# Un algorithme optimal de programmation dynamique sur architectures régulières

Clémentin Tayou Djamegni[a] et Maurice Tchuenté[b,c]

[a]Faculté des Sciences B.P 067 Dschang, Cameroun, dtayou@yahoo.com
[b]IRD, UR GEODES, 32 Avenue Henri Varagnat, 93143 Bondy Cedex, France
[c]Faculté des Sciences, B.P 812 Yaoundé, Cameroun

**RÉSUMÉ.** Ce papier étudie la résolution du problème de programmation dynamique sur les architectures régulières: Étant donné

$$w(i,j), 1 \le i < j \le n, \text{ et } c(i,i+1), 1 \le i < n,$$

calculer

$$c(i,j) = w(i,j) + min_{i<k<j} c(i,k) + c(k,j), \text{ pour } 1 \le i < j \le n.$$

Nous montrons que ce problème peut être résolu sur une architecture régulière optimale en temps et en surface. Cette architecture nécessite $2n-1$ unités de temps et $n^2/14 + O(n)$ processeurs élémentaires. Ce résultat améliore la meilleure borne précédemment connue, qui est de $n^2/10+O(n)$ processeurs.

**ABSTRACT.** In this paper, we are interested in the dynamic programming problem defined as follows. Given

$$w(i,j), 1 \le i < j \le n, \text{ and } c(i,i+1), 1 \le i < n,$$

compute

$$c(i,j) = w(i,j) + min_{i<k<j} c(i,k) + c(k,j), \text{ for } 1 \le i < j \le n.$$

Here, we show that this problem can be solved on a space-time optimal array of size $n^2/14 + O(n)$. This is an improvement over the best previously known bound, i.e. $n^2/10 + O(n)$.

**MOTS-CLÉS :** graphe de dépendance; fonction de temps;fonction d'allocation; ré-indexation; optimalité

**KEYWORDS :** dependence graph; timing function; allocation function; re-indexation; optimality

# 1. Introduction

Nous étudions ici le problème de Programmation Dynamique (PD) défini comme suit : Étant donné $w(i,j), 1 \leq i < j \leq n,$ et $c(i,i+1), 1 \leq i < n$, calculer $c(i,j) = w(i,j) + min_{i<k<j} c(i,k) + c(k,j),$ pour $1 \leq i < j \leq n$. Ce schéma peut être utilisé pour résoudre plusieurs problèmes d'optimisation combinatoire[1, 2, 3, 4, 5, 6]. Il traduit le fait que la solution optimale d'un problème peut être écrite sous la forme d'une équation récurrente qui implique les solutions optimales des sous problèmes. Le nombre total d'opérations arithmétiques requis par ce schéma est de l'ordre de $\Theta(n^3)$. Ce qui justifie l'étude de sa parallélisation.

La première implémentation de ce problème de PD sur des architectures régulières est due à Guibas, Kung et Thompson [1]. Leur réseau correspond à une architecture triangulaire et orthogonalement connecté de $n(n+1)/2$ Processeurs Elementaires (PEs) qui résout le problème de la PD en temps optimal $T = 2n - 1$. Le nombre de PEs a été réduit à $n^2/4 + O(n)$ par Gachet, Joinnault et Quinton [7], à $n^2/8 + O(n)$ par Louka et Tchuenté [2, 3], et plus tard à $n^2/10 + O(n)$ par Djamegni et Tchuenté [8]. Aucune de ces solutions n'est optimale en nombre de PEs par rapport à la fonction de temps choisie.

Dans ce papier, nous proposons un réseau optimal en temps et en surface qui nécessite $n^2/14 + O(n)$ PEs et $T = 2n - 1$ unités de temps. Ce qui améliore les meilleures bornes précédemment connues.

Le reste de ce papier est organisé comme suit. la section 2 étudie le graphe de dépendance du problème de programmation dynamique. La section 3 montre comment compresser ce graphe pour obtenir un autre graphe dit réduit qui se prête mieux à l'application des méthodes d'allocation des tâches aux processeurs. La section 4 propose un réseau optimal en temps et un surface. La section 5 conclut ce papier.

# 2. Le graphe de dépendance

Partant des équations récurrentes qui définissent le problème de PD, nous introduisons une troisième cordonnée à toutes les variables $c(i,j)$ pour distinguer les différents calculs qu'elles impliquent. D'où le Système d'Equations Récurrentes (SER) suivant [2, 15] :

$$\text{Initialisation : } \begin{array}{l} c(i,i+1,i+1) = c(i,i+1), \text{ pour } i = 1,...,n-1 \\ c(i,j,i) = +\infty, \text{ pour } 1 \leq i < j \leq n, j \neq i+1 \end{array} \quad [1]$$

$$\text{Calcul : } c(i,j,k) = min\{c(i,j,k-1), c(i,k) + c(k,j)\}, \ 1 \leq i < k < j \leq n \quad [2]$$

$$\text{Sortie : } c(i,j) = c(i,j,j) = w(i,j) + c(i,j,j-1), \text{ pour } 1 \leq i < j \leq n. \quad [3]$$

