UNIVERSITÉ LOUIS PASTEUR
STRASBOURG

PRÉSENTATION DES TRAVAUX

pour obtenir

L'HABILITATION À DIRIGER DES RECHERCHES

par

Philippe Clauss

Méthodes polyédriques pour la parallélisation et l'optimisation de programmes

Soutenue le 14 janvier 2000 devant le jury composé de :

Paul Feautrier	Rapporteurs
Christian Lengauer	
Jean-Jacques Pansiot	
Yves Robert	
Catherine Mongenet	Examinateurs
Guy-René Perrin	

Table des matières

Table des figures

Introduction

Un des principaux objectifs motivant les progrès technologiques en informatique est bien évidemment la quête d'une puissance de traitement de plus en plus importante. Les enjeux sont tant scientifiques que commerciaux, et cette quête permet de repousser les limites du possible en matière de simulation physique, de calcul de prévisions, d'analyses statistiques, de traitements en temps réel, d'animations graphiques... Cette puissance croissante donne à la science de nouveaux moyens d'accélérer ses avancées, et procure à la production logicielle grand public des moyens de séduction de plus en plus grands.

Mais il se pose d'importants problèmes d'écriture de logiciels profitant pleinement de cette puissance. En effet, sans prendre garde à l'adéquation entre le programme produit et la structure de la machine d'exécution, de grandes distorsions entre performance théorique et performance réelle peuvent apparaître. Ces nouvelles technologies remettent bien souvent en question les modes de raisonnement et les anciennes habitudes liées à la programmation des ordinateurs. C'est pourquoi se pose la nécessité de proposer des outils de transformation et de compilation de programmes, permettant d'exploiter au mieux ces capacités offertes.

L'organe central de cette augmentation de puissance est bien sûr le processeur. Les progrès dans ce domaine ont été, et sont toujours, de plus en plus rapides. Pourtant, l'état actuel de la technologie révèle des écarts non négligeables entre les vitesses des processeurs et celles de leurs organes périphériques : le processeur est souvent inactif car en attente de réponse de la part d'un de ses périphériques. Les réponses technologiques apportées à ce problème recouvrent deux approches : l'augmentation de vitesse des périphériques, et la capacité des processeurs à partager leur activité entre plusieurs traitements, réduisant ainsi leur temps d'inactivité. Bien que permettant d'améliorer sensiblement les performances, ces deux approches ne permettent pas d'apporter de réponse totalement satisfaisante.

En particulier, une des techniques est de hiérarchiser les mémoires des ordinateurs, en allant d'une mémoire d'accès très rapide, proche du processeur et de capacité de stockage réduite, appelée *mémoire cache*, aux mémoires d'accès plus lent mais possédant de grandes capacités de stockage, telles que les disques ou les bandes magnétiques. Ainsi, un processeur verra ses performances pleinement exploitées s'il n'accède qu'à sa mémoire cache. Mais cette mémoire ne possède qu'une taille limitée, due à son coût important, et des transferts de données entre tous les niveaux hiérarchiques de la mémoire sont indispensables. C'est là qu'une utilisation bien adaptée des processeurs peut sensiblement améliorer les performances d'exécution, par une réduction du nombre de ces transferts inter-mémoires.

Bien que la puissance des processeurs suive une continuelle évolution, la recherche de performances toujours meilleures passe par la conception d'ordinateurs plus ambitieux : les machines multi-processeurs ou machines parallèles. L'utilisation de plusieurs processeurs pour l'accomplissement d'un même traitement crée de nouveaux problèmes aussi bien au niveau technologique qu'en matière de raisonnement et de programmation. D'un point de vue technologique, ces dernières décennies ont vu l'apparition d'architectures matérielles très diverses, avec plus ou moins de succès. La tendance actuelle est aux architectures composées de noeuds de processeurs reliés par un réseau d'interconnexion très rapide, où chaque noeud possède une mémoire propre. Ces mémoires peuvent être vues par le système comme une mémoire globale découpée physiquement sur chacun des noeuds, ou comme une mémoire distribuée sur les noeuds, où chaque mémoire relève d'une gestion propre.

Parmi les matériels permettant des traitements parallèles, n'oublions pas les circuits dédiés à haute intégration, ainsi que les circuits programmables, les circuits FPGA. Ces circuits se composent de processeurs élémentaires, interconnectés de manière très régulière, souvent en connexion nord-sud-est-ouest (NEWS), et dédiés à un seul traitement : la programmation est fixée à la conception du circuit dans le cas des circuits dédiés, ou modifiée en reprogrammant les processeurs élémentaires et en reconfigurant le réseau de communication dans le cas des circuits FPGA. L'observation de la baisse du coût des matériels, au début des années 80, a déclenché un grand intérêt pour ces circuits et a stimulé d'importantes recherches en synthèse d'algorithmes parallèles pour ces architectures, sous la forme d'algorithmes et d'architectures *systoliques*. Bien que leur succès se soit amoindri depuis, les recherches en parallélisation automatique pour ces circuits ont démontré les possibilités de développement de méthodes automatiques de parallélisation de programmes. Ainsi, beaucoup des résultats obtenus ont aisément pu être transposés aux machines parallèles actuelles plus générales, moyennant la prise en compte de nouvelles contraintes matérielles, notamment en ce qui concerne le poids des communications inter-processeurs dans la performance globale des traitements.

Une programmation exploitant au mieux la puissance théorique d'une machine multi-processeurs peut être obtenue de manière expérimentale, en implémentant plusieurs solutions parallèles d'un traitement et en observant les performances obtenues. Toutefois, ce mode de programmation bien fastidieux nécessite de grandes compétences techniques de la part du programmeur. De plus, la mise en évidence du réel degré de parallélisme du traitement considéré peut s'avérer très difficile. D'où l'intérêt d'une parallélisation automatique, impliquant une analyse fine et complexe des traitements.

L'exécution d'un traitement sur plusieurs processeurs pose des problèmes particuliers de coopération entre ces processeurs. Comme sur une machine mono-processeur où le processeur est en attente de ses périphériques, un processeur sur une machine multi-processeurs peut être en attente de ses pairs et donc être inactif. Les réponses technologiques sont à la fois de même nature que celles citées plus haut dans le cas des machines mono-processeurs, et relatives à la répartition de la mémoire sur les noeuds et à la communication inter-noeuds : réseaux d'interconnexion de plus en plus rapides, organisation mémoire permettant le partage des données, ... Malgré ces progrès, la nature même du traitement et sa parallélisation peuvent induire de très mauvaises performances, en raison de la manière dont sont répartis les calculs sur les processeurs ainsi que celle des données en mémoire. Les méthodes de parallélisation ont donc pour but de générer des placements des calculs et des données, afin d'utiliser au mieux les possibilités matérielles et afin d'atteindre les meilleures performances possibles.

L'étude des programmes en vue de leur parallélisation revient à étudier leur structure, à la recherche des opérations pouvant s'exécuter de manière simultanée. Parmi les structures de contrôle d'un programme, les structures itératives, appelées communément *boucles* ou *nids de* *boucles* pour les structures imbriquées, contiennent potentiellement la majeure partie du parallélisme. Il est donc indiqué de s'intéresser prioritairement à la parallélisation de ces structures. De plus, certaines formes d'expression de ces boucles permettent de développer des outils de parallélisation et d'optimisation entièrement automatiques.

Nos travaux ont globalement été consacrés à ces domaines de parallélisation et d'optimisation automatiques pour les structures itératives, en proposant :

1. Des méthodes de placement des calculs sur les processeurs pour les algorithmes systoliques, permettant de maximiser l'efficacité des processeurs, pour des temps de calcul et un nombre de processeurs minimaux. Cet objectif revient à minimiser l'inactivité des processeurs. Les méthodes proposées se caractérisent par la non-linéarité des fonctions mettant en correspondance calculs et processeurs.

2. De nouveaux outils mathématiques permettant de résoudre de nombreux problèmes posés par la parallélisation et l'optimisation des programmes. Ces travaux ont donné lieu à des résultats dans les domaines des *polyèdres paramétrés* et des *polynômes d'Ehrhart*, sous la forme d'avancées théoriques et de développements d'outils logiciels. Le champ d'application de ces outils est vaste, dépassant même le domaine informatique.

3. Des méthodes automatiques de transformation de l'organisation des données en mémoire et de transformation de programmes, permettant d'optimiser l'utilisation de la mémoire et en particulier de la mémoire cache des processeurs. Cette partie est au centre de nos recherches actuelles et nous prévoyons de nouvelles avancées dans un avenir proche.

L'ensemble de ces travaux s'appuie sur une modélisation géométrique des structures itératives en terme de polyèdres et de polytopes convexes plongés dans le réseau des points à coordonnées entières. Nous présentons ce modèle au chapitre 1.

Chapitre 1

Les nids de boucles et le modèle polyédrique

Le modèle polyédrique, ou *polytope-model*, qui s'est révélé être bien adapté à la synthèse d'architectures systoliques au milieu des années 80, a également montré plus tard son efficacité dans la parallélisation des nids de boucles [62, 45]. Plusieurs environnements logiciels de parallélisation automatique basés sur ce modèle ont été développés : citons par exemple *PAF* développé dans l'équipe de P. Feautrier, *Loopo* [51] dans l'équipe de C. Lengauer et *PIPS* développé dans l'équipe de F. Irigoin [56]. Nos récentes extensions paramétriques de ce modèle, que nous présentons au chapitre 3, lui ouvre encore un plus vaste champ d'utilisations.

1.1 Définitions et notations

Les nids de boucles considérés dans nos travaux sont de la forme générale suivante :

for $i_1 = l_1$ to h_1
    ...
    for $i_2 = l_2(i_1)$ to $h_2(i_1)$
        ...
        for $i_3 = l_3(i_1, i_2)$ to $h_3(i_1, i_2)$
            ...
                ...
                    for $i_n := l_n(i_1, i_2, \ldots, i_{n-1})$ to $h_n(i_1, i_2, \ldots, i_{n-1})$
                        $\ldots Y(f_1(i_1, i_2, \ldots, i_n), f_2(i_1, i_2, \ldots, i_n), \ldots, f_m(i_1, i_2, \ldots, i_n)) \ldots$
                        ...

où les symboles l_q et h_q désignent des fonctions affines des indices de boucles extérieures, définissant respectivement les bornes inférieures et supérieures des valeurs que peuvent prendre l'indice i_q. Les symboles f_q désignent également des fonctions affines des indices de boucles et définissent les coordonnées des éléments du tableau Y référencés par une instruction du nid de boucles. Les termes constants de ces fonctions affines peuvent éventuellement contenir des paramètres de taille du problème. Mais dans ce cas, les opérations de transformations de nids de boucles peuvent nécessiter l'utilisation d'outils mathématiques symboliques particuliers, tels que ceux que nous avons développés et qui sont décrits au chapitre 3.

L'espace d'itération définit un polyèdre convexe P de dimension n, où chaque point entier contenu dans P est un vecteur d'itération noté $\overrightarrow{I} = (i_1, i_2, \ldots, i_n)$. Il est donc défini mathématiquement par un système d'inégalités linéaires conditionné par les bornes des indices de boucles :

$$P = \{(i_1, i_2, \ldots, i_n) \in \mathbb{Q}^n | \quad l_1 \leq i_1 \leq h_1, l_2(i_1) \leq i_2 \leq h_2(i_1), l_3(i_1, i_2) \leq i_3 \leq h_3(i_1, i_2), \ldots, l_n(i_1, i_2, \ldots, i_{n-1}) \leq i_n \leq h_n(i_1, i_2, \ldots, i_{n-1})\}$$

La mise sous forme matricielle de ces inégalités donne la définition générale suivante :

$$P = \{\overrightarrow{I} \in \mathbb{Q}^n | A.\overrightarrow{I} + \overrightarrow{B} \leq 0\}$$

où A est une matrice de taille $2n \times n$ et $\overrightarrow{B}$ un vecteur de taille n. Toutefois, le nombre de contraintes peut être sensiblement réduit par l'élimination des redondances. Cette opération est notamment réalisée par la librairie de calculs polyédriques *Polylib* [98]. La matrice A est alors de taille $s \times n$, avec $s \leq 2n$.

Les fonctions de référence à un tableau Y peuvent également être mises sous forme matricielle, $Y(f(\overrightarrow{I})) = Y(R.\overrightarrow{I} + \overrightarrow{o})$, où R est une matrice de taille $m \times n$ et $\overrightarrow{o}$ est un vecteur de taille m. Cette fonction d'accès $f(\overrightarrow{I})$ peut être vue comme une transformation affine de l'espace d'itération, comme cela a été décrit lors de nos travaux sur les polynômes d'Ehrhart de transformations affines de polytopes présentés dans [19].

Dans un nid de boucles, les itérations sont exécutées dans l'ordre lexicographique $\triangleleft$ des vecteurs d'itération :

$$\forall \ \overrightarrow{I}, \overrightarrow{J} \in P, \overrightarrow{I} \triangleleft \overrightarrow{J} \Leftrightarrow \exists \ p \leq n, \forall \ q \leq p-1, i_q = j_q \text{ et } i_p < j_p$$

1.2 Analyse des dépendances

L'application de transformations de parallélisation et d'optimisation nécessite de connaître les relations de dépendances entre les opérations. Deux opérations sont indépendantes si leur éventuel accès à une même adresse mémoire ne se fait qu'en lecture. De nombreuses méthodes de détection des dépendances ont été proposées. Citons notamment le test de U. Banerjee [4], les travaux de P. Feautrier [42], ceux de F. Irigoin *et al.* sur PIPS [56] et ceux de W. Pugh sur Omega [80].

On distingue classiquement trois types de dépendances : les dépendances de flot sont générées par une écriture suivie d'une lecture à une même adresse mémoire ; les dépendances de sortie sont générées par une écriture suivie d'une autre écriture ; les anti-dépendances sont générées par une lecture suivie d'une écriture.

On note $S(\overrightarrow{I}) \ \delta \ T(\overrightarrow{J})$ si l'itération $\overrightarrow{J}$ de l'instruction T dépend de l'itération $\overrightarrow{I}$ de l'instruction S. Les dépendances sont modélisées géométriquement par des vecteurs de dépendance $\overrightarrow{d_{S,T}} = \overrightarrow{J} - \overrightarrow{I}$ pour un même niveau de boucle. L'ordre partiel défini par les dépendances est alors représenté par un graphe de dépendance, où les noeuds sont les points entiers associés aux instances des instructions et où les arcs entre les noeuds symbolisent les dépendances entre les instances des instructions.

Dans le modèle polyédrique, la parallélisation d'un nid de boucles consiste à substituer à l'espace d'itération un espace $\mathcal{S} \times \mathcal{T}$, où $\mathcal{S}$ est l'espace des processeurs et $\mathcal{T}$ est l'espace temps. Cette transformation passe par la détermination d'une fonction de cadencement des instructions ainsi qu'une fonction d'allocation de ces instructions à un ensemble de processeurs.

1.3 Cadencement des opérations

Une fonction de cadencement d'un ensemble d'opérations associe à chaque opération une valeur prise dans un ensemble totalement ordonné. De plus, elle doit satisfaire les contraintes de précédence définies par les dépendances : si l'opération T dépend de l'opération S, alors S doit être exécutée avant T.

Dans le modèle polyédrique, on ne considère que des fonctions affines à image dans un ensemble discret, classiquement $\mathbb{Z}^d$ ou $\mathbb{Q}^d$. Suivant la valeur de d, on parle de cadencement monodimensionnel ou multidimensionnel.

Définition 1 (cadencement monodimensionnel) *Un cadencement monodimensionnel est une fonction t qui associe à chaque itération de chaque instruction un instant d'exécution de manière à respecter les dépendances :*

$$t : V \times \mathbb{Z}^n \to \mathbb{Z} \text{ tel que } S(\overrightarrow{I})\ \delta\ T(\overrightarrow{J}) \Longrightarrow t(S, \overrightarrow{I}) < t(T, \overrightarrow{J})$$

où V désigne l'ensemble des instructions.

Cette notion de cadencement a été étendue aux fonctions à valeurs dans $\mathbb{Q}$ [83], en considérant la partie entière d'une fonction à valeur rationnelle. Les fonctions de cadencement monodimensionnel sont des fonctions affines de la forme :

$$\begin{array}{rcl} V \times \mathbb{Z}^n & \to & \mathbb{Q} \\ t : (S, \overrightarrow{I}) & \to & \overrightarrow{\lambda_S} \cdot \overrightarrow{I} + \alpha_S \end{array}$$

où $\overrightarrow{\lambda_S}$ est un vecteur de $\mathbb{Q}^n$ et α_S est une constante rationnelle.

Les cadencements multidimensionnels ont été introduits et étudiés par Karp, Miller et Winograd [59], Lamport [61], Rao [84], Roychowdhury [86] ; Allen et Kennedy [1], Wolfe et Lam [102], Feautrier [43] et Darte et Vivien [96, 34] les ont largement étudiés et ont proposé des algorithmes de cadencement de nids de boucles.

Définition 2 (cadencement multidimensionnel) *Un cadencement multidimensionnel de dimension d est une fonction qui définit un ordre d'exécution par l'association d'un vecteur entier de dimension d à chaque itération de chaque instruction. Les vecteurs image sont ordonnés entre eux selon l'ordre lexicographique et sont choisis de manière à respecter les dépendances. C'est donc une application t de la forme :*

$$\begin{array}{rcl} V \times \mathbb{Z}^n & \to & \mathbb{Q}^d \\ t : (S, \overrightarrow{I}) & \to & \left(\overrightarrow{\lambda_S^1} \cdot \overrightarrow{I} + \alpha_S^1, \ldots, \overrightarrow{\lambda_S^d} \cdot \overrightarrow{I} + \alpha_S^d\right) \\ S(\overrightarrow{I})\ \delta\ T(\overrightarrow{J}) & \Longrightarrow & \left\lfloor\left(\overrightarrow{\lambda_S^1} \cdot \overrightarrow{I} + \alpha_S^1, \ldots, \overrightarrow{\lambda_S^d} \cdot \overrightarrow{I} + \alpha_S^d\right)\right\rfloor \lhd \left\lfloor\left(\overrightarrow{\lambda_T^1} \cdot \overrightarrow{J} + \alpha_T^1, \ldots, \overrightarrow{\lambda_T^d} \cdot \overrightarrow{J} + \alpha_T^d\right)\right\rfloor \end{array}$$

1.4 Allocation linéaire des opérations et des données à des processeurs

Une fonction d'allocation d'un ensemble d'opérations associe à chaque opération les coordonnées spatiales du processeur sur lequel elle est exécutée. Afin de ne pas réduire le parallélisme potentiel induit par le cadencement, cette fonction ne doit pas allouer au même processeur deux instructions possédant la même date d'exécution. Toutefois, cette contrainte forte n'est en réalité que rarement respectée pour l'implémentation finale d'un programme, car elle entraine en général un nombre de processeurs et un volume de communications trop importants.

On ne considère que des fonctions affines à image dans $\mathbb{Z}^m$, $m < n$, où n est la dimension de l'espace d'itération :

$$\begin{array}{rcl} V \times \mathbb{Z}^n & \rightarrow & \mathbb{Z}^m \\ alloc_S : (S, \overrightarrow{I}) & \rightarrow & \sigma_S \cdot \overrightarrow{I} + \beta_S \\ t\left(S(\overrightarrow{I})\right) = t\left(T(\overrightarrow{J})\right) & \Longrightarrow & alloc_S(\overrightarrow{I}) \neq alloc_T(\overrightarrow{J}) \end{array}$$

où σ_S est une matrice entière de taille $m \times n$ appelée *matrice d'allocation* et β_S est un scalaire. De telles fonctions allouent les opérations à des réseaux réguliers de processeurs, mais ne garantissent pas une utilisation complète de ces processeurs tout au long de l'exécution. C'est pourquoi plusieurs travaux, dont les notres présentés au chapitre 2, ont proposé d'autres techniques d'allocation consacrées à une minimisation du nombre de processeurs sans ralentissement du cadencement.

La distribution de la mémoire sur les processeurs et le coût élevé des communications entre ces processeurs posent un problème de placement des données et de minimisation des communications. De nombreux travaux traitent de ce problème : citons Ramanujam et Sadayappan [85], Li et Chen [65], Anderson et Lam [3], Darte et Robert [33], Feautrier [44], Bau *et al.* [6], Dion et Robert [35], Chatterjee *et al.* [14] et Loechner et Mongenet [76, 67, 70, 71]. Nos travaux sur les performances de cache, présentés au chapitre 4, ont un objectif très proche de ceux-ci, par la recherche d'une minimisation des transferts inter-mémoires.

1.5 Équations récurrentes affines et algorithmes systoliques

Les méthodes de synthèse d'algorithmes systoliques, sur lesquelles se basent nos travaux sur les allocations non-linéaires, ne considèrent que des structures itératives à *assignation unique* : les éléments de tableaux ne doivent être assignés qu'une seule fois par le programme. De plus, ces méthodes utilisent le formalisme des systèmes d'équations récurrentes affines :

$$X(\overrightarrow{I}) = g(\ldots, Y(f(\overrightarrow{I})), \ldots) \quad \overrightarrow{I} \in P$$

où g est une fonction de calcul de complexité $O(1)$ définissant l'instruction exécutée.

Sous nos hypothèses, il est possible de transformer un nid de boucles en nid de boucles (ou en équation récurrente affine) à assignation unique. Cette transformation consiste en une analyse exacte du flot de données pour les références aux éléments de tableaux. On peut par exemple utiliser la méthode développée par P. Feautrier et présentée dans [42]. Les seules dépendances subsistantes entre les calculs sont alors des dépendances de flot.

Les méthodes de synthèse d'algorithmes systoliques consistent en une analyse exacte des dépendances de flot entre les calculs, en une recherche de cadencement des calculs et d'une allocation de ces calculs à une grille de processeurs. De nombreuses méthodes de conception automatique d'architectures systoliques ont été proposées dans les années quatre-vingts, et ont parfois donné lieu à un développement logiciel : *Diastol* développé par P. Quinton [82], *Systol* développé par C. Mongenet [75] et *Advis* développé par D.I. Moldovan [78]. Certaines de ces méthodes et logiciels ont depuis été étendus à une parallélisation des équations récurrentes pour des machines cibles plus générales : *Opéra* de V. Loechner et C. Mongenet [68, 67] ou *Alpha* initié par P. Quinton, S. Rajopadhye et D.K. Wilde [99].

