RENPAR/CFSE/SYMPA
Paris, cité des sciences et de l'industrie, 24 – 27 avril 2001

Tiling pour une séquence de nids de boucles

Youcef BOUCHEBABA et Fabien COELHO

CRI, ENSMP 77305 Fontainebleau Cedex France
e-mail: {boucheba@cri.ensmp.fr, coelho@cri.ensmp.fr }

Résumé

La plupart des travaux sur le tiling sont appliqués à un seul nid de boucles. Le peu de travaux qui sont consacrés aux codes à plusieurs nids proposent de fusionner ces nids avant d'appliquer le tiling. Dans cet article on propose une méthode qui permet d'appliquer le tiling et la fusion simultanément sur une séquence de nids de boucles parallèles afin de pouvoir minimiser les mouvements de données et la consommation d'énergie et /ou de maximiser la vitesse d'exécution. Chacun de ces nids accède à des données suivant un *patron*. Toutes les données qui sont utilisées dans un nid sont calculées dans le nid précédent. Après l'application du tiling, on doit garantir que toutes les données qui sont nécessaires pour le calcul d'une itération ont déjà été calculées par les itérations précédentes. Pour cela on ajoute un délai pour les calculs de chaque nid. On propose une méthode pour mettre en relation les différents *patrons*, les délais ajoutés et les paramètres du tiling. On donne enfin une solution pour une classe de tiling.

Mots-clés : Tiling Hiérarchique, Fusion de boucles, Transformation de programmes.

1. Introduction

L'objectif de ces travaux est de minimiser l'énergie électrique consommée au cours de l'exécution des programmes de traitement du signal qui sont constitués de séquences de nids de boucles. La plus grande partie de cette consommation serait due aux transferts des données entre les différents niveaux de la hiérarchie mémoire [2]. Pour cela on transforme ces programmes en utilisant le tiling [4, 9, 12, 13, 11, 5], la fusion de boucles [14, 7, 8, 11, 6] et les transformations unimodulaires [1, 13, 10], afin de minimiser ces transferts.

Malheureusement, la plupart des travaux sur le tiling sont consacrés uniquement aux codes qui sont constitués d'un seul nid de boucles. Pour les codes qui sont constitués de plusieurs nids, certains travaux proposent d'appliquer la fusion de ces différents nids et ensuite appliquer le tiling au nid fusionné.

Irigoin et Triolet [4] ont modélisé le tiling par une matrice $H \in Q^{n*n}$ et ils ont donné une condition suffisante pour l'application de ce tiling. Par contre Xue [12] a proposé d'utiliser le tiling comme une transformation de boucles et a donné la condition nécessaire et suffisante à son application. Dans notre approche nous avons combiné la modélisation de Xue [12] avec les différentes techniques de fusion de boucles.

Les programmes considérés dans cet article sont des séquences de nids de boucles. Chacun de ces nids accède à des données suivant un *patron*. Ces données sont produites dans le nid précédent. Après l'application du tiling, on doit garantir que toutes les données qui sont nécessaires pour le calcul d'une itération ont déjà été calculées par les itérations précédentes. Pour cela, on ajoute un délai [8] à chaque nid qui sera calculé en fonction du tiling et du *patron*. Contrairement aux autres travaux qui déterminent quel est le tiling qui est valide en fonction des dépendances du code initial, nous déterminons quels sont les délais qu'on doit ajouter aux nids pour qu'un tiling donné soit valide. Dans cet article on s'intéresse uniquement au régime permanent, sans tenir compte des calculs aux bornes. Notre tiling est représenté par deux matrices : (1) une matrice de tiling hiérarchique à deux niveaux qui donne les différents coefficients des tuiles et (2) une matrice de permutation qui permet d'échanger plusieurs boucles et donc de spécifier l'ordonnancement des tuiles.

2. Exemple

Afin de mieux comprendre le problème traité, voici un exemple de code constitué de deux nids de boucles (Figure 1), auquel on applique le tiling qui est défini par la matrice de tiling hiérarchique A et la matrice de permutation P.

```
do (i = ...)
    do (j = ...)
        A1(i, j) = ....
    enddo
enddo
do (i = ...)
    do (j = ...)
        A2(i, j) = A1(i − 1, j) + A1(i, j − 1)+
                   A1(i, j) + A1(i, j + 1) + A1(i + 1, j)
    enddo
enddo
```

$$\mathcal{A} = \begin{pmatrix} 4 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 2 & 1 \end{pmatrix}$$

