized polyhedron P_N and its pseudo-polynomial enumerator is summarized in figure 1. The number of variables in (or dimension of) the pseudo-polynomial is equal to the number of (linearly independent) parameters defining P_N. The period of the pseudo-polynomial is equal to the denominator of the polyhedron P_N and the degree of the pseudo-polynomial is equal to the dimension of the polyhedron. And finally, the parameter subdomains over which different pseudo-polynomial are valid, are related to the subdomains where the vertex set of P_N doesn't change (or is stable). Hence, the derivation of the Ehrhart pseudo-polynomial can be done by counting some initial values of the enumerator and solving systems of symbolic rational linear equations. The number of unknown coefficients is of a pseudo-polynomial is $(degree + 1)period$ for a one-dimensional polynomial and $((degree + 1)period)^{dimension}$ for a multi-dimensional polyhedron. Since the number of equations of the system must be equal to the number of unknown values, the complexity depends on the number of parameters, the number of validity domains, the degree, and the denominator.

The number of unknowns can be reduced using the following theorem, conjectured by Ehrhart [10] and proved by Stanley [25].

Theorem 3 *Let P_n be a rational k-polytope and let the Ehrhart pseudo-polynomial be*

$$E_n = c_k(n)n^k + c_{k-1}(n)n^{k-1} + \ldots + c_0(n)$$

where $c_0, \ldots, c_k$ are periodic numbers of n. If for some $q \in [0, k]$, the affine span of every q-dimensional face of P_1 contains at least one lattice point, then for any $p \geq q$, $c_p(n)$ is constant, i.e. of period one.

Example 6

Consider the set of linear constraints $\left\{0 \leq i \leq \frac{1}{2}n, 0 \leq j \leq \frac{1}{2}m\right\}$ defining the rational 2-polytope $P_{n,m}$,

$$P_{n,m} = \{\left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \in \mathcal{Z}^2 \mid \left(\begin{smallmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{smallmatrix}\right)\left(\begin{smallmatrix} i \\ j \end{smallmatrix}\right) \leq \left(\begin{smallmatrix} 0 & 0 \\ \frac{1}{2} & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \end{smallmatrix}\right)\left(\begin{smallmatrix} n \\ m \end{smallmatrix}\right)\}$$

$P_{n,m}$ is the closed rectangle whose parametric vertices are $(0, 0)$, $(\frac{n}{2}, 0)$, $(0, \frac{m}{2})$, $(\frac{n}{2}, \frac{m}{2})$. The denominator of this polyhedron is 2, it is 2-dimensional and is a function of 2 parameters.

By looking at $P_{1,0}$, we observe that one vertex, $(\frac{1}{2}, 0)$, is not an integer point. Hence, the coefficient of n^0 is periodic. The other coefficients are constant, since all the edges, and obviously the polytope itself, contain at least one integer point.

By looking at $P_{0,1}$, we observe that one vertex, $(0, \frac{1}{2})$, is not an integer point. Hence, the coefficient of m^0 is periodic. The other coefficients are constant, since all the edges, and obviously the polytope itself, contain at least one integer point.

Moreover, the period of the periodic coefficients is equal to the denominator of $P_{n,m}$, i.e. 2. The pseudo-polynomial $E_{n,m} = c_1n^2 + c_2n + [c_3, c_4]$ can thus be determined by solving the following system of rational linear equations:

$$\begin{cases} c_3 & = E_{0,m} \\ c_1 + c_2 + c_4 & = E_{1,m} \\ 4c_1 + 2c_3 + c_3 & = E_{2,m} \\ 9c_1 + 3c_2 + c_4 & = E_{3,m} \end{cases}$$

Before we can solve this system, the values of $E_{0,m}$, $E_{1,m}$, $E_{2,m}$ and $E_{3,m}$ must be computed first.

We compute $E_{0,m} = c_5m^2 + c_6m + [c_7, c_8]_m$:

$$\begin{cases} c_7 & = E_{0,0} = 1 \\ c_5 + c_6 + c_8 & = E_{0,1} = 1 \\ 4c_5 + 2c_6 + c_7 & = E_{0,2} = 2 \\ 9c_5 + 3c_6 + c_8 & = E_{0,3} = 2 \end{cases}$$

We obtain $E_{0,m} = 1, 1, 2, 2$ by counting points in $P_{0,0}$ to $P_{0,3}$ and then solve for $c_5 \cdots c_8$ to obtain $E_{0,m} = \frac{1}{2}m + [1, \frac{1}{2}]_m$.

In a likewise manner we set up systems of equations for $E_{1,m}$, $E_{2,m}$, and $E_{3,m}$, which when solved, give us $E_{1,m} = \frac{1}{2}m + [1, \frac{1}{2}]_m$, $E_{2,m} = m + [2, 1]_m$, and $E_{3,m} = m + [2, 1]_m$.

Finally, we can symbolically solve the first set of equations to obtain:

$$\begin{aligned} E_{n,m} &= \left(\tfrac{1}{4}m + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m\right) n + \left[\tfrac{1}{2}m + \left[1, \tfrac{1}{2}\right]_m, \ \tfrac{1}{4}m + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m\right]_n \\ &= \tfrac{1}{4}nm + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_m n + \left[\tfrac{1}{2}, \tfrac{1}{4}\right]_n m + \left[\begin{array}{cc} 1 & 1/2 \\ 1/2 & 1/4 \end{array}\right]_{n,m} \end{aligned}$$

□

4 Description of the algorithm

To use Ehrhart's results to specify a pseudo-polynomial, we need four pieces of information:

1. the number of parameters
2. the dimension of the system
3. the denominator of system
4. the validity domain over which the polynomial is valid

The number of parameters and the dimension of the system are immediately available from the problem specification. To find the denominator and validity domains of the system, the vertices of the polyhedron P_n must first be computed. We use the algorithm of Loechner and Wilde [17] to determine the parametric coordinates of the vertices of any convex polytope and their associated definition domains. This algorithm is associated with a software package called the *polyhedral library* [28] which eliminates redundant constraints and performs logical operations on polyhedra.

In some problems, non-convex polytopes might have to be considered. In such a case, the Loechner/Wilde algorithm will have to be extended in order to consider Ehrhart's *normal* polyhedron, consisting of a union of convex polytopes $P_N^1, P_N^2, \ldots$ which can only intersect at their proper faces. This proposed extension consists of the following heuristic procedure:

- Compute all the vertices of each rational polytope P_N^i.
- Compute all the vertices of each intersection $P_N^i \cap P_N^j, j \neq i$.

4.1 An algorithm to find parametric vertices

The method presented below is fully explained in [17]. It computes the vertices $v_i(N)$ of a parametric polyhedron $P_N = \{x \in \mathcal{Q}^d \mid Ax \leq BN + b;\ Cx = DN + d\}$, where N is a vector of the form $\begin{pmatrix} n_1 \\ \vdots \\ n_p \end{pmatrix}$. If assume that the different n_j are linearly independent, then the following theorem can be shown:

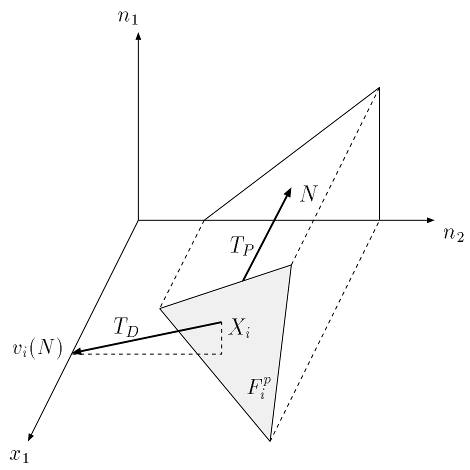


Figure 2: A p-face F_i^p (highlighted) and the computation of $v_i(N)$ as a function of N

Theorem 4 *Let $v_i(N)$ be a vertex of the polyhedron P_N. Let P' be a polyhedron in the combined data and parameter space corresponding to P_N. Then for each vertex $v_i(N)$ of P_N, there exists a unique corresponding p-face of P' denoted F_i^p such that $\begin{pmatrix} v_i(N) \\ N \end{pmatrix} = F_i^p \cap \left\{ \begin{pmatrix} x \\ n \end{pmatrix} \mid n = N \right\}$.*

Proof The proof can be found in [17]. It is based on the definition of $v_i(N)$ as a 0-face of P_N; $v_i(N)$ is the intersection of a supporting hyperplane $\mathcal{H}_i$ and P_N. In the combined space, $\mathcal{H}_i$ is the intersection of $\left\{ \begin{pmatrix} x \\ n \end{pmatrix} \mid n = N \right\}$ and a supporting hyperplane of P' denoted $\mathcal{H}'_i$. $\mathcal{H}'_i$ can be chosen such that $\mathcal{H}'_i \cap P'$ is a p-face of P' and the result follows.
□

Property 1 *The point $v_i(N)$ is only defined for values of N in the polyhedron $\left\{ N \mid \begin{pmatrix} v_i(N) \\ N \end{pmatrix} \in F_i^p \right\}$, which is a projection of F_i^p onto the parameter space.*

Proof Refer to figure 2. Since $\begin{pmatrix} v_i(N) \\ N \end{pmatrix} = F_i^p \cap \left\{ \begin{pmatrix} x \\ n \end{pmatrix} \mid n = N \right\}$, the validity domain for $v_i(N)$ (the range of N over which $v_i(N)$ exists) is $\left\{ N \mid \begin{pmatrix} v_i(N) \\ N \end{pmatrix} \in F_i^p \right\}$. It is a convex polyhedron since it is a projection of F_i^p and F_i^p is a convex polyhedron.
□

If F_i^p is a p-face of P' corresponding to a vertex $v_i(N)$, then all points $\begin{pmatrix} v_i(N) \\ N \end{pmatrix} \in F_i^p$ lie in the intersection of $d = \text{dimension}(P') - p$ facets of P' and thus must satisfy d linearly independent equations of the form $a^T \begin{pmatrix} v_i(N) \\ N \end{pmatrix} = c$, where a is a constant vector and c is a constant. Using these equations, we can solve for $v_i(N)$ as a vector of linear functions of N.

If such a system of equations cannot be solved, then there does not exist one-to-one correspondence between points in F_i^p and values of N. In this case, the p-face F_i^p not correspond to a vertex of P_N, and that face F_i^p can be ignored by the algorithm.

4.1.1 Partitioning the parameter space

The output of the above algorithm is a set of vertices and for each vertex, a set of constraints that must be verified by the parameters N for that vertex to be valid.