Initialement, ces méthodes ne considéraient que des cadencements monodimensionnels ainsi qu'une allocation des calculs d'un espace d'itération de dimension n vers un espace de dimension $n-1$ de processeurs. C'est sur ces hypothèses de construction d'architectures systoliques que se basent nos travaux sur les allocations non-linéaires présentés au chapitre 2. Les réseaux de processeurs synthétisés par les méthodes d'allocations linéaires présentent d'excellentes qualités de régularité et de localité des connexions. Mais une étude de l'activité des processeurs montre une activité simultanée que pour un faible sous-ensemble de ces processeurs. De plus, les programmes exécutés par les processeurs ont un grain de parallélisme trop fin pour être réaliste dans le cas de machines parallèles. C'est pourquoi nous avons proposé plusieurs méthodes d'allocations non-linéaires des calculs permettant de maintenir de bonnes propriétés de régularité et de localité et permettant de réduire le nombre de processeurs alloués, sans ralentir le programme parallèle.

Chapitre 2

Algorithmes systoliques et allocations non-linéaires

L'objectif initial de ces travaux concerne l'optimisation des réseaux systoliques, où seuls des cadencements monodimensionnels et des allocations vers un espace de dimension $n-1$ sont classiquement appliqués, n étant la dimension de l'espace d'itération. Nous nous limiterons donc dans cette section à des cadencements et allocations de ce type, bien que les notions présentées soient généralisables à des cadencements et allocations de dimensions quelconques.

À la fin des années quatre-vingts, plusieurs travaux ont proposé des solutions "optimales" en nombre de processeurs pour des algorithmes particuliers [7, 8, 60, 87, 88]. D'autres ont proposé des méthodologies de synthèse d'architectures minimales en nombre de processeurs [95, 100, 15, 24, 32, 12]. Certaines de ces méthodes [12, 32, 24] sont identiques et ne se distinguent que par la modélisation de l'activité des processeurs et le mode d'application. Elles s'apparentent à notre méthode du groupement présentée à la section 2.3.

Nous rappelons tout d'abord la modélisation de l'activité sur laquelle se basent les méthodes d'allocations non-linéaires présentées ensuite. Cette modélisation ainsi que les méthodes de sérialisation et de groupement ont été initialement présentées dans ma thèse de doctorat [15] et dans [24, 27]. Elles ont été enrichies depuis d'un mode plus formel d'application, ayant pour but la synthèse de réseaux de processeurs réguliers à connexions locales, et la détermination des fonctions d'allocations non-linéaires associées [16, 25, 26, 23, 17, 77]. C'est dans ce cadre qu'une troisième méthode, la méthode du décalage d'instructions, a également été développée [28].

2.1 Modélisation de l'activité

On considère ici un réseau de processeurs synthétisé selon la méthode décrite à la section précédente. La modélisation de l'activité des processeurs comporte deux aspects :

1. La durée d'activité : il s'agit ici du temps séparant la date de première et la date de dernière activation d'un processeur.

2. L'efficacité : les réseaux systoliques étant basés sur l'hypothèse d'une exécution synchrone du réseau rythmé par une horloge globale, et que le temps de calcul d'une itération est

d'un cycle d'horloge, on évalue ici le rapport entre le nombre de cycles passés à un calcul et les cycles d'inactivité.

Le domaine d'itération de l'algorithme définit un polyèdre convexe P caractérisé par un système d'inégalités linéaires. En remplaçant ces inégalités par des égalités, on obtient les équations des hyperplans contenant les facettes de P.

L'allocation linéaire des points de calcul peut être vue comme un partitionnement de P en droites parallèles au vecteur de projection $\overrightarrow{p}$ appelées *droites d'allocation*. Tous les points appartenant à une même droite correspondent à tous les calculs exécutés sur un même processeur. Par définition des ensembles convexes, toute droite qui traverse P intersecte au moins deux facettes de P. Les deux points entiers extrêmes des segments de ces droites, contenus entièrement dans P, sont associés aux calculs de première et de dernière activations du processeur correspondant. Les points d'intersection entre les droites d'allocation et les facettes de P n'étant pas nécessairement entiers, il s'agit de considérer les points entiers les plus extrêmes des segments de ces droites. Ces points extrêmes définissent les facettes épaisses. L'épaisseur de ces facettes est déterminée en effectuant le produit scalaire de $\overrightarrow{p}$ et des vecteurs $\overrightarrow{F_q}$ orthogonaux aux facettes de P. De plus, le système d'inégalités linéaires définissant une facette épaisse peut aisément être déduit de celui définissant P, en ajoutant une contrainte correspondant à la facette associée à laquelle on ajoute la valeur de l'épaisseur en partie constante de l'inégalité.

On appelle *facettes de premières activations* les facettes épaisses contenant les points de calculs associés aux premières activations des processeurs, et *facettes de dernières activations* les facettes épaisses contenant les points de calculs associés aux dernières activations des processeurs. Les intersections entre de telles facettes et un front de calcul définissent les points de calcul associés à des premières, ou dernières, activations simultanées. On détermine pour chaque épaisseur des facettes de premières (respectivement dernières) activations un vecteur $\overrightarrow{\mathcal{F}_q}$ (respectivement $\overrightarrow{\mathcal{L}_q}$) définissant une base unimodulaire de son intersection avec un front de calcul. L'application de la fonction d'allocation *alloc* à tous ces vecteurs génère des droites sur le réseau de processeurs, pour une instance donnée t_0 des fronts de calculs définie par l'équation $t(\overrightarrow{I}) = t_0$. Ces droites caractérisent les processeurs de premières, ou dernières, activations simultanées.

On représente les droites de premières activations d'un instant t_0, et les droites de dernières activations à $t_0 - 1$. Chaque paire composée d'une droite de premières et d'une droite de dernières activations définit un cône ou deux droites parallèles, dans le cas particulier où les facettes de premières et dernières activations considérées sont parallèles. Plus généralement dans le cas de facettes épaisses, chaque paire est composée de l'ensemble des droites associées à une facette épaisse de premières activations, et de l'ensemble des droites associées à une facette épaisse de dernières activations. On définit ainsi des *cônes épais*, ou deux *droites épaisses* parallèles. Les processeurs situés à l'intérieur du cône, ou entre les droites parallèles, représentent les processeurs actifs à l'instant t_0. On parle alors de *cônes d'activité*. De manière générale, le cône d'activité définit trois ensembles de processeurs selon le schéma de la figure 2.1 : les processeurs qui ne seront plus actifs, les processeurs actifs, et les processeurs non encore activés.

On démontre [15] que la position des cônes d'activité de n'importe quel cycle t_0 est déterminée par une translation, de vecteur constant, des sommets des cônes.

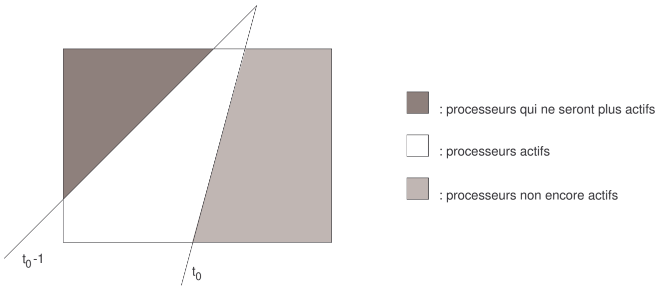

FIG. 2.1: *Un cône d'activité sur un réseau de processeurs*

L'efficacité des processeurs est déterminée par le produit $\overrightarrow{p} \cdot \overrightarrow{\lambda}$: ce produit définit le nombre de cycles d'inactivité séparant deux activations successives. Ainsi, un processeur du réseau est caractérisé par le ratio $R_{ef} = \frac{1}{\overrightarrow{p} \cdot \overrightarrow{\lambda}}$.

Ce ratio conditionne également notre modélisation en cônes d'activité. En effet, afin de respecter le rythme d'activité des processeurs, les cônes sont définis plus généralement par les premières activations d'un cycle t_0, et par les dernières activations non pas du cycle $t_0 - 1$, mais du cycle $t_0 - \left(\overrightarrow{p} \cdot \overrightarrow{\lambda}\right)$.

Cette modélisation met en évidence une faible utilisation des réseaux de processeurs synthétisés par les méthodes classiques décrites à la section précédente. C'est pourquoi, à partir de ces deux critères de durée d'activité et d'efficacité, nous développons trois méthodes d'allocations non-linéaires des calculs. Nous en présentons des versions constructives permettant de synthétiser des architectures régulières à connexions locales : la méthode de sérialisation, la méthode du groupement et la méthode du décalage d'instructions.

2.2 La méthode de sérialisation

Cette méthode se base uniquement sur la durée d'activation des processeurs et consiste à allouer à un même processeur les calculs effectués par des processeurs ayant des durées d'activation disjointes. La modélisation en terme de cônes d'activité permet de caractériser les processeurs candidats au regroupement sur un même processeur. On choisira des processeurs situés sur les arêtes des cônes, afin de minimiser le nombre de processeurs résultants. Nous avons montré [15] que, si l'efficacité des processeurs est de 1, cette méthode permet de déterminer des architectures optimales en place, c'est-à-dire où le nombre de processeurs est égal au nombre maximum d'activations simultanées. Chaque processeur résultant exécute alors successivement les calculs de chacun des processeurs dont il est issu.

Des nombreuses possibilités de regroupements, on ne retiendra que celles produisant une architecture régulière de processeurs interconnectés localement (entre voisins). Le respect de cette contrainte de régularité ne permet pas d'obtenir systématiquement un nombre de processeurs minimum, mais nous conservons une propriété d'*optimalité asymptotique*.

Notre approche constructive est basée sur la constatation que si les directions de sérialisations sont parallèles et constantes, alors le réseau résultant est régulier et toroïdal. Les

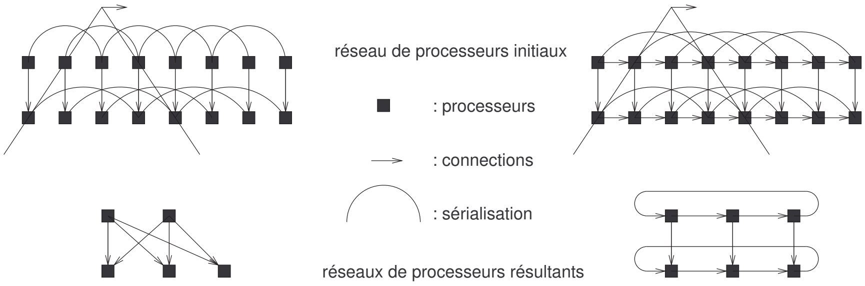

FIG. 2.2: *Sérialisation constructive et connexions toroïdales*

connexions toroïdales sont issues des connexions initiales entre les processeurs d'une même direction de sérialisation, comme nous le montrons à la figure 2.2 dans le cas d'un seul cône d'activité.

Plus formellement, les étapes de la méthode sont les suivantes :

1. Partitionnement de l'architecture initiale en segments π_q de droites parallèles selon une direction $\overrightarrow{\pi}$.

2. Pour chaque segment π_q, calcul du nombre maximum p_q de processeurs situés à l'intérieur des cônes d'activité.

3. Calcul du maximum de ces valeurs $p_{max} = MAX_q\ p_q$.

4. Pour chaque segment, sérialisation de longueur p_{max}.

La première étape consiste à déterminer une direction $\overrightarrow{\pi}$ de partitionnement. Soit np le nombre de segments de droites π_q. Le nombre total de processeurs résultant de la sérialisation est alors égal à $np \times p_{max}$. Un bon choix peut être celui qui minimise ce nombre.

Le calcul des valeurs p_q est effectué en déterminant le nombre maximum de points entiers situés à l'intersection entre l'hyperplan du domaine d'itération dont la projection est le segment π_q, et les fronts de calculs. On utilise ici notre méthode de calcul des polynômes d'Ehrhart décrite dans la section suivante. Un calcul similaire du parallélisme maximum est décrit dans [22].

Dans le but de minimiser le nombre de connexions de type toroïdal dans l'architecture résultante, notre approche consiste à déterminer un partitionnement de l'architecture initiale en parallélogrammes de volumes $np \times p_{max}$. Ces parallélogrammes sont générés par le vecteur $\overrightarrow{\pi}$ et d'autres vecteurs (un seul en dimension 2) dont le choix influence le type de connexions inter-processeurs comme cela est illustré par la figure 2.3. Les parallélogrammes définissent alors la forme de l'architecture finale.

On note $\overrightarrow{c}$ les coordonnées d'un processeur donnant sa position dans l'architecture initiale. Soit Π une matrice de vecteurs formant une base orthogonale avec $\overrightarrow{\pi}$. Tout processeur appartient à un segment de droite π_q où $q = \Pi \cdot \overrightarrow{c}$. Soient $alloc_{min}(q)$ et $alloc_{max}(q)$ les coordonnées des processeurs situés sur les bords d'un parallélogramme du partitionnement pour

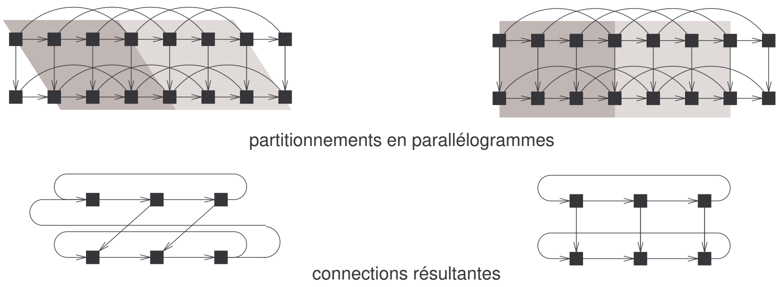

FIG. 2.3: *Influence des parallélogrammes sur le nombre de connexions toroïdales*

un segment de droite π_q donné. Elles sont définies par :

$$\begin{cases} alloc_{min}(q) & = & \overrightarrow{c_{min}} + (q - q_{min}) \cdot \Omega \\ alloc_{max}(q) & = & alloc_{min}(q) + (p_{max} - 1)\overrightarrow{\pi} \end{cases}$$

où $\overrightarrow{c_{min}}$ sont les coordonnées du premier processeur de la bordure du parallélogramme, situé sur le premier segment π_q, où q_{min} est l'indice de ce premier segment, et où Ω est la matrice des vecteurs colonnes définissant avec $\overrightarrow{\pi}$ le parallélogramme.

Finalement, la fonction d'allocation de la sérialisation est :

$$serialise(c) = c - ((c - alloc_{min}(q)) \div p_{max}) \times p_{max}$$

Remarquons que les coordonnées c résultent de l'allocation de points de l'espace d'itération, c'est-à-dire $c = alloc(\overrightarrow{I})$, et que la fonction d'allocation de la sérialisation peut donc être exprimée directement à partir de ces points :

$$alloc\_serialise(\overrightarrow{I}) = alloc(\overrightarrow{I}) - ((alloc(\overrightarrow{I}) - alloc_{min}(\Omega \cdot alloc(\overrightarrow{I}))) \div p_{max}) \times p_{max}$$

Les connexions entre les processeurs dans l'architecture finale sont les mêmes que dans l'architecture initiale, excepté pour les processeurs situés aux bords et dont les connexions initiales ont une direction non parallèle aux vecteurs définissant les parallélogrammes de partitionnement. Soit $\overrightarrow{m}_{init}$ le vecteur définissant une connexion initiale et soit $\overrightarrow{m}_{serial}$ le vecteur définissant la même connexion dans l'architecture finale.

- Si $\overrightarrow{m}_{init}$ est parallèle à un des bords des parallélogrammes, alors $\overrightarrow{m}_{serial} = \overrightarrow{m}_{init}$.
- Sinon, si le processeur considéré, de coordonnées c, appartient à un bord d'un parallélogramme,
  - alors $\overrightarrow{m}_{serial} = alloc_{min}(\Omega \cdot c + \Omega \cdot \overrightarrow{m}_{init}) - alloc_{max}(\Omega \cdot c)$
  - sinon $\overrightarrow{m}_{serial} = \overrightarrow{m}_{init}$.

Chaque processeur résultant de la sérialisation possède une durée d'activation qui est la somme des durées d'activation des processeurs initiaux dont il est issu. Toutefois, l'efficacité des processeurs initiaux est conservée par cette méthode. Les processeurs résultants peuvent donc être régulièrement inactifs entre leurs premières et dernières activations. La méthode du groupement présentée dans la suite permet de rendre actifs les processeurs à tous les cycles de leur durée d'activation, et de réduire encore leur nombre.

2.3 La méthode du groupement

Prenons l'hypothèse d'une durée d'activation infinie des processeurs. Soit np le nombre total de ces processeurs. À un certain cycle t_0, seulement un nombre de l'ordre de $np \times R_{ef}$ processeurs sont actifs. Au cycle $t_0 + 1$, un nombre du même ordre d'autres processeurs, sont actifs à leur tour, et ainsi de suite jusqu'au cycle $t_0 + \overrightarrow{p} \cdot \overrightarrow{\lambda} - 1$. Ensuite, les processeurs actifs du cycle t_0 sont à nouveau actifs à $t_0 + \overrightarrow{p} \cdot \overrightarrow{\lambda}$. Par conséquent, on distingue $\overrightarrow{p} \cdot \overrightarrow{\lambda}$ ensembles de processeurs : ceux qui sont actifs aux cycles t tels que $t \ mod \ (\overrightarrow{p} \cdot \overrightarrow{\lambda}) = 0$, ceux qui sont actifs aux cycles t tels que $t \ mod \ (\overrightarrow{p} \cdot \overrightarrow{\lambda}) = 1, \ldots,$ ceux qui sont actifs aux cycles t tels que $t \ mod \ (\overrightarrow{p} \cdot \overrightarrow{\lambda}) = \overrightarrow{p} \cdot \overrightarrow{\lambda} - 1$. On note ces ensembles $A_0, A_1, \ldots, A_s$ avec $s = \overrightarrow{p} \cdot \overrightarrow{\lambda} - 1$. A. Darte dans [32] propose une modélisation de ce phénomène en terme de *matrice d'activité*.

La méthode du groupement consiste à fusionner en un seul processeur, $(\overrightarrow{p} \cdot \overrightarrow{\lambda})$ processeurs initiaux, en en prenant un dans chacun des ensembles A_q. Une méthode constructive consiste à regrouper uniquement des processeurs initiaux voisins, et d'effectuer toujours les mêmes types de regroupements, afin de conserver la régularité et la localité des connexions dans l'architecture résultat. Cette méthode est appelée *passive clustering* par J. Bu *et al.* dans [12].

Les regroupements possibles dans l'architecture initiale sont définis par des motifs de regroupement construits de la manière suivante :

1. décomposition du vecteur d'allocation $\overrightarrow{p}$ en une somme de vecteurs $\overrightarrow{p} = \overrightarrow{p_0} + \overrightarrow{p_1} + \ldots + \overrightarrow{p_s}$, ou tout vecteur $\overrightarrow{p_q}$ est tel que $\overrightarrow{p_q} \cdot \overrightarrow{\lambda} = 1$.

2. application de la fonction d'allocation aux vecteurs $\overrightarrow{p_q}$. A partir de n'importe quel processeur initial, le parcours des vecteurs $alloc(\overrightarrow{p_q})$, dans un ordre quelconque, décrit un motif de regroupement possible.

Toute décomposition de $\overrightarrow{p}$ permet de définir un ou plusieurs motifs. On minimise le nombre de processeurs en résolvant un problème de pavage à partir des motifs ainsi définis de l'espace des processeurs initiaux. Les qualités de régularité et de localité sont garanties en n'utilisant qu'un seul motif sur l'ensemble de l'architecture initiale. Les qualités d'adéquation entre motifs et forme de l'architecture initiale peuvent amener à des processeurs résultats dont l'efficacité est inférieure à 1 : il s'agit des processeurs situés en bordure de l'architecture initiale. A. Darte dans [32] se limite à des motifs de forme parallépipédiques.

Dans le cas de l'utilisation d'un seul motif, la fonction d'allocation est aisée à déterminer. On partitionne l'espace d'itération P en $(\overrightarrow{p} \cdot \overrightarrow{\lambda})$ réseaux réguliers, ou lattices, de points. Chacun de ces lattices correspond aux calculs alloués aux processeurs d'un même ensemble A_q. Ils sont caractérisés par la valeur modulo $(\overrightarrow{p} \cdot \overrightarrow{\lambda})$ de leur composante temporelle $\left(\overrightarrow{I} \cdot \overrightarrow{\lambda}\right)$. À partir de la fonction d'allocation initiale *alloc*, on détermine une fonction d'allocation par lattice en ajoutant des constantes à la définition de la fonction. Ainsi, plusieurs points alloués initialement à des processeurs différents, puis regroupés par le motif choisi, seront alloués au même processeur résultat.

A. Darte [32] et J. Bu *et al.* [12] (*active clustering*) ont également proposé l'application de cette méthode à l'allocation des calculs à un nombre fixé de processeurs. Dans ce cas, un ralentissement du programme est bien sûr indispensable. Elle correspond alors à une méthode de partitionnement de type *LSGP* pour "localement séquentiel, globalement parallèle", qui s'oppose aux méthodes de type *LPGS*.

Bien que s'apparentant aux méthodes actuelles de partitionnement, ou *tiling*, des domaines d'itérations [55], par l'utilisation de motifs de recouvrement, la méthode du groupement ne considère pas les mêmes objectifs. Nous cherchons ici à maximiser l'utilisation des processeurs, alors que les méthodes de partitionnement cherchent à minimiser les volumes de communications entre processeurs, en regroupant des itérations pouvant s'exécuter sans communiquer.

2.4 La méthode du décalage d'instructions

Les deux méthodes rappelées ci-dessus abordent chacune un critère particulier d'activité, à savoir respectivement la durée d'activation et l'efficacité. Elles peuvent par conséquent être appliquées simultanément, afin de réduire au maximum le nombre de processeurs de l'architecture finale. Ce nombre peut-être optimal, c'est-à-dire égal au nombre maximum de calculs simultanés, mais cela sans garantir la régularité et la localité des connexions inter-processeurs ; ou quasi-optimal, dans le cas d'architectures finales régulières à connexions toroïdales.