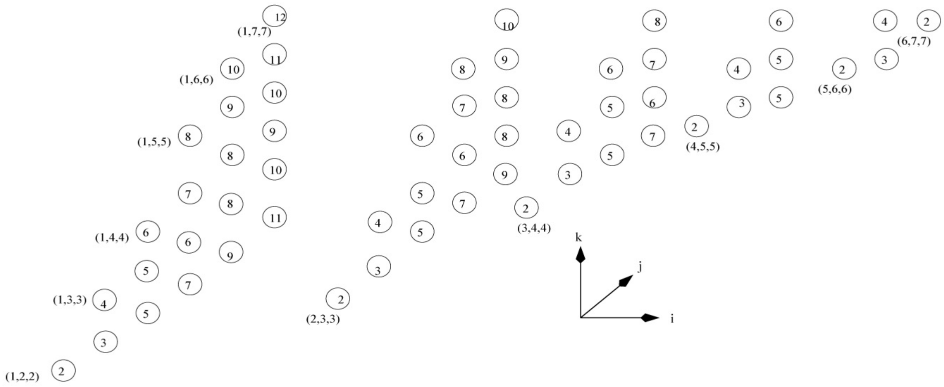


**Figure 1.** *Fonction de temps au plus tôt de la PD pour* $N = 7$

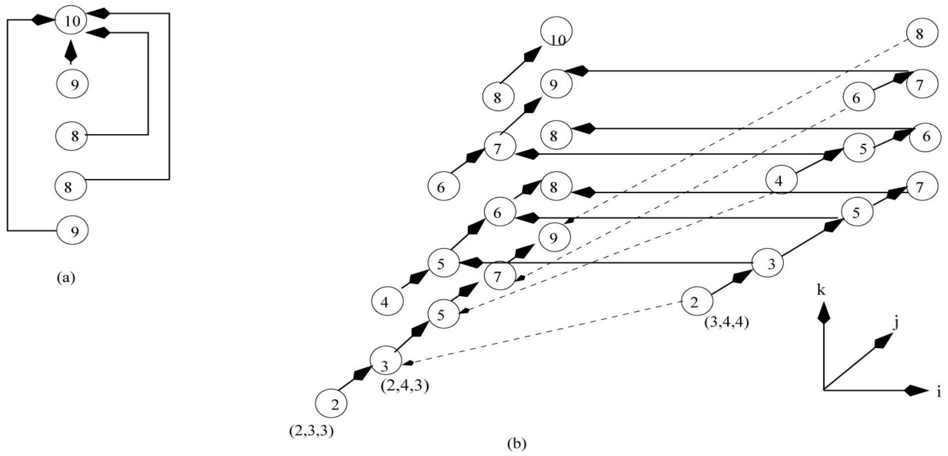


**Figure 2.** *Graphe de dépendances de la PD pour* $N = 7$

Les figures 1 et 2 illustrent le graphe de dépendances $G$ pour $n = 7$. Dans ce graphe, un noeud $(i, i+1, i+1)$ ne participe à aucun calcul. Son rôle est de dupliquer la valeur de $c(i, i+1)$ et d'envoyer deux copies aux noeuds $(i-1, i+1, i)$ et $(i, i+2, i+1)$. Une variable $c(i, j), j \neq i+1$ fonctionne comme suit :

1) Elle est initialisée à $+\infty$, et elle est calculée par les noeuds du segment vertical constitué des points $(i, j, k), k = i+1, ..., j$. Il n'y a aucune relation de dépendance entre les points de ce segment puisque les calculs qui conduisent au minimum dans (2) peuvent être effectués dans n'importe quelle ordre. Cependant, il y a un arc de dépendance du

noeud $(i, j, k)$ vers le noeud $(i, j, j)$, pour $k = i+1, ..., j-1$. En effet, l'opération (3) doit être précédée par toutes les opérations induites par (2). Ces dépendances correspondent aux arcs de la figure 2.a.

2) Dès que la valeur de la variable $c(i, j)$ est calculée au noeud $(i, j, j)$, $c(i, j)$ doit circuler à travers le graphe pour participer au calcul de $c(i-1, j), \ \dots, c(1, j)$. Plus précisément, ceci peut être modélisé par les équations suivantes :

$$\begin{array}{l} c(i-1,j,i) = min\{c(i-1,j,i-1), c(i-1,i) + c(i,j)\}, \\ c(i-2,j,i) = min\{c(i-2,j,i-1), c(i-2,i) + c(i,j)\}, \\ \dots \\ c(1,j,i) = min\{c(1,j,i-1), c(1,i) + c(i,j)\}. \end{array} \qquad [4]$$

Une copie de $c(i, j)$ est envoyée au noeud $(i-1, j, i)$ et ceci correspond aux arcs en pointillées de la figure 2.b. Partant du noeud $(i-1, j, i)$, cette copie circule à travers les noeuds successifs $(i-2, j, i), \ \dots, (1, j, i)$, suivant les arcs de direction $(-1, 0, 0)$, qui sont représentés horizontalement et dirigés de la droite vers la gauche sur figure 2.b.