$$\mathcal{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

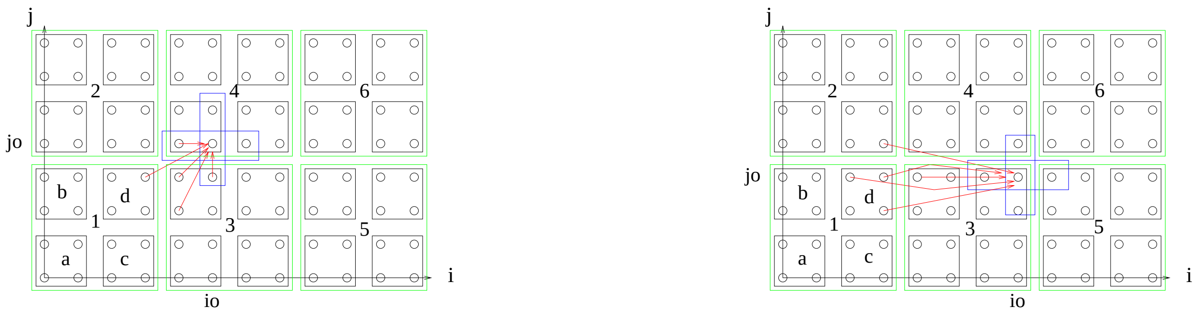
FIG. 1 – *Exemple de code*

La matrice A permet de transformer chaque point $(i, j)^t \in Z^2$ du domaine initial en un point $(i_1, i_2, i_3, j_1, j_2, j_3)^t \in Z^6$ avec $(i, j)^t = A\ .\ (i_1, i_2, i_3, j_1, j_2, j_3)^t$ (voir Figure 3).

La matrice P transforme chaque point $(i_1, i_2, i_3, j_1, j_2, j_3)^t \in Z^6$ en un point $(l_1, l_2, l_3, l_4, l_5, l_6)^t$ avec $(l_1, l_2, l_3, l_4, l_5, l_6)^t = P\ .\ (i_1, i_2, i_3, j_1, j_2, j_3)^t \in Z^6$ (voir Figure 3).

A chaque itération $(i, j)^t$ du deuxième nid du code de la Figure 1 on calcule l'élément $A_2(i, j)$ en fonction d'un *patron* de données du tableau A_1 qui sont produites dans le nid précédent. Ce *patron* est représenté par les vecteurs $(\alpha_1 = -1, \beta_1 = 0)^t$, $(\alpha_2 = 0, \beta_2 = -1)^t$, $(\alpha_3 = 0, \beta_3 = 0)^t$, $(\alpha_4 = 0, \beta_4 = 1)^t$ et $(\alpha_5 = 1, \beta_5 = 0)^t$.

Après l'application du tiling défini par les matrices A et P sur ce code on obtient un seul nid de boucles (voir Figure 3). Le corps de ce nid est constitué de deux instructions ($A_1() = ...$ et $A_2() = ...$) et son vecteur d'itération sera représenté par le vecteur $(l_1, l_2, l_3, l_4, l_5, l_6)^t$. Pour que ce tiling soit valide on doit assurer que tous les éléments du tableau A_1 qui sont nécessaires pour le calcul d'un élément du tableau A_2 à une itération $(l_1, l_2, l_3, l_4, l_5, l_6)^t$ ont déjà été calculés par les itérations précédentes. Pour satisfaire cette condition on avance le calcul de l'instruction ($A_1(i,j) = ...$) du code avant le tiling avec un délai $\vec{d} = (d_1, d_2)^t$, c'est à dire l'instruction ($A_1(i,j) = ...$) sera remplacé par ($A_1(i + d_1, j + d_2) = ...$).



(a) : $\vec{d} = (1, 1)$ (b) : $\vec{d} = (4, 0)$

FIG. 2 – *Deux délais valides pour le même tiling*

Si on prend $\vec{d} = (1, 1)^t$, à une itération $(i_0, j_0)^t$:

1. dans le premier nid, on calcule l'élément $A_1(i_0 + 1, j_0 + 1)$.
2. dans le deuxième nid, on utilise $A_1(i_0-1, j_0)$, $A_1(i_0, j_0-1)$, $A_1(i_0, j_0)$, $A_1(i_0, j_0+1)$ et $A_1(i_0+1, j_0)$.