La méthode du décalage d'instructions aborde simultanément les deux critères d'activité, et permet de produire des architectures régulières à connexions locales, sans connexion toroïdale. Elle se base également sur la modélisation en cônes d'activité de l'architecture initiale, obtenue par projection linéaire des calculs. L'idée de cette méthode est d'inverser l'architecture et le cône. Ainsi, la surface de processeurs décrite par le cône est l'architecture finale, optimale ou quasi-optimale en nombre de processeurs selon que l'on ait un objectif de régularité ou non. Les méthodes présentées précédemment consistent en une forme de "compression" de l'architecture initiale, alors que la méthode du décalage d'instructions ré-alloue les calculs différemment.

L'architecture initiale est tout d'abord partitionnée selon des segments de droites parallèles, dont le vecteur directeur $\vec{\pi}$ est appelé *direction de partitionnement*. On observe le déplacement des cônes d'activité sur l'architecture initiale durant une exécution, et sur un segment de droite particulier : lorsqu'un processeur apparaît pour la première fois à l'intérieur d'un cône, son calcul exécuté à cet instant est alloué au premier processeur résultat dans la direction de $\vec{\pi}$. Ses calculs suivants sont alloués au même processeur résultat, jusqu'à ce qu'un autre processeur du même segment de droite apparaisse dans le cône. À ce moment là, ses calculs sont alloués au deuxième processeur résultat de cette direction et ainsi de suite.

Par conséquent, cette méthode est basée sur une distribution, ou décalage, des calculs d'un processeur initial sur toute une ligne de processeurs résultats. On peut noter que le voisinage d'un processeur initial est quasiment identique pour les processeurs résultats et qu'ainsi la localité des connexions est garantie. Plus formellement, les étapes de la méthode sont :

1. partitionnement de l'architecture initiale en segments de droites parallèles de vecteur directeur $\vec{\pi}$,

2. pour chaque segment S_q, calcul du nombre maximum de processeurs actifs simultanément $act(S_q)$,

3. pour chaque segment S_q, décalage et allocation des calculs sur $act(S_q)$ processeurs.

La détermination de la direction de partitionnement $\vec{\pi}$ peut être conditionnée par la recherche du nombre minimum de processeurs résultats. Il s'agit alors de rechercher une direction

$\overrightarrow{\pi}$ pour laquelle la somme pour tous les segments S_q du nombre maximum de processeurs actifs simultanément, $\Sigma_q act(S_q)$, est minimale. Le calcul des quantités $act(S_q)$ est effectué en déterminant le nombre maximum de points entiers situés à l'intersection entre l'hyperplan du domaine d'itération dont la projection est le segment de droite S_q et les fronts de calculs. On utilise ici notre méthode de calcul des polynômes d'Ehrhart décrite dans la section suivante.

Si n est la dimension du domaine d'itération P, alors l'architecture initiale et l'architecture résultat sont de dimension $n-1$. Par conséquent, les coordonnées d'un processeur dans l'espace des processeurs résultats peuvent être exprimées dans une base orthogonale $(\overrightarrow{\pi}, \overrightarrow{\pi}_2, \ldots, \overrightarrow{\pi}_{n-1})$. L'indice q de tout segment S_q est le $(n-2)$-uplet $(\overrightarrow{\pi}_2 \cdot c, \ldots, \overrightarrow{\pi}_{n-1} \cdot c)$, où c désigne les coordonnées des processeurs initiaux dans la base initiale.

La première activation d'un processeur initial correspond à un point entier d'une facette de premières activations notée F_i. Désignons par $proc_i$ un tel processeur. L'allocation d'un calcul de $proc_i$ à un processeur résultat dépend de sa position dans son cône d'activité le long de son segment S_q, au cycle d'exécution du calcul considéré. On démontre dans [28] la proposition suivante :

Proposition 1 *La position d'un processeur initial $proc_i$, le long de son segment S_q et à l'intérieur de son cône d'activité, pour un certain cycle d'exécution t_0, est défini par :*

$$pos(proc_i, t_0) = (t_0 - f(proc_i)) \div \left(\overrightarrow{\lambda} \cdot \overrightarrow{\pi}_{F_i}\right) + 1$$

où

- *$\overrightarrow{\pi}_{F_i}$ est le vecteur directeur de l'intersection entre la facette de premières activations F_i et l'hyperplan du domaine d'itération dont la projection est le segment de droite S_q.*

- *$f(proc_i)$ est le cycle de première activation de $proc_i$, défini par :*

$$f(proc_i) = t\left(\left(\begin{array}{c} \sigma \\ \overrightarrow{F}_i \end{array}\right)^{-1} \cdot \left(\begin{array}{c} proc_i \\ b_i \end{array}\right)\right)$$

  où $\overrightarrow{F}_i$ et b_i sont tels que $\overrightarrow{F}_i \cdot \overrightarrow{I} = b_i$ est l'équation de l'hyperplan supportant la facette F_i.

On peut maintenant définir la fonction d'allocation des calculs des processeurs initiaux aux processeurs finaux par :

$$iDec(proc_i, t_0) = (pos(proc_i, t_0), q)$$

Notons que puisque les coordonnées $proc_i$ résultent de l'allocation de points de l'espace d'itération, c'est-à-dire $proc_i = alloc(\overrightarrow{I})$, et que t_0 est le cycle d'exécution du point de l'espace d'itération associé au calcul considéré, c'est-à-dire $t_0 = t(\overrightarrow{I})$, la fonction d'allocation peut directement être exprimée sur ces points $\overrightarrow{I}$. Cette nouvelle expression définit alors une fonction d'allocation non-linéaire de l'espace d'itération.

L'étape suivante consiste à synthétiser les connexions inter-processeurs dans l'architecture finale. Soient $proc_{out}$ un processeur émetteur de données et t_{out} un cycle d'émission d'une donnée, et soient $proc_{in}$ un processeur récepteur associé et t_{in} son cycle de réception. Une connexion entre ces deux processeurs est définie comme étant un couple $(\vec{c}, t_{\vec{c}})$ avec $\vec{c} = proc_{in} - proc_{out}$ et $t_{\vec{c}} = t_{in} - t_{out}$.

L'application de la fonction d'allocation $iDec$ à toute connexion initiale $(\vec{c}_{init}, t_{\vec{c}_{init}})$ résulte en un ensemble de connexions résultats $(\vec{c}_{res}, t_{\vec{c}_{res}})$. Puisque la méthode d'allocation n'affecte pas le cadencement initial, $t_{\vec{c}_{res}} = t_{\vec{c}_{init}} = t_{in} - t_{out}$. De plus, on a $\vec{c}_{res} = iDec(proc_{in}, t_{in}) - iDec(proc_{out}, t_{out})$. On démontre dans [28] la proposition suivante :

Proposition 2 *Les connexions inter-processeurs dans l'architecture finale sont déterminées par* $\vec{c}_{res} = (\vec{c}_{res_1}, \vec{\pi}_2 \cdot \vec{c}_{init}, \ldots, \vec{\pi}_{n-1} \cdot \vec{c}_{init})$*, où* $\vec{c}_{res_1}$ *prend toutes les valeurs entières définies par l'intervalle fermé :*

$$\left\lceil \frac{t_{\vec{c}_{init}} - \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ \vec{F}_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{c}_{init} \\ 0 \end{pmatrix} + 1}{\vec{\lambda} \cdot \vec{\pi}_{F_i}} \right\rceil - 1 .. \left\lfloor \frac{t_{\vec{c}_{init}} - \vec{\lambda} \cdot \begin{pmatrix} \sigma \\ \vec{F}_i \end{pmatrix}^{-1} \cdot \begin{pmatrix} \vec{c}_{init} \\ 0 \end{pmatrix} + 1}{\vec{\lambda} \cdot \vec{\pi}_{F_i}} \right\rfloor + 1$$

Dans les cas de circuits VLSI ou FPGA, les données d'entrée sont organisées en flots réceptionnés par des processeurs situés sur les bords du réseau. Cette organisation des données doit correspondre à l'exécution synchrone du réseau de processeurs ainsi qu'au cadencement des calculs. La méthode doit donc déterminer une nouvelle organisation des flots d'entrée de données pour l'architecture résultat.

Comme cela est spécifié dans la proposition 1, les calculs d'un processeur initial sont alloués aux processeurs résultats en blocs de $\vec{\lambda} \cdot \vec{\pi}_{F_i}$ calculs. Si l'on considère un processeur initial recevant des données, des blocs de $\phi = \vec{\lambda} \cdot \vec{\pi}_{F_i}$ données doivent être reçus par les processeurs résultats.

Considérons deux blocs successifs de ϕ données, reçus par un seul processeur initial, et devant être reçus par deux processeurs résultats voisins. Le processeur recevant le premier bloc de données le réceptionne aux cycles θ_1, $\theta_1 + 1$, ..., $\theta_1 + \phi - 1$. Le processeur recevant le deuxième bloc de données doit alors le réceptionner à partir du cycle $\theta_2 = \theta_1 + \phi$. Si l'on choisit d'introduire les données dans le réseau selon la direction de partitionnement $\vec{\pi}$, on observe que la dernière donnée du premier bloc chevauche la première donnée du second bloc, ce qui n'est pas satisfaisant. Par contre, en choisissant la direction opposée à $\vec{\pi}$, les deux blocs doivent être espacés d'un cycle de retard, ce qui est satisfaisant.

Chaque segment S_q résulte en $act(S_q)$ processeurs. Étant donné que ce nombre dépend du segment considéré, les calculs des processeurs initiaux récepteurs de données sont alloués à un nombre variable de processeurs résultats. Afin d'assurer une bonne synchronisation, des cycles de retard doivent être ajoutés en début de certains flots d'entrées. Pour un flot associé à un segment S_q, le nombre de cycles de retard à ajouter est égal à $act - act(S_q)$, où act est le maximum des $act(S_q)$.

Analysons l'optimalité des architectures résultats. La méthode n'affecte en rien l'ordonnancement temporel des calculs de la solution initiale. Par conséquent, si celle-ci est optimale en temps, l'algorithme parallèle résultat de la méthode l'est également. C'est uniquement pour l'optimalité en place que la méthode intervient.

Par définition de la méthode, tous les calculs d'un processeur initial $proc_i$ sont alloués à un nombre $\left\lceil \frac{d(proc_i)}{\vec{\lambda} \cdot \vec{\pi}_{F_i}} \right\rceil$ de processeurs résultats, où $d(proc_i)$ désigne le nombre de calculs, ou la durée d'activité, du processeur initial. Si l'on considère tout un segment S_q de processeurs initiaux $proc_i$, l'allocation de leurs calculs nécessite $np_q = MAX_i \left\lceil \frac{d(proc_i)}{\overrightarrow{\lambda} \cdot \overrightarrow{\pi}_{F_i}} \right\rceil$ processeurs résultats. Si $act(S_q) < np_q$, le nombre de processeurs résultats pour le segment S_q n'est pas optimal. Le cas le plus défavorable est celui où le nombre de processeurs initiaux d'un segment S_q est inférieur à np_q : dans ce cas, la méthode augmente le nombre de processeurs. Les cas les plus favorables sont ceux où les segments S_q sont tels que $act(S_q) \geq np_q$: le nombre de processeurs résultats de chaque segment S_q est optimal, sans que l'architecture soit nécessairement optimale dans sa globalité. Mais dans ces cas, le ratio np/pp, où $np = \Sigma_q np_q$ et pp est le nombre maximum de calculs simultanés, tend vers une valeur constante lorsque l'on fait croître les paramètres de taille du problème traité. Il s'agit alors d'architectures asymptotiquement optimales, représentant un bon compromis entre optimalité et régularité.

2.5 Conclusion

Bien que dédiées à l'optimisation d'architectures systoliques, ces méthodes pourraient trouver de nouveaux champs d'application pour le placement des données et des calculs sur des machines parallèles plus générales. Elles devraient alors être redéfinies afin de prendre en compte d'autres contraintes telles que la localité des accès mémoire.

La considération de paramètres de taille dans les problèmes traités nécessite des évaluations symboliques du parallélisme potentiel, du nombre de processeurs résultants, de la latence de l'algorithme, ... Nos techniques d'allocations non-linéaires utilisaient initialement des méthodes peu générales pour ces évaluations. Ce manque a motivé nos recherches sur une méthode d'évaluation symbolique du nombre de points entiers dans un polytope paramétré, ainsi que dans la projection affine d'un polytope, présentés au chapitre suivant.

Chapitre 3

Polynômes d'Ehrhart et polyèdres paramétrés

La modélisation des équations récurrentes et des nids de boucles par des polyèdres ou polytopes convexes permet l'utilisation d'outils mathématiques existants portant sur la géométrie convexe et l'arithmétique en nombres entiers. Toutefois, la résolution de problèmes informatiques spécifiques a parfois nécessité de la part des chercheurs un travail de recherche mathématique plus général. En effet, de nombreux problèmes informatiques se réduisent souvent à un seul problème mathématique plus général. Citons notamment dans ce cadre les recherches menées par P. Feautrier [41] sur la programmation linéaire en nombres entiers, celles de W. Pugh sur les simplifications des formules de Presburger [80], ou celles de D.K. Wilde sur les opérations géométriques appliquées aux polyèdres convexes [98].

Nos travaux sur les polyèdres paramétrés et les polynômes d'Ehrhart répondent également à ce critère de problème général mathématique. En effet, de nombreux calculs liés à l'analyse de performance, à l'analyse de programmes et aux choix de transformations de programmes, se réduisent au problème mathématique général de dénombrement de points entiers contenus dans un polytope convexe. De plus, les programmes destinés à la parallélisation sont en général des applications scientifiques manipulant de très importants volumes de données, variant à chaque exécution. Ceci implique l'utilisation de paramètres de taille des problèmes, et donc nécessite une résolution symbolique. Un tel raisonnement s'avère également indiqué dans le but de conditionner les transformations d'optimisation et de parallélisation d'un programme, notamment selon des contraintes liées aux caractéristiques de la machine cible : nombre de processeurs, type d'architecture, taille mémoire, organisation et taille des mémoires caches, type de processeur, ...

Des travaux antérieurs au nôtre ont eu le même objectif d'une méthode de dénombrement automatique. M. Haghighat et C. Polychronopoulos présentent dans [52, 53] une méthode de calcul de volume. Les résultats donnés par leur méthode ont une forme complexe composée de fonctions *min* et *max*. Ferrante, Sarkar et Thrash présentent dans [46] une méthode de calcul du nombre de locations mémoires et de lignes de cache référencées par une boucle. Ils ne considèrent pas les unions de polytopes, leurs résultats ne sont pas symboliques et ne constituent que des approximations. La méthode développée par Pugh [81] consiste en un ensemble de techniques dédiées à la simplification des équations initiales. Leur application est souvent intuitive et nécessite l'utilisation de formules standards sur les sommes de puissances de nombres. Tawbi et Feautrier [93, 94] utilisent une technique de partitionnement du domaine d'itération et d'élimination de variables dans un ordre prédéfini. Leurs résultats ne sont pas systématiquement exacts, mais des approximations.

Nos résultats décrits dans ce chapitre ont donné lieu aux publications [18, 21, 19, 22, 20], où plusieurs exemples d'applications sont présentés. Grâce à leur caractère général, ils sont aujourd'hui largement utilisés dans la communauté scientifique informatique, et servent de support au développement de méthodes dédiées à divers problèmes de parallélisation et d'optimisation de programmes. Citons par exemple S. Ghosh *et al.* [48, 49], qui modélisent les défauts de cache d'un programme par une équation en nombres entiers appelée *cache miss equation*, et qui dénombrent ses solutions par un polynôme d'Ehrhart ; citons M. Cosnard *et al.* [31], P. Boulet et X. Redon [10], Loechner et Mongenet [71], qui déterminent des volumes de communications par des polynômes d'Ehrhart ; citons également Heine et Slowik [54], qui caractérisent les placements de données sur des machines de type NUMA par la localité des accès mémoires, modélisés par des polynômes d'Ehrhart. Bien d'autres utilisations peuvent être envisagées. Nous nous attachons nous-même à montrer la portée des polynômes d'Ehrhart, notamment dans le contexte de l'optimisation des performances de caches, présentée au chapitre 4.

Rappelons tout d'abord quelques notions de géométrie des nombres [50, 89, 9] et de combinatoire énumérative [91, 92].

3.1 Quelques définitions

L'espace affine d'un polyèdre P est le plus petit sous-espace affine de $\mathbb{Q}^d$ qui le contient. La dimension d'un polyèdre P est la dimension de son espace affine. Un polyèdre de dimension k sera appelé un k-polyèdre dans la suite.

Un *lattice* est l'ensemble des points $\mathbb{L}^d = \{\alpha_1\overrightarrow{g}_1 + \ldots + \alpha_d\overrightarrow{g}_d | \alpha_1, \ldots, \alpha_d \in \mathbb{Q}\}$, où $\overrightarrow{g}_1, \ldots, \overrightarrow{g}_d$ sont des vecteurs linéairement indépendants de $\mathbb{Q}^d$ et sont appelés *base du lattice*. L'ensemble de tous les points entiers est appelé *lattice standard* noté $\mathbb{Z}^d$. Soit G la matrice dont les vecteurs colonnes sont les vecteurs $\overrightarrow{g}_1, \ldots, \overrightarrow{g}_d$, le lattice $\mathbb{L}^d$ est constitué des points $G \cdot x, x \in \mathbb{Z}^d$, c'est-à-dire $\mathbb{L}^d = G \cdot \mathbb{Z}^d$. Par conséquent, tout lattice peut être obtenu par une transformation non-singulière quelconque G [50]. Nous ne considérerons donc dans la suite, et sans perte de généralité, que le lattice standard $\mathbb{Z}^d$, tout lattice $\mathbb{L}^d$ pouvant être ramené à $\mathbb{Z}^d$ par la transformation inverse G^{-1}. Le *dénombrant* d'un ensemble S est le nombre de points de $S \cap \mathbb{L}^d$.

L'intersection entre un lattice $\mathbb{L}^d$ et l'enveloppe convexe d'un nombre fini de points x_1, …, x_{m+1} de $\mathbb{Q}^d$ est appelé un *lattice polytope* P de $\mathbb{L}^d$. Si $x_1, \ldots, x_{m+1}$ sont des points affine-indépendants, c'est-à-dire tels que $x_2 - x_1, x_3 - x_1, \ldots, x_{m+1} - x_1$ sont linéairement indépendants, alors l'enveloppe convexe de $x_1, \ldots, x_{m+1}$ est appelé un *lattice simplexe* S de $\mathbb{L}^d$ de dimension m, dont les sommets sont $x_1, \ldots, x_{m+1}$. Un lattice polytope ou un lattice simplexe sont définis par un ensemble d'inégalités linéaires de la forme $\{x \in \mathbb{L}^d | A \cdot x \leq \overrightarrow{b}\}$, où A est une matrice d'entiers de taille $m \times d$ et $\overrightarrow{b}$ est un vecteur d'entiers de taille m.

Si $\overrightarrow{c}$ est un vecteur non nul et γ est défini par $\gamma = MAX\{\overrightarrow{c} \cdot x | A \cdot x \leq \overrightarrow{b}\}$, l'hyperplan affine défini par $\{x | \overrightarrow{c} \cdot x = \gamma\}$ est appelé *hyperplan tangent* à P. Les *faces* de P sont $\emptyset$, P, et les intersections de P avec ses hyperplans tangents. Chaque face de P est elle-même un polyèdre. Si P est un k-polyèdre, les faces de dimensions 0, 1 et $k-1$ sont appelées respectivement *sommets*, *arêtes* et *facettes*.

Nous ne considérons pas uniquement les polyèdres convexes, mais également une certaine classe de polyèdres non-convexes définis par E. Ehrhart sous le nom de *polyèdres normaux* [37, 39]. Un polyèdre normal est un polyèdre fini admettant une division simpliciale rectilinéaire finie. C'est donc l'union d'un nombre fini de simplexes $S_1, S_2, \ldots S_n$ de mêmes dimensions k et tels que :

- l'intersection entre deux simplexes est vide ou est constituée d'une face commune,
- chaque facette d'un simplexe appartient à au plus un autre simplexe, les facettes appartenant à un seul simplexe constituant la bordure du polyèdre,
- les simplexes peuvent être ordonnés de telle manière que pour tout $h < n$, si $P_h = S_1 \cup S_2 \cup \ldots \cup S_h$, l'intersection $P_h \cap S_{h+1}$ est soit vide, soit l'union de facettes communes entre S_{h+1} et la bordure de P_h.

Un lattice polytope P de $\mathbb{L}^d$ est un $\mathbb{L}^d$*-polytope* si tous ses sommets appartiennent à $\mathbb{L}^d$. Si $\mathbb{L}^d = \mathbb{Z}^d$, alors P est un *polytope entier*. Si au moins un sommet de P n'appartient pas à $\mathbb{L}^d$ et possède des coordonnées rationnelles, alors P est un *polytope rationnel*. Puisque, par définition, tout lattice $\mathbb{L}^d$ peut être déduit de $\mathbb{Z}^d$ par une transformation non-singulière G, tout $\mathbb{L}^d$-polytope est équivalent à un polytope entier. Par conséquent, les résultats donnés dans la suite et concernant les polytopes entiers peuvent être directement étendus aux $\mathbb{L}^d$-polytopes. On utilisera donc dans la suite le terme générique de *polytope entier*.

Les polytopes concernés par les polynômes d'Ehrhart ont la particularité de dépendre linéairement de paramètres entiers. Plus formellement, les inégalités linéaires qui les définissent peuvent contenir de tels paramètres dans leur partie constante : $\{x \in \mathbb{Q}^d | A \cdot x \leq B \cdot N + \overrightarrow{c}\}$, où N est un vecteur symbolique de p paramètres, A et B sont des matrices entières constantes de tailles respectives $m \times d$ et $m \times p$, et $\overrightarrow{c}$ est un vecteur constant de m entiers.

