Localité de données et compression mémoire dans les opérations *stencil*

Benoît Meister

LSIIT/ICPS - ULP
Pôle API - bd Sébastien Brant
F-67400 Illkirch

meister@icps.u-strasbg.fr

RÉSUMÉ. *L'un des objectifs principaux actuels en optimisation de programmes et en parallélisation est l'amélioration de la localité des accès aux données manipulées par les programmes traités. Celle-ci se sépare en deux aspects. Des accès rapprochés dans le temps à un même ensemble de données physiquement regroupées en mémoire définissent la* localité spatiale*, alors que des accès à une même donnée séparés par un temps suffisamment court définissent la* localité temporelle*. Cet article décrit un ensemble de méthodes d'optimisation des localités temporelle et spatiale, et une méthode de parallélisation sans communication inter-processeurs, dans le cas d'opérations* stencil*. Celles-ci sont constituées de nids de boucles parcourant des grilles régulières en effectuant en chaque point un calcul de voisinage. Les techniques employées consistent en un réordonnancement des itérations et en une réorganisation des données en mémoire. Elles utilisent le modèle polyédrique des nids de boucles et la technique de linéarisation de polytopes présentée par Clauss et al. Les méthodes proposées sont indépendantes de l'architecture machine visée, et entraînent une compression mémoire.*

ABSTRACT. *A major goal in nowadays program optimisation, as well as in parallelism, is the improvement of data locality in programs. Locality of the data accesses is composed of two aspects. Repeated accesses to a same data within a short interval of time define temporal locality, while spatial locality is defined by repeated accesses to data included in a same block of memory within a short interval of time. This article describes a method to optimize temporal and spatial locality for stencil-based operations, which are composed of loop nests computing a neighbourhood operation on a uniform grid. The techniques shown here consist in affine loop nest transformations, and in reordering data in memory. They are based on the polytope model and use a polytope linearization technique (presented by Clauss et al). In favourable cases, the resulting program is a set of loop nests which can be executed in parallel without any interprocessor communications. The proposed method is architecture-independent, and moreover leads to memory contraction.*

MOTS-CLÉS : *Localité spatiale et temporelle, compression mémoire, mémoire cache, TLB, nids de boucles, optimisation de performances, polytopes paramétrés, polynômes d'Ehrhart*

KEYWORDS: *Spatial and Temporal Locality, memory contraction, cache memory, TLB, loop nests, performance optimization, parameterized polytopes, Ehrhart polynomials*

1. Introduction

Dans cet article, nous traitons le cas des opérations *stencil*, c'est-à-dire des nids de boucles dans lesquels les références à un même tableau sont égales, à une translation près, et dans lesquelles on n'accède pas plusieurs fois à une même donnée par la même référence (pas d'*auto-réutilisation*). Ce modèle regroupe beaucoup d'algorithmes classiques tels que les calculs aux éléments finis, la surrelaxation, les filtres numériques en traitement d'image et de son, ... Dans notre modèle, l'auto-réutilisation est toutefois permise si les réutilisations proviennent d'indices de boucles non présents dans les fonctions de références. Par exemple, lorsque l'on répète plusieurs fois une boucle ne portant pas d'auto-réutilisation. L'accès à une même donnée par des références distinctes lors d'une même itération sera considéré comme un cas particulier de réutilisation de groupe.

Il est connu que la performance des programmes en général est très dépendante des accès mémoire provoqués lors de leur exécution. Dans le cas de systèmes enfouis, qui connaissent aujourd'hui un essor important, la consommation en énergie électrique et le coût sont aussi très dépendants de ces accès. De plus, les contraintes de taille mémoire y sont très fortes, et la réduction de la quantité de mémoire nécessaire à l'éxécution d'un programme sur de tels systèmes a un impact direct sur la faisabilité de ceux-ci. Les ordinateurs actuels comportent plusieurs niveaux hiérarchiques de mémoire, dans lesquels les plus *bas* niveaux sont de grande taille mais d'accès lent (disque, mémoire principale), alors que les niveaux les plus *élevés* sont rapides mais de petite taille (caches, registres). Les architectures mémoire des systèmes enfouis peuvent être plus simples (limitées à un seul niveau mémoire avec un processeur dédié), mais tendent de plus en plus à être dotées de processeurs généraux et d'une mémoire hiérarchique, cela afin d'étendre leurs fonctionnalités logicielles.

Il est donc préférable que les programmes accèdent le plus souvent aux hauts niveaux de hiérarchie. Pour cela, deux principes de base sont à considérer, tous deux liés à l'implémentation physique des caches et des mémoires : la *localité spatiale* et la *localité temporelle*. La localité temporelle caractérise l'accès répété à une même donnée avant qu'elle ne soit remplacée. Dans le cas d'un défaut de cache ou de page mémoire, une ligne est chargée dans le cache, ou une page dans la mémoire principale, ce qui est coûteux en temps. C'est pourquoi, lorsque plusieurs adresses contigües sont accédées consécutivement avant que la ligne de cache ou la page ne soit remplacée, on parle de localité spatiale. Améliorer la localité temporelle et spatiale consiste donc à minimiser le nombre de défauts de cache, le nombre de défauts de page, ainsi que le nombre de défauts de TLB (*Translation Lookaside Buffer*) et de chargements des registres. Rappelons que le TLB est un tableau intégré au processeur qui procure une correspondance rapide entre l'adresse virtuelle d'une page et son adresse physique en mémoire.

Le premier objectif de notre méthode est de rendre consécutifs le plus grand nombre d'accès à une même donnée définis par le programme original. On recherche pour cela une direction de parcours des itérations donnant un nombre de réutilisations maximal entre plusieurs itérations consécutives et respectant les dépendances. On teste ensuite si l'ensemble des itérations de l'opération est décomposable en sous-ensembles accédant à des groupes de données disjoints. On peut alors exécuter en parallèle ces sous-ensembles sur autant de processeurs que de groupes de données disjoints, pour obtenir un code à gros grain de parallélisme de type SPMD. Finalement, on réorganise les données de façon à ce que toutes les itérations accèdent successivement à des données consécutives en mémoire, par une technique utilisant les polynômes d'Ehrhart introduite dans [CLA 00], adaptée au cas des opérations stencil. Cette transformation a pour but l'amélioration de la localité spatiale du programme, et sa compression mémoire.

Dans la section 2, nous présentons le schéma général des opérations stencil et des références aux tableaux, et nous introduisons notre modélisation géométrique. Pour des raisons d'intelligibilté, nous présentons d'abord notre méthode d'optimisation dans le cas d'un nid de boucles simple. La section 3 décrit l'algorithme de choix de la direction de parcours du nid de boucles. Dans la section 4, nous traitons de la décomposition de boucles afin de générer un programme parallèle sans communication. La section 5 décrit la réorganisation des données en mémoire et ses conséquences sur l'écriture de la nouvelle boucle. La section 6 conclut en situant cet article dans notre schéma général d'optimisation.

2. Prérequis

2.1. *Opérations stencil*

Nous nous intéressons à des fonctions d'accès affines à des tableaux, ces fonctions pouvant être paramétriques. Il y a réutilisation de groupe lorsque la boucle considérée utilise plusieurs fois une même donnée par des références différentes. Il est intéressant de considérer la réutilisation de groupe lorsqu'il n'y a pas d'auto-réutilisation ou lorsque les auto-réutilisations sont portées par des indices de boucles non contenues dans les références. Dans le cas contraire, une meilleure optimisation est généralement obtenue en ne considérant que l'auto-réutilisation [LOE 02, LOE 01].