Les éléments $A_1(i_0 - 1, j_0)$, $A_1(i_0, j_0 - 1)$, $A_1(i_0, j_0)$, $A_1(i_0, j_0 + 1)$ et $A_1(i_0 + 1, j_0)$ sont calculés dans le premier nid respectivement aux itérations $(i_0 - 2, j_0 - 1)^t$, $(i_0 - 1, j_0 - 2)^t$, $(i_0 - 1, j_0 - 1)^t$, $(i_0 - 1, j_0)^t$ et $(i_0, j_0 - 1)^t$. On peut vérifier qu'après le tiling défini par les matrices A et P, $\forall (i_0, j_0)^t$: les itérations $(i_0-2, j_0-1)^t$, $(i_0-1, j_0-2)^t$, $(i_0-1, j_0-1)^t$, $(i_0-1, j_0)^t$ et $(i_0, j_0-1)^t$ sont exécutées avant l'itération $(i_0, j_0)^t$, par conséquent le tiling défini par A et P sur le code de la Figure 1 est valide. En réalité le choix de $\vec{d} = (1, 1)^t$ rend les boucles i et j après la fusion des deux nids complètement permutables [9]. Quel que soit le tiling orthogonal appliqué, il sera valide ($\mathrm{Figure}\ 2(a)$).

De même si on choisit $\vec{d} = (4, 0)^t$, le tiling défini par les matrices A et P est valide. Par contre les boucles i et j du code fusionné ne sont pas complètement permutables, par conséquent le choix d'une autre taille de tuile, par exemple 5x5, n'est pas nécessairement valide ($\mathrm{Figure}\ 2(b)$).

3. Code en entrée

Au vu des applications concernées, on suppose que le code en entrée est une séquence de nids de boucles à deux dimensions, qui ont le même domaine d'itérations et dont les bornes sont des constantes ($\mathrm{Figure}\ 3(a)$). Chacun de ces nids accède à des données suivant un *patron* qui est représenté implicitement dans ce nid et donné explicitement par l'utilisateur sous forme d'un ensemble de vecteurs $S_k = \{(\alpha_{k,1}, \beta_{k,1})^t, (\alpha_{k,2}, \beta_{k,2})^t, \ldots, (\alpha_{k,m_k}, \beta_{k,m_k})^t\}$.

Notre méthode est généralisable pour les séquences de nids de dimensions quelconques et il n'est pas nécessaire que les domaines d'itérations des différents nids soient les mêmes ou que leurs bornes soient des constantes.

4. Transformations de programme

En plus des ensembles de vecteurs représentant les *patrons*, l'utilisateur doit nous fournir une matrice de tiling hiérarchique A et une matrice de permutation P.

4.1. Tiling hiérarchique

La matrice de tiling hiérarchique A, permet de transformer le code en $\mathrm{Figure}\ 3(a)$ en un nid intermédiaire qui est donné en $\mathrm{Figure}\ 3(c)$. Cette matrice nous donne les différents facteurs du tiling et elle doit avoir la forme suivante :

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{2,4} & a_{2,5} & 1 \end{pmatrix}$$

.

Les éléments de la matrice A doivent satisfaire les conditions suivantes :

$a_{1,1} > 0$, $a_{1,2} > 0$, $a_{2,4} > 0$, $a_{2,5} > 0$, $a_{1,2}$ divise $a_{1,1}$, $N_{i2} = a_{1,1} \div a_{1,2}$, $N_{i3} = a_{1,2}$, $a_{2,5}$ divise $a_{2,4}$, $N_{j2} = a_{2,4} \div a_{2,5}$ et $N_{j3} = a_{2,5}$. Les constantes $N_{..}$ sont les bornes supérieures des boucles initiales et finales apparaissant aux figures $(3(a))$ et $(3(c))$.

En conséquence $(i, j)^t = A \,.\, (i_1, i_2, i_3, j_1, j_2, j_3)^t$ et $(i_1 = \lfloor \frac{i}{a_{1,1}} \rfloor, i_2 = \lfloor \frac{i \bmod a_{1,1}}{a_{1,2}} \rfloor, i_3 = i \bmod a_{1,2}, j_1 = \lfloor \frac{j}{a_{2,4}} \rfloor, j_2 = \lfloor \frac{j \bmod a_{2,4}}{a_{2,5}} \rfloor, j_3 = j \bmod a_{2,5})^t$

4.2. Échange de boucles

On sait que, dans certains cas, il est préférable d'échanger des boucles. Pour cela, l'utilisateur doit nous fournir une matrice P de permutation de dimension 6. Cette matrice permet de transformer le nid de la $\mathrm{Figure}\ 3(c)$ en nid de la $\mathrm{Figure}\ 3(d)$, en permutant plusieurs boucles en même temps.