In order to use these results, we need to know the set of distinct parameter subdomains which partition complete *sets* of parameterized vertices, i.e. to transform the collection of (subdomain, vertex) pairs into a collection of (subdomain, set of vertices) pairs, where each subdomain (specified by set of conditions on N) is a partition in which a complete set of vertices of P_N is stable. These partitions are obtained by intersecting the individual domains of all of the vertices in each set.

Ehrhart's theorem (Theorem 1) and our extension to it (Theorem 2) can only be applied to bordered or homothetic polyhedra. These partitions are essential for the application of these theorems because they separate a parameterized polyhedron into a set of bordered and homothetic polyhedra. For each bordered or homothetic polyhedron, a different pseudo-polynomial enumerator is computed.

4.2 Computing periodic coefficients

A pseudo polynomial of degree $p-1$ and period q is written:

$$E_n = [u_{p-1,q-1}, \cdots u_{p-1,0}]_n\, n^{p-1} + [u_{p-2,q-1}, \cdots u_{p-2,0}]_n\, n^{p-2} + \cdots + [u_{0,q-1}, \cdots u_{0,0}]_n\, n^0$$

The values $u_{j,k}$ (where $0 \leq j < p$ and $0 \leq k < q$) are unknowns. To solve for them, we substitute in different values of n to generate different equations.

Normally, a system of pq equations would need to be solved to find pq unknown coefficients taking $\mathcal{O}(p^3n^3)$ time. However, in this case, we can break the problem down and solve p sets of q-equations, which is done in $\mathcal{O}(pn^3)$ time.

Let $n = n_0$ and $k = (n_0 \bmod q)$. Then substituting n for n_0 in the pseudo-polynomial, we get:

$$u_{p-1,k}n_0^{p-1} + u_{p-2,k}n_0^{p-2} + \cdots + u_{0,k} = E(n_0)$$

If we choose a set of q values for n starting at n_0 and then increasing by strides of q, i.e., $n_i = n_0 + iq$, $0 \leq i < q$, and then substitute those values in the pseudo-polynomial, we arrive at the following system of equations.

$$u_{p-1,k}n_0^{p-1} + u_{p-2,k}n_0^{p-2} + \cdots + u_{0,k} \;=\; E(n_0)$$

$$\begin{aligned} u_{p-1,k}n_1^{p-1} + u_{p-2,k}n_1^{p-2} + \cdots + u_{0,k} &= E(n_1) \\ &\vdots \\ u_{p-1,k}n_{q-1}^{p-1} + u_{p-2,k}n_{q-1}^{p-2} + \cdots + u_{0,k} &= E(n_{q-1}) \end{aligned}$$

By striding the value of n by q, the index $k = (n \bmod q)$ remains constant and the same set of u_{jk} values are selected from the periodic coefficients for each equation. This system of equations can be put into matrix format $A\vec{u} = \vec{E}$ as follows:

$$\begin{bmatrix} n_0^{p-1} & n_0^{p-2} & \cdots & 1 \\ n_1^{p-1} & n_1^{p-2} & \cdots & 1 \\ & \vdots & & \\ n_{q-1}^{p-1} & n_{q-1}^{p-2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{p-1,k} \\ u_{p-2,k} \\ \vdots \\ u_{0,k} \end{bmatrix} = \begin{bmatrix} E(n_0) \\ E(n_1) \\ \vdots \\ E(n_{q-1}) \end{bmatrix}$$

Matrix A will always be an integer matrix and the inverse of A will always be a rational matrix. By inverting matrix A, we can solve for vector $\vec{u}$, obtaining $\vec{u} = A^{-1}\vec{E}$. Two cases arise:

1. The $E(n_j)$'s are constant. They are obtained by counting the number of integer points in the polyhedron $P(n_j)$ (n_j being a fixed integer). This is done by scanning the polyhedron using the algorithm described in [16].

2. The $E(n_j)$'s are not constant. This happens when the enumerator E is a function of more than one parameter, $E(n, m, \cdots)$. In this case, the enumerator $E(n_j, m, \cdots)$ is computed by setting parameter n to n_j and recursively calling the procedure to compute the formula for $E(m, \cdots) = E(n, m, \cdots) \mid_{n=n_0}$. This recursion will eventually terminate in case 1 when all parameters have been fixed. The recurrent procedure returns a pseudo-polynomial formula and thus the resulting $\vec{E}$ is a vector of pseudo-polynomials. The matrix-vector product $A^{-1}\vec{E}$ must therefore be done symbolically and procedures to perform symbolic multiplication and addition of pseudo-polynomials are used to compute $u_{j,k}$.

This entire procedure is then repeated for other sets of values for n, that have different values of $k = (n \bmod q)$, until all of the u_{jk} have been solved.

4.3 Simplifying pseudo-polynomials

The pseudo-polynomial returned by the above procedure is not usually in simplest form. A procedure called *Reduce* is called to reduce the pseudo-polynomial E. It is defined as follows:

Reduce(E)

1. If E is not a simple rational number, (it is a polynomial of degree $p-1$, or a periodic number of period p), then for $0 \leq j < p$ recursively call Reduce($E.u_j$).

2. If E is a periodic number ($E = [u_{p-1}, \cdots, u_0]$), then
   (a) Try to reduce the period of E.
   The period of E can be reduced from p to q if
   $$\exists\, q \;:\; 1 \leq q < p;\; \frac{p}{q} \in \mathcal{Z} \left(\forall\, j, c \;:\; 0 \leq j < q;\; 0 \leq c < \frac{p}{q}\;\; (u_k = u_{cq+k})\right)$$
   (b) Try to reduce the strength of E.
   If the period of E is one ($E = [u_0]$), reduce it from a periodic number to a simple (non-periodic) rational number u_0.
3. If E is a polynomial ($E = u_{p-1}x^{p-1} + u_{p-2}x^{p-2} + \cdots + u_0 x^0$), then
   (a) Try to reduce the degree of E.
   The degree of E can be reduced from $p-1$ to $q-1$ if
   $$\exists\, q \;:\; 1 \leq q < p\,(\forall\, j \;:\; q \leq j < p\;\; (u_j = 0))$$
   (b) Try to reduce the strength of E.
   If the degree of E is zero ($E = u_0 x^0$), reduce it from a polynomial to a simple rational number u_0.

Example 7

$$\left[\frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}\right]_n n^2 + \left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right]_n n + \left[0, -\frac{1}{4}, 0, -\frac{1}{4}, 0, -\frac{1}{4}\right]_n$$

is reduced to

$$\frac{3}{4}n^2 + \frac{1}{2}n + \left[0, -\frac{1}{4}\right]_n$$

□

5 Examples and related work

All the examples considered in this section come from several related works [13, 14, 27, 26, 11, 21] and were also handled by W. Pugh in [21], except example 13 whose aim is to show the generality of our method.

Example 8

M. Haghighat and C. Polychronopoulos present a method for volume computation in [13, 14]. Their first example is $\sum_{i=1}^{n} \sum_{j=3}^{i} \sum_{k=j}^{5} 1$. This sum defines a set of linear constraints $P_N = \{1 \leq i \leq n, 3 \leq j \leq i, j \leq k \leq 5\}$. The Loechner/Wilde algorithm computes the following vertices for the polytope P_n:

- (1) if $3 \leq n \leq 5$, the vertices are $(3,3,5)$, $(3,3,3)$, $(n,3,5)$, $(n,3,3)$, $(n,n,5)$, (n,n,n).
- (2) if $n > 5$, the vertices are $(3,3,5)$, $(3,3,3)$, $(n,3,5)$, $(n,3,3)$, $(5,5,5)$, $(n,5,5)$.

Hence, P_n is an integral 3-polytope and the associated pseudo-polynomial $E_n = c_1n^3 + c_2n^2 + c_3n + c_4$ is computed for each of the two domains.
Domain 1: From $n = 3$ to 5 we count $E_n = 3, 8, 14$. Since we have only three equations and four unknowns, we can arbitrarily set c_1 to zero, reducing the degree of the polynomial enumerator to 2. The remaining coefficients are solved for using the following system of equations:

$$(1)\left\{\begin{array}{ll} c_1 & = 0 \\ 9c_2 + 3c_3 + c_4 & = E_3 = 3 \\ 16c_2 + 4c_3 + c_4 & = E_4 = 8 \\ 25c_2 + 5c_3 + c_4 & = E_5 = 14 \end{array}\right.$$

The solution of this system results in $E_n = \frac{1}{2}n^2 + \frac{3}{2}n - 6$.

Our result is different from that computed by Pugh [21], because Pugh only used two points $n = 3$ to 4 where as our algorithm used $n = 3$ to 5. Both are correct over their respective ranges of n.
Domain 2: From $n = 5$ to 8 we count $E_n = 14, 20, 26, 32$.

$$(2)\left\{\begin{array}{ll} 125c_1 + 25c_2 + 5c_3 + c_4 & = E_5 = 16 \\ 216c_1 + 36c_2 + 6c_3 + c_4 & = E_6 = 20 \\ 343c_1 + 49c_2 + 7c_3 + c_4 & = E_7 = 26 \\ 512c_1 + 64c_2 + 8c_3 + c_4 & = E_8 = 32 \end{array}\right.$$

The solution of this system results in $E_n = 6n - 16$.
Summary The enumerator is thus

$$E_n = \left\{\begin{array}{ll} \frac{1}{2}n^2 + \frac{3}{2}n - 6 & \text{if} \quad 3 \leq n \leq 5 \\ 6n - 16 & \text{if} \quad 5 \leq n \end{array}\right.$$

This is equivalent to the result derived by W. Pugh. On the other hand, M. Haghighat and C. Polychronopoulos derived the enumerator:

$$\mu(\min(n-2,3))(-(\min(n,5))^3 + 15(\min(n,5))^2 - 38\min(n,5) + 24)/6 + 6\max(n-5,0)$$

where $\mu(x)$ is defined to be 1 if x is positive and 0 otherwise. The enumerator evaluates to the same values as ours, but the form is quite different, and more complicated, because of the $\mu(x)$, *min* and *max* functions they employ. Our method never generates $\mu(x)$, *min* or *max* functions.
□

Example 9

The second example in [13, 14] is

$$\sum_{i=1}^{2n} \sum_{j=1}^{min(i,2n-i)} 1$$

This sum defines a set of linear constraints $P_n = \{1 \le i \le 2n, 1 \le j \le i, i+j \le 2n\}$. The Loechner/Wilde algorithm computes the following vertices for the polytope P_n:

- if $n \ge 1$, the vertices are $(n,n), (2n-1,1), (1,1)$.