De nombreux travaux se sont intéressés à la localité de données [O'B 99, KAN 98] et dans la recherche de parallélisme sans communication [LIM 95, SHI 96]. Mais ces travaux ne traitent que d'auto-réutilisation. De plus, ils ne tiennent pas toujours compte des bornes des boucles, et leur approche est non-paramétrique. D'autres chercheurs ont traité le cas des opérations stencil. Mais les techniques développées sont dépendantes du problème traité [WEI 00] ou de l'architecture cible [HAR 99], par exemple lorsqu'elles consistent en un tuilage [BAS 98].

Les références auxquelles nous nous intéressons sont des fonctions affines paramétriques inversibles. Aussi nous utiliserons le terme référence pour désigner aussi bien la référence elle-même que sa fonction d'indexation, représentée par une matrice. Nous définissons une relation de réutilisation d'une même donnée entre deux références R_α et R_β par la transformation $\mathcal{T}_{\alpha\beta}$ qui associe à une itération I accédant une donnée par la référence R_α, l'itération $\mathcal{T}_{\alpha\beta}.I$ qui accède à la même donnée par la référence R_β : $\mathcal{T}_{\alpha\beta} = R_\beta^{-1}.R_\alpha$. Comme nous traitons uniquement la réutilisation de groupe, les références sont des fonctions inversibles, et nous ne parlerons de relations de réutilisation qu'entre deux références distinctes.

On modélise les opérations stencil par un nid de boucles parcourant un ensemble d'instructions de la forme générale suivante :

$$Y(B.I) = X_1(A_1.I + v_{X_1,1}) \otimes X_1(A_1.I + v_{X_1,2}) \otimes ... \otimes X_1(A_1.I + v_{X_1,p}) \otimes ...$$
$$... \otimes X_t(A_t.I + v_{X_t,1}) \otimes ... \otimes X_t(A_t.I + v_{X_t,q})$$

où chaque référence à un tableau accédé possède la même fonction d'accès à un vecteur de translation près, Y est le tableau résultat, les X_i sont les tableaux de données et les $v_{X_{i,j}}$ le j-ième vecteur stencil du tableau X_i. S'il existe i tel que $X_i = Y$, alors il y a des dépendances de données si l'intersection entre les éléments de X_i et de Y accédés par la boucle est non vide.

On utilise le modèle polyédrique [ANC 91, FEA 96] en représentant un tel nid de boucles par un polytope convexe paramétré, où chaque point à coordonnées entières est associé à une itération.

On remarque que la relation de réutilisation de groupe pour les opérations stencil entre un $X_i(A_i.I + v_{X_i,\alpha})$ et un $X_i(A_i.I + v_{X_i,\beta})$ est une translation, définie par un vecteur $\mathcal{T}_{\alpha\beta}$, c'est pourquoi on parlera désormais de *vecteur de réutilisation*. On remarque qu'il est inutile de déterminer $\mathcal{T}_{\beta\alpha}$, car pour tout $\mathcal{T}_{\alpha\beta}$, il existe un $\mathcal{T}_{\beta\alpha}$ qui est l'opposé de $\mathcal{T}_{\alpha\beta}$. On appelle *pgcd d'un vecteur* le plus grand commun diviseur commun de ses composantes.

2.2. *Notations matricielles*

Dans les sections suivantes, nous exprimerons les transformations affines paramétriques par des matrices homogènes. Une fonction affine de variable I s'écrit :

$$\mathcal{T}.\begin{pmatrix} I \\ 1 \end{pmatrix} = A.I + c$$

On appelle A la partie linéaire de $\mathcal{T}$, notée $\mathcal{L}(\mathcal{T})$, et le vecteur c la partie constante de $\mathcal{T}$, notée $cst(\mathcal{T})$.

Dans le cas des opérations stencil, une référence R à un tableau s'exprime comme une matrice carrée inversible de dimension $n + 1$, où n est la profondeur du nid de boucles. Ses vecteurs-colonnes forment une base d'un réseau régulier de points entiers, de dimension $n + 1$, que l'on notera $\epsilon(R)$.
L'ensemble des données touchées par un nid de boucles à travers une référence est un *Z-polytope*, défini par l'intersection d'un réseau de points entiers avec un polytope. Quinton et al proposent dans [QUI 97] une étude détaillée de cet objet mathématique

et de ses propriétés. Les auteurs montrent que tout Z-polytope dans un espace de dimension n s'exprime sous la forme d'un triplet (B, F, P), où P est un polytope ,B est une matrice de rang n et F est une matrice. Ils démontrent aussi que l'ensemble des Z-polytopes est stable par intersection. Dans le cas de Z-polytopes de dimension géométrique n, F est de dimension n. Appelons *réseau support* d'un Z-polytope le réseau engendré par les vecteurs-colonnes de la matrice $B.F$

Soit deux matrices carrées inversibles M_1 et M_2 de dimension n, formées des vecteurs de base de réseaux de points. Appellons *réseau commun* de M_1 et M_2 le réseau formé des points appartenant à la fois à $\epsilon(M_1)$ et à $\epsilon(M_2)$, que nous noterons $\mathcal{C}(\epsilon(M_1), \epsilon(M_2))$. Celui-ci est également appelé *plus petit commun multiple* de M_1 et M_2.
Soit $H_0 = S(\mathcal{L}(M_1), \mathcal{L}(M_2))$ le réseau support des parties linéaires de M_1 et M_2. La condition d'existence d'un réseau commun est que $v = cst(M_1) - cst(M_2)$ appartienne à $\epsilon(H_0)$. Cette condition équivaut a la suivante : $H_0^{-1}.v$ doit être un vecteur d'entiers.

2.3. *Intersection de Z-polytopes*

Soit deux Z-polytopes $\Delta_1 = \epsilon(M_1) \cap P_1$ et $\Delta_2 = \epsilon(M_2) \cap P_2$, où P_1 et P_2 sont des polytopes. L'intersection de ces deux Z-polytopes est le Z-polytope $\Delta_{12} = \mathcal{C}(\epsilon(M_1), \epsilon(M_2)) \cap (P_1 \cap P_2)$.

Afin d'analyser les dépendances de données, on teste si l'intersection de deux Z-polytopes (qui est un Z-polytope) contient des points de son réseau ou non (on parlera de vacuité du Z-polytope). Soit un Z-polytope Δ_1 de polytope P_1, et dont le réseau a comme vecteurs de base les vecteurs-colonnes d'une matrice M_1. Si l'on exprime Δ_1 dans la base des vecteurs-colonnes de M_1, on obtient un Z-polytope $\Delta_1' = \mathbb{Z} \cap P_1'$, où P_1' est défini par la relation :

$$\begin{array}{ccc} P_1 & \longrightarrow & P_1' = M_1^{-1}.P_1 \\ I = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} & \longrightarrow & I' = \begin{pmatrix} i_1' \\ i_2' \\ \vdots \\ i_n' \end{pmatrix} = M_1^{-1}.I \end{array}$$

Le nombre de points du Z-polytope Δ_1 est égal au nombre de points à coordonnées entières contenus dans le polytope P_1'. Ce nombre est égal à $\mathcal{E}(P_1')$, le polynôme d'Ehrhart de P_1'. On peut donc parler du polynôme d'Ehrhart d'un Z-polytope, $\mathcal{E}(\Delta_1)$.

3. Optimisation de la localité temporelle

Notre méthode d'amélioration de la localité temporelle consiste à sélectionner une famille de couples de références à un même tableau (R_α, R_β), puis transformer l'espace des itérations afin que les itérations utilisant une donnée par R_α et les itérations utilisant la même donnée par R_β (c'est-à-dire les itérations liées par $\mathcal{T}_{\alpha\beta}$) soient exécutées consécutivement ou de manière très rapprochée.