3.2 Les polynômes d'Ehrhart

Rappelons tout d'abord certains résultats d'E. Ehrhart afin de pouvoir présenter plus aisément leurs extensions dans la suite. Nous proposons également ici quelques améliorations et généralisations. Ses résultats trouvent leurs origines dans les travaux d'Euler, basés sur les notions de récurrence eulérienne et de fractions génératrices rationnelles [92]. Leurs démonstrations peuvent être trouvées dans [36, 37, 38, 39]. Certains aspects de ses travaux ont été formalisés et étendus par I.G. Macdonald [72, 73] et R.P. Stanley [91, 92].

Définition 3 (fraction rationnelle génératrice) *Une fraction rationnelle génératrice est de la forme :*

$$F(x) = \sum_{n \geq 0} f(n) x^n = \frac{P(x)}{Q(x)}$$

en prenant l'hypothèse que $Q(0) \neq 0$.

Définition 4 (relation de récurrence eulérienne) *Soit le produit :*

$$\Pi(u) = (1-u^{a_1})(1-u^{a_2})\dots(1-u^{a_s}) \qquad (1)$$

où tout $a_i \in \mathbb{N}$*. Lorsqu'on développe et simplifie ce produit, et que l'on remplace chaque terme* u^r *par* u_{n-r}*, on obtient une relation de récurrence linéaire homogène, notée* $\{\Pi(u)\} = 0$ *et appelée relation de récurrence eulérienne. Le terme général* $f(n)$ *d'une telle relation est déterminé par des conditions aux limites et par une fraction rationnelle génératrice* $\frac{P(u)}{\Pi(u)} = \sum_{n\geq 0} f(n)u^n$*.*

Euler a montré que le nombre de solutions entières de tout système de la forme $\sum a_i x_i = n, x_i \geq 0$, est déterminé par le produit $\Pi(1-t^{a_i})$.

Le terme général des récurrences eulériennes peut s'exprimer par des polynômes particuliers appelés *pseudo-polynômes*. Nous donnons ici la définition originale d'Ehrhart, que nous étendons au paragraphe suivant.

Définition 5 (nombre périodique) *Un nombre périodique* $u_n = [u_0, u_1, \dots, u_{p-1}]$ *est égal à l'élément dont le rang est égal à n modulo p, p étant appelé la période de* u_n*.*

$$u_n = \begin{cases} u_0 & si\ n\ modulo\ p = 0 \\ u_1 & si\ n\ modulo\ p = 1 \\ \dots & \\ u_{p-1} & si\ n\ modulo\ p = p-1 \end{cases}$$

Donnons quelques exemples de nombres périodiques :

- Le n-ième nombre décimal du nombre décimal périodique $4, 285\,285\,\dots$ est le nombre périodique $[5, 2, 8]$.
- Le terme général de la série $1 + \frac{1}{3} - \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \frac{1}{11} - \dots = \frac{\pi}{2\sqrt{2}}$ est $\frac{[-1,1,1,-1]}{2n-1}$.
- $(-1)^n = [1, -1]$.
- $cos\left(n\frac{2\pi}{3}\right) = \frac{[2,-1,-1]}{2}$.
- $sin\left(n\frac{\pi}{2}\right) - cos\left(n\frac{\pi}{2}\right) = [-1, 1, 1, -1]$.

Définition 6 (pseudo-polynôme) *Un polynôme* $f(n)$ *est un pseudo-polynôme si certains de ses coefficients sont des nombres périodiques au lieu d'être constants. Le plus petit commun multiple des périodes de ses coefficients périodiques est sa pseudo-période. Un pseudo-polynôme est caractérisé par son degré et sa pseudo-période.*

L'application de ces notions aux polyèdres passe par une classification de ceux-ci en termes de *polyèdres homothétiques* et de *polyèdres bordés*. Nous en donnons ici une version plus générale que celle d'Ehrhart, et en donnerons également une extension au paragraphe suivant.

Définition 7 (polyèdre homothétique) *Soit un polyèdre* P_n *défini par des équations et inéquations linéaires de la forme* $\sum a_i x_i = bn$, $\sum a_i x_i < bn$, $\sum a_i x_i \leq bn$, *où les* a_i *et* b *sont des constantes entières et* n *est un paramètre entier positif. Le polyèdre* P_n *est homothétique si il existe* $n_{min}, n_{max} \in \mathbb{Z}$ *tels que* $\forall\ n_{min} < n \leq n_{max}$, P_n *est le résultat de l'homothétie de* $P_{n_{min}}$ *de centre l'origine du repère* O *et de rapport* n*.*

Cette définition correspond à celle donnée par Ehrhart. Une autre définition équivalente est qu'un polyèdre est homothétique si les coordonnées de ses sommets sont des fonctions linéaires du paramètre n.

Définition 8 (polyèdre bordé) *Soit un polyèdre P_n défini par des équations et inéquations linéaires de la forme $\sum a_i x_i = bn + c$, $\sum a_i x_i < bn + c$, $\sum a_i x_i \leq bn + c$, où les a_i, b et c sont des constantes entières et n est un paramètre entier positif. Le polyèdre P_n est bordé si en enlevant les constantes c des équations et inéquations qui le définissent, on obtient un polyèdre homothétique P'_n, appelé noyau de P_n.*

Une autre définition équivalente est qu'un polyèdre est bordé si les coordonnées de ses sommets sont des fonctions affines du paramètre n. C'est cette notion d'homothétie des polyèdres paramétrés qui permet d'associer leur dénombrant à des relations de récurrences. En général, les polyèdres paramétriques peuvent être définis comme une collection de polyèdres homothétiques ou bordés, définis sur des domaines adjacents du paramètre n. Voyons un exemple simple dû à D.K. Wilde :

Exemple 1 Soient les trois polyèdres paramétrés suivants :

$$\begin{aligned} P_n^1 &= \{x \in \mathbb{Q} | x \geq 0, x \leq n, 2x \leq n + 6\} \\ P_n^2 &= \{x \in \mathbb{Q} | x \geq 0, n \geq 0, 2x \leq n + 6\} \\ P_n^3 &= \{x \in \mathbb{Q} | 0 \leq x \leq n\} \end{aligned}$$

Le polyèdre P_n^1 n'est ni homothétique, ni bordé, mais est égal, selon la valeur du paramètre n, au polyèdre bordé P_n^2 ou au polyèdre homothétique P_n^3, c'est-à-dire :

$$P_n^1 = \begin{cases} P_n^3 \text{ si } 0 \leq n \leq 6 \\ P_n^2 \text{ si } 6 \leq n \end{cases}$$

□

Avant de donner le résultat fondamental d'Ehrhart, il nous faut encore définir les termes de *dénominateur* et de *produit caractéristique* pour les polyèdres rationnels.

Définition 9 (dénominateur) *Le dénominateur d'un polyèdre rationnel est le plus petit commun multiple des dénominateurs de ses sommets.*

Définition 10 (produit caractéristique) *Le produit caractéristique de tout polyèdre rationnel est défini par l'équation* (1)*, où les a_i sont les dénominateurs de ses sommets.*

Le résultat majeur d'Ehrhart est le suivant :

Théorème 1 (théorème d'Ehrhart) *Le dénombrant d_n de tout k-polyèdre homothétique ou bordé P_n, dont le produit caractéristique est $\Pi(t)$, vérifie la relation de récurrence eulérienne $\{\Pi(d)\} = 0$. Par conséquent, d_n est un polynôme en n de degré k si P_n est entier, et est un pseudo-polynôme en n de degré k dont la pseudo-période est le dénominateur de P_n si P_n est rationnel.*

Le dénombrant d_n d'un polyèdre homothétique-bordé peut donc être défini par une relation de récurrence linéaire, et son terme général peut être calculé en utilisant les fractions génératrices rationnelles. Voyons comment peuvent s'effectuer ces calculs sur un exemple simple.

Exemple 2 Soit le rectangle paramétré P_n défini par :

$$P_n = \left\{ x \in \mathbb{Q}^2 | \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 0 \\ \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix} \times n \right\}$$

Les sommets de P_n ont pour coordonnées $(0,0)$, $(\frac{1}{2}n, 0)$, $(0, \frac{1}{2}n)$ et $(\frac{1}{2}n, \frac{1}{2}n)$. Leurs dénominateurs étant $1, 2, 2, 2$, le produit caractéristique de P_n est donc :

$$\Pi(t) = (1-t)(1-t^2)^3 = 1 - t - 3t^2 + 3t^3 + 3t^4 - 3t^5 - t^6 + t^7$$

On en déduit la relation de récurrence $\{\Pi(d) = 0\}$:

$$d_n - d_{n-1} - 3d_{n-2} + 3d_{n-3} + 3d_{n-4} - 3d_{n-5} - d_{n-6} + d_{n-7} = 0$$

Afin de déterminer la fraction génératrice $\frac{P(t)}{\Pi(t)} = \sum_0^\infty d_n t^n$, on utilise le théorème suivant démontré par R.P. Stanley [92].

Théorème 2 *La fraction rationnelle génératrice associée au dénombrant de tout polyèdre homothétique ou bordé est de degré inférieur à 0.*

Par conséquent, étant donné que le dénominateur $\Pi(t)$ de la fraction génératrice pour notre exemple est de degré 7, il suffit de disposer des 7 premiers termes de la somme $\sum_0^\infty d_n t^n$ pour déterminer le numérateur $P(t)$.

En instanciant le rectangle de $n = 0$ à $n = 6$, on compte les valeurs $d_n = 1, 1, 4, 4, 9, 9, 16$. D'où :

$$\frac{P(t)}{\Pi(t)} = \sum_0^\infty d_n t^n = 1 + t + 4t^2 + 4t^3 + 9t^4 + 9t^5 + 16t^6 + \ldots$$

On en déduit la valeur du numérateur :

$$P(t) = \Pi(t) \sum_0^\infty d_n t^n = 1 - t^4$$

Par simplification, on obtient :

$$\frac{P(t)}{\Pi(t)} = \frac{1+t^2}{(1-t^2)^2(1-t)}$$

La décomposition en séries entières de cette fraction résulte en :

$$\sum_0^\infty \frac{(n+2)(n+1)}{4} t^n + \frac{1}{8} \sum_0^\infty t^n + \frac{1}{8} \sum_0^\infty (-1)^n t^n + \frac{1}{4} \sum_0^\infty (-1)^n (n+1) t^n$$

La somme des termes généraux s'exprime aisément sous la forme d'un pseudo-polynôme :

$$d_n = \frac{1}{4}n^2 + [1, \frac{1}{2}]n + [1, \frac{1}{4}]$$

Remarquons que l'expression simplifiée de la fraction génératrice comporte un dénominateur, notons-le $\omega(t)$, définissant une relation de récurrence $\{\omega(d) = 0\}$ de moindre profondeur que $\{\Pi(d) = 0\}$:

$$d_n - d_{n-1} - 2d_{n-2} + 2d_{n-3} + d_{n-4} - d_{n-5} = 0$$

Cette version est appelée *produit caractéristique réduit*, et a été introduite par Ehrhart dans [37].

□

La détermination du produit caractéristique réduit a donné lieu à une conjecture d'Ehrhart [39], démontrée plus tard par R.P. Stanley [90] et P. McMullen [74] :

Théorème 3 *Soit P_n un k-polyèdre homothétique-bordé dont le dénombrant d_n est défini par le pseudo-polynôme :*

$$d_n = c_k(n)n^k + c_{k-1}(n)n^{k-1} + \ldots + c_0(n)$$

où $c_0(n), \ldots, c_k(n)$ sont des nombres périodiques de n. Si pour une valeur $q \in [0, k]$, l'espace affine de chaque face de dimension q de P_n contient un point entier, alors pour tout $r \geq q$, $c_r(n)$ est un coefficient constant, et non périodique. En considérant la plus petite valeur de q vérifiant cette propriété, on appelle grade *du pseudo-polynôme la valeur $g = q - 1$.*

Théorème 4 *Le produit caractéristique réduit associé à un pseudo-polynôme de degré k, de grade g et de pseudo-période p est $(1-t)^{k-g}(1-t^p)^{g+1}$.*

Mais l'utilisation des fractions rationnelles génératrices et leur décomposition en séries entières nous amènent à considérer des fonctions harmoniques (sinus, cosinus, ...), ne correspondant pas exactement au domaine des nombres rationnels. Par conséquent, une autre méthode de calcul des polynômes d'Ehrhart est la suivante : Étant donné le théorème principal d'Ehrhart, le degré et la pseudo-période sont connus, et donc le nombre de coefficients du pseudo-polynôme. Il suffit alors de connaître le nombre de points entiers de P_n pour un nombre de valeurs de n égal au nombre de coefficients du pseudo-polynôme, et de résoudre un système d'équations linéaires. De plus, la connaissance du grade du pseudo-polynôme permet de réduire le nombre de coefficients périodiques à déterminer.

Exemple 3 Reprenons l'exemple du rectangle paramétré P_n. En analysant les faces de dimension 0 de P_n, c'est-à-dire ses sommets, on remarque que trois d'entre elles ne contiennent pas de point entier lorsque n est impair. De même, on remarque pour ses faces de dimension 1, c'est-à-dire ses arêtes, que deux d'entre elles appartiennent à un espace affine qui ne contient pas de point entier lorsque n est impair. Il n'y a que l'espace contenant P_n lui-même qui contient des points entiers pour toutes valeurs de n (par exemple le point $(0, 0)$). Par conséquent, le coefficient du terme de plus haut degré, c'est-à-dire de degré 2, n'est pas périodique mais constant. Il s'agit donc de calculer deux coefficients périodiques, de période 2, donc $2 \times 2 = 4$ valeurs, et un coefficient constant, soit 5 inconnues. On dispose de 5 valeurs initiales du nombre de points entiers, pour $n = 0$ à 4 : $d_n = 1, 1, 4, 4, 9$, d'où :

$$d_n = c_1 n^2 + [c_2, c_3]n + [c_4, c_5] \text{ avec } c_1, c_2, c_3, c_4, c_5 \text{ tels que}$$

$$\begin{cases} & & & & c_4 & = 1 & \text{pour } n = 0 \\ c_1 & + & c_3 & + & c_5 & = 1 & \text{pour } n = 1 \\ 4c_1 & + & 2c_2 & + & c_4 & = 4 & \text{pour } n = 2 \\ 9c_1 & + & 3c_3 & + & c_5 & = 4 & \text{pour } n = 3 \\ 16c_1 & + & 4c_2 & + & c_4 & = 9 & \text{pour } n = 4 \end{cases}$$

□

C'est cette dernière méthode de calcul qui a été choisie pour l'implémentation logicielle. Les choix d'implémentations des différentes étapes de calculs sont décrits à la sous-section 3.4.

Il est à noter que ces résultats concernent d'une manière générale une classe de polyèdres plus généraux appelés *polyèdres normaux* par Ehrhart, et dont la définition a été rappelée au début de cette section. Moyennant une détermination des sommets, et un comptage de quelques valeurs initiales, les pseudo-polynômes associés à ce type de polyèdres peuvent être déterminés directement. D'une manière générale, on démontre aisément que ces résultats sont valides pour toute union et toute intersection de polyèdres convexes.

Les résultats présentés ici, sous réserve des quelques améliorations et généralisations, constituent les travaux d'Ehrhart sur lesquels nous nous sommes basés pour nos extensions : les polyèdres paramétrés considérés lors de la modélisation des programmes informatiques dépendent souvent de plusieurs paramètres. De plus, nous verrons qu'il peut être opportun d'introduire des paramètres supplémentaires dans l'expression des problèmes afin d'obtenir des résultats constructifs.

3.3 Extensions des polynômes d'Ehrhart

Ehrhart, pressentant l'existence de ces polynômes pour un nombre quelconque de paramètres, a énoncé la conjecture suivante dans [39] :

Conjecture 1 (conjecture d'Ehrhart) *Le dénombrant d'un système diophantien linéaire, dépendant de plusieurs paramètres entiers positifs, s'exprime par plusieurs pseudo-polynômes sur différents intervalles.*

Nous nous sommes attachés à démontrer cette conjecture, et à développer une méthode de calcul de ces pseudo-polynômes à plusieurs variables et de leurs intervalles associés. Cette extension passe par la mise à jour des définitions de la sous-section 3.2.

Définition 11 (fraction rationnelle génératrice à plusieurs variables) *Une fraction rationnelle génératrice à p variables est de la forme :*

$$F(x_1, x_2, \ldots, x_p) = \sum_{n_1 \geq 0} \sum_{n_2 \geq 0} \cdots \sum_{n_p \geq 0} f(n_1, n_2, \ldots, n_p) x_1^{n_1} x_2^{n_2} \ldots x_p^{n_p} = \frac{P(x_1, x_2, \ldots, x_p)}{Q(x_1, x_2, \ldots, x_p)}$$

en prenant l'hypothèse que $Q(0, 0, \ldots, 0) \neq 0$.

Définition 12 (relation de récurrence eulérienne multi-dimensionnelle) *Soit le produit :*

$$\Pi(u_1, u_2, \ldots, u_p) = (1 - u_1^{a_{11}} u_2^{a_{12}} \ldots u_p^{a_{1p}})(1 - u_1^{a_{21}} u_2^{a_{22}} \ldots u_p^{a_{2p}}) \ldots (1 - u_1^{a_{p1}} u_2^{a_{p2}} \ldots u_p^{a_{pp}}) \quad (2)$$

où tout $a_{ij} \in \mathbb{N}$. Lorsqu'on développe et simplifie ce produit, et que l'on remplace chaque terme $u_1^{r_1} u_2^{r_2} \ldots u_p^{r_p}$ par $u_{n_1-r_1, n_2-r_2, \ldots, n_p-r_p}$, on obtient une relation de récurrence linéaire multi-dimensionnelle, notée $\{\Pi(u_1, u_2, \ldots, u_p)\} = 0$, et appelée relation de récurrence eulérienne multi-dimensionnelle. Le terme général $f(n_1, n_2, \ldots, n_p)$ d'une telle relation est déterminé par des conditions aux limites et par une fraction rationnelle génératrice à plusieurs variables $\frac{P(u_1, u_2, \ldots, u_p)}{\Pi(u_1, u_2, \ldots, u_p)} = \sum_{n_1 \geq 0} \sum_{n_2 \geq 0} \cdots \sum_{n_p \geq 0} f(n_1, n_2, \ldots, n_p) u_1^{n_1} u_2^{n_2} \ldots u_p^{n_p}$.

Ce terme général est un pseudo-polynôme à plusieurs variables, dont les coefficients sont des nombres périodiques multi-dimensionnels.

Définition 13 (nombres multi-périodiques) *Soit le vecteur $N = (n_j)_{j=1..p}$. Un nombre p-périodique u_N est défini par un tableau U de dimension p et de taille $s_1 \times s_2 \times \ldots \times s_p$ de la manière suivante : $u_N = U[i_1, i_2, \ldots, i_p]$ si pour tout j de 1 à p, $i_j = n_j \bmod s_j$. Le vecteur $s = (s_j)$ est appelé la multi-période de u_N et le plus petit commun multiple des s_j est appelé la période de u_N.*

Par exemple, $(-1)^{n-m}$ peut être défini par le nombre multi-périodique $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{(n,m)}$.
De cette définition, on peut aisément déduire la définition des pseudo-polynômes à plusieurs variables, comme polynômes multi-variables à coefficients multi-périodiques. La définition des polyèdres homothétiques et bordés à plusieurs paramètres est plus délicate.

Définition 14 (polyèdre homothétique à plusieurs paramètres) *Soit un polyèdre P_N, avec $N = (n_1, n_2, \ldots, n_p)$, défini par des équations et inéquations linéaires de la forme $\sum a_i x_i = \sum b_j n_j$, $\sum a_i x_i < \sum b_j n_j$, $\sum a_i x_i \leq \sum b_j n_j$, où les a_i et b_j sont des constantes entières et les n_j sont des paramètres entiers positifs. Le polyèdre P_N est* homothétique *s'il est tel que $\forall\ n_j, \exists\ n_{j,min}, n_{j,max} \in \mathbb{Z}$ tel que $\forall\ n_{j,min} < n_j \leq n_{j,max}$, $P_{0,0,\ldots,0,n_j,0,\ldots,0}$ est le résultat de l'homothétie de $P_{0,0,\ldots,0,n_{j,min},0,\ldots,0}$ de centre l'origine du repère O et de rapport n_j.*

Une définition équivalente est que P_N est homothétique si les coordonnées de tous ses sommets s'expriment comme des fonctions linéaires des paramètres n_j.

Définition 15 (polyèdre bordé à plusieurs paramètres) *Soit un polyèdre P_N défini par des équations et inéquations linéaires de la forme $\sum a_i x_i = \sum b_j n_j + c$, $\sum a_i x_i < \sum b_j n_j + c$, $\sum a_i x_i \leq \sum b_j n_j + c$, où les a_i, b_j et c sont des constantes entières et les n_j sont des paramètres entiers positifs. Le polyèdre P_N est* bordé *si en enlevant les constantes c des équations et inéquations qui le définissent, on obtient un polyèdre homothétique P'_N appelé le noyau de P_N.*

Une définition équivalente est que P_N est bordé si les coordonnées de tous ses sommets s'expriment comme des fonctions affines des paramètres n_j.

Nous introduisons ici la notion de *dénominateur relatif à un paramètre*.

Définition 16 (dénominateur relatif) *Le* dénominateur relatif *au paramètre n d'un point rationnel est le plus petit commun multiple des dénominateurs des facteurs de n dans les expressions affines sur les paramètres de ses coordonnées. Le dénominateur relatif au paramètre n*

d'un polyèdre rationnel est le plus petit commun multiple des dénominateurs relatifs à n de ses sommets.