Hence, P_n is an integral 2-polytope and the associated pseudo-polynomial enumerator $E_n = c_1 n^2 + c_2 n + c_3$ is formulated.

From $n = 1$ to 3 we count $E_n = 1, 4, 9$. We solve for the unknown coefficients using the following system of equations:

$$\begin{cases} c_1 + c_2 + c_3 & = E_1 = 1 \\ 4c_1 + 2c_2 + c_3 & = E_2 = 4 \\ 9c_1 + 3c_2 + c_3 & = E_3 = 9 \end{cases}$$

The solution of this system results in $E_n = n^2$, which is the same result as W. Pugh. In [13, 14], Haghighat and Polychronopoulos do not describe their technique in detail. They define a number of rules in order to transform expressions, but nothing is said on how to decide which rule to apply and when. As in [27], they assume the summation must be performed in a predetermined order.

□

Example 10

We now handle Pugh's "more elaborate" example of [21], and show that our method does not encounter any difficulty whatsoever, while Pugh had to introduce additional techniques which were not elaborated in his paper. This example is the set of linear constraints $P_n = \{1 \le i;\ j \le n;\ 2i \le 3j\}$.

The Loechner/Wilde algorithm computes the following vertices for the so-defined polytope P_n:

- if $n \ge \frac{2}{3}$, the vertices are $(\frac{3n}{2}, n), (1, \frac{2}{3}), (1, n)$

Hence, P_n is a rational 2-polytope with a denominator of 6. The associated enumerator is a pseudo-polynomial of degree 2 and period 6:

$$E_n = [c_1, c_2, c_3, c_4, c_5, c_6]n^2 + [c_7, c_8, c_9, c_{10}, c_{11}, c_{12}]n + [c_{13}, c_{14}, c_{15}, c_{16}, c_{17}, c_{18}]$$

From $n = 1$ to 18 we count E_n = 1, 4, 8, 14, 21, 30, 40, 52, 65, 80, 96, 114, 133, 154, 176, 200, 225, 252. The 18 equations are solved as 6 systems of 3 equations a piece:

$$\begin{cases} c_2 + c_8 + c_{14} & = E_1 = 1 \\ 49c_2 + 7c_8 + c_{14} & = E_7 = 40 \\ 169c_2 + 13c_8 + c_{14} & = E_{13} = 133 \end{cases} \quad \begin{cases} 4c_3 + 2c_9 + c_{15} & = E_2 = 4 \\ 64c_3 + 8c_9 + c_{15} & = E_8 = 52 \\ 196c_3 + 14c_9 + c_{15} & = E_{14} = 154 \end{cases}$$

$$\begin{cases} 9c_4 + 3c_{10} + c_{16} & = E_3 = 8 \\ 81c_4 + 9c_{10} + c_{16} & = E_9 = 65 \\ 225c_4 + 15c_{10} + c_{16} & = E_{15} = 176 \end{cases} \quad \begin{cases} 16c_5 + 4c_{11} + c_{17} & = E_4 = 14 \\ 100c_5 + 10c_{11} + c_{17} & = E_{10} = 80 \\ 256c_5 + 16c_{11} + c_{17} & = E_{16} = 200 \end{cases}$$

$$\begin{cases} 25c_6 + 5c_{12} + c_{18} & = E_5 = 21 \\ 121c_6 + 11c_{12} + c_{18} & = E_{11} = 96 \\ 289c_6 + 17c_{12} + c_{18} & = E_{17} = 225 \end{cases} \quad \begin{cases} 36c_1 + 6c_7 + c_{13} & = E_6 = 30 \\ 144c_1 + 12c_7 + c_{13} & = E_{12} = 114 \\ 324c_1 + 18c_7 + c_{13} & = E_{18} = 252 \end{cases}$$

The solution of these systems results in

$$E_n = \left[\frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}\right]_n n^2 + \left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right]_n n + \left[-\frac{1}{4}, 0, -\frac{1}{4}, 0, -\frac{1}{4}, 0\right]_n$$

which can be directly simplified to

$$E_n = \frac{3}{4}n^2 + \frac{1}{2}n + \left[-\frac{1}{4}, 0\right]_n$$

Pugh [21] uses a different technique to derive an enumerator of

$$\begin{cases} \frac{(n+n \ mod \ 2)(3n+3(n \ mod \ 2)-2)}{8} & if \ 1 \leq n \\ \frac{3(n-n \ mod \ 2)(n-n \ mod \ 2+2)}{8} & if \ 2 \leq n \end{cases}$$

This expression is then intuitively simplified to $\frac{3n^2+2n-n \ mod \ 2}{4}$ which evaluates to the same values as our pseudo-polynomial enumerator. This example shows that Pugh's technique tended to complicate answers when considering rational polytopes, and hence, does not allow to consider systematically any set of rational linear constraints. Our method has no difficulty handling any set of rational linear constraints without any restriction.

The method described by Pugh in [21] consists of a set of tools and techniques dedicated to simplifying the initial set of constraints. Applying all of these tools could be expensive for large sets of constraints with rational coefficients. Moreover, their application is often rule driven, in some cases involving the addition of auxiliary variables and the use of standard formulas for sums of powers. (Pugh expects it will be sufficient to hard code the formulas for powers up to 10.) And finally, the resulting formula involves the use of modulo functions. This shows some of the limitations in the general use of Pugh's method.

□

6 Application to the analysis of nested-loop programs

In this section, we give several examples of how the method is applied to applications which need to analyze nested-loops.

Example 11

An example in [11] asks to calculate the number of memory locations touched in a Successive Over-Relaxation (SOR) code:

```
for i := 2 to n-1 do
  for j := 2 to n-1 do
      a(i,j) = (2*a(i,j) + a(i-1,j) + a(i+1,j)
              + a(i,j-1) + a(i,j+1)) / 6
```

The elements of a touched by this loop are

$$\begin{array}{ll} & \{(i,j)|2 \le i,j \le n-1\} \\ \cup & \{(i-1,j)|2 \le i,j \le n-1\} \\ \cup & \{(i+1,j)|2 \le i,j \le n-1\} \\ \cup & \{(i,j-1)|2 \le i,j \le n-1\} \\ \cup & \{(i,j+1)|2 \le i,j \le n-1\} \end{array} = \begin{array}{ll} & \{(i,j)|2 \le i,j \le n-1\} \\ \cup & \{(i,j)|1 \le i \le n-2, 2 \le j \le n-1\} \\ \cup & \{(i,j)|3 \le i \le n, 2 \le j \le n-1\} \\ \cup & \{(i,j)|2 \le i \le n-1, 1 \le j \le n-2\} \\ \cup & \{(i,j)|2 \le i \le n-1, 3 \le j \le n\} \end{array}$$

The parametric vertices of this union of polytopes P_n are computed by using the heuristic procedure described in subsection 4.1. It gives the following results: $(1,2)$, $(2,1)$, $(1,n-1)$, $(2,n)$, $(n-1,n)$, $(n,n-1)$, $(n-1,1)$ and $(n,2)$. Hence, it is an integral 2-polytope and the associated Ehrhart polynomial is $E_n = c_1 n^2 + c_2 n + c_3$.

From $n = 4$ to 6 we count $E_n = 12, 21, 32$.

$$\begin{cases} 16c_1 + 4c_2 + c_3 & = E_4 = 12 \\ 25c_1 + 5c_2 + c_3 & = E_5 = 21 \\ 36c_1 + 6c_2 + c_3 & = E_6 = 32 \end{cases}$$

The solution of this system gives the result $E_n = n^2 - 4$ which is the same answer that Pugh gave.

□

Example 12

An extension of our method to several size parameters has been implemented. It is applied to an example handled by Tawbi in [26] which is $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{m} 1$. This sum defines a set of linear constraints $\{1 \le i \le n, 1 \le j \le i, j \le k \le m\}$. The Loechner/Wilde algorithm computes the following vertices for the so-defined polytope $P_{n,m}$:

- (1) if $1 \le n \le m$, the vertices are $(1,1,m)$, $(1,1,1)$, $(n,1,m)$, $(n,1,1)$, (n,n,m), (n,n,n).
- (2) if $1 \le m \le n$, the vertices are $(1,1,m)$, $(1,1,1)$, $(n,1,m)$, $(n,1,1)$, (m,m,m), (n,m,m).

Hence, $P_{n,m}$ is an integral 3-polytope and the associated pseudo-polynomial is of the form $E_{n,m} = \sum_{i=0}^{3} \sum_{j=0}^{3-i} c_{ij} n^i m^j$ and has to be computed for each of the two domains, (1) and (2).

Domain 1: Our method, applied to several size parameters, uses a symbolic resolution by first considering the polynomial $E_{n,m} = d_3(m)n^3 + d_2(m)n^2 + d_1(m)n + d_0(m)$ where:

$$\begin{aligned} d_3(m) &= c_{33}m^3 + c_{32}m^2 + c_{31}m + c_{30} \\ d_2(m) &= c_{23}m^3 + c_{22}m^2 + c_{21}m + c_{20} \\ d_1(m) &= c_{13}m^3 + c_{12}m^2 + c_{11}m + c_{10} \\ d_0(m) &= c_{03}m^3 + c_{02}m^2 + c_{01}m + c_{00} \end{aligned}$$

Symbolically evaluating $E_{n,m}$ at $n = 1,2,3,4$, yields the following symbolic system of equations.

$$\begin{cases} d_3(m) + d_2(m) + d_1(m) + d_0(m) & = E_{1,m} \\ 8d_3(m) + 4d_2(m) + 2d_1(m) + d_0(m) & = E_{2,m} \\ 27d_3(m) + 9d_2(m) + 3d_1(m) + d_0(m) & = E_{2,m} \\ 64d_3(m) + 16d_2(m) + 4d_1(m) + d_0(m) & = E_{4,m} \end{cases}$$

In order to solve for the symbolic coefficients $d_3(m)$, $d_2(m)$, $d_1(m)$, and $d_0(m)$, the enumerators $E_{1,m}$, $E_{2,m}$, $E_{3,m}$ and $E_{4,m}$ have to be computed.