On observe que les opérations stencil peuvent contenir plusieurs relations de réutilisation définies par des vecteurs colinéaires. Lorsque l'on détermine les relations de réutilisations, on obtient un ensemble de vecteurs que l'on peut classer en familles de vecteurs colinéaires. On choisit alors comme nouvelle direction de parcours, la direction correspondant à un vecteur appartenant à la famille de vecteurs colinéaires la plus représentée. Appelons-la *famille privilégiée*. Ce choix permet de maximiser le nombre de réutilisations temporelles sur une direction de parcours.

Nous traiterons l'exemple suivant tout au long de l'article. Celui-ci ne comporte pas de dépendances pour des raisons de clarté, mais nous indiquerons à quelles étapes du processus elles doivent être prises en compte. On peut éventuellement contourner les problèmes de dépendances en séparant un tableau sur lequel portent des dépendances en un tableau résultat et un tableau de données, puis en mettant à jour le tableau de données ou en inversant le rôle des deux tableaux. Cette technique est également utilisée dans la sur-relaxation, afin d'accélérer la convergence vers le résultat cherché. Elle est avantageuse dans le cas où la quantité de mémoire disponible n'est pas un facteur critique. Ce choix est dépendant de l'architecture cible, il ne peut donc être intégré dans notre méthode d'optimisation.

Exemple 1

```
do i = 1,N
    do j = 3i,3N
       Y(i,j) = ( X(i+1,j) + X(i-1,j) + X(i,j+1) + X(i,j-1)) / 4.0
    enddo
enddo
```

Ce qui nous donne les vecteurs de réutilisation suivants :

$$\mathcal{T}_{14} = \mathcal{T}_{32} = (1,1) \qquad \mathcal{T}_{13} = \mathcal{T}_{42} = (1,-1) \qquad \mathcal{T}_{12} = (2,0) \qquad \mathcal{T}_{34} = (0,2)$$

Ici, la famille de $\mathcal{T}_{14}$ et celle du vecteur $\mathcal{T}_{13}$ sont celles qui contiennent le plus de vecteurs colinéaires. Nous pouvons choisir l'une des deux arbitrairement comme famille privilégiée : comme elles contiennent le même nombre de vecteurs colinéaires, elles sont porteuses du même potentiel de réutilisation. Choisissons par exemple celle de $\mathcal{T}_{13}$. ■

L'objectif est de générer une nouvelle boucle de parcours des itérations telle que toutes les itérations de la boucle la plus interne accèdent à une même donnée à travers les références liées par les vecteurs de la famille privilégiée.

Géométriquement, un parcours est caractérisé par un nombre de vecteurs égal à la profondeur de la boucle considérée. Ainsi, la famille privilégiée est associée à un vecteur v_{int} définissant la boucle la plus interne du nouveau parcours. Ce vecteur doit être colinéaire aux vecteurs de la famille, mais doit aussi être tel que chacun des vecteurs v_k de la famille privilégiée en est un multiple entier. Il est déterminé par :

$$v_{int} = \frac{v_k}{\alpha_k}$$

où α_k est le pgcd de v_k.

On construit une matrice M^{-1} de transformation unimodulaire du domaine d'itérations respectant les dépendances et telle que celui-ci soit parcouru selon v_{int} dans la boucle la plus interne. C'est une matrice telle que, pour tous les vecteurs de dépendance d_i, $M^{-1}.d_i \succ 0$, où $\succ$ définit la relation de supériorité lexicographique, et telle que le n-ième vecteur-colonne de M doit être égal à v_{int}, où n est la dimension de l'espace d'itérations.

Les transformations unimodulaires de domaines d'itérations pour l'amélioration de la localité de données ont déja été proposées dans d'autres travaux [KAN 98]. Toutefois, les auteurs ne cherchent pas à maximiser le nombre de réutilisations dans le cas d'opérations stencil comportant plusieurs instructions.

3.1. *Construction de la matrice* M

Le pgcd de v_{int} est égal à 1. Il existe donc toujours au moins une matrice unimodulaire M_0 avec pour n-ième vecteur-colonne le vecteur v_{int}. Celle-ci peut-être construite, par exemple par ajout de vecteurs-colonnes de pgcd égaux à 1, et tels que les sous-déterminants de M_0 d'ordre 2 à n soient égaux à ± 1. L'ensemble $\mathcal{M}$ des matrices M^{-1} telles que M est unimodulaire avec v_{int} pour n-ième vecteur-colonne est caractérisé par les contraintes suivantes sur ses vecteurs-lignes u_i :

- $u_i \cdot v_{int} = 0, \forall\, i \in [1..n-1]$ (première condition)
- $u_n \cdot v_{int} = 1$ (deuxième condition)

Chacune de ces équations définit un hyperplan de dimension $n - 1$. Des techniques de recherche de solutions respectant les dépendances ont déjà été élaborées. Celle de Darte et al. [DAR 00] se base sur une modélisation polyédrique des dépendances. Cette technique d'ordonnancement parallèle d'instructions tient compte d'un placement préalable des instructions sur des processeurs virtuels. Elle permet de traiter la parallélisation de boucles pouvant porter du parallélisme non borné.

Dans le cas où la construction de M a échoué, c'est-à-dire que l'on n'a trouvé aucune matrice M respectant les dépendances, nous devons choisir un autre vecteur de parcours de la boucle la plus interne v_{int}, et recommencer la construction d'une autre matrice U. On élimine alors la famille privilégiée de la liste des familles de vecteurs disponibles, et on détermine une nouvelle famille privilégiée parmi les familles restantes. Lorsque deux familles contiennent le même nombre de vecteurs de réutilisation, on privilégie celle qui contient le plus grand nombre de dépendances. On augmente ainsi la probabilité d'existence d'une solution, puisqu'on s'assure que la transformation M^{-1} respecte au moins les dépendances de la famille choisie. Dans le pire des cas, aucune des directions données par les familles privilégiées ne permet la construction d'une matrice M valide. On prend alors la matrice identité, ce qui revient à laisser le nid de boucles inchangé.

Si la construction de la matrice M a réussi, on peut utiliser celle-ci pour transformer le nid de boucles afin d'en améliorer la localité temporelle.

3.2. *Transformation du domaine d'itérations*

On peut transformer le domaine d'itérations D par la transformation M^{-1} :

$$\begin{array}{ccc} D & \longrightarrow & D' = M^{-1}.D \\ I = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} & \longrightarrow & I' = \begin{pmatrix} i'_1 \\ i'_2 \\ \vdots \\ i'_n \end{pmatrix} = M^{-1}.I \end{array}$$

Comme M^{-1} respecte les dépendances, la transformation du nid de boucles correspondant conserve la sémantique originale de ce nid de boucles.

Exemple 2

Ici, nous obtenons $v_{int} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Prenons $M = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$

Les bornes du nid de boucles qui correspond au domaine d'itérations $D' = M^{-1}.D$ sont obtenues par projection de Fourier-Motzkin [ANC 91], ce qui nous donne la nouvelle boucle :

```
do i=  4 , 4*N
    do j= max( -3*N , -i+1 ) , FLOOR(-3*i/4)
        Y(i+j,-j) = ( X(i+j+1,-j) + X(i+j-1,-j)
          +X(i+j,-j+1) + X(i+j,-j-1) )/4.0
    enddo
enddo
```

Une donnée accédée à l'itération (i_0, j_0) par $R_1 = X(i + j + 1, -j)$, (resp. R_4) est réaccédée par $R_3 = X(i + j, -j - 1)$, (resp. R_2) à l'itération $(i_0, j_0 + 1)$: deux itérations consécutives réutilisent 2 données. ■

4. Décomposition de l'espace des itérations

A présent, on détermine la possibilité de décomposer le domaine des itérations en sous-domaines accédant à des ensembles de données disjoints, afin d'exécuter ces sous-domaines sur plusieurs processeurs sans communication. Comme il est toujours possible d'éliminer la communication de données uniquement lues (en les dupliquant), nous nous intéresserons dans cette section aux *dépendances de données*, qui correspondent aux réutilisations entre une référence accédant *en écriture* à un tableau et une autre référence à ce même tableau. Les *vecteurs de dépendances* sont donc un sous-ensemble des vecteurs de réutilisation. Nous considérons les vecteurs de dépendances lexicographiquement positifs, qui définissent chacun un ordre partiel d'exécution entre les itérations du nid de boucles.