On peut maintenant définir le produit caractéristique d'un polyèdre à plusieurs paramètres :

Définition 17 (produit caractéristique d'un polyèdre à plusieurs paramètres) *Le produit caractéristique de tout polyèdre rationnel à plusieurs paramètres P_N est défini par l'équation* (2), *où les a_{ij} sont les dénominateurs relatifs au paramètre n_j pour le sommet s_i.*

Nous énonçons le théorème d'extension des polynômes d'Ehrhart à un nombre quelconque de paramètres de la manière suivante :

Théorème 5 (théorème d'Ehrhart étendu) *Le dénombrant d_N de tout k-polyèdre homothétique ou bordé à plusieurs paramètres P_N, dont le produit caractéristique est $\Pi(t_1, t_2, \ldots, t_p)$, vérifie la relation de récurrence eulérienne multi-dimensionnelle $\{\Pi(d)\} = 0$. Pas conséquent, d_N est un polynôme multi-variable en $n_1, n_2, \ldots, n_p$ de degré k si P_N est entier, et est un pseudo-polynôme en $n_1, n_2, \ldots, n_p$ de degré k dont la multi-période des coefficients périodiques est donnée par les dénominateurs relatifs à tous les paramètres n_j, si P_N est rationnel.*

La démonstration de ce théorème se trouve dans [22]. Une partie de cette démonstration, concernant le degré des pseudo-polynômes, est due à D.K. Wilde et se trouve dans [29]. Le domaine des fractions rationnelles génératrices à plusieurs variables ne fournit pas, à ce jour, des méthodes de calculs suffisamment générales permettant de construire un algorithme de détermination des pseudo-polynômes à plusieurs variables. Toutefois, il pourra sans doute procurer des développements intéressants dans l'avenir. Par conséquent, nous étendons notre méthode basée sur la résolution de systèmes linéaires, en passant par la résolution de systèmes linéaires symboliques. Décrivons cette méthode sur l'exemple simple suivant :

Exemple 4 Considérons un rectangle dépendant de deux paramètres n et m, dont la définition est la suivante :

$$P_{n,m} = \left\{ x \in \mathbb{Q}^2 | \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \cdot x \leq \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \cdot \begin{pmatrix} n \\ m \end{pmatrix} \right\}$$

Les sommets de $P_{n,m}$ ont pour coordonnées $(0,0)$, $(\frac{n}{2}, 0)$, $(0, \frac{m}{2})$ et $(\frac{n}{2}, \frac{m}{2})$.

D'après le théorème 5, le polynôme d'Ehrhart de $P_{n,m}$ a la forme générale suivante :

$$d_{n,m} = c_1 n^2 + c_2 m^2 + c_3 n + c_4 m + c_5 nm + c_6$$

où les coefficients c_i sont des nombres 2-périodiques en n et m. Ce pseudo-polynôme à 2 variables peut également être vu comme un pseudo-polynôme de pseudo-polynômes à une seule variable. Par exemple $d_{n,m} = c_1(m)n^2 + c_2(m)n + c_3(m)$, où les coefficients $c_i(m)$ sont des nombres 1-périodiques en n dont les valeurs sont des pseudo-polynômes en m, ou alors $d_{n,m} = c_1(n)m^2 + c_2(n)m + c_3(n)$, où les coefficients $c_i(n)$ sont des nombres 1-périodiques en m dont les valeurs sont des pseudo-polynômes en n.

Afin de réduire le nombre de coefficients à calculer, on utilise le théorème 3. En analysant $P_{n,0}$, on remarque que le sommet $(\frac{n}{2}, 0)$ n'est pas toujours entier. Par contre, l'arête et par conséquent le polyèdre lui-même contiennent toujours au moins un point entier. Les mêmes observations peuvent être faites pour $P_{0,m}$. Par conséquent, $d_{n,m} = c_1(m)n^2 + c_2(m)n + [c_3(m), c_4(m)]_n$, où les $c_i(m)$ désignent des pseudo-polynômes en m.

On détermine symboliquement ces coefficients en posant le système d'équations linéaires :

$$\begin{cases} c_3(m) & = d_{0,m} \\ c_1(m) + c_2(m) + c_4(m) & = d_{1,m} \\ 4c_1(m) + 2c_2(m) + c_3(m) & = d_{2,m} \\ 9c_1(m) + 3c_2(m) + c_4(m) & = d_{3,m} \end{cases}$$

Afin de résoudre ce système, les pseudo-polynômes $d_{0,m}$, $d_{1,m}$, $d_{2,m}$ et $d_{3,m}$ doivent être calculés. Par exemple et d'après les observations précédentes, $d_{0,m} = c_5 m^2 + c_6 m + [c_7, c_8]$.

$$\begin{cases} c_7 & = d_{0,0} \\ c_5 + c_6 + c_8 & = d_{0,1} \\ 4c_5 + 2c_6 + c_7 & = d_{0,2} \\ 9c_5 + 3c_6 + c_8 & = d_{0,3} \end{cases}$$

On compte le nombre de points entiers dans $P_{0,0}$, $P_{0,1}$, $P_{0,2}$ et $P_{0,3}$. On obtient $1, 1, 2, 2$, d'où $d_{0,m} = \frac{1}{2}m + [1, \frac{1}{2}]$. En procédant de même pour les trois autres pseudo-polynômes, et en reportant les résultats dans le premier système d'équations, on obtient finalement :

$$d_{n,m} = \frac{1}{4}nm + \left[\frac{1}{2}, \frac{1}{4}\right]_m n + \left[\frac{1}{2}, \frac{1}{4}\right]_n m + \left[\begin{array}{cc} 1/4 & 1/2 \\ 1/2 & 1 \end{array}\right]_{n,m}$$

C'est cette méthode de calcul qui a été implémentée dans notre logiciel, dont la réalisation a nécessité la conception d'autres algorithmes décrits à la sous-section suivante.

3.4 Implémentation logicielle des polynômes d'Ehrhart

Cette implémentation a été réalisée exclusivement par Vincent Loechner et Doran K. Wilde, que je tiens à remercier chaleureusement pour cette contribution essentielle. Ce logiciel est encore en perpétuelle évolution et amélioration. Nous ne décrivons ici que quelques aspects essentiels.

Un polyèdre dont la définition dépend linéairement de plusieurs paramètres, comme nous l'avons décrit dans l'exemple 3.2, est une collection de polyèdres homothétiques ou bordés définis sur des domaines de définition adjacents des paramètres. En considérant les définitions équivalentes des polyèdres homothétiques et bordés, c'est-à-dire relatives à une expression respectivement linéaire et affine sur les paramètres des coordonnées des sommets, il en résulte qu'un polynôme d'Ehrhart est associé à chaque domaine de définition des paramètres où les sommets ont une expression linéaire ou affine constante, et où leur nombre est constant. La connaissance de ces domaines de définition des paramètres est donc essentielle, afin de pouvoir déterminer les polynômes d'Ehrhart associés à chacun de ces domaines, et cela pour n'importe quelle définition paramétrique d'un polyèdre. De plus, la connaissance des coordonnées des sommets permet de déterminer le dénominateur du polyèdre, et donc la période des coefficients du polynôme.

3.4.1 Détermination des polyèdres homothétiques et bordés et de leur dénominateur

Une automatisation du calcul des polynômes d'Ehrhart nécessite donc la détermination des sommets paramétriques d'un polyèdre et des domaines de définition des paramètres associés. Ce développement a été réalisé par V. Loechner et D.K. Wilde, et implémenté dans la librairie polyédrique *Polylib*. Nous donnons ici les grandes lignes de leur algorithme qui est présenté en détail et dont les démonstrations se trouvent dans [69, 67].

Un polytope paramétré peut être représenté comme un polyèdre non-paramétré dans l'espace combiné des variables et des paramètres de la manière suivante :

$$P' = \left\{ \left(\begin{array}{c} x \\ N \end{array} \right) \in \mathbb{Q}^{d+p} \middle| [A| - B] \left(\begin{array}{c} x \\ N \end{array} \right) \leq \overrightarrow{c} \right\}$$

La propriété suivante relie le polyèdre combiné P' avec son polytope paramétré équivalent P_N :

Propriété 1 *Soit S_{N_0} le sous-espace affine de $\mathbb{Q}^{d+p}$ associé à une valeur $N = N_0$:*

$$S_{N_0} = \left\{ \left(\begin{array}{c} x \\ N_0 \end{array} \right) \in \mathbb{Q}^{d+p} \right\}$$

L'intersection de P' avec S_{N_0} instancie le paramètre à un vecteur constant N_0. Le résultat de cette intersection est le polytope P_{N_0} exprimé dans l'espace combiné $\mathbb{Q}^{d+p}$:

$$P' \cap S_{N_0} = \left\{ \left(\begin{array}{c} x \\ N \end{array} \right) \in \mathbb{Q}^{d+p} | x \in P_{N_0}, N = N_0 \right\}$$

P_{N_0} peut être obtenu par projection de cette intersection sur l'espace des variables $\mathbb{Q}^d$.

On considère dans la suite que dans le système de contraintes définissant le polytope paramétré P_N, il n'y a pas d'égalité de la forme $D \cdot N + \overrightarrow{d} = 0$, où D est une matrice non-nulle et $\overrightarrow{d}$ un vecteur. S'il existe une telle égalité, celle-ci est détectée et supprimée. La vérification de la propriété suivante en dépend :

Propriété 2 *Soit $s_i(N)$ un sommet paramétré du polytope P_N. Il existe une face de dimension p de P', notée F_i^p, telle que $\left(\begin{array}{c} s_i(N) \\ N \end{array} \right) = F_i^p \cap S_N$.*

Soit π la fonction de projection des points de $\mathbb{Q}^{d+p}$ sur l'espace des paramètres $\mathbb{Q}^p$: $\pi \left(\begin{array}{c} x \\ N \end{array} \right) = N$.

Propriété 3 *Le sommet $s_i(N)$ correspondant à la face F_i^p n'est défini que pour les valeurs de N contenues dans le polyèdre $\pi(F_i^p)$.*

Soit $\mathcal{B}$ une base du plus petit espace affine contenant F_i^p et X_i les coordonnées de $\begin{pmatrix} s_i(N) \\ N \end{pmatrix}$ dans la base $\mathcal{B}$. On peut aisément déterminer deux transformations affines T_D et T_P telles que :

$$\begin{array}{rcll} N & = & T_P(X_i) & \quad (1) \\ s_i(N) & = & T_D(X_i) & \quad (2) \end{array}$$

Si la transformation T_P est bijective, alors l'intersection de F_i^p avec S_N est un point unique, et F_i^p correspond à un sommet de P_N. L'équation (1) peut alors être réécrite : $T_P^{-1}(N) = X_i$. En substituant X_i dans l'équation (2) on obtient : $s_i(N) = T_D(T_P^{-1}(N))$.

Inversement, si la transformation T_P n'est pas bijective, alors il n'y a pas qu'un seul point dans F_i^p correspondant à une valeur de N, mais tout un polyèdre de dimension supérieure ou égale à 1. L'intersection de ce polyèdre avec S_N ne correspond pas à un sommet de P_N, mais à une face de dimension égale à celle du polyèdre.

Finalement, l'algorithme est le suivant :

1. Détermination de toutes les faces F_i^p.
2. Pour chacune de ces faces :
   - Détermination de T_D et T_P.
   - Si T_P est bijective, détermination de $T_D \circ T_P^{-1}$ et déduction du sommet correspondant. Calcul de $\pi(F_i^p)$ définissant l'ensemble des contraintes sur N, c'est-à-dire le domaine de définition du sommet.

Afin de caractériser les polyèdres homothétiques et bordés, les sommets doivent être regroupés par domaines de définitions adjacents des paramètres. Pour cela, on effectue un partitionnement du domaine des paramètres à partir de la projection des faces de dimension p de P'. Ainsi, pour chaque N d'un certain domaine de définition, l'ensemble des faces de dimension p intersectées par S_N est le même. Par conséquent, l'ensemble des sommets paramétrés de P_N est également le même.

Tout le domaine de définition des paramètres est $\pi(P')$, c'est-à-dire la projection de P' sur le domaine des paramètres $\mathbb{Q}^p$. La projection des faces convexes de dimension p de P' forme un ensemble de polyèdres qui recouvrent $\pi(P')$, P' étant entièrement défini par ses faces de dimension p. L'algorithme détermine un ensemble de polyèdres $D = \{d_j | 1 \leq j \leq v\}$, où v est le nombre de domaines de définition, tel que :

- d_j est un polyèdre de dimension p,
- $d_j \cap d_k$ est un polyèdre de dimension inférieure à p, $\forall j \neq k$,
- $d_1 \cup d_2 \cup \ldots \cup d_v = \pi(P')$,
- $\forall N \in d_j$, $S_N \cap P'$ correspond à un ensemble unique de sommets paramétrés.

L'algorithme construit le partitionnement de D en ajoutant successivement chaque face projetée $f = \pi(F_i^p)$ à l'ensemble des domaines de définition. A chaque itération de l'algorithme, un nouvel ensemble de domaines est construit à partir de l'ensemble D de l'itération précédente, en retenant :

- toutes les intersections $d_j \cap f$, pour tous les domaines $d_j \in D$,

– toutes les différences $d_j - f$, pour tous les domaines $d_j \in D$,

– la différence entre f et l'union de tous les domaines $d_j \in D$: $f - (\cup_{d_j \in D} d_j)$.

L'étape suivante consiste, pour chaque domaine de définition ainsi déterminé, à calculer le polynôme d'Ehrhart associé.

3.4.2 Le calcul des coefficients périodiques

Considérons un pseudo-polynôme de degré $k-1$ et de période q :

$$EP(n) = [u_{k-1,q-1}, \ldots, u_{k-1,0}]_n n^{k-1} + [u_{k-2,q-1}, \ldots, u_{k-2,0}]_n n^{k-2} + \ldots + [u_{0,q-1}, \ldots, u_{0,0}]_n n^0$$

Les inconnues sont les valeurs u_{ij}, avec $0 \leq i < k$ et $0 \leq j < q$. Pour les calculer, on instancie plusieurs fois n afin de générer un nombre suffisant d'équations. Ainsi, un système de kq équations devrait être résolu afin de déterminer kq coefficients en un temps $\mathcal{O}(k^3q^3)$. Mais dans notre cas, il est possible de réduire la complexité en ne résolvant que q systèmes de k équations, en un temps $\mathcal{O}(k^3q)$.

Soit $n_0 > 0$ et $v = n_0 \; mod \; q$. En substituant n_0 à n dans le pseudo-polynôme on obtient :

$$EP(n_0) = u_{k-1,v} n_0^{k-1} + u_{k-2,v} n_0^{k-2} + \ldots + u_{0,v}$$

Si l'on choisit un ensemble de p valeurs de n, en commençant par n_0 et en augmentant par pas de q, c'est-à-dire $n_i = n_0 + iq$, $0 \leq i < p$, et en substituant ces valeurs à n dans le pseudo-polynôme, on obtient le système d'équations suivant :

$$\left\{ \begin{array}{rcl} u_{k-1,v} n_0^{k-1} + u_{k-2,v} n_0^{k-2} + \ldots + u_{0,v} & = & EP(n_0) \\ u_{k-1,v} n_1^{k-1} + u_{k-2,v} n_1^{k-2} + \ldots + u_{0,v} & = & EP(n_1) \\ & \vdots & \\ u_{k-1,v} n_{k-1}^{k-1} + u_{k-2,v} n_{k-1}^{k-2} + \ldots + u_{0,v} & = & EP(n_{k-1}) \end{array} \right.$$

Ce système est mis sous la forme matricielle $A \cdot \overrightarrow{u} = \overrightarrow{E}$:

$$\begin{bmatrix} n_0^{k-1} & n_0^{k-2} & \ldots & 1 \\ n_1^{k-1} & n_1^{k-2} & \ldots & 1 \\ & & \vdots & \\ n_{k-1}^{k-1} & n_{k-1}^{k-2} & \ldots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{k-1,v} \\ u_{k-2,v} \\ \vdots \\ u_{0,v} \end{bmatrix} = \begin{bmatrix} EP(n_0) \\ EP(n_1) \\ \vdots \\ EP(n_{k-1}) \end{bmatrix}$$

La matrice A est toujours inversible et sa matrice inverse A^{-1} est rationnelle. On détermine $\overrightarrow{u}$ par $\overrightarrow{u} = A^{-1} \cdot \overrightarrow{E}$. On distingue deux cas :

1. Le vecteur $\overrightarrow{E}$ est un vecteur de constantes. Celles-ci sont obtenues par comptage du nombre de points entiers dans les polyèdres $P_{n_0}, P_{n_1}, \ldots, P_{n_{k-1}}$. Ce comptage est effectué par parcours des polyèdres en utilisant l'algorithme décrit dans [97].

2. Le vecteur $\overrightarrow{E}$ n'est pas un vecteur de constantes. Il s'agit du cas où le pseudo-polynôme EP est multi-variable, $EP(n, m, \ldots)$, c'est-à-dire lorsque le problème initial comporte

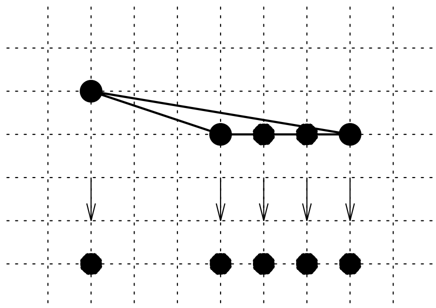

FIG. 3.1: *Illustration de la projection d'un polytope de points entiers*

plusieurs paramètres. Dans ce cas, tous les pseudo-polynômes $EP(n_j, m, \ldots)$ sont déterminés en appelant récursivement la procédure décrite ici. Ces appels récursifs se terminent par le cas 1, lorsque tous les paramètres sont instanciés. Le résultat de cette procédure récurrente est un pseudo-polynôme et le vecteur $\overrightarrow{E}$ est alors un vecteur de pseudo-polynômes. Le produit $A^{-1} \cdot \overrightarrow{E}$ doit donc être effectué symboliquement, et des procédures spécifiques de multiplication et d'addition de pseudo-polynômes sont utilisées pour calculer les valeurs u_{ij}.

L'ensemble de cette procédure est répété pour toutes les autres valeurs de $n \ mod \ q$, jusqu'à ce que toutes les valeurs u_{ij} aient été calculées.

Les pseudo-polynômes ainsi déterminés peuvent souvent être simplifiée. Une procédure de simplification décrite dans [29] a également été implémentée, consistant à réduire la période des coefficients périodiques en cas de valeurs identiques pour plusieurs valeurs de $n \ mod \ q$.

3.5 Les polynômes d'Ehrhart de transformations affines de polyèdres

La projection affine d'un polytope pose une difficulté supplémentaire. Les points entiers contenus dans un polytope P_N sont organisés en un réseau régulier, ou lattice. Si l'on applique une projection affine à un tel polytope, la projection affine des points entiers du polytope n'est plus, en général, un réseau régulier de points. On ne peut donc plus modéliser ces points par leur enveloppe convexe, comme il est montré sur l'exemple de la figure 3.1.

Le dénombrement des points résultants d'une telle projection affine ne peut donc plus s'effectuer par la méthode décrite dans les paragraphes précédents. Dans le cas plus général d'une transformation affine d'un polytope paramétrique, nous obtenons le résultat suivant démontré dans [19] :

Théorème 6 *Le nombre de points entiers, résultants d'une transformation affine des points entiers contenus dans un polytope paramétrique, est défini sur plusieurs domaines adjacents de valeurs des paramètres, par différents polynômes d'Ehrhart.*

La preuve de ce théorème découle du raisonnement développé dans la suite. Nous définissons les transformations affines de la manière suivante :

$$\begin{array}{rlll} & P \subset \mathbb{Z}^k & \longrightarrow & M \subset \mathbb{Z}^p \\ Trans\_Aff : & X & \longmapsto & Y = (F_r(X) + f_r)_{1 \le r \le p} \end{array}$$

où $F_r(X)$ désigne une combinaison linéaire à coefficients entiers de X et f_r une constante entière. Notre objectif est donc le dénombrement des points entiers résultants de l'application de $Trans\_Aff$ à tous les points entiers de P. Ce problème est équivalent à déterminer le nombre de valeurs distinctes de Y atteintes par $Trans\_Aff$.

Soit le polytope paramétrique P_Y suivant :

$$P_Y = \{X \in \mathbb{Z}^n | X \in P, Y = (F_r(X) + f_r)_{1 \le r \le p}\}$$

Ce polytope caractérise tous les points de P qui résultent en une même valeur Y. Les polynômes d'Ehrhart de P_Y peuvent être calculés, donnant le nombre de points de P résultant en une même valeur Y, et définis sur plusieurs domaines adjacents de valeurs de Y.

Puisque P est un ensemble borné de points entiers, l'ensemble de toutes les valeurs Y atteintes par $Trans\_Aff$ est également borné. Par conséquent, l'union des domaines de valeurs sur lesquels les polynômes d'Ehrhart de P_Y sont définis l'est aussi.

Malheureusement, dénombrer les points dans cette union de domaines ne correspond pas au nombre exact de valeurs de Y atteintes par $Trans\_Aff$. Certaines valeurs incluses dans les domaines ne sont pas atteintes : le polynôme d'Ehrhart pour ces valeurs est égal à zéro. Il s'agit donc d'exclure ces valeurs du dénombrement.

Soit $pe(Y)$ un polynôme d'Ehrhart défini sur un domaine D de valeurs pouvant être atteintes Y, pour lequel les coefficients périodiques sont étendus à une taille commune dans chacune de leur dimension. Soit S le vecteur des tailles des coefficients périodiques dans toutes leurs dimensions. On note Y *modulo* S la conjonction de tous les termes y_r *modulo* $s_r, 1 \le r \le p$. Soit V un vecteur d'entiers de taille p tel que pour tout $1 \le r \le p, 0 \le v_r < s_r$.

Un polynôme d'Ehrhart étant caractérisé par des coefficients périodiques, il peut également être vu comme un *polynôme périodique*. Par conséquent, pour tout Y tel que Y *modulo* $S = V$, un polynôme de la forme suivante est défini :

$$pe_V(Y) = \sum_{i_1=0}^{k} \sum_{i_2=0}^{k-i_1} \cdots \sum_{i_p=0}^{k-i_1-i_2-\ldots-i_{p-1}} c_{i_1,i_2,\ldots,i_p} \; y_1^{i_1} y_2^{i_2} \ldots y_p^{i_p}$$

où les $c_{i_1,i_2,\ldots,i_p}$ sont des nombres rationnels constants.