For $n = 1$ and from $m = 1$ to 4, we count $E_{n,m} = 1, 2, 3, 4$.

$$\begin{cases} c_1 + c_2 + c_3 + c_4 & = E_{1,1} = 1 \\ 8c_1 + 4c_2 + 2c_3 + c_4 & = E_{1,2} = 2 \\ 27c_1 + 9c_2 + 3c_3 + c_4 & = E_{1,3} = 3 \\ 64c_1 + 16c_2 + 4c_3 + c_4 & = E_{1,4} = 4 \end{cases}$$

This system results in $E_{1,m} = m$. In the same way, we compute $E_{2,m} = 3m - 1$, $E_{3,m} = 6m - 4$ and $E_{4,m} = 10m - 10$, giving the equations

$$\begin{cases} d_3(m) + d_2(m) + d_1(m) + d_0(m) & = m \\ 8d_3(m) + 4d_2(m) + 2d_1(m) + d_0(m) & = 3m - 1 \\ 27d_3(m) + 9d_2(m) + 3d_1(m) + d_0(m) & = 6m - 4 \\ 64d_3(m) + 16d_2(m) + 4d_1(m) + d_0(m) & = 10m - 10 \end{cases}$$

The expressions for $d_1(m), d_2(m), d_3(m)$ and $d_4(m)$ can be symbolically computed as: $d_1 = -\frac{1}{6}$, $d_2 = \frac{m}{2}$, $d_3 = \frac{m}{2} + \frac{1}{6}$ and $d_4 = 0$. Thus, for domain $1 \leq n \leq m$, the enumerator $E_{n,m}$ is derived as:

$$\begin{aligned} E_{n,m} &= -\tfrac{1}{6}n^3 + \tfrac{m}{2}n^2 + (\tfrac{1}{2}m + \tfrac{1}{6})n + 0 \\ &= -\tfrac{n^3}{6} + \tfrac{n^2 m}{2} + \tfrac{nm}{2} + \tfrac{n}{6} \end{aligned}$$

Domain 2: The same work is done for domain (2), i.e. $1 \leq m \leq n$, resulting in $E_{n,m} = -\frac{m^3}{6} + \frac{m^2 n}{2} + \frac{mn}{2} + \frac{m}{6}$, which matches the result of Pugh. Our method can be applied the same way when considering several size parameters in the case of rational polytopes, case which is not systematically handled by Pugh's techniques. Tawbi in [27, 26] first used a polyhedral splitting technique to characterize the definition domains, and an elimination of the variables in a predetermined order. However, her method does not allow systematically computing exact formulae like our method.

□

Example 13

We propose a more complex example than any previously published, requiring the use of the entire method we have described. This example involves rational parameterized polytopes depending on three size parameters. Let us consider the following loop nest:

```
for i := 0 to n do
  for j := 0 to i + m/2 do
    for k := 0 to i - n + p do
      a(i,j,k) = a(i,j,k-1) + a(k,j,k)*a(i,k,k)
```

We want to compute the number of flops in order to evaluate the execution time of this code segment. This loop nest is modeled by the convex polytope $P_N = \{z \in \mathcal{Z}^3 \mid Az \leq$

$BN + C\}$ where

$$A = \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ -2 & 2 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix} \quad N = \begin{pmatrix} n \\ m \\ p \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Loechner/Wilde algorithm computes the following vertices for the so-defined polytope P_N:

• (1) if $n \geq p$: $(n-p, \frac{2n+m-2p}{2}, 0)$, $(n-p, 0, 0)$, $(n, \frac{2n+m}{2}, p)$, $(n, \frac{2n+m}{2}, 0)$, $(n, 0, p)$, $(n, 0, 0)$.

• (2) if $n \leq p$: $(0, \frac{m}{2}, -n+p)$, $(0, \frac{m}{2}, 0)$, $(0, 0, -n+p)$, $(0, 0, 0)$, $(n, \frac{2n+m}{2}, p)$, $(n, \frac{2n+m}{2}, 0)$, $(n, 0, p)$, $(n, 0, 0)$.

As an example, we now compute the pseudo-polynomial enumerator on the first definition domain.

The denominator of P_N is 2. We count $((k+1)d)^q = 512$ initial values of $E(n, m, p)$. The lowest initial value of n is 9 since we consider the first definition domain ($n \geq p$) and since for any initial value of n, 8 initial values of p have to be considered. For example, with $n = 9$ we get the following initial counts:

n=9 p m	1	2	3	4	5	6	7	8
1	29	56	90	130	175	224	276	330
2	32	62	100	145	196	252	312	375
3	32	62	100	145	196	252	312	375
4	35	68	110	160	217	280	348	420
5	35	68	110	160	217	280	348	420
6	38	74	120	175	238	308	384	465
7	38	74	120	175	238	308	384	465
8	41	80	130	190	259	336	420	510

Each line of this array allows resolving a system of equations to determine $E(x, y, p)$, for all $x \in [9..16]$ and $y \in [1..8]$. For example, the first system dedicated to the determination of $E(9, 1, p) = [c_2, c_1]_p \, n_3^3 + [c_4, c_3]_p \, n_3^2 + [c_6, c_5]_p \, n_3 + [c_8, c_7]_p$ is

$$n = 9, m = 1 : \left\{ \begin{array}{lll} p = 1 : & c_1 + c_3 + c_5 + c_7 & = 29 \\ p = 2 : & 8c_2 + 4c_4 + 2c_6 + c_8 & = 56 \\ p = 3 : & 27c_1 + 9c_3 + 3c_5 + c_7 & = 90 \\ p = 4 : & 64c_2 + 16c_4 + 4c_6 + c_8 & = 130 \\ p = 5 : & 125c_1 + 25c_3 + 5c_5 + c_7 & = 175 \\ p = 6 : & 216c_2 + 36c_4 + 6c_6 + c_8 & = 224 \\ p = 7 : & 343c_1 + 49c_3 + 7c_5 + c_7 & = 276 \\ p = 8 : & 512c_2 + 64c_4 + 8c_6 + c_8 & = 330 \end{array} \right.$$

resulting in $E(9, 1, p) = -\frac{1}{6}p^3 + \frac{9}{2}p^2 + \frac{44}{3}p + 10$.

Solving for the other polynomials, $E(9,2,p) \dots E(9,8,p)$, allows resolving systems to determine $E(9,m,p) = [c_2, c_1]_m\, m^3 + [c_4, c_3]_m\, m^2 + [c_6, c_5]_m\, m + [c_8, c_7]_m$ with this system of equations:

$$n = 9 : \begin{cases} m = 1 : & c_1 + c_3 + c_5 + c_7 & = -\frac{1}{6}p^3 + \frac{9}{2}p^2 + \frac{44}{3}n_3 + 10 \\ m = 2 : & 8c_2 + 4c_4 + 2c_6 + c_8 & = -\frac{1}{6}p^3 + 5p^2 + \frac{97}{6}p + 11 \\ m = 3 : & 27c_1 + 9c_3 + 3c_5 + c_7 & = -\frac{1}{6}p^3 + 5p^2 + \frac{97}{6}p + 11 \\ m = 4 : & 64c_2 + 16c_4 + 4c_6 + c_8 & = -\frac{1}{6}p^3 + \frac{11}{2}p^2 + \frac{53}{3}p + 12 \\ m = 5 : & 125c_1 + 25c_3 + 5c_5 + c_7 & = -\frac{1}{6}p^3 + \frac{11}{2}p^2 + \frac{53}{3}p + 12 \\ m = 6 : & 216c_2 + 36c_4 + 6c_6 + c_8 & = -\frac{1}{6}p^3 + 6p^2 + \frac{115}{6}p + 13 \\ m = 7 : & 343c_1 + 49c_3 + 7c_5 + c_7 & = -\frac{1}{6}p^3 + 6p^2 + \frac{115}{6}p + 13 \\ m = 8 : & 512c_2 + 64c_4 + 8c_6 + c_8 & = -\frac{1}{6}p^3 + \frac{13}{2}p^2 + \frac{62}{3}p + 14 \end{cases}$$

In this case, the following result is obtained:

$$E(9,m,p) = \tfrac{1}{4}mp^2 + \tfrac{3}{4}mp + \tfrac{1}{2}m - \tfrac{1}{6}p^3 + \left[\tfrac{9}{2}, \tfrac{17}{4}\right]_m p^2 + \left[\tfrac{44}{3}, \tfrac{167}{12}\right]_m p + \left[10, \tfrac{19}{2}\right]_m$$

In a similar way, $E(10,m,p)$, $E(11,m,p)$, $\dots$, $E(16,m,p)$ are computed. Finally, $E(n,m,p)$ is determined by resolving the system shown below:

$$\begin{cases} n = 9 : & 729c_1 + 81c_3 + 9c_5 + c_7 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [\frac{9}{2}, \frac{17}{4}]_m p^2 + [\frac{44}{3}, \frac{167}{12}]_m p + [10, \frac{19}{2}]_m \\ n = 10 : & 1000c_2 + 100c_4 + 10c_6 + c_8 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [5, \frac{19}{4}]_m p^2 + [\frac{97}{6}, \frac{185}{12}]_m p + [11, \frac{21}{2}]_m \\ n = 11 : & 1331c_1 + 121c_3 + 11c_5 + c_7 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [\frac{11}{2}, \frac{21}{4}]_m p^2 + [\frac{53}{3}, \frac{203}{12}]_m p + [12, \frac{23}{2}]_m \\ n = 12 : & 1728c_2 + 144c_4 + 12c_6 + c_8 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [6, \frac{23}{4}]_m p^2 + [\frac{115}{6}, \frac{221}{12}]_m p + [13, \frac{25}{2}]_m \\ n = 13 : & 2197c_1 + 169c_3 + 13c_5 + c_7 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [\frac{13}{2}, \frac{25}{4}]_m p^2 + [\frac{62}{3}, \frac{239}{12}]_m p + [14, \frac{27}{2}]_m \\ n = 14 : & 2744c_2 + 196c_4 + 14c_6 + c_8 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [7, \frac{27}{4}]_m p^2 + [\frac{133}{6}, \frac{257}{12}]_m p + [15, \frac{29}{2}]_m \\ n = 15 : & 3375c_1 + 225c_3 + 15c_5 + c_7 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [\frac{15}{2}, \frac{29}{4},]_m p^2 + [\frac{71}{3}, \frac{275}{12}]_m p + [16, \frac{31}{2}]_m \\ n = 16 : & 4096c_2 + 256c_4 + 16c_6 + c_8 & = & \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + [8, \frac{31}{4}]_m p^2 + [\frac{151}{6}, \frac{293}{12}]_m p + [17, \frac{33}{2}]_m \end{cases}$$

The following result is obtained, giving the number of flops executed by the loop nest when $n \geq p$:

$$E(n,m,p) = \frac{1}{2}np^2 + \frac{3}{2}np + n + \frac{1}{4}mp^2 + \frac{3}{4}mp + \frac{1}{2}m - \frac{1}{6}p^3 + \left[0, -\frac{1}{4}\right]_m p^2 + \left[\frac{7}{6}, \frac{5}{12}\right]_m p + \left[1, \frac{1}{2}\right]_m$$

Likewise, when $n \leq p$:

$$E(n,m,p) = -\tfrac{1}{6}n^3 - \tfrac{1}{4}n^2 m + n^2\left[\tfrac{1}{2}p, \tfrac{1}{2}p - \tfrac{1}{4}\right]_m + \tfrac{1}{2}nmp + \tfrac{1}{4}nm + n\left[\tfrac{3}{2}p + \tfrac{7}{6}, p + \tfrac{11}{12}\right]_m + \\ \tfrac{1}{2}mp + \tfrac{1}{2}m + \left[p + 1, \tfrac{1}{2}p + \tfrac{1}{2}\right]_m$$

□

Example 14

For our last example, we show an application of parametric analysis. We derive the maximum amount of parallelism found in a small nested-loop program.