L'ensemble des points générés par tous les vecteurs de dépendances, à partir d'un point quelconque, forme un réseau régulier de points entiers. L'ensemble des itérations est donc l'union disjointe de tous les réseaux engendrés. Ceux-ci peuvent donc définir des programmes qui ne partagent aucune donnée. Une solution parallèle sans communication consiste donc à placer chaque programme sur un processeur différent.

D'Hollander [D'H 92] propose une méthode d'indiçage et de parcours des réseaux d'itérations disjoints. L'auteur calcule une matrice triangulaire inférieure par combinaison linéaire entière des vecteurs-colonnes de la matrice de dépendances. Cette matrice constitue une base du réseau recherché. Son déterminant est égal à la *partitionabilité* de l'espace d'itérations, c'est-à-dire au nombre de processeurs sur lesquels on pourra exécuter des réseaux d'itérations disjoints sans communication interprocesseur. Il déduit ensuite de la valeur du k-ième élément diagonal de la matrice obtenue ($k \in [1..n]$) le nombre de réseaux disjoints que l'on peut parcourir selon le k-ième indice de boucle, et un algorithme de génération de la boucle transformée. Toutefois, la méthode proposée nécessite un calcul des bornes de boucles à chaque itération des boucles externes. De plus, elle produit des nids de boucles non normalisés car parcourant le domaine d'itérations final avec des sauts d'itérations différents de un. Le programme résultant peut alors être plus difficilement analysé et optimisé par un compilateur.

Nous proposons ici une autre technique, qui fonctionne par transformations affines du domaine d'itérations correspondant au nid de boucles. Celle-ci utilise la mise sous forme normale de Hermite, ainsi qu'une décomposition de matrice en produit de matrices particulières.

Considérons la matrice de dépendances Q, dont les vecteurs-colonnes sont les vecteurs de dépendances, lexicographiquement positifs. Pour développer notre technique, nous supposerons d'abord que l'espace vectoriel engendré par les vecteurs de dépendances est de dimension n. Puis nous montrerons comment étendre notre technique au cas où l'espace engendré est de dimension inférieure à n.

4.1. *Cas où la dimension de l'espace engendré est égale à* n

Nous proposons d'utiliser la forme normale gauche de Hermite $[H\ 0]$ de la matrice de dépendances. H sera dans ce cas une matrice carrée, triangulaire inférieure, dont les éléments sont positifs. Nous utilisons alors une décomposition particulière de H pour générer de manière automatique un ensemble de nids de boucles normalisés accédant à des données disjointes en respectant les dépendances. Les nids de boucles obtenus parcourent aux niveaux les plus externes des réseaux d'itérations disjoints. Les indices des boucles internes, ainsi que les références aux tableaux, sont alors paramétrés par ces indices de boucles externes.

Les vecteurs-colonnes de H constituent une base du réseau. On note $det(H)$ le déterminant de H. Si $|det(H)| = 1$ (H unimodulaire), alors l'espace d'itération n'est défini que par le réseau standard $\mathbb{Z}^n$. Il n'est alors pas possible de décomposer les itérations en sous-ensembles accédant à des données disjointes.

Par contre si $|det(H)| = r > 1$, on peut partitionner l'espace d'itération en r réseaux disjoints tels que les itérations définies par un de ces réseaux accèdent à des données qui ne sont jamais accédées par les itérations définies par les autres réseaux. Classiquement, r est appelé *déterminant* du réseau.

Nous illustrerons cette technique par l'exemple suivant :

Exemple 3

```
do i = 1,N
    do j = 1, i
        A(2*i+j,i+j) = A(2*i+j+5,i+j+3) + A(2*i+j+2)(i+j+2)
                               + A(2*i+j+3)(i+j+1)
    enddo
enddo
```

Ici, les vecteurs de dépendances sont les vecteurs de réutilisation suivants :

$$\mathcal{T}_{21} = (2,1) \qquad \mathcal{T}_{31} = (0,2) \qquad \mathcal{T}_{41} = (2,-1)$$

La matrice de dépendances est donc :

$$\begin{pmatrix} 2 & 0 & 2 \\ 1 & 2 & -1 \end{pmatrix}$$

et sa forme normale gauche de Hermite H est :

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

dont le déterminant est 4. Nous pouvons donc séparer les itérations de cette opération stencil en quatre sous-ensembles que l'on pourra exécuter en parallèle sans communication. ■

Lorsque, pour tout $j \in [1..n]$, les éléments du vecteur-colonne $H_{\cdot j}$ sont divisibles par H_{jj}, la méthode de D'Hollander produit un nid de boucles ne contenant pas de calcul supplémentaire des bornes de boucles. Dans le cas général, une analyse plus fine est nécessaire.

Nous proposons de décomposer H en un produit de matrices carrées de rang n :

$$H = N_m \cdot U_m \cdot N_{k-1} \ldots N_1 \cdot U_1 \cdot N_0$$

, où $m \leq n$, et tel que :

– Les matrices N_i, que l'on appelle *matrices de compression*, sont des matrices diagonales à coefficients entiers positifs,

– les matrices U_i sont des matrices triangulaires inférieures unimodulaires entières à coefficients diagonaux positifs,

– le j-ième élément diagonal de la matrice N_i est égal au *pgcd* du j-ième vecteur-colonne de la matrice $A_i = N_m \cdot U_m \cdot N_{m-1} \ldots U_{i+1} \cdot N_i$.

Il est démontrable qu'une telle décomposition existe toujours pour une matrice H de rang n. Il existe également un algorithme de construction d'une telle décomposition. Pour des raisons de place, la preuve d'existence et l'algorithme ne seront pas développés dans cet article.

Soit $H' = U_m \cdot U_{m-1} \ldots U_2 \cdot U_1$. On applique au domaine d'itérations D la transformation suivante :

$$\begin{array}{ccc} D & \longrightarrow & D_0 = H'.{N_0}^{-1}.{U_1}^{-1} \ldots N_{m-1}^{-1}.U_m^{-1}.N_m^{-1}.D \\ I = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} & \longrightarrow & U = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = H'.{N_0}^{-1}.{U_1}^{-1} \ldots N_{m-1}^{-1}.U_m^{-1}.N_m^{-1}.I \end{array}$$

où l'application de N_i^{-1}, $i \in [0..m]$ se traduit par la transformation de nid de boucles suivante, pour tout élément $\rho_k = N_{i_{k,k}} > 1$:

– remplacer l'indice de boucle i_k de profondeur k par un indice paramétré $w_{k,l}$, à l'aide du changement de variable $i_k = \rho_k.w_{k,l} + l$ pour $l \in [0..\rho_k - 1]$.

– générer une boucle supplémentaire, externe au nid de boucles, parcourant les valeurs prises par le (nouveau) paramètre l.

Le niveau de boucle externe généré n'est pas pris en compte par les transformations suivantes, qui sont toujours appliquées au n niveaux de boucles les plus internes. La transformation N_i^{-1} a pour effet la création de boucles de parcours de réseaux disjoints, qui peuvent être parcourus en parallèle. Géométriquement, elle correspond à une *compression* du domaine d'itérations. Les transformations U_i^{-1} correspondent à des changements de base unimodulaires qui permettent que les compressions soient appliquées selon les vecteurs de base des réseaux disjoints.