Proposition 3 *Si tous les coefficients de $pe_V(Y)$ sont égaux à zéro, le nombre de valeurs Y dans D atteintes par $Trans\_Aff$ et telles que Y modulo $S = V$, est égal à zéro. Sinon, il est égal à $N_V - N_R$, où N_V et N_R sont définis comme suit :*

- *N_V est le nombre de points entiers contenus dans le polytope défini par :*

$$\{Y \in \mathbb{Z}^p \mid Y \in D, Y \text{ } modulo \text{ } S = V\}$$

- *N_R est le nombre de racines entières R de $pe_V(Y)$ telles que R modulo $S = V$ et $R \in D$.*

Les domaines de valeurs des paramètres étant adjacents, ils doivent tout d'abord être transformés en domaines disjoints, par remplacement d'inégalités larges en inégalités strictes.

Proposition 4 *Le nombre total de valeurs atteintes par $Trans\_Aff$ est obtenu en effectuant le dénombrement décrit précédemment pour tous les vecteurs V et pour tous les domaines disjoints D, et en sommant tous les résultats.*

Notre méthode se résume donc par les étapes suivantes :

1. Calcul des polynômes d'Ehrhart de

$$P_Y = \{X \in \mathbb{Z}^n | X \in P, Y = (F_r(X) + f_r)_{1 \leq r \leq p}\}$$

   et des domaines associés de valeurs des paramètres. Transformation de ces domaines en domaines disjoints.

2. Pour chacun des domaines D de valeurs des paramètres et pour chaque vecteur V, si le polynôme associé $pe_V(Y)$ n'est pas égal à zéro pour toutes valeurs Y pour lesquelles il est défini (coefficients nuls), calcul de $N_V - N_R$.

3. La somme de tous les résultats obtenus en 2 est le nombre de valeurs atteintes par $Trans\_Aff$.

La dernière difficulté concerne le dénombrement des points entiers de polytopes définis par des contraintes utilisant l'opération *modulo*.

Proposition 5 *Soit C un système de contraintes, définissant des points entiers $Y = (y_r)$, et pour lequel les contraintes non-linéaires sont de la forme, y_r modulo $s_r = v_r$, où s_r et v_r sont des constantes entières, $0 \leq v_r < s_r$.*

Soit C' un système de contraintes linéaires, définissant des points entiers $Z = (z_r)$, qui est déduit de C en supprimant toutes les contraintes non-linéaires, y_r modulo $s_r = v_r$, et en remplacant tout y_r par $s_r z_r + v_r$ dans toutes les autres contraintes.

Le nombre de points entiers Y définis par C est égal au nombre de points entiers Z définis par C'.

La détermination du nombre de racines entières des polynômes $pe_V(Y)$ peut s'avérer difficile et complexe dans les cas de degrés supérieurs à 3 ou de polynômes multi-variables. De plus, la considération de paramètres de taille dans le polytope initial amène à des déterminations symboliques de ces racines. Le domaine mathématique des *bases de Gröbner* [11] peut constituer une réponse à ces problèmes. Toutefois, ce domaine est encore peu développé en matière de résolution symbolique. Une autre solution consiste à utiliser l'outil de programmation linéaire en nombres entiers *PIP* développé par P. Feautrier [41], en vérifiant avec cet outil l'existence d'un point entier sur chaque ensemble P_Y.

Un exemple d'application au dénombrement des éléments de tableau accédés par une boucle est présenté dans [19].

Chapitre 4

Optimisation des performances de cache

4.1 Introduction

Une grande différence de rapidité sépare les processeurs des mémoires. Ainsi, la performance d'un programme est très dépendante de ses accès en mémoire. Les mémoires caches, d'accès beaucoup plus rapide que la mémoire principale, constituent la solution matérielle apportée à ce problème. Toutefois, leurs capacités de stockage étant très réduites, à cause notamment de leur coût élevé, toutes les données nécessaires à un programme ne peuvent pas y être mémorisées pendant toute l'exécution. Lorsque le processeur, exécutant un programme, requiert une donnée, celle-ci est recherchée dans la mémoire cache. L'éventuelle non-présence de la donnée entraîne alors le chargement d'un bloc mémoire dans le cache, puis enfin le chargement de la donnée par le processeur. Cet événement de non-présence d'une donnée dans le cache, appelé communément *défaut de cache*, est très coûteux en temps et pénalise la performance des programmes. L'écriture d'un programme performant consiste donc à minimiser le nombre de défauts de cache, en essayant de regrouper les réutilisations d'une même donnée dans le temps. On distingue deux types de réutilisations des données :

- La réutilisation temporelle : il s'agit de la réutilisation d'une seule et même donnée durant l'exécution.
- La réutilisation spatiale : la mise à jour du cache s'effectue par le chargement d'un bloc mémoire constitué de plusieurs données stockées de manière contiguë. La taille de ce bloc dépend du type de mémoire cache implémentée. La réutilisation spatiale concerne la réutilisation d'un tel bloc : si un tel bloc a été chargé pour l'utilisation d'une donnée particulière, les autres données du bloc sont elles aussi utilisées par le processeur.

On déduit de cette caractérisation deux objectifs d'optimisation des performances : l'optimisation de la localité temporelle et l'optimisation de la localité spatiale.

La plupart des applications scientifiques sont constituées de nids de boucles accédant à de grands tableaux multi-dimensionnels. Mais, malheureusement, ces boucles sont généralement écrites sans prendre en compte la performance des mémoires caches. Plusieurs travaux ont mis en évidence l'utilité de techniques de compilation pour optimiser la localité des données. On peut distinguer deux types d'approches : celles basées sur des transformations de boucles, et celles basées sur des réorganisations du stockage des données en mémoire.

Dans le cadre de la première approche, citons Wolfe [101], Banerjee [5] et Li et Pingali [64, 66] qui ont largement contribué au développement de la théorie en la matière. La deuxième approche possède plusieurs avantages. Tout d'abord, certains nids de boucles ne peuvent pas être transformés à cause des dépendances entre les instructions, alors que les transformations du stockage des données n'affectent en rien ces dépendances. Ensuite, même si une transformation de boucles est possible, une réorganisation de données peut s'avérer plus efficace. De plus, contrairement aux transformations de boucles, les réorganisations des données n'affectent pas tous les tableaux référencés par une boucle. Dans ce cadre, Kandemir *et al.* [57, 58] ont proposé une méthode de réorganisation des données en mémoire pour les nids de boucles, en ne considérant que la boucle la plus interne et en limitant les transformations de stockage à des inversions de lignes et de colonnes dans les tableaux. Leung et Zahorjan [63] se limitent quant à eux à des transformations unimodulaires. Récemment, O'Boyle et Knijnenburg [79] ont proposé une généralisation de ces transformations aux transformations affines bijectives.

Notre ambition est encore plus générale et inclut l'ensemble de ces approches. En effet, nos travaux dans le domaine tendent à proposer de meilleures organisations des données par transformations non-linéaires des accès mémoires. Ainsi, nos résultats peuvent consister en des transformations linéaires, lorsque celles-ci correspondent à une localité optimale des accès, ou en des transformations polynômiales. Contrairement aux autres approches, cette approche est plus générale et plus efficace pour plusieurs critères d'optimisation :

- l'optimisation est effectuée pour tous les niveaux de boucles,
- toute référence à des éléments de tableau définie par une fonction affine des indices de boucles est prise en compte,
- on ne considère que les éléments de tableau effectivement accédés, afin d'éviter le chargement inutile de données dans la mémoire cache,
- les boucles dont les bornes sont paramétrées et les tableaux de tailles variables sont considérés, amenant à l'expression de transformations symboliques.

Nos premiers résultats recouvrant ces objectifs concernent l'optimisation de la localité spatiale, que nous décrivons dans la suite.

Les optimisations des localités temporelle et spatiale sont difficiles à concilier dans une même approche. Mais il semble intuitivement clair que les méthodes de transformations de programmes sont adaptées à l'optimisation de localité temporelle, par réordonnancement des accès aux éléments de tableau. De même, les méthodes de réorganisation des données sont adaptées à l'optimisation de localité spatiale. L'association de ces deux approches constitue donc *a priori* la meilleure voie pour une résolution globale des performances de cache d'un programme.

Dans le contexte du travail actuel de thèse d'Andry Randrianatoavina, que j'encadre en collaboration avec Christine Eisenbeis, nous proposons une approche originale pour ce problème. Celle-ci est basée sur une modélisation des accès mémoire des nids de boucles par des *fenêtres de références* [47, 40] relatives à un instant d'exécution. L'ambition de ce modèle repose sur une paramétrisation qui conditionne les transformations de programmes et de données.

Nous présentons ces deux voies de recherches dans les sections suivantes.

4.2 Optimisation de la localité spatiale par réorganisation des données

4.2.1 Objectif de la méthode

La technique présentée est illustrée par le nid de boucles suivant :

Exemple 5

```
for i1 = 1 to p
  for i2 = i1 to 2 * i1 - 1
    for i3 = i1 - i2 to i1 + i2
      X(i1,i2,i3) := Y(i1 + i2 - 1, i1 + i3 + 2, i2 + i3) + 10
```

où p est un paramètre entier positif. On analyse les accès au tableau Y. Ceux-ci sont définis par une fonction affine f du vecteur d'itération $\overrightarrow{I} = (i_1, i_2, i_3)$, $f(\overrightarrow{I}) = R \cdot \overrightarrow{I} + \overrightarrow{o}$, où R est appelée *matrice d'accès* et $\overrightarrow{o}$ est appelé *vecteur de déplacement* :

$$R = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \text{ et } \overrightarrow{o} = \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix}$$

□

Notre objectif est d'optimiser la localité spatiale pour de tels accès à un tableau, en déterminant une réorganisation de ses éléments de telle manière que tout bloc de données chargé dans la mémoire cache, sera entièrement utilisé par le nid de boucles, et par des itérations successives. Afin d'analyser les accès, on définit des vecteurs $\overrightarrow{U}_m$ caractérisant l'ordre d'utilisation des éléments pour un niveau de boucle m.

Définition 18 *Le vecteur d'utilisation du niveau de boucle m, $\overrightarrow{U}_m$, est égal à la m-ième colonne de la matrice d'accès R.*

Interprétons ces vecteurs sur notre exemple :

Exemple 6 *(suite)* Les vecteurs d'utilisation sont :

$$\overrightarrow{U_1} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \overrightarrow{U_2} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{ et } \overrightarrow{U_3} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

Considérons un élément de tableau $Y(i,j,k)$ référencé par le nid de boucles à une certaine itération (i_1, i_2, i_3). Si $i_3 < i_1 + i_2$, l'élément qui sera référencé ensuite sera $Y((i,j,k) + \overrightarrow{U_3}) = Y(i, j+1, k+1)$ à l'itération $(i_1, i_2, i_3 + 1)$. Plus généralement, pour tout q tel que $1 \leq q \leq i_1 + i_2 - i_3$, l'élément $Y((i,j,k) + q \times \overrightarrow{U_3}) = Y(i, j+q, k+q)$ sera référencé à l'itération $(i_1, i_2, i_3 + q)$.

Ensuite, lorsque $q = i_1 + i_2 - i_3$, l'élément référencé après $Y(i, j+q, k+q)$ sera $Y(i, j+q, k+q) + \overrightarrow{U_2}) = Y(i+1, j+q, k+q+1)$. Les références suivantes peuvent ensuite à nouveau être déterminées à l'aide du vecteur $\overrightarrow{U_3}$, et ainsi de suite. □

Ces vecteurs permettent de caractériser les réutilisations apparaissant dans le nid de boucles. Si au moins un vecteur $\overrightarrow{U}_m$ est nul, des éléments de tableau sont réutilisés temporellement dans les boucles internes $i_{m+1}, \ldots, i_n$, autrement dit l'utilisation d'une même suite d'éléments se répète dans les boucles internes. Si deux ou plusieurs vecteurs sont linéairement dépendants, c'est-à-dire si R est une matrice singulière, certains éléments sont réutilisés temporellement.

Lorsqu'un tableau est référencé plusieurs fois dans un même corps de boucle, il est difficile de concilier les optimisations spatiales pour toutes ses références. Nos résultats à ce jour ne considèrent pas ce cas, qui sera traité dans nos travaux futurs.

4.2.2 Les données effectivement utilisées

Soit $Y(\overrightarrow{I_0}) = Y(i0_1, i0_2, \ldots, i0_n)$ un élément d'un tableau Y référencé par un nid de boucles. L'ensemble des itérations référençant cet élément $Y(\overrightarrow{I_0})$ est défini par :

$$P(\overrightarrow{I_0}) = \{\overrightarrow{I} \in P | R \cdot \overrightarrow{I} + \overrightarrow{o} = \overrightarrow{I_0}\}$$

Si $P(\overrightarrow{I_0})$ est vide, alors $Y(\overrightarrow{I_0})$ n'est jamais référencé par le nid de boucles et par conséquent, ne doit jamais être chargé dans la mémoire cache. Il peut être utile, notamment dans le cas d'un compilateur, de savoir si un élément est effectivement utilisé, et si oui à quelle fréquence. En calculant le polynôme d'Ehrhart $EP(\overrightarrow{I_0})$ de $P(\overrightarrow{I_0})$, on fournit une fonction donnant cette information.

Si $P(\overrightarrow{I_0})$ se réduit à une seule itération, alors le tableau Y n'est pas réutilisé temporellement. Dans ce cas, la méthode que nous présentons ici donne la réorganisation de données la meilleure possible : on affecte chaque donnée d'un nouvel indice, correspondant à la position de l'itération qui la référence, selon l'ordre d'exécution de toutes les itérations. En d'autres termes, si la q-ième itération référence $Y(\overrightarrow{I_0})$, alors $Y(\overrightarrow{I_0})$ sera placé en mémoire principale à l'adresse physique $b + q \times w$, où b est l'adresse de base du tableau Y, et w est la taille en octets d'un élément.

Bien qu'une optimisation aussi bien temporelle que spatiale nécessiterait également des transformations de programmes, le gain de performance dû à notre optimisation spatiale est déjà important, car il garantit que tout bloc chargé dans le cache est entièrement utilisé avant d'être écrasé par un nouveau chargement. En cas de réutilisation temporelle, un bloc déjà chargé précédemment peut l'être à nouveau. C'est ce besoin de rechargement d'un même bloc qui doit être localisé, si possible, par une transformation de programme.

Dans le cas d'une réutilisation temporelle, c'est-à-dire lorsque $P(\overrightarrow{I_0})$ définit plusieurs itérations, on choisit de considérer la première itération, selon l'ordre lexicographique, de cet ensemble. Ce minimum doit donc être calculé symboliquement, par notre méthode décrite dans [22] et utilisant la librairie *Polylib*, ou en utilisant l'outil *PIP* [41]. Comme la même suite de références se répète plusieurs fois, ce choix garantit une localité spatiale optimale à chaque apparition de cette suite. De plus, si cette suite correspond à une taille mémoire inférieure ou égale à la taille de la mémoire cache, celle-ci peut être chargée entièrement et la localité temporelle est alors également optimisée.

Exemple 7 *(suite)* L'ensemble des itérations qui référencent un élément $Y(i0_1, i0_2, i0_3)$ est défini par :

$$\begin{aligned} P(i0_1, i0_2, i0_3) &= \{(i_1, i_2, i_3) \in P | i_1 + i_2 - 1 = i0_1, i_1 + i_3 + 2 = i0_2, i_2 + i_3 = i0_3\} \\ &= \{(i_1, i_2, i_3) \in P | i_1 = \tfrac{i0_1 + i0_2 - i0_3 - 1}{2}, i_2 = \tfrac{i0_1 - i0_2 + i0_3 + 3}{2}, i_3 = \tfrac{-i0_1 + i0_2 + i0_3 - 3}{2}\} \end{aligned}$$

Remarquons que $P(i0_1, i0_2, i0_3)$ est vide lorsque au moins une des fractions $\frac{i0_1+i0_2-i0_3-1}{2}$, $\frac{i0_1-i0_2+i0_3+3}{2}$ ou $\frac{-i0_1+i0_2+i0_3-3}{2}$ n'est pas entière, ou n'est pas une valeur appartenant à l'espace d'itération P. Remarquons également que $P(i0_1, i0_2, i0_3)$ définit au plus une seule itération, et qu'il n'y a donc pas de réutilisation temporelle pour Y. □

L'ensemble des données effectivement utilisées est défini par :

$$D_Y = \{\overrightarrow{I_0} | \overrightarrow{I_0} = A \cdot \overrightarrow{I} + \overrightarrow{a}, \overrightarrow{I} \in P\}$$

Ces données sont associées aux points entiers résultant de la transformation affine $(R, \overrightarrow{a})$ de l'espace d'itération P. Nous avons présenté dans [19], et rappelé à la section 3.5, une méthode de dénombrement exact de ces données effectivement référencées. Un tel ensemble n'est pas en général un réseau régulier, ou lattice, de points entiers, à cause de "trous" apparaissant dans le réseau. Mais pour notre méthode, une détermination exacte de D_Y n'est pas nécessaire. L'enveloppe convexe de D_Y, $Conv(D_Y)$, est suffisante afin de réduire les réponses de notre méthode à ce qui est strictement utile. Pour calculer cette enveloppe convexe, nous utilisons notre programme de recherche de sommets paramétriques et des domaines de définition présenté dans [69, 22] et rappelé à la sous-section 3.4.1. En calculant les sommets paramétriques de $P(\overrightarrow{I_0})$, les domaines de définitions obtenus correspondent à l'enveloppe convexe rationnelle des valeurs prises par $\overrightarrow{I_0}$, c'est-à-dire à $Conv(D_Y)$.

Exemple 8 *(suite)* L'ensemble des données effectivement référencées est défini par :

$$\begin{aligned} D_Y = \{(i0_1, i0_2, i0_3) | \quad & i0_1 = i_1 + i_2 - 1, i0_2 = i_1 + i_3 + 2, i0_3 = i_2 + i_3, \\ & 1 \leq i_1 \leq p, i_1 \leq i_2 \leq 2i_1 - 1, i_1 - i_2 \leq i_3 \leq i_1 + i_2\} \end{aligned}$$

On calcule l'enveloppe convexe de D_Y, $Conv(D_Y)$:

$$\begin{aligned} Conv(D_Y) = \{(i0_1, i0_2, i0_3) | \quad & i0_1 + i0_2 \leq i0_3 + 2p + 1, 3i0_3 + 7 \leq i0_1 + 3i0_2, \\ & i0_2 \leq i0_3 + 2, i0_1 + i0_2 \leq 3i0_3 + 1, \\ & i0_2 + i0_3 \leq 3i0_1 + 5\} \end{aligned}$$

□

4.2.3 Le placement des données en mémoire

Le minimum lexicographique de $P(\overrightarrow{I_0})$ définit la première itération qui référence un élément $Y(\overrightarrow{I_0})$. Soit $\overrightarrow{I}_{min}$ ce point, dont les coordonnées sont des fonctions affines de $\overrightarrow{I_0}$. La position de cette itération, selon l'ordre d'exécution, nous donne la nouvelle position de la donnée $Y(\overrightarrow{I_0})$ en mémoire principale, relativement à l'adresse de base b du tableau Y. Cette nouvelle position garantit l'optimalité spatiale, car la donnée référencée juste avant, ou juste après $Y(\overrightarrow{I_0})$, sera contiguë à $Y(\overrightarrow{I_0})$ en mémoire.

La position de l'itération $\overrightarrow{I}_{min}$ est déterminée en calculant le nombre d'itérations exécutées avant $\overrightarrow{I}_{min}$. L'ensemble de ces itérations est défini par :

$$P(\overrightarrow{I_{min}}) = \{\overrightarrow{I} \in P | \overrightarrow{I} \lhd \overrightarrow{I_{min}}\}$$

où $\triangleleft$ symbolise l'ordre lexicographique. Afin de calculer le nombre d'itérations défini par $P(\overrightarrow{I_{min}})$, l'inégalité lexicographique doit tout d'abord être transformée en un ensemble d'inégalités linéaires. Cette transformation consiste en une décomposition de $P(\overrightarrow{I_{min}})$ en une union de polyèdres convexes disjoints :

$$P(\overrightarrow{I_{min}}) = P_1(\overrightarrow{I_{min}}) \cup P_2(\overrightarrow{I_{min}}) \cup P_3(\overrightarrow{I_{min}}) \cup \ldots \cup P_n(\overrightarrow{I_{min}})$$

avec

$$\begin{array}{ll} P_1(\overrightarrow{I_{min}}) & = \{\overrightarrow{I} \in P | i_1 < i_{1,min}\} \\ P_2(\overrightarrow{I_{min}}) & = \{\overrightarrow{I} \in P | i_1 = i_{1,min}, i_2 < i_{2,min}\} \\ P_3(\overrightarrow{I_{min}}) & = \{\overrightarrow{I} \in P | i_1 = i_{1,min}, i_2 = i_{2,min}, i_3 < i_{3,min}\} \\ \ldots & \\ P_n(\overrightarrow{I_{min}}) & = \{\overrightarrow{I} \in P | i_1 = i_{1,min}, i_2 = i_{2,min}, \ldots, i_{n-1} = i_{n-1,min}, i_n \leq i_{n,min}\} \end{array}$$

Puisque $\overrightarrow{I_{min}}$ est défini comme une fonction affine de $\overrightarrow{I_0}$, $P(\overrightarrow{I_{min}})$ est une union de polyèdres paramétrés par $\overrightarrow{I_0}$. En déterminant les polynômes d'Ehrhart $EP_q(\overrightarrow{I_0})$ de chacun de ces polyèdres, et en déduisant le polynôme somme, on obtient le nombre d'itérations définies par $P(\overrightarrow{I_{min}})$ et paramétré par $\overrightarrow{I_0}$. Ce polynôme définit une fonction de placement des éléments $Y(\overrightarrow{I_0})$ en mémoire principale, en donnant leur position relative à l'adresse de base b du tableau Y.

Cette fonction de placement peut constituer, pour un compilateur, le processus d'évaluation des références de tableaux : au lieu d'évaluer classiquement les références à un tableau [2], on évalue le polynôme d'Ehrhart déterminé préalablement, pour une localité spatiale optimale. Cette opération de compilation peut être vue comme la production du code exécutable suivant :

for $i_1 = l_1$ to h_1
  for $i_2 = l_2(i_1)$ to $h_2(i_1)$
    ...
      for $i_n = l_n(i_1, i_2, ..., i_{n-1})$ to $h_n(i_1, i_2, ..., i_{n-1})$
        $\ldots Y(EP(A \cdot (i_1, i_2, \ldots, i_n)^T + (o_1, o_2, \ldots, o_n)^T)) \ldots$

où le tableau Y est vu maintenant comme un tableau unidimensionnel.