Comme il est préférable de parcourir d'une manière contiguë les éléments d'une tuile, il est souhaitable que la matrice P vérifie ces deux conditions de contiguïté de parcours (I) :

- $\forall a,b,c\ : (P_{a,1} = P_{b,2} = P_{c,3} = 1) \Rightarrow a < b < c.$
- $\forall a,b,c\ : (P_{a,4} = P_{b,5} = P_{c,6} = 1) \Rightarrow a < b < c.$

Comme la matrice P permet de transformer le nid b en le nid c, on a alors la relation suivante : $(l_1, l_2, l_3, l_4, l_5, l_6)^t = P \,.\, (i_1, i_2, i_3, j_1, j_2, j_3)^t$

5. Fusion sans tiling

Pour pouvoir fusionner tous les nids dans un seul, on doit s'assurer que tous les éléments du tableau A_k qui sont nécessaires pour le calcul d'un élément $A_{k+1}(i, j)$ à une itération $(i, j)^t$ dans le nid fusionné, ont déjà été calculés par les itérations précédentes. Pour satisfaire cette condition on avance le calcul de chaque élément $A_k(i, j)(1 \leq k \leq n-1)$ avec un délai $\vec{d}_k = (d_{k,1}, d_{k,2})^t$. Ces différents délais doivent satisfaire :

$\vec{d}_k \succeq \max_{\succeq}^{m}((\alpha_{k+1,m}, \beta_{k+1,m})^t + \vec{d}_{k+1}) : (1 \leq k \leq n-1)$ et $\vec{d}_n = (0, 0)^t$.

Les conditions de légalité de la fusion de boucles sans tiling ont été données par beaucoup d'auteurs [14, 7, 8, 11, 6]. Par contre la fusion et le tiling simultanément n'a été traité par personne.

6. Fusion avec tiling

Le calcul des différents délais $\vec{d}_k$ pour la fusion avec tiling du code de la Figure 3(a), nécessite plus d'efforts. Soient les notations suivantes :

$\vec{i} = (i, j)^t$ le vecteur d'itération avant l'application du tiling, $\vec{i'} = (i_1, i_2, i_3, j_1, j_2, j_3)^t$ le vecteur d'itération après l'application du tiling défini par A, $\vec{l} = (l_1, l_2, l_3, l_4, l_5, l_6)^t$ le vecteur d'itération après l'application de la permutation P et $\vec{d}_k$ le délai associé au nid k.

Le tiling défini par les matrices A et P est représenté par une transformation $\hbar$.

$\hbar : Z^2 \to Z^6$

$\vec{i} \longmapsto \vec{l} = P \,.\, (\lfloor \frac{i}{a_{1,1}} \rfloor, \lfloor \frac{i \bmod a_{1,1}}{a_{1,2}} \rfloor, i \bmod a_{1,2}, \lfloor \frac{j}{a_{2,4}} \rfloor, \lfloor \frac{j \bmod a_{2,4}}{a_{2,5}} \rfloor, j \bmod a_{2,5})^t.$

Le nid n ne nécessite pas de délai, c'est à dire $\vec{d}_n = (0, 0)^t$.

A l'itération $\vec{i}_0 = (i_0, j_0)^t$ du nid $k+1$ $(1 \leq k \leq n-1)$ on utilise les éléments $A_k(i_0 + \alpha_{k+1,m} + d_{k+1,1},\ j_0 + \beta_{k+1,m} + d_{k+1,2})_{m=1..m_{k+1}}$.

L'élément $A_k(i_0 + \alpha_{k+1,m} + d_{k+1,1},\ j_0 + \beta_{k+1,m} + d_{k+1,2})$ est calculé dans le nid k à l'itération $(i_0 + \alpha_{k+1,m} + d_{k+1,1} - d_{k,1},\ j_0 + \beta_{k+1,m} + d_{k+1,2} - d_{k,2})^t$.