Exemple 4 La matrice H se décompose ainsi :

$$H = N_1 \cdot U_1 \cdot N_0$$

c'est-à-dire

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

Nous avons ici $H' = U_1$. La transformation consiste donc à appliquer $U_1 \cdot N_0^{-1} \cdot U_1^{-1} \cdot N_1^{-1}$ au domaine d'itérations. Nous détaillons cette transformation, en notant i' et j' les nouveaux indices de boucles pour chaque transformation. Pour des raisons de lisibilité, ces indices seront renommés i et j dans le nid de boucles transformé.

1) Application de N_1^{-1} :

$$i = 2i' + l_1 \Rightarrow i' = \frac{i - l_1}{2}$$

$$j' = j$$

```
do l1 = 0 , 1
  do i = CEILING((1-l1)/2) , FLOOR((N-l1)/2)
    do j = 1 , 2i+l1
      A(4*i+2*l1+j,2*i+l1+j) = A(4*i+2*l1+j+5,2*i+l1+j+3)
       + A(4*i+2*l1+j+2,2*i+l1+j+2) + A(4*i+2*l1+j+3,2*i+l1+j+1)
    enddo
  enddo
enddo
```

Ici, une première séparation d'indice a eu lieu sur l'indice i. Celle-ci s'est accompagnée de la création d'un niveau de boucle externe d'indice l_1.

2) Application de U_1^{-1} :

$$i' = i$$

$$j' = j - i$$

```
do l1 = 0 , 1
  do i = CEILING((1-l1)/2) , FLOOR((N -l1)/2)
    do j = 1-i , i+l1
      A(5*i+j+2*l1,3*i+j+l1) = A(5*i+j+2*l1+5,3*i+j+l1+3)
        + A(5*i+j+2*l1+2,3*i+j+l1+2) + A(5*i+j+2*l1+3,3*i+j+l1+1)
    enddo
  enddo
enddo
```

On observe ici que les vecteurs de dépendances sont $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ et $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$, ce qui montre que l'on peut "séparer" l'indice j.

3) C'est précisément l'effet de l'application de N_0^{-1} :

$$i' = i$$

$$j = 2j' + l_2 \Rightarrow j' = \frac{j - l_2}{2}$$

```
do l1 = 0 , 1
  do l2 = 0 , 1
    do i = CEILING((1-l1)/2) , FLOOR((N-l1)/2)
      do j = CEILING((1-i-l2)/2), FLOOR((i+l1-l2)/2)
        A(5*i+2*j+2*l1+l2,3*i+2*j+l1+l2) =
                A(5*i+2*j+2*l1+l2+5,3*i+2*j+l1+l2+3)
                     +  A(5*i+2*j+2*l1+l2+2,3*i+2*j+l1+l2+2)
                          +  A(5*i+2*j+2*l1+l2+3,3*i+2*j+l1+l2+1)
      enddo
    enddo
  enddo
enddo
```

4) Application de U_1 :

$$i' = i$$

$$j' = i + j$$

```
do l1 = 0,1
  do l2 = 0,1
    do i = CEILING((1-l1)/2) , FLOOR((N-l1)/2)
      do j = CEILING((i-l2+1)/2) , FLOOR((3*i+l1-l2)/2)
        A(3*i+2*j+2*l1+l2,i+2*j+l1+l2) =
                A(3*i+2*j+2*l1+l2+5,i+2*j+l1+l2+3)
                      + A(3*i+2*j+2*l1+l2+2,i+2*j+l1+l2+2)
                            + A(3*i+2*j+2*l1+l2+3,i+2*j+l1+l2+1)
      enddo
    enddo
  enddo
enddo
```

Le parcours des boucles d'indices l_1 et l_2 ne portant aucune dépendance, l'exécution parallèle de ces boucles se fait sans communication interprocesseur. ■

Les itérations accédant à un même réseau sont caractérisées par une même valeur du vecteur d'indices $(l_1, ..., l_m)$. Les itérations de même vecteur d'indices $(l_1, ..., l_m)$ peuvent donc être placées sur le même processeur. On obtient ainsi un programme de type SPMD (Programme unique sur des ensembles données multiples) sans communication entre les processeurs. Ce grain de parallélisation est favorable aux machines de type NUMA, ainsi qu'aux réseaux de machines. Toutefois, on pourra préférer une parallélisation à grain fin, dans le cadre d'un modèle de machine SIMD. Cette opération nécessite alors de transformer notre solution à travers un échange de boucles tel que les boucles parallèles d'indices l_1 à l_m soient les boucles les plus internes.

Notons M la matrice par laquelle on transforme le domaine d'itérations : $M = H'.{N_0}^{-1}.{U_1}^{-1} \ldots N_{m-1}^{-1}.U_m^{-1}.N_m^{-1}$

Proposition 1 *Soit Q une matrice formée des vecteurs de dépendance, tous lexicographiquement positifs, associés à un nid de boucles de domaine d'itérations D. Soit M la matrice de transformation obtenue à partir de cette matrice de dépendance. La transformation $M.D$ respecte les dépendances.*

Preuve Les matrices U_i et $N_i, i \in [0..m]$, issues de la décomposition de H sont triangulaires inférieures d'éléments diagonaux positifs. Par stabilité de l'ensemble des matrices triangulaires inférieures d'éléments diagonaux positifs par produit matriciel, $H' = U_m.U_{m-1} \ldots U_0$ est elle aussi triangulaire inférieure à éléments diagonaux positifs. Par stabilité de l'ensemble des matrices triangulaires inférieures d'éléments diagonaux positifs par inversion, il en est de même pour la matrice M. La multiplication à gauche d'une matrice Q formée de vecteurs-colonnes lexicographiquement positifs par une matrice triangulaire inférieure à éléments diagonaux positifs donne une matrice formée de vecteurs-colonnes lexicographiquement positifs. La matrice de dépendances obtenue après application de M sur D s'exprimant comme $M.Q$, celle-ci est donc formée de vecteurs-colonnes lexicographiquement positifs. La transformation par M du nid de boucles de domaine d'itérations D respecte donc les dépendances. ■

De plus, il est aisément démontrable (comme M est triangulaire inférieure à éléments positifs) que le vecteur de parcours de la boucle la plus interne du nid de boucles obtenu est colinéaire à celui du nid de boucles initial. On voit également que l'ordre de parcours des itérations d'un même réseau par la boucle la plus interne est inchangé.

4.2. *Cas où la dimension de l'espace engendré est inférieure à* n

Lorsque l'espace vectoriel engendré par les vecteurs de dépendance est de dimension p inférieure à n, cela signifie que le nid de boucles est porteur d'un *parallélisme non borné*, c'est-à-dire que le nombre de processeurs sur lesquels on peut exécuter des itérations distinctes est dépendant des bornes des boucles. La méthode présentée dans la section 4.1 extrait le *parallélisme borné* d'un nid de boucles. Ici, les deux formes de parallélisme coexistent.

Notre approche est la suivante :

1) $B = Q$
Pour f valant de p à n, répéter le traitement suivant :

2) calculer une base B_f du réseau de dimension f engendré par les vecteurs-colonnes de B (par élimination entre vecteurs-colonnes),

3) construire une base de dimension $f+1$ en y ajoutant un vecteur linéairement indépendant (un “faux vecteur de dépendance”)

4) $B = B_f$

Soit G la matrice $n \times n$ obtenue. On applique ensuite la méthode développée dans la section 4.1 à partir de la forme normale de Hermite de la matrice G.