Consider the following loop nest:

```
for i:= 0 to n do
  for j:= 0 to i do
    for k:= 0 to i-j do
      Statement(s)
```

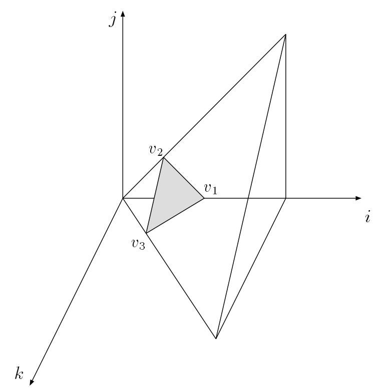


Figure 3: T_{n,t_0} in the first validity domain

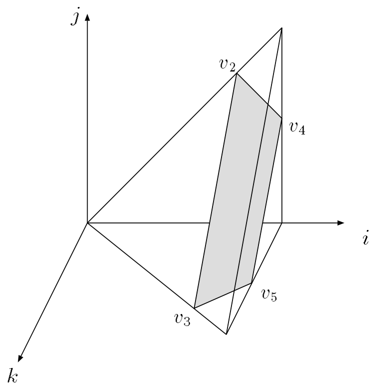


Figure 4: T_{n,t_0} in the second validity domain

This loop nest is modeled by the one-parameter polyhedral iteration space defined by:

$$P_n = \{\begin{pmatrix} i \\ j \\ k \end{pmatrix} \in \mathcal{Z}^3 | \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} n\}$$

Assume that the dependence analysis of this loop nest yields an optimal linear schedule defined by $t(i,j,k) = i + j + k$: any iteration (i,j,k) is computed at instant $i+j+k$. We use our method to determine the parametric number of simultaneous calculations $par(n,t_0)$ occuring at any instant t_0, in order to determine the parametric maximum parallelism $maxpar(n) = MAX_{t_0}\{par(n,t_0)\}$, i.e. the maximum number of simultaneous calculations.

From a geometrical point of view, all the points (i,j,k) of P_n resulting in the same value of $t_0 = t(i,j,k)$, belong to the two-parameters rational 2-polytope T_{n,t_0} defined by $T_{n,t_0} = P_n \cap \{(i,j,k) \in \mathcal{Z}^3 | i+j+k = t_0\}$. The number of points in T_{n,t_0} is $par(n,t_0)$.

Giving this last parametric definition of T_{n,t_0} as an input to the parametric vertices finding algorithm, it outputs the following results as shown on figures 3 and 4:

- if $0 \leq t_0 \leq n$, the parametric vertices are $v_1 = (t_0, 0, 0)$, $v_2 = \left(\frac{t_0}{2}, \frac{t_0}{2}, 0\right)$ and $v_3 = \left(\frac{t_0}{2}, 0, \frac{t_0}{2}\right)$.
- if $n \leq t_0 \leq 2n$, the parametric vertices are $v_2 = \left(\frac{t_0}{2}, \frac{t_0}{2}, 0\right)$, $v_3 = \left(\frac{t_0}{2}, 0, \frac{t_0}{2}\right)$, $v_4 = (n, t_0 - n, 0)$ and $v_5 = (n, 0, t - n)$.
- otherwise, T_{n,t_0} does not exist and $par(n,t_0) = 0$.

Hence, $par(n,t_0)$ is defined on two validity domains as pseudo-polynomials in n and t_0, of degree 2, and of pseudo-period 2. According to theorem 3 and by looking at $T_{1,0}$, it is deduced that all the coefficients of n^q, $q \leq 2$, are constant: $par(n,t_0) = c_1 n^2 + c_2 n + c_3$.

(i) We determine c_1, c_2, c_3 by solving the following system of equations on the first validity domain:

$$\begin{cases} 36c_1 + 6c_2 + c_3 & = par(6, t_0) \\ 49c_1 + 7c_2 + c_3 & = par(7, t_0) \\ 64c_1 + 8c_2 + c_3 & = par(8, t_0) \end{cases}$$

Solving this system yields the equations for $par(6, t_0)$, $par(7, t_0)$ and $par(8, t_0)$. According to theorem 3 and by looking at $T_{1,1}$, it is deduced that only the coefficient of t_0^2 is constant. For example, $par(6, t_0) = c_1 t_0^2 + [c_3, c_2]t_0 + [c_5, c_4]$, such that:

$$\begin{cases} c_5 & = par(6,0) = 1 \\ c_1 + c_2 + c_4 & = par(6,1) = 1 \\ 4c_1 + 2c_3 + c_5 & = par(6,2) = 3 \\ 9c_1 + 3c_2 + c_4 & = par(6,3) = 3 \\ 16c_1 + 4c_3 + c_5 & = par(6,4) = 6 \end{cases}$$

where the values of $par(6,0), \ldots, par(6,4)$ are given by some initial counts.
$\Rightarrow par(6, t_0) = \frac{1}{8}t_0^2 + [\frac{3}{4}, \frac{1}{2}]t_0 + [1, \frac{3}{8}]$. In the same way, we determine that the other pseudo-polynomials are the same, i.e $\forall\ n \in [6..8]$, $par(n, t_0) = \frac{1}{8}t_0^2 + [\frac{3}{4}, \frac{1}{2}]t_0 + [1, \frac{3}{8}]$. Solving the first system gives the parallelism rate of any instant t_0 on the first validity domain:

$$\forall\ 0 \le t_0 \le n,\ par(n, t_0) = \frac{1}{8}t_0^2 + \left[\frac{3}{4}, \frac{1}{2}\right] t_0 + \left[1, \frac{3}{8}\right]$$

Observe that the parallelism rate does not depend on n in this validity domain.
(ii) We now consider the second validity domain.

$$\begin{cases} 36c_1 + 6c_2 + c_3 & = par(6, t_0) \\ 49c_1 + 7c_2 + c_3 & = par(7, t_0) \\ 64c_1 + 8c_2 + c_3 & = par(8, t_0) \end{cases}$$

Solving this system yields the equations for $par(6, t_0)$, $par(7, t_0)$ and $par(8, t_0)$. For example,
$par(6, t_0) = c_1 t_0^2 + [c_3, c_2]t_0 + [c_5, c_4]$, such that:

$$\begin{cases} 36c_1 + 6c_3 + c_5 & = par(6,6) & = 10 \\ 49c_1 + 7c_2 + c_4 & = par(6,7) & = 9 \\ 64c_1 + 8c_3 + c_5 & = par(6,8) & = 12 \\ 81c_1 + 9c_2 + c_4 & = par(6,9) & = 9 \\ 100c_1 + 10c_3 + c_5 & = par(6,10) & = 11 \end{cases}$$

$\Rightarrow par(6, t_0) = -\frac{3}{8}t_0^2 + [\frac{25}{4}, 6]t_0 + [-14, -\frac{117}{8}]$. In the same way, we determine the other pseudo-polynomials:

$$\begin{aligned} par(7, t_0) &= -\tfrac{3}{8}t_0^2 + [\tfrac{29}{4}, 7]t_0 + [-20, -\tfrac{165}{8}] \\ par(8, t_0) &= -\tfrac{3}{8}t_0^2 + [\tfrac{33}{4}, 8]t_0 + [-27, -\tfrac{221}{8}] \end{aligned}$$

Solving the first system gives the parallelism rate of any instant t_0 on the second validity domain:

$$\forall\ n \le t_0 \le 2n, par(n,t_0) = nt_0 - \tfrac{1}{2}n^2 + \tfrac{1}{2}n - \tfrac{3}{8}t_0^2 + \left[\tfrac{1}{4}, 0\right]_{t_0} t_0 - \left[1, \tfrac{3}{8}\right]_{t_0}$$

(iii) From these results, the maximum parallelism is deduced:

- On the first validity domain ($0 \le t_0 \le n$), $par(n,t_0)$ is maximum when t_0 is maximum, i.e. when $t_0 = n$. Hence, $maxpar(n) = \frac{1}{8}n^2 + \left[\frac{3}{4}, \frac{1}{2}\right] n + \left[1, \frac{3}{8}\right]$.
- On the second validity domain ($n \le t_0 \le 2n$), we take the derivative of $par(n,t_0)$ with respect to t_0: $par'(n,t_0) = n - \frac{3}{4}t_0 + \left[\frac{1}{4}, 0\right]_{t_0}$.

$$\begin{aligned} & par'(n,t_0) \le 0 \\ \Leftrightarrow\ & n - \tfrac{3}{4}t_0 + \left[0, \tfrac{1}{4}\right]_{t_0} \le 0 \\ \Leftrightarrow\ & t_0 \ge \tfrac{4n-[1,0]}{3} \end{aligned}$$

  Hence, $par(n,t_0)$ increases from $t_0 = n$ to $t_0 = \frac{4n-[1,0]}{3}$, and then decreases until $t_0 = 2n$: it is maximum for $t_0 = t_{max} = \frac{4n-[1,0]}{3}$. Since t_{max} is not always integral, a case study is necessary to get the exact value of $maxpar(n) = par(n, t_{max})$, checking the possible values of $\lfloor t_{max} \rfloor$ and $\lceil t_{max} \rceil$. It yields the following answer: $maxpar(n) = \frac{1}{6}n^2 + \frac{5}{6}n - \left[1, 1, \frac{2}{3}\right]$.
- In conclusion, the maximum parallelism of the loop nest is:

$$\forall\ n \ge 0,\ maxpar(n) = \tfrac{1}{6}n^2 + \tfrac{5}{6}n - \left[1, 1, \tfrac{2}{3}\right]$$

□

7 Non-linear extensions

All these results concern normal polyhedra defined by a combination of linear constraints. When considering some nonlinear ones, the transformation rules given in this section have to be applied first.