Théorème1 :

Le tiling défini par les matrices A et P est valide si et seulement si :

$\forall k, \forall (i_0, j_0)^t, \forall m :$

$\hbar((i_0 + \alpha_{k+1,m} + d_{k+1,1} - d_{k,1}, j_0 + \beta_{k+1,m} + d_{k+1,2} - d_{k,2})^t) \preceq \hbar((i_0, j_0)^t) \quad \text{(II)}$

$$\text{(II)} \Leftrightarrow \forall k, \forall\, (i_0, j_0)^t, \forall m : P \begin{pmatrix} \lfloor \frac{i_0+\alpha_{k+1,m}+d_{k+1,1}-d_{k,1}}{a_{1,1}} \rfloor \\ \lfloor \frac{(i_0+\alpha_{k+1,m}+d_{k+1,1}-d_{k,1}) \bmod a_{1,1}}{a_{1,2}} \rfloor \\ (i_0 + \alpha_{k+1,m} + d_{k+1,1} - d_{k,1}) \bmod a_{1,2} \\ \lfloor \frac{j_0+\beta_{k+1,m}+d_{k+1,2}-d_{k,2}}{a_{2,4}} \rfloor \\ \lfloor \frac{(j_0+\beta_{k+1,m}+d_{k+1,2}-d_{k,2}) \bmod a_{2,4}}{a_{2,5}} \rfloor \\ (j_0 + \beta_{k+1,m} + d_{k+1,2} - d_{k,2}) \bmod a_{2,5} \end{pmatrix} \preceq P \begin{pmatrix} \lfloor \frac{i_0}{a_{1,1}} \rfloor \\ \lfloor \frac{i_0 \bmod a_{1,1}}{a_{1,2}} \rfloor \\ i_0 \bmod a_{1,2} \\ \lfloor \frac{j_0}{a_{2,4}} \rfloor \\ \lfloor \frac{j_0 \bmod a_{2,4}}{a_{2,5}} \rfloor \\ j_0 \bmod a_{2,5} \end{pmatrix}$$

Preuve :

Voir le rapport interne du CRI [3].

On peut remarquer que le calcul de $\vec{d}_1$ dépend de $\vec{d}_2$, $\vec{d}_2$ dépend de $\vec{d}_3$,.... et $\vec{d}_{n-2}$ dépend de $\vec{d}_{n-1}$. Pour calculer les différents $\vec{d}_i$ on commence par calculer $\vec{d}_{n-1}$, ensuite $\vec{d}_{n-2}$.... et enfin on calcule $\vec{d}_1$.

Il existe une solution indépendante de la matrice A, pour une matrice P qui vérifie les conditions de contiguïté de parcours (I). Cette solution est donnée par :

- $\vec{d}_k = ((\max_m \alpha_{k+1,m}, \max_m \beta_{k+1,m})^t + \vec{d}_{k+1}) : (1 \leq k \leq n-1)$
- $\vec{d}_n = (0, 0)^t$.

```
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_1(i, j) = ...
    enddo
enddo
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_2(i, j) = F_{i=1}^{m_2}(A_1(i + α_{2,i}, j + β_{2,i}))
    enddo
enddo
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_3(i, j) = F_{i=1}^{m_3}(A_2(i + α_{3,i}, j + β_{3,i}))
    enddo
enddo
.
.
do (i = 0, i = N_i)
    do (j = 0, j = N_j − 1)
        A_n(i, j) = F_{i=1}^{m_n}(A_{n−1}(i + α_{n,i}, j + β_{n,i}))
    enddo
enddo
```

(a) code initial

```
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_1(i + d_{1,1}, j + d_{1,2}) = ...
    enddo
enddo
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_2(i + d_{2,1}, j + d_{2,2}) =
        F_{i=1}^{m_2}(A1(i + α_{2,i} + d_{2,1}, j + β_{2,i} + d_{2,2}))
    enddo
enddo
do (i = 0, i = N_i − 1)
    do (j = 0, j = N_j − 1)
        A_3(i + d_{3,1}, j + d_{3,2}) =
        F_{i=1}^{m_3}(A2(i + α_{3,i} + d_{3,1}, j + β_{3,i} + d_{3,2}))
    enddo
enddo
.
.
do (i = 0, i = N_i)
    do (j = 0, j = N_j − 1)
        A_{n−1}(i + d_{n−1,1}, j + d_{n−1,2}) =
        F_{i=1}^{m_n}(A_{n−2}(i + α_{n−1,i} + d_{n−1,1}, j + β_{n−1,i} + d_{n−1,2}))
    enddo
enddo
do (i = 0, i = N_i)
    do (j = 0, j = N_j − 1)
        A_n(i, j) = F_{i=1}^{m_n}(A_{n−1}(i + α_{n,i}, j + β_{n,i}))
    enddo
enddo
```