Les vecteurs que l'on ajoute à la base déterminent le parallélisme borné du domaine d'itérations. Le nombre de réseaux d'itérations disjoints correspondant au parallélisme borné est donné par le déterminant de G. Ce déterminant est dépendant des vecteurs que l'on ajoute à la base. De cette manière, les vecteurs choisis déterminent le parallélisme borné *et* non borné du nid de boucles obtenu.

Une solution consiste à ajouter des vecteurs de la base canonique à la base des vecteurs de dépendances. Ajouter le j-ième vecteur de la base entraîne que le j-ième niveau de boucle sera porteur de parallélisme non borné. De plus, le j-ième élément diagonal de la forme normale de Hermite obtenue sera alors égal à 1, ce qui entraîne que le j-ième indice de boucle ne sera pas “séparé”. En revanche, ce j-ième niveau de boucle pourra être exécuté en parallèle (parallélisme non borné). En utilisant cette solution, le déterminant de G est compris entre la valeur minimale et maximale des sous-déterminants d'ordre p de la matrice B_p.

Exemple 5 Considérons le nid de boucles suivant :

```
do i = 1 , N
  do j = 1 , N
    do k = 1 , N
      C(i,j,k) = C(i-1,j-2,k-4) + C(i-2,j-3,k-5)
    enddo
  enddo
enddo
```

La matrice de dépendances correspondante est la suivante :

$$B_2 = \begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}$$

Si l'on souhaite y ajouter des vecteurs de la base canonique, la matrice G peut prendre trois formes :

1) $G = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 0 \\ 4 & 5 & 0 \end{pmatrix}$, dont le déterminant est 2, et la forme normale de Hermite est $H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{pmatrix}$. Notre méthode extraira deux réseaux disjoints du nid de boucles

de départ. On pourra aussi exécuter la boucle d'indice i en parallèle.

2) $G = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 4 & 5 & 0 \end{pmatrix}$, dont le déterminant est 3, et la forme normale de Hermite est $H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 3 \end{pmatrix}$. Notre méthode extraira trois réseaux disjoints du nid de boucles de départ. La boucle d'indice j pourra être exécutée en parallèle.

3) $G = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 1 \end{pmatrix}$, dont le déterminant est 1, et la forme normale de Hermite est $H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Dans ce cas, tout le parallélisme sera porté par l'indice k. La boucle d'indice k sera donc la seule à être exécutée en parallèle. ■

Le choix des vecteurs portant le parallélisme non borné peut aussi être dirigé par l'architecture cible. En particulier, on préférera choisir un vecteur plaçant le parallélisme sur les indices les plus internes dans le cas d'une machine SIMD, et inversement si le grain de parallélisme recherché est plus gros. Souhaitant présenter une méthode indépendante de l'architecture cible, nous ne pouvons y intégrer ce dernier critère.

Lorsque l'on s'interdit la duplication de données, il est nécessaire de prendre en compte tous les vecteurs de réutilisation. Nous avons appliqué cette méthode à l'exemple optimisé en section 3, en considérant tous les vecteurs de réutilisation. On obtient un programme parallèle sans communication interprocesseur ni duplication de données, représenté ci-dessous. L'exécution de la boucle d'indice l est rendue parallèle par la directive OpenMP PARALLEL DO.

Exemple 6

```
!$ OMP PARALLEL DO
do l = 0,1   /* rho1 = 2 */
    do i=  CEILING((4-l)/2) , FLOOR((4*N-l)/2)
        do j= max( -3*N , -2*i-l+1 ) , FLOOR((-6*i-3*l)/4)
        Y(2*i+j+l,-j) = ( X(2*i+j+l+1,-j) + X(2*i+j+l-1,-j)
           + X(2*i+j+l,-j+1) + X(2*i+j+l,-j-1) )/4.0
        enddo
    enddo
enddo
!$END PARALLEL DO
```

■

Bien qu'en théorie les domaines de données accédés soient disjoints, en pratique ils peuvent se situer sur une même page mémoire. Le chargement d'une donnée provoque le chargement de la page correspondante dans l'espace mémoire du processeur qui l'utilise. Si un autre processeur nécessite des données qui sont sur cette page pour faire son calcul, celle-ci devra migrer vers ce processeur, ce qui représente des communications. Dans la section suivante, nous réorganisons les données pour que celles-ci soient placées en mémoire dans l'ordre dans lequel elles sont accédées par le nid de boucles correspondant. Comme chaque nid de boucles (donc chaque processeur) accède à des données disjointes, les données accédées par un processeur seront contigües en mémoire, ce qui éliminera les possibilités de migration de page. Ainsi, seules les données réellement accédées par un processeur seront chargées par ce processeur.

5. Optimisation spatiale

On améliore la localité spatiale en faisant en sorte que les données auxquelles on accède à la même itération, successivement ou dans un temps rapproché, soient physiquement proches en mémoire. Dans nos travaux précédents [CLA 00, LOE 02], nous décrivons une méthode basée sur les polynômes d'Ehrhart [CLA 98], qui réindice les données dans l'ordre dans lequel elles sont accédées par le nid de boucles. Ceci nous donne une nouvelle fonction d'adressage pour l'ensemble des données accédées par cette référence.

Les opérations stencil consistant en un calcul de voisinage, la probabilité que deux données utilisées à la même itération, ou dans un court intervalle de temps, soient rapprochées en mémoire est naturellement forte. Nous allons placer les données en mémoire de façon à ce que cette probabilité soit maximale. Mais notre méthode doit être indépendante de l'architecture cible, par exemple de la taille d'une ligne de cache.

5.1. *Principe de la méthode*

Premièrement, on détermine l'ordre dans lequel les itérations sont parcourues par le nid de boucles, c'est-à-dire une fonction qui à toute itération $I_0 = \begin{pmatrix} i_{1,0} \\ i_{2,0} \\ \cdots \\ i_{n,0} \end{pmatrix}$ associe sa position dans le parcours. Puis, on associe la donnée d_0 accédée par I_0 à cette position, pour en déduire sa nouvelle adresse mémoire. Cette association est unique pour toute référence inversible. La position de I_0 est égale au nombre de points du domaine d'itérations D qui sont lexicographiquement inférieurs ou égaux à I_0. Pour cela, nous déterminons le polyèdre borné $P(I_0)$ contenant I_0 et les points de D qui lui sont lexicographiquement inférieurs. La position de I_0 sera alors le nombre de points à coordonnées entières contenus dans ce polytope. L'ordre lexicographique étant une relation non linéaire, le polyèdre borné $P(I_0)$ est à priori non-convexe. Il s'exprime par l'union de polytopes convexes disjoints :

$$P(I_0) = P_1(I_0) \cup P_2(I_0) \cup P_3(I_0) \cup \ldots \cup P_n(I_0)$$

avec

$$\begin{aligned}
&P_1(I_0) = \{I \in D | i_1 < i_{1,0}\} \\
&P_2(I_0) = \{I \in D | i_1 = i_{1,0}, i_2 < i_{2,0}\} \\
&P_3(I_0) = \{I \in D | i_1 = i_{1,0}, i_2 = i_{2,0}, i_3 < i_{3,0}\} \\
&\ldots \\
&P_n(I_0) = \{I \in D | i_1 = i_{1,0}, i_2 = i_{2,0}, i_3 = I_{3,0} \ldots i_n \leq i_{n,0}\}
\end{aligned}$$

Les $P_i(I_0)$ sont des polytopes : le nombre de points à coordonnées entières contenus dans chacun de ces polytopes est leur polynôme d'Ehrhart, exprimé en fonction de I_0. Comme ces polytopes sont disjoints, le nombre de points entiers contenus dans $P(I_0)$ est la somme des nombres de points contenus dans chaque $P_i(I_0)$. La position d'une itération est donc déterminée par le polynôme d'Ehrhart

$$\mathcal{E}(I_0) = \sum_k \mathcal{E}_k(I_0)$$

où $\mathcal{E}_k(I_0)$ est le polynôme d'Ehrhart de $P_k(I_0)$.