Exemple 9 *(suite)* Le polyèdre $P(i0_1, i0_2, i0_3)$ définit au plus un point entier associé à une itération référençant un élément $Y(i0_1, i0_2, i0_3)$. Cette itération est définie par $\overrightarrow{I_{min}} = (\frac{i0_1+i0_2-i0_3-1}{2}, \frac{i0_1-i0_2+i0_3+3}{2}, \frac{-i0_1+i0_2+i0_3-3}{2})$. Le nombre d'itérations exécutées avant $\overrightarrow{I_{min}}$ est déterminé en calculant le polynôme d'Ehrhart de l'union de polyèdre suivante :

$$\begin{array}{l} P(\frac{i0_1+i0_2-i0_3-1}{2}, \frac{i0_1-i0_2+i0_3+3}{2}, \frac{-i0_1+i0_2+i0_3-3}{2}) \\ = \{(i_1, i_2, i_3) \in P | i_1 < \frac{i0_1+i0_2-i0_3-1}{2}, (i0_1, i0_2, i0_3) \in Conv(D_Y)\} \\ \cup \{(i_1, i_2, i_3) \in P | i_1 = \frac{i0_1+i0_2-i0_3-1}{2}, i_2 < \frac{i0_1-i0_2+i0_3+3}{2}, (i0_1, i0_2, i0_3) \in Conv(D_Y)\} \\ \cup \{(i_1, i_2, i_3) \in P | i_1 = \frac{i0_1+i0_2-i0_3-1}{2}, i_2 = \frac{i0_1-i0_2+i0_3+3}{2}, i_3 \leq \frac{-i0_1+i0_2+i0_3-3}{2}, \\ \qquad (i0_1, i0_2, i0_3) \in Conv(D_Y)\} \end{array}$$

Sur le premier polyèdre, notre programme de calcul du polynôme d'Ehrhart donne le résultat suivant :

$$\begin{array}{l} EP_1(i0_1, i0_2, i0_3) = -\frac{1}{8}i0_3^3 + \frac{3}{8}i0_2 i0_3^2 + \frac{3}{8}i0_1 i0_3^2 - \frac{3}{4}i0_3^2 - \frac{3}{8}i0_2^2 i0_3 - \frac{3}{4}i0_1 i0_2 i0_3 + \frac{3}{2}i0_2 i0_3 \\ -\frac{3}{8}i0_1^2 i0_3 + \frac{3}{2}i0_1 i0_3 - \frac{11}{8}i0_3 + \frac{1}{8}i0_2^3 + \frac{3}{8}i0_1 i0_2^2 - \frac{3}{4}i0_2^2 + \frac{3}{8}i0_1^2 i0_2 - \frac{3}{2}i0_1 i0_2 + \frac{11}{8}i0_2 + \frac{1}{8}i0_1^3 \\ -\frac{3}{4}i0_1^2 + \frac{11}{8}i0_1 - \frac{3}{4} \end{array}$$

Sur le deuxième polyèdre, on obtient :

$$EP_2(i0_1, i0_2, i0_3) = i0_1 i0_3 - i0_1 i0_2 + i0_3 - i0_2 + 2i0_1 + 2$$

et sur le troisième :

$$EP_3(i0_1, i0_2, i0_3) = \frac{3}{2}i0_3 - \frac{1}{2}i0_2 - \frac{1}{2}i0_1 + \frac{3}{2}$$

Finalement, l'adresse mémoire de tout élément $Y(i0_1, i0_2, i0_3)$, résultant en une localité spatiale optimale, est :

$$EP(i0_1, i0_2, i0_3) \times w + b = (EP_1(i0_1, i0_2, i0_3) + EP_2(i0_1, i0_2, i0_3) + EP_3(i0_1, i0_2, i0_3)) \times w + b$$

où b est l'adresse de base du tableau Y.

Par exemple, considérons trois itérations successives $(3, 5, 8)$, $(4, 4, 0)$ et $(4, 4, 1)$. Les éléments référencés par ces itérations sont respectivement $Y(7, 13, 13)$, $Y(7, 6, 4)$ et $Y(7, 7, 5)$. On vérifie que notre fonction d'évaluation de références résulte en des adresses mémoires contiguës pour ces éléments :

$$\begin{array}{lll} EP(7,13,13) & = 15+16+11 & = 42 \\ EP(7,6,4) & = 42+0+1 & = 43 \\ EP(7,7,5) & = 42+0+2 & = 44 \end{array}$$

Par conséquent, $Y(7, 13, 13)$, $Y(7, 6, 4)$ et $Y(7, 7, 5)$ seront mémorisés aux adresses $42 \times w + b$, $43 \times w + b$ et $44 \times w + b$, pour une localité spatiale optimale. □

Il est à noter que si le tableau Y n'est pas réutilisé temporellement, alors seul l'évaluation de l'adresse du premier élément de tableau est nécessaire. En effet, les adresses des éléments suivants peuvent être obtenues par simple incrémentation de cette adresse. En cas de réutilisation temporelle, le nombre d'évaluations peut sensiblement être réduit en effectuant de telles incrémentations pour chaque réutilisation d'une même suite d'éléments, et en mémorisant l'adresse du premier élément lors de la première et unique évaluation.

4.2.4 Conclusion et perspectives

L'objectif de ne considérer que les éléments effectivement accédés, qui n'est pas couvert par les autres méthodes existantes, nécessite un placement des données défini par une fonction non-linéaire. Cette particularité permet d'envisager notamment l'application de notre méthode à l'optimisation des traitements de matrices creuses.

Plusieurs développements sont à prévoir pour notre approche dans un futur proche : étendre la technique aux références multiples à un même tableau, développer une méthode non-linéaire de transformation de programme pour optimiser la localité temporelle, etc...

Parallèlement à l'objectif d'optimisation des performances de caches, l'enjeu de ces travaux se situe également dans le cadre de la compilation pour les processeurs à parallélisme d'instructions : une optimisation de la localité spatiale d'un programme améliore l'exploitation des unités SIMD d'un processeur, et plus généralement l'exploitation des unités fonctionnelles parallèles. C'est d'ailleurs ici que repose notre motivation à développer une activité dans ce domaine de compilation pour les processeurs à parallélisme explicite.

4.3 Les fenêtres de références généralisées (GRW)

4.3.1 Introduction

Les fenêtres de références ont été introduites pour la premiere fois par Gannon, Jalby et Gallivan [47, 40] afin de définir un modèle permettant d'évaluer les propriétés de localité d'un programme et de conditionner des transformations optimisantes. Dans la littérature actuelle, la fenêtre de référence évalue le nombre de défauts de cache dus aux capacités de taille.

Cette notion a été étendue afin de prendre en compte les défauts de caches dus aux conflits : les défauts causés par de nombreuses réutilisations du même bloc dans la mémoire cache. L'idée principale est de considérer chaque bloc dans la mémoire cache comme étant de la mémoire locale et d'évaluer la fenêtre de référence associée à ce bloc. Typiquement, dans les mémoires caches à accès direct, si la taille de la fenêtre de référence généralisée est plus grande que 1 à un moment de l'exécution du programme et pour un bloc donné dans la mémoire cache, des défauts de cache vont se produire après cet instant.

L'exploitation de ce modèle à la définition de transformations de programmes et de données est la prochaine étape.

4.3.2 Définition formelle

Afin de définir la *GRW*, nous nous basons sur la formulation initiale d'une fenêtre de référence : Étant donné un point p dans la trace de l'exécution d'un programme, la fenêtre de référence est l'ensemble des emplacements mémoires accédés dans le passé avant et au point p et qui seront réaccédés dans le futur après et au point p. Si la taille $|GRW|$ de la fenêtre de référence n'est jamais plus grande que la taille de la mémoire locale, toutes les réutilisations possibles seront exploitées et aucun accès mémoire ne sera utile pour recharger les données une fois qu'elles ont été chargées dans la mémoire cache.

Dans le cas d'une mémoire cache associative par bloc, nous étendons cette définition en deux temps. Dans un premier temps, les fenêtres de références sont étendues afin de prendre en compte la localité spatiale. Basiquement, ceci consiste à considérer les blocs mémoires au lieu des emplacements mémoires. Dans un second temps, l'associativité est prise en compte en ne considérant que les blocs mémoires qui vont se placer sur un bloc fixé de la mémoire cache. La fenêtre de référence généralisée se formule alors ainsi :

- pour un nid de boucles à bornes paramétrées affines où chaque instruction contient une référence paramétrée affine à un tableau,
- pour un bloc s fixé dans la mémoire cache,
- pour une instruction S_0 et un vecteur d'itération $\vec{i_0}$,

$$GRW^s_{S_0(\vec{i_0})} = \{B \in M_s, \exists S, \exists \vec{i}, S(\vec{i}) \leq S_0(\vec{i_0}), B = BN(A_S(\vec{i})), s = CN(A_S(\vec{i}))\}$$

$$\bigcap$$

$$\{B \in M_s, \exists T, \exists \vec{j}, S_0(\vec{i_0}) \leq T(\vec{j}), B = BN(A_T(\vec{j})), s = CN(A_T(\vec{j}))\}$$

avec :

- M_s l'ensemble des blocs de la mémoire principale qui vont se charger dans le bloc s de la mémoire cache,
- S et T deux instructions dans le code source,
- $\vec{i}$,$\vec{j}$ deux itérations dans le nid de boucle considéré,
- Pour une instruction S , $S(\vec{i})$ représente la position de l'instruction S à l'itération $\vec{i}$ selon l'ordre lexicographique d'exécution de toutes les itérations, et $A_S(\vec{i})$ l'adresse absolue accédée par l'instance de S à l'itération $\vec{i}$,
- Les fonctions BN et CN associent à une adresse absolue dans la mémoire principale, le numéro de bloc dans la mémoire principale et dans la mémoire cache.

La taille de la fenêtre $|GRW|$ est le nombre de points entiers de l'ensemble GRW. La détermination de cette taille nécessite une reformulation équivalente. Afin de simplifier, nous supposons que deux tableaux ne partagent pas le même bloc dans la mémoire principale : deux références n'accèdent pas à un même bloc sauf dans le cas où elles accèdent le même tableau.

Pour deux instructions S et T qui accèdent au même tableau, la GRW associée est :

$$M_s \cap BN\{A_S(\vec{i}), S(\vec{i}) \leq S_0(\vec{i_0})\} \cap BN\{A_T(\vec{j}), S_0(\vec{i_0}) \leq T(\vec{j})\}$$

La fenêtre de référence generalisée est l'union des ensembles des fenêtres de références généralisées de telles paires d'instructions. Par comparaison au formalisme des *cache miss equation* introduit par Ghosh *et al.* [48], ce formalisme entraine une complexité de calcul inférieure. Toutefois, les ensembles élémentaires ne sont pas en général des polyèdres mais des projections de polyèdres, ce qui les rendent plus difficile à manipuler, spécialement quand on énumère les points entiers [19, 22].

4.3.3 Perspectives de développement

La considération de blocs de mémoire entraine la prise en compte de contraintes à opérations modulo et division entière dans la définition des fenêtres de références généralisées. De plus, l'introduction de certains paramètres tels que la taille des blocs résulte en des contraintes non-linéaires, où ces paramètres sont coefficients des variables considérés. Par conséquent, ce modèle pose des problèmes mathématiques fondamentaux.

Son intérêt repose sur une analyse globale du comportement du nid de boucle et de la mémoire cache considérés. De futurs développements tenteront d'affiner ce modèle afin qu'il puisse conditionner des transformations de programmes et d'organisation de données, non plus en traitant les programmes instruction par instruction et référence par référence, mais en regroupant tous ces facteurs. Ce modèle devra donc être enrichi de paramètres conditionnant de telles transformations.

Chapitre 5

Projets de recherches

Mes objectifs couvrent deux aspects principaux : un aspect "fondamental" recouvrant le développement d'outils géométriques et mathématiques, et un aspect de résolution de problèmes d'optimisation et de parallélisation de programmes, proche des architectures matérielles.

5.1 La conception d'outils géométriques et mathématiques

Les méthodes et outils mathématiques que nous désirons continuer de proposer doivent être utilisables en informatique pour permettre de repousser les techniques empiriques de programmation des ordinateurs, et en particulier des ordinateurs multi-processeurs. Avec nos résultats sur les polynômes d'Ehrhart et les polyèdres paramétrés, nous avons montré qu'une avancée fondamentale dans le domaine des mathématiques calculables par ordinateur permet de grandes avancées dans le domaine de la compilation de programmes. En effet, beaucoup de questions donnant lieu à thèmes de recherches ont trouvé ici des réponses quasi immédiates.

Dans un avenir proche, il est nécessaire de pouvoir traiter les cas d'inéquations non-linéaires. Le domaine des bases de Gröbner [11] semble constituer une réponse possible, bien que le cas général des inéquations, ainsi que le cas général d'équations paramétriques, n'ont été à ce jour que peu traités en mathématiques. Nous nous attacherons donc à tenter de faire progresser ces aspects.

5.2 Optimisation de l'utilisation mémoire

Nous comptons proposer des méthodes automatiques de transformation de programmes et d'organisation mémoire des données, ainsi que de nouveaux processus de compilation, permettant d'optimiser l'utilisation de l'ensemble de la hiérarchie mémoire des ordinateurs. De plus, le cas des machines multi-processeurs pose des problèmes supplémentaires de cohérence des données stockées dans les mémoires locales (notamment les mémoires caches) des processeurs, ou plus généralement des noeuds de l'architecture. Nous pensons pouvoir proposer des solutions efficaces à ces problèmes grâce à l'utilisation de nos outils mathématiques et de leurs futures extensions.

Notre approche dans ce domaine est double : un ensemble de transformations efficaces, basées sur la non-linéarité, telles que celle décrite à la section 4.2 ; ou une modélisation paramétrique des programmes, où l'optimisation mémoire consiste en une maximisation ou une minimisation des paramètres, réalisée par l'application de transformations adaptées (section 4.3).

5.3 Compilation de programmes pour les processeurs à parallélisme explicite (EPIC)

Avec l'arrivée prochaine de la nouvelle génération de processeurs Intel à architecture 64 bits (IA-64), dont le premier sera l'*Itanium*, apparaissent sur le marché des processeurs à parallélisme d'instructions explicite (EPIC - *explicitly parallel instruction computing*). Ces processeurs, dotés de plusieurs unités fonctionnelles, sont capables d'exécuter plusieurs instructions simultanément, et peuvent donc offrir des performances élevées, grâce à cette capacité à déclencher plusieurs instructions qualifiées de parallèles par le compilateur.

Toutefois, ces améliorations technologiques ne peuvent être exploitées qu'avec un code exécutable efficace, décrivant explicitement les instructions pouvant être exécutées en parallèle. C'est donc au compilateur qu'il revient de détecter le parallélisme dans les programmes sources, et de formater des mots d'instructions devant être exécutées en parallèle par les unités fonctionnelles.

En association avec ses moyens de calculs parallèles, le processeur est également capable d'effectuer des exécutions spéculatives afin de gagner encore en performance : par exemple, deux branches d'une conditionnelle commencent à être exécutées simultanément avant de connaître le résultat d'un test.

Notre ambition est donc, à partir d'une connaissance détaillée de l'architecture des processeurs EPIC, et du savoir-faire actuel en matière de parallélisation automatique et d'optimisation de programmes, de proposer des méthodes de compilation efficaces pour ces processeurs. Cet objectif nécessitera un investissement dans le domaine du parallélisme d'instruction et de l'analyse de programmes dépassant l'analyse des nids de boucles. De plus, le caractère spéculatif des exécutions offre de nouvelles possibilités au parallélisme, où les dépendances entre instructions ne sont plus un frein à l'accélération des traitements.

Bibliographie

[1] J.R. Allen et K. Kennedy. Automatic Translation of Fortran Programs to vector form. *ACM Transactions on Programming Languages and Systems*, 9(4):491-542, octobre 1987.

[2] A.V. Aho, R. Sethi et J.D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley, 1987.

[3] J.M. Anderson et M.S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices*, 28(6):112-115, 1993.

[4] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Pub., Norwell, MA, USA, 1988.

[5] U. Banerjee. Unimodular transformations of double loops. *Proc. of Third Workshop on Progr. Lang. Compilers for Parallel Comput.*, August 1990.

[6] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali et P. Stodghill. Solving alignment using elementary linear algebra. In *Languages and Compilers for Parallel Computing.* LNCS 892, Springer-Verlag, 1995.

[7] A. Benanini et Y. Robert. Spacetime-minimal systolic arrays for Gaussian eliminnation and the Algebraic Path Problem. *In S.Y. Kung and E. Swartzlander editors, International Conference on Application-Specific Array Processors*, Princeton, New Jersey, Sept. 1990, IEEE Computer Society Press.

[8] J.C. Bermond, C. Peyrat, I. Sakho et M. Tchuente. Parallelization of Gauss elimination algorithm on systolic arrays. Rapport de recherche 430, Université Paris-Sud, Centre d'Orsay, Laboratoire de recherche en informatique, juillet 1988.

[9] M.S. Bazaraa, H.D. Sherali et C.M. Shetty. *Nonlinear Programming, Theory and Algorithms, second edition.* Wiley, New York, 1993.

[10] Pierre Boulet, Xavier Redon. Communication Pre-evaluation in HPF. Euro-Par'98 Parallel Processing, LNCS 1470:263-272, 1998.

[11] B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. *Multidimensional Systems Theory*, N.K. Bose, ed., D. REidel Publishing Co., Netherlands, pp 184-232, 1985.

[12] J. Bu, E.F. Depreterre et P. Dewilde. A design methodology for fixed-size systolic arrays. *In S.Y. Kung and E. Swartzlander editors, International Conference on Application-Specific Array Processors*, pages 591-602, Princeton, New Jersey, Sept. 1990, IEEE Computer Society Press.

[13] P.R. Capello. A spacetime-minimal systolic array for matrix product. *In Systolic array processors*, J. McCanny *et al.* eds, Prentice Hall (1989), pp. 347-356.

[14] S. Chatterjee, J.R. Gilbert, L. Oliker, R. Schreiber et T.J. Sheffler. Algorithms for automatic alignment of arrays. *Journal of Parallel and Distributed Computing*, 38(2):145-157, novembre 1996.

[15] Ph. Clauss. *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* Thèse de l'Université de Franche-Comté, Besançon, mai 1990.

[16] Ph. Clauss. Mapping systolic algorithms on Distributed Memory Computers. *EDMCC2*, Munich, avril 1991, LNCS no. 487, Spinger Verlag.

[17] Ph. Clauss. An efficient allocation strategy for mapping affine recurrences into space and time optimal regular processor arrays. In C. Jesshope, V. Jossifov and W. Wilhelmi editors, *Int. Conf. PARCELLA'94*, pages 257-266, Potsdam, Germany, Sept. 1994, Akademie Verlag.

[18] 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.

[19] Ph. Clauss. Handling Memory Cache Policy with Integer Points Countings. LNCS 1300, pp 285-293, *Euro-Par'97*, Passau, Germany, August 1997.

[20] Ph. Clauss. Sur les extensions et les utilisations en Informatique d'un résultat Mathématique : Les polynômes d'Ehrhart. *Bulletin de l'APMEP (Association Française des Professeurs de Mathématiques de l'Enseignement Public)*, numéros 418 et 419, septembre et octobre 1998.

[21] Ph. Clauss, V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, Chicago, Illinois, August 1996.

[22] Ph. Clauss et V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, Vol. 19, No. 2, Kluwer Academic Pub., July 1998.

[23] Ph. Clauss, C. Mongenet. Synthesis aspects in the design of efficient processor arrays from affine recurrence equations. *Journal of Symbolic Computation* 15, 547-569, Academic Press (1993).

[24] Ph. Clauss, C. Mongenet, G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. *In S.Y. Kung and E. Swartzlander editors, International Conference on Application-Specific Array Processors*, pages 4-18, Princeton, New Jersey, Sept. 1990, IEEE Computer Society Press.

[25] Ph. Clauss, C. Mongenet et G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the Algebraic Path Problem. *Int. workshop "Algorithms and Parallel VLSI Architectures II"*, Bonas, juin 1991, North-Holland Pub.

[26] Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the Algebraic Path Problem : A new contribution. *Parallel Computing*, 18 (1992).

[27] Ph. Clauss, C. Mongenet, G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays, *Journal of VLSI Signal Processing*, 4, pages 27-36 (1992).

[28] Ph. Clauss et G.-R. Perrin. Optimal mapping of systolic algorithms by regular instruction shifts. *Int. Conf. on Application-Specific Array Processors, ASAP'94*, pages 224-235, San Francisco, California, IEEE Computer Society Press.

[29] Ph. Clauss, V. Loechner et D.K. Wilde. Deriving Formulae to Count Solutions to Parameterized Linear Systems using Ehrhart Polynomials: Applications to the Analysis of Nested-Loop Programs. Rapport de recherche ICPS 97-01, 1997.

[30] Ph. Clauss et B. Meister. Automatic Memory Layout Transformations to Optimize Spatial Locality in Parameterized Loop Nests. *4th Annual Workshop on Interaction between Compilers and Computer Architectures, INTERACT-4*, Toulouse, France, 9 janvier 2000.

[31] Michel Cosnard, Emmanuel Jeannot et Laurence Rougeot, Low Memory Cost Dynamic Scheduling of Large Coarse Grain Task Graphs, 1998 IEEE International Parallel Processing Symposium (IPPS'98), Orlando, Florida.

[32] A. Darte. Regular Partitionning for synthesizing fixed-size systolic arrays. *Integration, the VLSI Journal*, 12:293-301, 1991.

[33] A. Darte et Y. Robert. On the alignment problem. *Parallel Processing Letters*, 4(3):259-270, 1994.

[34] A. Darte et F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *International Journal of Parallel Programming*, Vol. 25, No. 6, pp. 447-497, 1997.

[35] M. Dion et 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, mai 1995, Milan, Italie.