This set of constraints depends on some positive integral size parameters. Any constraint C_k is of the form

$$C_k : \sum_i a_i \alpha_i \le \sum_j b_j n_j + c$$

where the a_i's, b_i's and c are rational constants, the n_j's are the size parameters and the α_i's can be of the form

$$x_i \quad (x_i\ mod\ d_i) \quad \lfloor \tfrac{x_i}{d_i} \rfloor \quad \lceil \tfrac{x_i}{d_i} \rceil \quad (d_i | x_i)$$

where d_i is an integral constant, $(d_i|x_i)$ means that d_i evenly divides x_i and x_i is any rational linear combination of the free variables. When the form of any α_i yields a nonlinear constraint, this can easily be transformed into a linear one through some elementary transformations, as those presented by W. Pugh in [21]:

• If a term $\lfloor \frac{x_i}{d_i} \rfloor$ appears in C_k, a free variable z is added to the set of free variables, and constraint C_k is replaced with $\{d_i z \leq x_i < d_i(z+1), C'_k\}$ where C'_k is C_k with $\lfloor \frac{x_i}{d_i} \rfloor$ replaced with z.
• If a term $\lceil \frac{x_i}{d_i} \rceil$ appears in C_k, a free variable z is added to the set of free variables, and constraint C_k is replaced with $\{d_i(z-1) < x_i \leq d_i z, C'_k\}$ where C'_k is C_k with $\lceil \frac{x_i}{d_i} \rceil$ replaced with z.
• If a term $(x_i \ mod \ d_i)$ appears in C_k, a free variable z is added to the set of free variables, and constraint C_k is replaced with $\{d_i z \leq x_i < d_i(z+1), C'_k\}$ where C'_k is C_k with $(x_i \ mod \ d_i)$ replaced with $x_i - z$.
• A constraint $(d_i | x_i)$ is equivalent to $(x_i \ mod \ d_i = 0)$ and the precedent rule can be applied.
• If a term of the form $\lfloor \frac{U}{u} \rfloor$ appears in the right side of the constraint C_k, since rational constraints are considered by our method, the floor can be avoided and the whole constraint is multiplied by u.

8 Conclusion

We have presented a general method, based on the vertices of parameterized polytopes and Ehrhart polynomials, for deriving a symbolic formula for the number of integer solutions to a set of parameterized linear constraints. The enumerator is formulated symbolically as a pseudo-polynomial in terms of parameters, The method we have presented is robust and handles any set of rational parameterized linear constraints without any restriction. Moreover, the method gives a closed form formula which easily evaluates to exact counts. The procedure is entirely computable and has been implemented along with the parametric vertices finding algorithm of Loechner and Wilde [17] and operates in association with the *polyhedral library* [28].

Being able to find symbolic enumerators has many applications in the analysis and transformation of scientific programs. We have given several examples in this paper. There are many other applications in the analysis and transformation of loop nests, some of which can be found in works cited here above or in [4, 5]. Other applications may also be found in other fields as sequential program analysis, combinatorics and optimization.

We have compared our method with other similar methods. M. Haghighat and C. Polychronopoulos present in [13, 14] a method for volume computation. The form of their answers is different because of the *min* and *max* expressions they introduce. Their results tend to be more complicated.

Ferrante, Sarkar and Thrash in [11] present methods for computing the number of distinct memory locations and cache lines accessed by a loop nest. These methods cannot handle unions of polytopes, do not compute symbolic answers, often compute only an approximation, and use expensive methods to handle the cache lines touched by a set of references.

The method described by Pugh in [21] consists of a set of tools and techniques dedicated to simplifying the initial set of constraints. Applying all of these tools could be expensive for large sets of constraints with rational coefficients. Moreover, their application is often rule driven, in some cases involving the addition of auxiliary variables and the use of standard formulas for sums of powers. (Pugh expects it will be sufficient to hard code the formulas for powers up to 10.) And finally, the resulting formula involves the use of modulo functions.

Tawbi in [27, 26] first uses a polyhedral splitting technique to characterize the validity domains, and an elimination of the variables in a predetermined order. However, her method does not systematically compute exact answers.

After doing these comparisons, we make these observations:

- summations over several variables should not presume an order in which to perform the summation (as pointed out by Pugh in [21]),
- a general technique must also consider rational polytopes, since nested loops models often yield rational affine loop bounds, and since it is useful to systematically handle a wide variety of applications in the parallelization of scientific programs.
- in order to be simple, the result should be expressed without any complex function as *min*'s, *max*'s, *mod*'s, floors or ceilings. In this respect, periodic numbers, introduced by Ehrhart [10], are a good alternative and can undoubtedly be used in other situations.
- exact formulations can be found.

This is the first method to be proposed that enumerates both integer and rational polytopes using a single comprehensive algorithm. Moreover, the formulation, using Ehrhart pseudo-polynomials, is simple and gives exact parametric expressions for the number of integer points.

References

[1] A. I. Barvinok. A polynomial-time algorithm for counting integral points in polyhedra when the dimension is fixed. In *Proc. of the 34th Symp. on the Foundations of Computer Science (FOCS'93)*, pages 566-572. IEEE Computer Society Press, New York, 1993.

[2] A. I. Barvinok. Computing the Ehrhart Polynomial of a Convex Lattice Polytope. *Discrete Comput. Geom.*, 12:35-48,1994.

[3] M.S. Bazaraa, H.D. Sherali and C.M. Shetty. *Nonlinear Programming, Theory and Algorithms, second edition.* Wiley, New York, 1993.

[4] Ph. Clauss. The volume of a lattice polyhedron to enumerate processors and parallelism. Research Report ICPS 95-11, 1995. `http://icps.u-strasbg.fr/pub-95/pub-95-11.ps.gz`

[5] Ph. Clauss. Counting Solutions to Linear and Nonlinear Constraints through Ehrhart polynomials: Applications to Analyze and Transform Scientific Programs. *10th ACM Int. Conf. on Supercomputing*, Philadelphia, May 1996. Also available as Research Report ICPS 95-20. `http://icps.u-strasbg.fr/pub-96/pub-95-20.ps.gz`

[6] M. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967-974, 1988.

[7] E. Ehrhart. Sur les polyèdres rationnels homothétiques à n dimensions. *C.R. Acad. Sci. Paris*, 254:616-618, 1962.

[8] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire I. *J. Reine Angew. Math.*, 226:1-29, 1967.

[9] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire II. *J. Reine Angew. Math.*, 227:25-49, 1967.

[10] E. Ehrhart. *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*. International Series of Numerical Mathematics, vol.35, Birkhäuser Verlag, Basel/Stuttgart, 1977.

[11] J. Ferrante, V. Sarkar and W. Thrash. On estimating and enhancing cache effectiveness. *Advances in Languages and Compilers for Parallel Processing*, pages 328-343. The MIT Press, 1991.

[12] P.M. Gruber and C.G. Lekkerkerker. *Geometry of Numbers*. North-Holland, Amsterdam, 1987.

[13] M. Haghighat and C. Polychronopoulos. Symbolic analysis: A basis for parallelization, optimization and scheduling of programs. *U. Banerjee et al., editor, Languages and Compilers for Parallel Computing*. Springer-Verlag, August 1993. LNCS 768, *Proc. of the 6th annual workshop on Programming Languages and Compilers for Parallel Computing*.

[14] M. Haghighat and C. Polychronopoulos. Symbolic analysis: A basis for parallelization, optimization and scheduling of programs. Technical Report 1317, CSRD, Univ. of Illinois, August 1993.

[15] D. J. Kuck. *The Structure of Computers and Computations*, J. Wiley and Sons, N.Y., 1978.

[16] H. Le Verge, V. Van Dongen, and D. Wilde. La synthèse de nids de boucles avec la bibliothèque polyédrique. *RenPar'6*, Jun 1994. English version "Loop Nest Synthesis Using the Polyhedral Library" in IRISA technical report 830, May 1994.

[17] V. Loechner and D. K. Wilde. Parameterized polyhedra and their vertices. Research Report ICPS 96-09, Submitted for publication, 1996. `http://icps.u-strasbg.fr/pub-96/pub-96-09.ps.gz`

[18] I. G. Macdonald. The volume of a lattice polyhedron. *Proc. Camb. Phil. Soc.*, 59:719-726, 1963.

[19] I. G. Macdonald. Polynomials associated with finite cell-complexes. *J. London Math. Soc.*, 4(2):181-192, 1971.

[20] T. H. Mattheiss and D. Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5(2):167-185, 1980.

[21] W. Pugh. Counting Solutions to Presburger Formulas: How and Why. *Proc. of the 1994 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1994.

[22] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

[23] H. S. Stone and D. Thiebaut. Footprints in the cache. *Proc. ACM SIGMETRICS 1986*, pp. 4-8, May 1986.

[24] R. P. Stanley. *Combinatorics and Commutative Algebra*. Birkhäuser, Boston, 1983.

[25] R. P. Stanley. *Enumerative Combinatorics Volume I*. Wadsworth & Brooks/Cole, Monterey, California, 1986.

[26] N. Tawbi. Estimation of nested loop execution time by integer arithmetics in convex polyhedra. *Proc. of the 1994 International Parallel Processing Symposium*, April 1994.

[27] N. Tawbi and P. Feautrier. Processor allocation and loop scheduling on multiprocessor computers. *Proc. of the 1992 International Conference on Supercomputing*, pages 63-71, July 1992.

[28] D. K. Wilde. A library for doing polyhedral operations. *Master's thesis*, Oregon State University, Corvallis, Oregon, Dec 1993. Also published as IRISA technical report PI 785, Rennes, France, Dec 1993.

Bibliographie

[AALT95] S.P. Amarasinghe, J.M. Anderson, M.S. Lam, and C.W. Tseng. The SUIF compiler for scalable parallel machines. In *Proceedings of the seventh SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[AI91] C. Ancourt and F. Irigoin. Scanning polyhedra with DO loops. In *Proc. ACM SIGPLAN '91*, pages 39–50, june 1991.