Elle nous donne donc, dans le cas où un tableau n'est accédé qu'à travers une seule référence R inversible, l'ordre dans lequel les données sont parcourues. L'indice d'ordre pour lequel la donnée d_0 est accédée par le nid de boucles est dans ce cas :

$$A(d_0) = \mathcal{E}(R^{-1}(d_0))$$

Le principe est de réorganiser les données en mémoire par cette fonction. La nouvelle adresse d'une donnée d_0 sera $A(d_0) + b$, où b est l'adresse de base de l'ensemble de données. L'adaptation de cette technique aux cas où des données sont accédées par des références non inversibles est développé dans [LOE 02].

Ce procédé a pour conséquence importante que les données qui ne sont pas accédées par le nid de boucles ne sont pas réorganisées en mémoire (elles sont éliminées). Notre méthode constitue donc de surcroît une technique de compression mémoire.

Notre méthode repose sur un placement contigü en mémoire des données accédées successivement. Le parcours du nid de boucle donné par notre méthode est tel qu'un maximum de données réutilisées soient accédées consécutivement ou dans un temps très court : pour deux références R_α et R_β de la famille privilégiée, les itérations I et J telles que $J = \mathcal{T}_{\alpha\beta}.I$ seront exécutées de façon successive ou rapprochée.

L'itération I accède à la donnée $R_\beta.I$ et à $R_\alpha.I = R_\beta.T_{\alpha\beta}.I = R_\beta.J$. Les données accédées à la même itération par R_α et R_β seront donc placées en mémoire de façon contigüe ou rapprochée : il y aura localité spatiale des données accédées. Ainsi, la réorganisation spatiale sera déterminée par le vecteur de réutilisation $\mathcal{T}_{\alpha\beta}$ de la famille privilégiée : tous les accès effectués à travers les références reliées par $\mathcal{T}_{\alpha\beta}$ seront optimisés spatialement.

5.2. *Adaptation de la méthode*

On veut réorganiser en mémoire les éléments du tableau accédé, en fonction de leur ordre d'accès. On ne peut indicer un ensemble de données par un ensemble d'itérations que si l'on peut définir une bijection entre ces deux ensembles. En particulier, les deux ensembles doivent avoir un même nombre d'éléments, ce qui n'est pas le cas pour les opérations stencil : des données distinctes d'un même tableau sont accédées lors d'une même itération, et il y a plus de données accédées que d'itérations. Deux solutions s'offrent à nous : l'une consistant à extraire du domaine de données un ensemble isomorphe au domaine d'itérations (méthode découpante), et l'autre consistant à utiliser un domaine d'itérations virtuel isomorphe au domaine de données (méthode de l'enveloppe convexe).

Ces deux méthodes définissent une nouvelle organisation des données pour une référence particulière R_p, puis expriment la nouvelle fonction d'accès pour les autres références au même tableau en fonction de la fonction d'accès obtenue pour R_p. Pour profiter de l'impact du choix de la direction de parcours sur la localité spatiale, on choisit R_p parmi les références utilisées pour le calcul des vecteurs de réutilisation de la famille privilégiée. Si le tableau considéré n'est pas accédé par une telle référence, on procèdra tout de même à l'amélioration de la localité spatiale des données de ce tableau.

Le principe de la méthode découpante est de séparer en mémoire le domaine de données C accédé non uniquement par R_p des autres données, et de les réorganiser séparément. Une partie des itérations accèdera à C par toutes les autres références. Une autre partie accèdera à C par seulement 2 références, une autre par 3, ... ce qui entraîne un premier découpage du domaine d'itérations et donc du nid de boucles. Un second découpage a lieu lorsqu'on considère les domaines de données accédés par plusieurs références (R_p non incluse), pour chaque partition de l'ensemble des références. Lorsque l'on optimise la localité spatiale des données de plusieurs tableaux, les découpages engendrés se superposent, ce qui augmente encore la complexité du code source et des calculs. Dans le cas des opérations stencil, nous préférons la méthode de l'enveloppe convexe, qui est mieux adaptée et qui permet de générer un code moins volumineux.

5.3. *Méthode de l'enveloppe convexe*

Pour obtenir une bijection entre le domaine d'itérations et le domaine de données accédé, on considère une référence particulière R_p pour laquelle on veut calculer une nouvelle fonction d'adressage. Soit P l'union des domaines de données accédées par toutes les références au tableau considéré. Afin que toutes les références partagent une même fonction d'adressage après optimisation spatiale, nous calculons une fonction d'adressage pour la référence R_p en faisant comme si cette seule référence permettait d'accéder à tout le domaine de données P à travers un domaine d'itérations suffisant. C'est pourquoi on parle de *domaine d'itérations virtuel*, que l'on notera V. Ce domaine, qui s'exprime comme $R_p^{-1}.P$, est à priori non-convexe. Or l'expression d'un polynôme d'Ehrhart est calculée pour un polyèdre convexe, ce qui nous obligerait à découper ce polyèdre en une union de polyèdres convexes. C'est pourquoi, afin de simplifier l'expression du polynôme d'Ehrhart et donc de la nouvelle fonction d'adressage, on considèrera comme domaine d'itérations virtuel un polyèdre convexe contenant $R_p^{-1}.P$. Deux solutions viennent naturellement à l'esprit, l'une consistant a prendre l'enveloppe convexe de P, et l'autre étant de prendre la plus petite image de D par homothétie contenant P. Le domaine d'itérations virtuel correspondant sera alors l'image de ce polyèdre par R_p^{-1}.

Les opérations stencil sont des opérations de voisinage, c'est pourquoi les domaines de données non accédés par R_p sont d'un ordre de grandeur inférieur au domaine de données accédé par R_p. Cela implique aussi que la distance entre le domaine de données accédé par R_p et les domaines de données non accédés par R_p sera faible. Ces deux propriétés garantissent un faible nombre de *trous* dans le domaine d'itérations virtuel, c'est-à-dire une faible proportion d'adresses ne correspondant à aucune donnée accédée par la nouvelle fonction d'adressage. Le choix d'un domaine d'itérations virtuel égal à l'enveloppe convexe de $R_p^{-1}.P$ a pour avantage de produire un nombre minimal de trous. Par contre, il peut rendre l'expression du polynôme d'Ehrhart plus complexe que si l'on choisissait le plus petit polyèdre de même forme que le domaine d'itérations initial.

Le domaine d'itérations reste le même, seules les fonctions d'accès ayant changé. L'adresse d'une donnée d accédée par une référence R à l'itération i où la référence privilégiée accède à la donnée d_p par une fonction d'adressage $A(d_p)$ est égale à $A(R.R_p^{-1}.d_p)$.