(b) code après l'ajout des délais

```
do (i_1 = ...)
 do (i_2 = 0, i_2 = N_{i2} − 1)
  do (i_3 = 0, i_3 = N_{i3} − 1)
   do (j_1 = ...)
    do (j_2 = 0, j_2 = N_{j2} − 1)
     do (j_3 = 0, j_3 = N_{j3} − 1)
      A_1(...) =
      A_2(...) = F_{i=1}^{m_2}(A_1())
      .
      .
      A_n(...) = F_{i=1}^{m_n}(A_{n−1}())
     enddo
    enddo
   enddo
  enddo
 enddo
enddo
```

(c) code après le tiling défini par A

```
do (l_1 = ...)
 do (l_2 = ...)
  do (l_3 = 0, l_3 = N_{l3} − 1)
   do (l_4 = 0, l_4 = N_{l4} − 1)
    do (l_5 = 0, l_5 = N_{l5} − 1)
     do (l_6 = 0, l_6 = N_{l6} − 1)
      A_1(...) =
      A_2(...) = F_{i=1}^{m_2}(A_1())
      .
      .
      A_n(...) = F_{m=1}^{m_n}(A_{n−1}())
     enddo
    enddo
   enddo
  enddo
 enddo
enddo
```

(d) code après la permutation définie par P

FIG. 3 – *Les codes des différentes étapes*

Le choix de cette solution rend la fusion des différents nids possibles et les boucles i et j du code fusionné complètement permutables. On sait que si un nid de boucles est complètement permutable, on peut lui appliquer n'importe quel tiling parallèle aux axes[9] (P doit satisfaire I).

7. Conclusion

L'application du tiling et fusion simultanément sur une séquence de nids de boucles n'a été traitée par aucun auteur. Par contre de nombreux travaux se sont intéressés à ces deux problèmes isolément. Dans notre approche, ces deux problèmes sont traités simultanément en combinant les travaux de de Xue [12] qui a modélisé le tiling (un seul nid de boucles) par une transformation de boucles et les autres travaux sur la fusion de plusieurs nids de boucles [14, 7, 8, 11, 6]. Notre approche consiste à avancer le calcul de chaque nid de façon qu'après le tiling toutes les données qui sont nécessaires pour une itération donnée ont été déjà produites par les itérations précédentes.

Notre approche nous donne un système d'inéquations qui met en relation les délais ajoutés aux différents nids, les vecteurs des différents *patrons*, la matrice A et la matrice P. Pour ce système d'inéquations on a donné une solution pour une classe de tiling.

Dans cet article on n'a pas discuté les différentes solutions qui sont dépendantes des matrices A et P. Dans les travaux futurs on étudiera ces différentes solutions afin d'optimiser certains critères, comme par exemple la taille de l'espace d'itérations, le volume mémoire, etc.

Bibliographie

1. U. Banerjee. Loop transformations for restructuring compilers. *Kluwer Academic Publishers*, 93.
2. F. Cathoor et al. Custom memory management methodology-exploration of memory organisation for embedded multimedia system design. *Kluwer Academic Publishers,ISBN 3-540-64105-X*, 98.
3. Y. Bouchebaba et Fabien Coelho. Tiling pour une séquence de nids de boucles (version étendue). *Rapport interne du CRI : A/322*, 2001.
4. F. Irigoin and R. Triolet. Super-node partitioning. *In Proc. 15th Annual ACM Symp. Principles of Programming Languages.*, pages 319–329, 1988.
5. M. Kandemir, A. Choudhary, and J. Ramanujam. I/O-conscious tiling for disk-resident data sets. *Europar*, 1999.
6. T. SS. Abdelrahman M. Manjikian. Fusion of loops for parallelism and locality. *IEEE trans. on paral*, 97.
7. K.S. McKinley, S. Carr, and C-W. Tseng. Improving data locality with loop transformations. *ACM Trans. on Programming Languages and Systems*, 18(4):424–453, 96.
8. W. Pugh and E. Rosser. Iteration space slicing for locality. *LCPC99*, pages 165–184, 1999.
9. M. E. Wolf. Improving locality and parallelism in nested loops. *PhD thesis, University of stanford*, 1992.
10. M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE trans. on paral*, 2, 91.
11. M. Wolfe. High performance compilers for parallel computing. *Addison-Wesley*, 1996.
12. J. Xue. On tiling as a loop transformation. *Parallel Processing Letters*, 1997.
13. J. Xue and ChuaHuang Huang. Reuse-driven tiling for improiving data locality. *Parallel programming*, 98.
14. H. P. Zima and B. M. Chapman. Supercompilers for parallel and vector computers. *Addison-Wesley*, 1, 1990.