[AI92] C. Ancourt and F. Irigoin. Automatic code distribution. In *The third Workshop on Compilers for Parallel Computers*, pages 39–50, june 1992.

[AICK93] C. Ancourt, F. Irigoin, F. Coelho, and R. Keryell. A linear algebra framework for static HPF code distribution. In *The fourth Workshop on Compilers for Parallel Computers*, December 1993.

[AK84] J. R. Allen and K. Kennedy. Automatic loop interchange. *SIGPLAN Notices*, 19(6):233–248, June 1984. Proc. of the ACM SIGPLAN'84 symposium on Compiler Construction.

[AL93] J.M. Anderson and M.S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices*, 28(6):112–125, 1993.

[Alb97] O. Albiez. Génération de code dans le cadre des transformations affines par variable. Rapport de DEA, ICPS, Strasbourg, France, 1997. ftp://dpt-info.u-strasbg.fr/pub/recherche/Parallelisme/DEA_97_Albiez.ps.gz.

[Anc91] C. Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales.* PhD thesis, Université de Paris VI, 1991.

[ANS93] ANSI. *The Programming Language ADA Reference Manual.* Springer-Verlag London, Ltd, 1993.

[Ban93] U. Banerjee. *Loop Transformations for Restructuring Compilers - The Foundations.* Kluwer Academic Publishers, 1993.

[BENP93] U. Banerjee, R. Eigenman, A. Nicolau, and A. Padua. Automatic program parallelization. *Proceedings of the IEEE*, 81(2):211–243, February 1993.

[BKK+95] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill. Solving alignment using elementary linear algebra. In *Languages and Compilers for Parallel Computing*. LNCS 892, Springer-Verlag, 1995.

[BL92] M. Barnett and C. Lengauer. Unimodularity considered non-essential. In *Proc. CONPAR 92 - VAPP V*, pages 659–664. Springer Verlag, 1992. LNCS 634.

[Bla63] W.A. Blankinship. A new version of the euclidean algorithm. *American Mathematical Monthly*, 7:742–745, 1963.

[CFR+91] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. An efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, October 1991.

[CFR95] J.-F. Collard, P. Feautrier, and T. Risset. Construction of DO loops from systems of affine constraints. *Parallel Processing Letters*, 5(3):421–436, 1995.

[CGO+96] S. Chatterjee, J.R. Gilbert, L. Oliker, R. Schreiber, and T.J. Sheffler. Algorithms for automatic alignment of arrays. *Journal of Parallel and Distributed Computing*, 38(2):145–157, November 1996.

[Cha93] Z. Chamski. Fast and efficient generation of loop bounds. Technical Report 771, IRISA, Rennes, France, Oct 1993.

[Che65] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5(2):228–133, 1965.

[CL96] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. In Fortes, Mongenet, Parhi, and Taylor, editors, *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, pages 415–424. IEEE Computer Society Press, August 1996.

[Cla90] Ph. Clauss. *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* PhD thesis, Université de Franche-Comté, Besançon, France, May 1990.

[Cla95] Ph. Clauss. The volume of a lattice polyhedron to enumerate processors and parallelism. Technical Report 95-11, ICPS, http://icps.u-strasbg.fr/pub-95/pub-95-11.ps.gz, 1995.

[Cla96] Ph. Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *10th ACM Int. Conf. on Supercomputing, Philadelphia*, 1996.

[Cla97] Ph. Clauss. Handling memory cache policy with integer points countings. In Lengauer, Griebl, and Gorlatch, editors, *Euro-Par'97, Passau*, pages 285–293. Springer-Verlag, LNCS 1300, August 1997.

[CLW97] Ph. Clauss, V. Loechner, and D. K. Wilde. Deriving formulae to count solutions to parameterized linear systems using Ehrhart polynomials: Applications to the analysis of nested-loop programs. Technical Report 97-05, ICPS, http://icps.u-strasbg.fr/pub-97/pub-97-01.ps.gz, 1997.

[CP93] P. Clermont and N. Paris. HyperC: Portable parallel programming in C. In H. J. Siegel, editor, *Proceedings of the 8th International Parallel Processing Symposium, Cancn, Mexico*, pages 682–687. IEEE, 1993.

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

[DG96] C.G. Diderich and M. Gengler. Solving the constant-degree parallelism alignment problem. In *Proceedings of EuroPar '96*, volume 1123 of *LNCS*, pages 451–454. Springer-Verlag, August 1996.

[DI85] J.M. Delosme and I.C.F. Ipsen. n illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *Second. Int. Symposium on VSLI Technology, Systems and Applications*, pages 268–273, 1985.

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

[DR92] A. Darte and Y. Robert. Scheduling uniform loop nests. Research Report 92-10, LIP, Ecole Normale Supérieure de Lyon, France, February 1992. Published in IEEE Transactions on Parallel and Distributed Systems, Vol 5(8), 1994.

[DR94] A. Darte and Y. Robert. On the alignment problem. *Parallel Processing Letters*, 4(3):259–270, 1994.

[DR95] M. Dion and Y. Robert. Mapping affine loop nests : new results. In B. Hertzberger and G. Serazzi, editors, *Int. Conf. on High-Performance Computing and Networking*, pages 184–189. Springer Verlag, May 1995. Milan, Italy.

[Ehr62] E. Ehrhart. Sur les polyèdres rationnels homothétiques à n dimensions. *C.R. Acad. Sci. Paris*, 254:616–618, 1962.

[Ehr67a] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire i. *Reine Angew. Math.*, 226:1–29, 1967.

[Ehr67b] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire ii. *Reine Angew. Math.*, 227:25–49, 1967.

[Ehr77] E. Ehrhart. *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*, volume 35 of *International Series of Numerical Mathematics*. Birkhäuser Verlag, Basel/Stuttgart, 1977.

[Fea91] P. Feautrier. Dataflow analysis of scalar and array references. *Int. J. of Parallel Programming*, 20(1):23–53, February 1991.

[Fea92a] P. Feautrier. Some efficient solutions to the affine scheduling problem, part 1 : one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, October 1992.

[Fea92b] P. Feautrier. Some efficient solutions to the affine scheduling problem, part 2 : multidimensional time. *Int. J. of Parallel Programming*, 21(6), December 1992.

[Fea94] P. Feautrier. Towards automatic distribution. *Parallel processing letters*, 4(3):233–244, 1994.

[FFW85] J.A.B. Fortes, K.S. Fu, and B.W. Wah. Systematic approaches to the design of algorithmically specified systolic arrays. In *Int. Conf. on Acoustics, Speech and Signal Processing*, 1985.

[FHK+91] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu. Fortran D language specification. Technical Report CRPC-TR90079, Rice University, 1991. http://www.crpc.rice.edu/CRPC/softlib/TRs_online.html.

[FQ88] F. Fernandez and P. Quinton. Extension of chernikova's algorithm for solving general mixed linear programming problems. Technical Report 437, IRISA, Rennes, France, 1988.

[FR94] K. Fukuda and V. Rosta. Combinatorial face enumeration in convex polytopes. *Computational Geometry*, 4:191–198, 1994.

[FT89] P. Feautrier and N. Tawbi. Résolution de systèmes d'inéquations linéaires ; mode d'emploi du logiciel PIP. Technical Report 90-2, Laboratoire MASI, Université P. et M. Curie, Paris, January 1989.

[GBD+] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, a user's guide and tutorial.* MIT Press.

[GL87] P.M. Gruber and C.G. Lekkerkerker. *Geometry of Numbers.* North-Holland, Amsterdam, 1987.

[Gol56] A. J. Goldmann. Resolution and separation theorems for polyhedral convex sets. In Kuhn and Tucker, editors, *Linear Inequalities and Related Systems.* Princeton University Press, Princeton, NJ, 1956.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 1978.

[HPF93] High Performance Fortran Forum. *High Performance Fortran language specification*, draft edition, May 1993. Available as technical report CRPC-TR92225, Rice University.

[HS91] C.H. Huang and P. Sadayappan. Communication-free hyperplane partitioning of nested loops. In Banerjee, Gelernter, Nicolau, and Padua, editors, *Languages and Compilers for Parallel Computing*, pages 186–200. Springer Verlag, 1991. LNCS No 589.

[INM88] INMOS ltd. *OCCAM 2 reference manual*, prentice-hall international edition, 1988.

[KL78] H.T. Kung and C.E Leiserson. Systolic arrays for VLSI. In *Sparse matrix proceedings*, pages 245–282. SIAM, 1978.

[KLS90] K. Knobe, J. D. Lukas, and G. L. Steele. Data optimization: Allocation of arrays to reduce communications on simd machines. *Journal of Parallel and Distributed Computing*, 8, 1990.

[KM92] K. Kennedy and K. McKinley. Optimizing for parallelism and data locality. In *Proc. of the 1992 ACM Int. Conf. on Supercomputing*, 1992.

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

[KP93] W. Kelly and W. Pugh. A framework for unifying reordering transformations. Technical Report 3193, Dpt. of Computer Science, University of Maryland, April 1993.

[KPR95] W. Kelly, W. Pugh, and E. Rosser. Code generation for multiple mappings. In *5th Symposium on Frontiers of Massively Parallel Computation*, pages 332–341, Feb. 1995. McLean, Virginia.

[Kuc78] D. J. Kuck. *The Structure of Computers and Computations.* J. Wiley and Sons, 1978.

[Kun82] H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.

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

[Lam94] M. S. Lam. Locality optimizations for parallel machines. In B. Buchberger and J. Volkert, editors, *Third Joint Int. Conference on Vector and Parallel Processing, CONPAR 94 - VAPP VI*, pages 17–28. Springer Verlag, 1994. LNCS 854.

[LC91] J. Li and M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of Parallel and Distributed Computing*, 13(2), 1991.

[Le 92] H. Le Verge. A note on chernikova's algorithm. Technical Report 635, IRISA, Rennes, France, 1992.

[LEZ96] W. Löwe, J. Eisenbiegler, and W. Zimmermann. Optimizing parallel programs on machines with expensive communications. In *Europar '96 Parallel Processing, LNCS 1124*, pages 602–610, 1996.

[LL94] A. W. Lim and M.S. Lam. Communication-free parallelization via affine transformations. In *Proc. of the 7th Worshop on Languages and Compilers for Parallel Computing*, 1994.