[36] E. Ehrhart. Sur les polyèdres rationnels homothétiques à n dimensions. Compte-rendu de l'Académie des Sciences, Paris, 254:616-618, 1962.

[37] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire I. *Journal Reine Angew. Math.*, 226:1-29,1967.

[38] E. Ehrhart. Sur un problème de géométrie diophantienne linéaire II. *Journal Reine Angew. Math.*, 227:25-49,1967.

[39] E. Ehrhart. *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*. International Series of Numerical Mathematics, vol. 35, Birkhäuser Verlag, Basel/Stuttgart, 1977.

[40] C. Eisenbeis, W. Jalby, D. Windheiser et F. Bodin. A strategy for array management in local memory. Rapport de recherche INRIA RR-1262, juillet 1990.

[41] P. Feautrier. Parametric Integer Programming. *RAIRO, Recherche Opérationnelle*, 22:213-268, September 1988.

[42] P. Feautrier. Dataflow Analysis of scalar and array references. *Int. J. of Parallel Programming*, Vol. 20, No. 1, pp. 23-53, 1991.

[43] P. Feautrier. Some efficient solutions to the affine scheduling problem, part II: multidimensionnal time. *Int. J. Parallel Programming*, 21(6):389-420, décembre 1992.

[44] P. Feautrier. Towards automatic distribution. *Parallel Processing Letters*, 4(3):233-244, 1994.

[45] P. Feautrier. Automatic Parallelization in the Polytope Model. Rapport technique 1996/8, PRiSM, Université de Versailles, juin 1996.

[46] J. Ferrante, V. Sarkar et W. Thrash. On estimating and ehancing cache effectiveness. *Advances in Languages and Compilers for Parallel Processing*, pp. 328-343, The MIT Press, 1991.

[47] D. Gannon, W. Jalby et K. Gallivan. Strategies for Cache and Local Memory Management by Global Program Transformation. Proceedings of the International Conference on Supercomputing, Springer Verlag, New York, 1987 and Journal of Parallel and Distributed Computing, Oct. 1988.

[48] S. Ghosh, M. Martonosi et S. Malik. Cache Miss Equations: An Analytical Representation of Cache Misses. Proceedings of the 11th ACM International Conference on Supercomputing, Vienna, Austria, July 1997.

[49] S. Ghosh, M. Martonosi et S. Malik. Precise Miss Analysis for Program Transformations with Caches of Arbitrary Associativity. Proceedings of the 8th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-VIII, San Jose, California, October 4-7, 1998.

[50] P.M. Gruber et C.G. Lekkerkerker. *Geometry of numbers*. North-Holland, Amsterdam, 1987.

[51] M. Griebl et Christian Lengauer. The Loop Parallelizer *LooPo.Proc. Sixth Workshop on Compilers for Parallel Computers, Michael Gerndt editor, Konferenzen des Forschungszentrums Jülich series*, vol. 21, Forschungszentrum Jülich pub., 1996, pp 311-320.

[52] M. Haghighat et C. Polychronopoulos. Symbolic Analysis: A basis for parallelization, optimization and scheduling of programs. *U. Banerjee et al., editor, Languages and Compilers for Parallel Computing*. LNCS 768, Springer-Verlag, août 1993. *Proc. 6th annual workshop on Programming Languages and Compilers for Parallel Computing.*

[53] M. Haghighat et C. Polychronopoulos. Symbolic Analysis: A basis for parallelization, optimization and scheduling of programs. Technical Report 1317, CSRD, Univ. of Illinois, août 1993.

[54] F. Heine et A. Slowik. Volume Driven Data Distribution for NUMA-Machines. Research Report, University of Paderborn, Germany, 1999.

[55] D.E. Hudak et S.G. Abraham, editors. *Compiling Parallel Loops for high performance computers*. Kluwer Academic Publishers, 1993.

[56] F. Irigoin, P. Jouvelot et Rémy Triolet. Semantical interprocedural parallelization: an overview of the PIPS project. In *Proc. of the 1991 ACM Int. Conf. on Supercomputing*, Cologne, RFA, juin 1991.

[57] M. Kandemir, A. Choudhary, J. Ramanujam, et P. Banerjee. Optimizing Spatial Locality in Loop Nests using Linear Algebra. *Proc. 7th Int. Workshop on Compilers for Parallel Computers*, Sweden, June 1998.

[58] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, et J. Ramanujam. A Hyperplane Based Approach for Optimizing Spatial Locality in Loop Nests. *Proc. 11th ACM Int. Conf. on Supercomputing*, Melbourne, Australia, July 1998.

[59] R.M. Karp, R.E. Miller et S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563-590, juillet 1967.

[60] S.Y. Kung, S.C. Lo, P.S. Lewis. Optimal Systolic Design for the transitive closure and the shortest path problems. *IEEE Trans. on Computers*, Vol. C-36, No. 5, mai 1987.

[61] L. Lamport. The parallel execution of DO loops. *Communications of the ACM*, 17(2):83-93, février 1974.

[62] C. Lengauer. Loop Parallelization in the Polytope Model. *CONCUR'93, E. Best editor*, Springer-Verlag, LNCS 715, 1993, pp. 398–416.

[63] S.-T. Leung et J. Zahorjan. Optimizing data locality by array restructuring. University of Washington, Department of Computer Science and Engineering. Technical Report 95-09-01, 1995.

[64] W. Li. *Compiling for NUMA parallel machines*. Ph.D. dissertation, Dept. Computer Science, Cornell University, Ithaca, NY, 1993.

[65] J. Li et M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of Parallel and Distributed Computing*, 13(2), 1991.

[66] W. Li et K. Pingali. A singular loop transformation framework based on nonsigular matrices. *Int. J. of Parallel Programming*, 22(2):183-205, 1994.

[67] V. Loechner. *Contribution à l'étude des polyèdres paramétrés et applications en parallélisation automatique*. Thèse de l'Université Louis Pasteur, Strasbourg, décembre 1997.

[68] V. Loechner et C. Mongenet. A toolbox for affine recurrence equations parallelization. *HPCN'95*, Milan, mai 1995, LNCS 919, Springer Verlag, pp. 263-268.

[69] V. Loechner et D.K. Wilde. Parameterized polyhedra and their vertices. *Int. J. of Parallel Programming*, Vol. 25, No. 6, 1997.

[70] V. Loechner et C. Mongenet. Communication optimization for affine recurrence equations using broadcast and locality. Technical report RR 98-03, ICPS. A paraître dans *Int. J. of Parallel Programming*, février 2000.

[71] V. Loechner et C. Mongenet. Memory Cost Due to Anticipated Broadcast. *8th Workshop on Compilers for Parallel Computers*, Aussois (France), January 2000.

[72] I.G. Macdonald. The volume of a lattice polyhedron. *Proc. Camb. Phil. Soc.*, 59:719-726, 1963.

[73] I.G. Macdonald. Polynomials associated with finite cell-complexes. *J. London Math. Soc.*, 4(2):181-192, 1971.

[74] P. McMullen. Lattice invariant valuations on rational polytopes. *Arch. Math. (Basel)*, 31:509-516, 1978/79.

[75] C. Mongenet. *Une méthode de conception d'algorithmes systoliques, résultats théoriques et réalisation.* Thèse de l'INPL, Nancy, 1985.

[76] C. Mongenet. Affine dependence classification for communications minimization. *Int. J. of Parallel Programming*, 25(6), décembre 1997.

[77] C. Mongenet, Ph. Clauss, G.-R. Perrin. Geometrical Tools to map Systems of Affine Recurrence Equations on Regular Arrays. *Acta Informatica* 31, 137-160 (1994).

[78] D.I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE Conf.*, Vol. 71-1, janvier 1983, pp. 113-120.

[79] M.F.P. O'Boyle et P.M.W. Knijnenburg. Nonsingular Data Transformations: Definition, Validity, and Applications. *Int. J. of Parallel Programming*, Vol. 27, No. 3, pp. 131-159, juin 1999.

[80] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102-114, August 1992.

[81] W. Pugh. Counting Solutions to Presburger Formulas: How and Why. *Proc. of the 1994 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 1994.

[82] P. Quinton, P. Gachet. Manuel d'utilisation de *Diastol.* PI 233, IRISA Rennes, 1984.

[83] P. Quinton. The systematic design of systolic arrays. In F. Fogelman Soulie, Y. Robert et M. Tchuente, éditeurs, *Automata Networks in Computer Science*, chapitre 9, pages 229-260, Manchester University Press, 1987.

[84] S.K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, octobre 1985.

[85] J. Ramanujam et P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Transactions on parallel and distributed systems*, 2(4):472-482, octobre 1991.

[86] V.P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms.* PhD thesis, Stanford University, décembre 1988.

[87] C.J. Scheiman et P.R. Cappello. A Processor-Time Minimal Systolic Array for Transitive Closure. *In S.Y. Kung and E. Swartzlander editors, International Conference on Application-Specific Array Processors*, pp. 19-30, Princeton University, NJ, septembre 1990, IEEE Computer Society Press.

[88] C. Scheiman and P. Cappello. A Processor-Time-Minimal Schedule for the Standard Tensor Product Algorithm. *Int. Conf. on Application-Specific Array Processors, ASAP'94*, pages 176-187, San Francisco, California, IEEE Computer Society Press.

[89] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, New York, 1986.

[90] R.P. Stanley. Decompositions of rational convex polytopes. *Annals of Discrete Math.*, 6:333-342, 1980.

[91] R.P. Stanley. *Combinatorics and Commutative Algebra.* Birkhäuser, Boston, 1983.

[92] R.P. Stanley. *Enumerative Combinatorics Volume I.* Wadsworth & Brooks/Cole, Monterey, California, 1986.

[93] N. Tawbi et P. Feautrier. Processor allocation and loop scheduling on multiprocessor computers. *Proc. of the 1992 Int. Conf. on Supercomputing*, pp. 63-71, juillet 1992.

[94] N. Tawbi. Estimation of nested loop execution time by integer arithmetics in convex polyhedra. *Proc. of the 1994 Int. Parallel Processing Symposium*, avril 1994.

[95] M. Tchuente et I. Sakho. Méthode de conception d'algorithmes parallèles pour réseaux réguliers. Techniques et Sciences Informatiques (TSI), 0752-4072/89/01/63-10/ 3,10.

[96] F. Vivien. *Détection de parallélisme dans les boucles imbriquées.* Thèse de l'ENS Lyon, décembre 1997.

[97] H. Le Verge, V. Van Dongen et D.K. Wilde. La synthèse de nids de boucles avec la bibliothèque polyédrique. *RenPar'6*, juin 1994. Version en anglais : Loop Nest Synthesis Using the Polyhedral Library. IRISA technical report 830, May 1994.

[98] D.K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, 1993. Also published as IRISA Technical Report PI 785, Rennes, France, 1993.

[99] D.K. Wilde et O. Sié. Regular Array Synthesis using Alpha. *Int. Conf. on Application Specific Array Processors, ASAP'94*, San Francisco, California, août 1994.

[100] Y. Wong, J-M. Delosme. Optimization of processor count for systolic arrays. Research report YALEU/DCS/RR-697, May 1989.

[101] M.E. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1996.

[102] M.E. Wolfe et M.S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452-471, octobre 1991.

Annexe 1

Collaborations et encadrements

Collaborations et programmes de recherches

Mes activités de recherche ont donné lieu à des échanges réguliers avec les meilleures équipes françaises du domaine dans le cadre :

- du PRC C3 (Coopération, Concurrence et Communication) du CNRS de 1987 à 1992,
- du GDR 1169 PRS (Parallélisme, Réseau et Système) du CNRS et du MESR de 1993 à 1997,
- du thème iHPerf du GDR ARP (Architecture, Réseaux, Parallélisme) du CNRS et du MESR depuis 1998.

Mes recherches sur les polynômes d'Ehrhart m'ont donné l'occasion de rencontrer plusieurs fois Eugène Ehrhart à la bibliothèque de mathématiques de l'ULP ou à son domicile. Sans qu'il s'agisse réellement d'une collaboration, mes conversations avec cet homme exceptionnel sont pour moi inoubliables. Monsieur Ehrhart est le parfait représentant d'une recherche passionnée et désintéressée.

Une collaboration fructueuse avec D. K. Wilde, de l'Université de Brigham Young, USA, lors de sa venue à l'ICPS en tant que chercheur invité pour une période de 3 mois, a permis le développement du logiciel de calcul des polynômes d'Ehrhart. Cette collaboration s'est également concrétisée par l'écriture d'un article [29].

Une collaboration sur le thème de l'optimisation des performances de caches est en cours avec C. Eisenbeis de l'INRIA de Rocquencourt, avec qui je co-encadre un étudiant en thèse à l'ULP. Ce travail est basé sur notre deuxième approche de modélisation paramétrique des accès mémoire.

L'optimisation des mémoires caches est un thème important d'une collaboration naissante entre l'ICPS et la société Silicon Graphics Inc. (centre de recherche de Minneapolis, USA), dont l'objectif est le développement d'un transformateur automatique de programme Fortran 90 en programme parallèle OpenMP.

Activités d'encadrement

À ce jour, mes activités d'encadrement consistent en :

- l'encadrement d'un étudiant de DEA à l'ULP en 1995/96, sur le thème des extensions et applications des polynômes d'Ehrhart ;
- le co-encadrement en 1996/1997 de V. Loechner, étudiant en thèse sous la direction de C. Mongenet, sur le thème des polyèdres paramétrés et des polynômes d'Ehrhart [67] ;
- l'encadrement d'un stagiaire de licence d'informatique à l'ICPS en juillet et août 1999, sur le thème de l'optimisation de la localité spatiale ; ce travail a donné lieu à l'écriture d'un article accepté au workshop *INTERACT-4* [30] ;
- le co-encadrement en cours, avec C. Eisenbeis de l'INRIA, du travail de thèse d'Andry Randrianatoavina, sur le thème de l'optimisation des performances de caches.

Annexe 2

Liste des publications de Philippe Clauss

Journaux internationaux

Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the Algebraic Path Problem : A new contribution. *Parallel Computing*, 18 (1992).

Ph. Clauss, C. Mongenet, G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays, *Journal of VLSI Signal Processing*, 4, pages 27-36 (1992).

Ph. Clauss, C. Mongenet. Synthesis aspects in the design of efficient processor arrays from affine recurrence equations. *Journal of Symbolic Computation* 15, 547-569, Academic Press (1993).

C. Mongenet, Ph. Clauss, G.-R. Perrin. Geometrical Tools to map Systems of Affine Recurrence Equations on Regular Arrays. *Acta Informatica* 31, 137-160 (1994).

Ph. Clauss and V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, Vol. 19, No. 2, Kluwer Academic Pub., July 1998.

Conférences internationales avec comité de lecture

Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthesis of efficient systolic arrays for matrix algebra problems. *Int. Symposium on High Performance Computer Systems*, Paris, décembre 1987, North-Holland Pub.

Ph. Clauss et G.-R. Perrin. Synthesis of process arrays. *CONPAR'88*, Manchester, septembre 1988, Cambridge University Press.

Ph. Clauss, C. Mongenet, G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. *In S.Y. Kung and E. Swartzlander editors, International Conference on Application-Specific Array Processors*, pages 4-18, Princeton, New Jersey, Sept. 1990, IEEE Computer Society Press.

Ph. Clauss. Mapping systolic algorithms on Distributed Memory Computers. *EDMCC2*, Munich, avril 1991, LNCS no. 487, Spinger Verlag.

C. Mongenet, Ph. Clauss, G.-R. Perrin. A geometrical coding to compile affine recurrences equations. *5th Int. Parallel Symposium*, mai 1991, IEEE Computer Society Press.

Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the Algebraic Path Problem. *Int. workshop "Algorithms and Parallel VLSI Architectures II"*, Bonas, juin 1991, North-Holland Pub.

Ph. Clauss. An efficient allocation strategy for mapping affine recurrences into space and time optimal regular processor arrays. In C. Jesshope, V. Jossifov and W. Wilhelmi editors, *Int. Conf. PARCELLA'94*, pages 257-266, Potsdam, Germany, Sept. 1994, Akademie Verlag.

Ph. Clauss et G.-R. Perrin. Optimal mapping of systolic algorithms by regular instruction shifts. *Int. Conf. on Application-Specific Array Processors, ASAP'94*, pages 224-235, San Francisco, California, IEEE Computer Society Press.

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.

Ph. Clauss et V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, Chicago, Illinois, August 1996.

Ph. Clauss. Handling Memory Cache Policy with Integer Points Countings. LNCS 1300, pp 285-293, *Euro-Par'97*, Passau, Germany, August 1997.

Ph. Clauss et B. Meister. Automatic Memory Layout Transformations to Optimize Spatial Locality in Parameterized Loop Nests. *4th Annual Workshop on Interaction between Compilers and Computer Architectures, INTERACT-4*, Toulouse, France, 9 janvier 2000.

Journaux nationaux

Ph. Clauss. Sur les extensions et les utilisations en Informatique d'une découverte Mathématique : Les polynômes d'Ehrhart. *Bulletin de l'APMEP (Association Française des Professeurs de Mathématiques de l'Enseignement Public)*, numéros 418 et 419, septembre et octobre 1998.

Conférences nationales

Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthèse de réseaux systoliques hétérogènes avec *Systol.* Journées Architectures C^3, Sophia-Antipolis, juillet 1987, Bigre+Globule n° 56.

Ph. Clauss. Sur le dénombrement des solutions d'un système linéaire en nombres entiers paramétrique. Présentation aux *Rencontres sur les problèmes linéaires en nombres entiers, PL-NE'96*, INRIA Rocquencourt, 28-30 octobre 1996.

Ph. Clauss. Les polyèdres d'Ehrhart pour l'analyse et la transformation de programmes. Présentation à la journée PRS *Parallélisme de données*, ENS Lyon, janvier 1996.

Contributions à des livres

Ph. Clauss, C. Mongenet, G.-R. Perrin. Implantations optimales d'équations récurrentes affines. *Algorithmique parallèle, collection études et recherches en informatique*, Masson 1992.

Rapports de recherches

Ph. Clauss. Advances in parameterized linear diophantine equations for precise program analysis. ICPS RR 98-02, septembre 1998.

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. ICPS RR 97-01, April 1997.

Ph. Clauss, The Volume of a Lattice Polyhedron to Enumerate Processors and Parallelism, research report ICPS 95-11, 1995.

Thèse

Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones. Thèse de l'Université de Franche-Comté, Besançon, mai 1990.

Annexe 3

Curriculum-Vitae

Philippe Clauss
né le 26 décembre 1964 à Barr (Bas-Rhin)
Célibataire

Formation :

1982 Baccalauréat série C à Sélestat (67).

1984 B.T.S. Services Informatiques à Strasbourg.

1986 Maîtrise d'informatique mention assez-bien, Université de Franche-Comté, Besançon.

1987 DEA Automatique, Informatique et Robotique mention assez-bien, Université de Franche-Comté, Besançon. Titre : *Contribution à la synthèse de réseaux systoliques.* Responsables de stage : Catherine Mongenet et Guy-René Perrin.

1990 Thèse de Doctorat en informatique mention très honorable soutenue le 25 mai 1990, Université de Franche-Comté, Besançon. Titre : *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* Directeur de thèse : Guy-René Perrin. Jury : Yves Robert (président), Patrice Quinton (rapporteur), Maurice Tchuente (rapporteur), Didier Arquès (examinateur), Catherine Mongenet (examinateur), Guy-René Perrin (examinateur).

Prises de fonction :

- Maître de conférences à l'Université Robert Schuman de 1991 à 1996.
- Maître de conférences à l'Université Louis Pasteur depuis 1996.

Responsabilités collectives :

- Responsable de la mention *calcul parallèle* du DESS Informatique de l'ULP depuis 1998.

- Membre titulaire de la commission de spécialistes section 27 de l'ULP depuis 1999. Membre suppléant de 1998 à 1999.
- Membre titulaire de la commission de spécialistes section 27 de l'Université de Metz depuis 1997.
- Membre titulaire de la commission de spécialistes section 27 de l'École des Mines de Nancy depuis 1999. Membre suppléant de 1998 à 1999.
- Organisateur du *forum des emplois en informatique* de l'ULP en 1999.
- Membre du comité de programme de RenPar'10 en juin 1998.
- Membre du bureau de direction du département informatique de l'ULP depuis 1997, responsable de la communication.

Activités d'enseignement :

- Cours d'architectures parallèles en DESS informatique.
- Cours de programmation en langage à parallélisme de données en DESS informatique.
- Cours d'outils de parallélisation et d'optimisation de programmes en DEA informatique.
- Cours, TD et TP de génie logiciel en licence et maîtrise d'informatique.
- TD de mathématiques pour l'informatique en IUP GMI première année.
- TD de bases de données en IUP GMI deuxième année.
- Cours de Fortran et parallélisme en troisième année de l'École Nationale Supérieure de Physique de Strasbourg.
- TP de programmation C en licence EEA.

Annexe 4

Publications jointes

Dans cette annexe se trouvent quelques publications résumant mes travaux :

Ph. Clauss, C. Mongenet, G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the Algebraic Path Problem : A new contribution. *Parallel Computing*, 18 (1992).

Ph. Clauss et G.-R. Perrin. Optimal mapping of systolic algorithms by regular instruction shifts. *Int. Conf. on Application-Specific Array Processors, ASAP'94*, pages 224-235, San Francisco, California, IEEE Computer Society Press.

Ph. Clauss and V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, Vol. 19, No. 2, Kluwer Academic Pub., juillet 1998.

Ph. Clauss. Handling Memory Cache Policy with Integer Points Countings. LNCS 1300, pp 285-293, *Euro-Par'97*, Passau, Germany, août 1997.

Ph. Clauss. Sur les extensions et les utilisations en Informatique d'une découverte Mathématique : Les polynômes d'Ehrhart. *Bulletin de l'APMEP (Association Française des Professeurs de Mathématiques de l'Enseignement Public)*, numéros 418 et 419, septembre et octobre 1998.

Ph. Clauss and B. Meister. Automatic Memory Layout Transformations to Optimize Spatial Locality in Parameterized Loop Nests. *4th Annual Workshop on Interaction between Compilers and Computer Architectures, INTERACT-4*, Toulouse, France, 9 janvier 2000.