Exemple 7
Nous prenons ici l'enveloppe convexe de P comme domaine d'itérations virtuel. La boucle obtenue est la suivante :

```
!$PARALLEL DO
do l = 0,1
    do i=  CEILING((4-l)/2) , FLOOR((4*N-l)/2)
        j= max( -3*N , -2*i-l+1 )
        EP=i*i/4+7*i/2+j+l+4+(MOD(i,2))/4
```

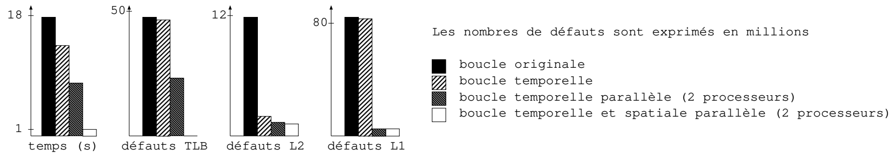

Figure 1. *Mesures de performances pour N=5000*

```
        do j= max( -3*N , -2*i-l+1 ) , FLOOR((-6*i-3*l)/4)
              Y(EP,l) = X(EP,l) + X(EP-CEILING(i/2)-3,l)
                         + X(EP+1,l) + X(EP-CEILING(i/2)-2,l)
              EP=EP+1
        enddo
    enddo
enddo
!$END PARALLEL DO
```

où X_i et Y_i sont les adresses de base des tableaux X et Y pour $l = i$. Nous avons utilisé le compilateur SGI F90 de notre machine Origin 2000 à mémoire virtuellement partagée, avec l'option d'optimisation -O3. Pour empêcher les optimisations du compilateur d'interférer avec celles que nous voulons mesurer, nous avons supprimé les possibilités d'échange et déroulement de boucles, et de tuilage. Nous avons exécuté le programme obtenu pour N=5000 sur un processeur MIPS R12000 pour les versions originale et avec optimisation temporelle, et sur deux processeurs R12000 pour les versions parallèle avec optimisation temporelle, et parallèle avec optimisations temporelle et spatiale. On obient respectivement un temps d'exécution de 17,525 et 13,40 pour les deux premières versions, contre respectivement 7,76 et 0,94 pour les versions parallèles. Les processeurs R12000 sont dotés de 32Ko de mémoire cache de niveau 1, de 8 Mo de cache de niveau 2, et la taille des pages du TLB était fixée à 64Ko. Pour chaque version du nid de boucles, nous avons mesuré le nombre de défauts de TLB et de cache à l'aide de l'outil d'analyse de performances *perfex*, qui utilise les compteurs d'événements du processeur. Ces mesures sont représentées figure 1.

■

Cette méthode d'optimisation spatiale a pour résultat un nid de boucle simple, ce qui économise beaucoup de calculs, notamment de découpage. Elle résulte en une légère modification des conditions de localité spatiale aux itérations où on accède à des données non accédées par R_p. En effet, les données accédées par R_p n'y sont pas contigües en mémoire : le stockage contigûdes données accédées par R_p est interrompu par le stockage des données accédées uniquement par les autres références. Par contre, la localité spatiale avec les données accédées simultanément par les autres références est conservée, contrairement au résultat de la méthode découpante. Cette méthode sera donc conservée préférablement à la méthode découpante.

6. Conclusion

Notre méthode d'optimisation stencil s'inscrit dans le cadre d'un ensemble de techniques d'optimisation de la localité de données de programmes contenant des nids boucles parcourant des tableaux multidimensionnels. Ces techniques sont indépendantes de l'architecture cible considérée, et sont paramétriques. Elles utilisent la librairie *Polylib*, dévelopée à l'IRISA de Rennes puis étendue aux polyèdres paramétrés et aux polynômes d'Ehrhart [CLA 98] à l'Université Louis Pasteur de Strasbourg.

D'autres optimisations, dépendantes de l'architecture cible, peuvent être envisagées en plus de celle-ci, comme par exemple le tuilage. Pour affiner ces optimisations, on pourra alors profiter du fait qu'une des conséquences de notre méthode est que le nombre de réutilisations porté par les boucles externes est réduit. Ces dernières sont donc de bonnes candidates à une parallélisation minimisant les communications. Dans le cas de plusieurs nids de boucles, il peut être intéressant de considérer d'autres transformations de domaine d'itérations, comme par exemple la fusion de boucles. Il faut alors noter l'interférence de celle-ci avec notre méthode d'optimisation de la localité temporelle. En effet, les contraintes de maximisation du nombre de réutilisations à l'intérieur des boucles ne sont pas forcément compatibles et s'additionnent aux contraintes donnant une fusion de boucle optimale.

Notre méthode adresse, en plus de l'amélioration de la localité de données, le probléme de la compression mémoire. La réorganisation des données par les polynômes d'Ehrhart entraîne une réduction du nombre des données adressées par la totalité du programme.

7. Bibliographie

[ANC 91] ANCOURT C., IRIGOIN F., « Scanning Polyhedra with DO Loops », *Proc. of the 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, april 1991, p. 39–50.

[BAS 98] BASSETTI F., DAVIS K., QUINLAN D., « Optimizing transformations of stencil operations for parallel objectoriented scientific frameworks on cachebased architectures », *D. Caromel et al., editor, International Scientific Computing in Object-Oriented Parallel Environments*, ISCOPE, Springer, 1998, vol. 1505 of LNCS.

[CLA 98] CLAUSS P., LOECHNER V., « Parametric Analysis of Polyhedral Iteration Spaces », *Journal of VLSI Signal Processing*, vol. 19, n° 2, 1998, p. 179-194, Kluwer Academic Pub.

[CLA 00] CLAUSS P., MEISTER B., « Automatic Memory Layout Transformations to Optimize Spatial Locality in Parameterized Loop Nests », *ACM SIGARCH Computer Architecture News*, vol. 28, n° 1, 2000, p. 11–19.

[DAR 00] DARTE A., DIDERICH C., GENGLER M., VIVIEN F., « Scheduling the Computations of a Loop Nest with Respect to a Given Mapping », *EuroPar'00*, 2000, p. 405-414.

[D'H 92] D'HOLLANDER E., « Partitioning and labeling of loops by unimodular transformations », *IEEE Transactions on parallel and distributed systems*, vol. 3, n° 4, 1992, p. 465–476.

[FEA 96] FEAUTRIER P., « Distribution automatique des données et des calculs », *Technique et science informatique*, vol. 15, n° 5, 1996, p. 529–557.

[HAR 99] HARPER J. S., KERBYSON D. J., NUD G. R., « Analytical Modeling of Set-Associative Cache Behavior », *IEEE transactions on computers*, vol. 48, n° 10, 1999.

[KAN 98] KANDEMIR M., CHOUDHARY A., RAMANUJAM J., BANERJEE P., « Optimizing Spatial Locality in Loop Nests using Linear Algebra », *7th International Worksop on Compilers for parallel Computers*, June 1998, Sweden.

[LIM 95] LIM A., LAM M., « Communication-free parallelization via affine transformations », *K. Pingali et al., editors, Languages and Compilers for Parallel Computing, LNCS*, vol. 892, 1995, p. 92–106, Springer-Verlag.

[LOE 01] LOECHNER V., MEISTER B., CLAUSS P., « Data Sequence Locality : a Generalization of Temporal Locality », SAKELLARIOU R., Ed., *Euro-Par 2001 Parallel Processing*, Manchester, UK, August 2001, Springer.

[LOE 02] LOECHNER V., MEISTER B., CLAUSS P., « Precise data locality optimization of nested loops », *To appear in the Journal of Supercomputing*, , 2002.

[O'B 99] O'BOYLE M., KNIJNENBURG P., « Nonsingular Data Transformations : Definition, Validity, and Applications », *Int. J. of Parallel Programming*, vol. 27, n° 3, 1999, p. 131-159.

[QUI 97] QUINTON P., RAJOPADHYE S. V., RISSET T., « On Manipulating Z-polyhedra using a Canonical Representation », *Parallel Processing Letter*, vol. 7, n° 2, 1997, p. 181-194.

[SHI 96] SHIH K.-P., SHEU J.-P., HUANG C.-H., « Statement-level Communication-free Partitioning Techniques for Parallelizing Compilers », *9th Workshop on Languages and Compilers for Parallel Computing*, 1996, August.

[WEI 00] WEISS C., KOWARSCHIK M., RUEDE U., KARL W., « Cache-aware multigrid methods for solving poisson's equation in two dimensions », rapport, 2000, To appear in Computing, Springer Verlag, 2000. 21.