[LM97] V. Loechner and C. Mongenet. Solutions to the communication minimization problem for affine recurrence equations. In Lengauer, Griebl, and Gorlatch, editors, *Europar'97*, pages 328–337. Springer-Verlag, August 1997. LNCS 1300.

[LMQ91] H. Le Verge, Ch. Mauras, and P. Quinton. The Alpha language and its use for the design of systolic arrays. *Journal of VLSI and signal processing*, 3:173–182, 1991.

[LP92] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. In *Proc. of the 5th Workshop on Languages and Compilers for Parallel Computing*, Aug. 1992. Yale University.

[LVW94] H. Le Verge, V. Van Dongen, and D. K. Wilde. Loop nest synthesis using the polyhedral library. Technical Report 830, IRISA, Rennes, France, May 1994. http://www.irisa.fr/EXTERNE/bibli/pi/pi830.html.

[LW97] V. Loechner and D. K. Wilde. Parameterized polyhedra and their vertices. *International Journal of Parallel Programming*, 25(6), December 1997.

[Mac63] I. G. Macdonald. The volume of a lattice polyhedron. *Proc. Camb. Phil. Soc.*, 59:719–726, 1963.

[Mac71] I. G. Macdonald. Polynomials associated with finite cell-complexes. *London Math. Soc.*, 4(2):181–192, 1971.

[Mau89] Ch. Mauras. *Alpha, un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones.* PhD thesis, Université de Rennes I, Dec 1989.

[MCP91] C. Mongenet, Ph. Clauss, and G.R. Perrin. A geometrical coding to compile affine recurrence equations on regular arrays. In *5th International Parallel Processing Symposium, IPPS'91*, 1991.

[MCP94] C. Mongenet, Ph. Clauss, and G.R. Perrin. Geometrical tools to map systems of affine recurrence equations on regular arrays. *Acta Informatica*, 31:137–160, 1994.

[MF86] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transactions on Computers*, 35(1):1–12, 1986.

[Min96] H. Minkowski. *Geometrie der Stahlen (Erste Lieferung).* Teubner, Leipzig (reprinted : Chelsea, New York, 1967), 1896.

[Min11] H. Minkowski. *Gesammelte Abhandlungen, Band I, II.* Teubner, Leipzig (reprinted : Chelsea, New York, 1967), 1911.

[Mol83] D.I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, 71(1):113–120, January 1983.

[Mon85] C. Mongenet. *Une Méthode de Conception d'Algorithmes Systoliques, Résultats Théoriques et réalisation.* PhD thesis, Institut National Polytechnique de Lorraine, Nancy, France, 1985.

[Mon97] C. Mongenet. Affine dependence classification for communications minimization. *International Journal of Parallel Programming*, 25(6), December 1997.

[MP87] C. Mongenet and G.R. Perrin. Synthesis of systolic arrays for inductive problems. In *Conference on Parallel Architectures and Languages Europe, PARLE'87*, pages 260–277. Springer Verlag, 1987. LNCS 259.

[MPI95] MPI Forum. *MPI: A message-passing Interface Standard*, 1995.

[MQRS90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In Kung, Schwartzlander, Fortes, and Przytula, editors, *Int. Conf. on Application Specific Array Processors*, pages 100–110. IEEE Computer Society Press, 1990.

[MR80] T. H. Mattheiss and D. Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5(2):167–185, 1980.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. *Contributions to the Theory of Games*. Kuhn and Tucker, Eds., Princeton University Press, Princeton, NJ, (Reprinted in : *Theodore S. Motzkin: Selected Papers*, Cantor, Gordon, Rotschild, Eds., Birkhäuser, Boston, 1983, pp. 81–103), 1953.

[Par92] N. Paris. *Définition de PompC*. Laboratoire d'Informatique, Ecole Normale Supérieure, 1992. Technical Report LIENS-92-05.

[Pla95] A. Platonoff. *Contribution à la distribution automatique des données pour machines massivement parallèles*. PhD thesis, Université Paris 6, Ecole Nationale Supérieure des Mines de Paris, March 1995.

[PMM92] D. M. Pase, T. MacDonald, and A. Meltzer. MPP Fortran programming model. Technical report, Cray Research, Inc, Eagan, Minnesota, 1992.

[PS95] P. Pineo and M.L. Soffa. A practical approach to single assignment code. In P. Evripidou L. Bic, W. Böhm and J.L. Gaudiot, editors, *Proc. of the IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques, PACT'95, Limassol, Cyprus*, pages 149–158. ACM Press, June 1995.

[Pug92] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Comm. of the ACM*, 8:102–114, August 1992.

[QRR97] P. Quinton, S. V. Rajopadhye, and T. Risset. On manipulating z-polyhedra using a canonical representation. *Parallel Processing Letter*, 7(2):181–194, 1997.

[QRW95a] P. Quinton, S. Rajopadhye, and D. Wilde. Deriving imperative code from functional programs. In *FPCA'95*, June 1995. La Jolla, California.

[QRW95b] P. Quinton, S. Rajopadhye, and D. Wilde. On deriving data parallel code from a functional program. In *9th Int. Parallel Processing Symposium*, April 1995. Santa Barbara,CA.

[Qui84a] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrence equations. In *IEEE 11th Int. Symp. on Computer Architecture*, pages 208–214, 1984.

[Qui84b] P. Quinton. The systematic design of systolic arrays. Technical report, Microelectronics Center of North Carolina, July 1984.

[QV89] P. Quinton and V. Van Dongen. The mapping of linear recurrence equations on regular arrays. *The Journal of VLSI Signal Processing*, 1:95–113, 1989.

[Ram92] J. Ramanujam. Non-unimodular transformations of nested loops. In *Supercomputing 92*, pages 214–223. IEEE Computer Society Press, November 1992.

[Rao85] S.K. Rao. *Regular iterative algorithms and their implementation on processor arrays.* PhD thesis, Stanford University, Information systems Lab., 1985.

[Ref96] C. Reffay. *Génération de code parallèle à partir de systèmes d'équations récurrentes.* PhD thesis, Université de Franche-Comté, 1996.

[RF90] S. V. Rajopadhye and R. M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing*, 4:163–189, 1990.

[RPF86] S. V. Rajopadhye, S. Purushothaman, and R. M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In K. Nori, editor, *Foundations of Software Technology and Theoretical Computer Science*, pages 488–503. Springer-Verlag, 1986. LNCS 241.

[RS91] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Transactions on parallel and distributed systems*, 2(4):472–482, October 1991.

[Rub75] D. Rubin. Vertex generation and cardinality constrained linear programs. *Operations Research*, 23(3):555–565, May 1975.

[Sch86] A. Schrijver. *Theroy of Linear and Integer Programming.* John Wiley and Sons, 1986.

[Sta83] R. P. Stanley. *Combinatorics and Commutative Algebra.* Birkhäuser, Boston, 1983.

[Sta86] R. P. Stanley. *Enumerative Combinatorics Volume I.* Wadsworth & Brooks/Cole, Monterey, 1986.

[TT93] J. Teich and L. Thiele. Partitioning of processor arrays: A piecewise regular approach. *Integration: the VLSI journal*, 14(3):297–332, Feb 1993.

[VQ88] V. Van Dongen and P. Quinton. Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays. In K. Bromley et al., editor, *Int. Conf. on Systolic Arrays*, pages 473–482. IEEE Computer Society Press, 1988.

[WD89] Y. Wong and J.M. Delosme. Optimisation of processor count for systolic arrays. Technical report, Yale University, Computer science Dpt., 1989.

[Wil93] D.K. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, 1993.

[Wil94] D.K. Wilde. The Alpha language. Technical Report 827, IRISA, Rennes, France, May 1994. http://www.irisa.fr/EXTERNE/bibli/pi/pi827.html.

[Wil95] D. Wilde. *From Alpha to Imperative Code: A Transformational Compiler for an Array Based Functional Language.* PhD thesis, Oregon State University, 1995.

[WL91] M.E. Wolf and M.S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, October 1991.

[Wol91] M. Wolfe. The tiny loop restructuring research tool. In *International Conference on Parallel Processing*, pages II. 46–53, 1991.

[Wol96] M. Wolfe. *High Performance Compilers for Parallel Computing.* Addison Wesley, 1996.

[WR95] D. Wilde and S. Rajopadhye. The naive execution of affine recurrence equations. In P. Cappello, C. Mongenet, G.-R. Perrin, P. Quinton, and Y. Robert, editors, *Int. Conf. on Application Specific Array Processors, ASAP'95*, pages 1–12. IEEE Computer Society Press, 1995.

[WR97] D. Wilde and S. Rajopadhye. Memory reuse analysis in the polyhedral model. *Parallel Processing Letter*, 7(2):203–215, 1997.

[YC88] Y. Yaacobi and P.R. Cappello. Converting affine recurrence equations to quasi-uniform recurrence equations. In *Third International workshop on parallel computation and VLSI theory*, 1988.

[ZBC+92] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald. Vienna Fortran - a language specification, version 1.1. Technical Report 92-4, Vienna University, 1992.

[ZC90] H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers.* ACM Press, 1990.

[Zie95] G. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate texts in Mathematics.* Springer-Verlag, 1995.

Bibliographie personnelle

Articles publiés dans des revues scientifiques internationales

[1] V. Loechner and D. K. Wilde. Parameterized polyhedra and their vertices. *International Journal of Parallel Programming*, 25(6), December 1997.

[2] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 1998. To appear.

Conférences internationales avec comité de lecture et actes publiés

[3] V. Loechner and C. Mongenet. A toolbox for affine recurrence equations parallelization. In B. Hertzberger and G. Serazzi, editors, *Int. Conf. on High-Performance Computing and Networking*, pages 263–268. Springer Verlag, May 1995. LNCS 919.

[4] V. Loechner and C. Mongenet. Opera : A toolbox for loop parallelization. In Jelly, Gordon, and Croll, editors, *First International Workshop on Software Engineering for Parallel and Distributed Systems*, pages 134–145. Chapman&Hall, March 1996.

[5] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. In Fortes, Mongenet, Parhi, and Taylor, editors, *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, pages 415–424. IEEE Computer Society Press, August 1996.

[6] V. Loechner and C. Mongenet. Solutions to the communication minimization problem for affine recurrence equations. In Lengauer, Griebl, and Gorlatch, editors, *Europar'97*, pages 328–337. Springer-Verlag, August 1997. LNCS 1300.

Communications nationales

[7] V. Loechner. Polyèdres paramétrés : recherche de sommets et applications. In *RenPar'8, Bordeaux*, 1996. Poster.