3) D'une manière similaire, $c(i, j)$ est utilisée pour le calcul de $c(i, j+1), \ \dots, c(i, n)$. Plus précisément, $c(i, j)$ participe aux calculs suivants :

$$\begin{array}{l} c(i,j+1,j) = min\{c(i,j+1,j-1), c(i,j) + c(j,j+1)\}, \\ c(i,j+2,j) = min\{c(i,j+2,j-1), c(i,j) + c(j,j+2)\}, \\ \dots \\ c(i,n,j) = min\{c(i,n,j-1), c(i,j) + c(j,n)\}. \end{array} \qquad [5]$$

En conséquence, une copie est envoyée aux noeuds $(i, j+1, j), \ \dots, (i, n, j)$. Ceci correspond aux arcs de direction $(0, 1, 0)$ dans la figure 2.b. Pour des raisons de clarté, nous n'avons pas dessiné tous les arcs du graphe de dépendances $G = (D, U)$ dans la figure 2. Sur cette figure, nous avons dessiné les noeuds de $G$, en nous basant sur une représentation qui met en évidence les tranches verticales $V_r = \{(x, y, z) \in D : i = r\}, 1 \leq r \leq n$. Ces tranches verticales sont perpendiculaires à la direction $\vec{i}$. Nous avons illustré dans la figure 2.b les arcs entre deux tranches adjacentes, $V_r$ et $V_{r+1}$. Intéressons nous maintenant à la fonction de temps optimale au plus tôt. Dans cette fonction de temps, une tâche qui correspond à un noeud $u$ du graphe de dépendances est exécutée à une date qui correspond à la valeur du plus long chemin qui se termine sur le noeud $u$. Cette fonction de temps $T$ est illustrée à la figure 1 : $T(i, j, k) = j-i+max\{k-i, j-k\}$ Dans tous les algorithmes basés sur cette fonction de temps, il existe des tâches ayant une même date d'exécution qui appartiennent à une même ligne verticale. Ces tâches doivent être exécutées simultanément. Plus précisément, si $i < k < h < j$ et $k-i = j-h$ alors $T(i, j, k) = T(i, j, h)$. En restreignant notre étude au cas particulier où deux tâches ayant une même date d'exécution et participant au calcul d'un même coefficient $c(i, j)$ sont exécutées par un même

PE, la relation de récurrence du problème de PD étudié ici peut être ré-écrite de la manière suivante :

Initialisation : $c(i, i+1, i+1) = c(i, i+1),$ pour $i = 1, ..., n-1$
$c(i, j, i) = +\infty,$ pour $1 \leq i < j \leq n, j \neq i+1$ [6]

Calcul : $c(i, j, p) = \min\{c(i, j, p-1), c(i, k) + c(k, j), c(i, h) + c(h, j)\}$
où $k - i = j - h = p,$ pour $p = 1, 2, ..., d = \lfloor (j-i-1)/2 \rfloor.$ [7]

Sortie : $c(i, j) = c(i, j, d+1) = w(i, j) + c(i, j, d)$ [8]

## 3. Le graphe réduit

Nous avons vu dans la section précédente que le temps minimal requis pour la parallélisation du SER du problème de PD d'ordre $n$ étudié ici est de $2n-1$ unités de temps, et que la fonction de temps optimale au plus tôt est $T(i, j, k) = j - i + \max\{k-i, j-k\}$. Dans toutes les solutions proposées dans la littérature, deux tâches ayant une même date d'exécution et participant au calcul d'un même coefficient $c(i, j)$ sont exécutées par un même PE. Cependant, lors de la dérivation de nouveaux réseaux, les auteurs utilisent toujours le graphe initial. Ce qui obscurcit la définition de la fonction d'allocation au moins pour deux raisons :

1) Premièrement, les dépendances non-locales, i.e les dépendances qui impliquent des noeuds non voisins, apparaissent dans le graphe G. Ceci est le cas des flèches en pointillées de figure 2. Cette situation est particulièrement gênante puisque la fonction d'allocation doit conduire à un réseau localement connecté.

2) Deuxièmement, le fait que deux tâches ayant une même date d'exécution sont affectées à un même PE n'offre pas la possibilité d'exploiter toute la puissance de la méthode de partitionnement, ni celle de la nouvelle méthode d'allocation.

Ici, nous adoptons une approche basée sur une fusion préliminaire des noeuds d'un même segment vertical qui sont exécutés à une même date. Pour ce faire, il est important de noter que la fusion de deux noeuds $u$ et $u'$ peut se faire de deux manières. On peut soit garder $u$ et remplacer tous les arcs $(u\prime, v)$ et $(w, u\prime)$ de $G$ par les arcs $(u, v)$ et $(w, u)$ de $G\prime$, soit garder $u\prime$ et remplacer tous les arcs $(u, v)$ et $(w, u)$ de $G$, par les arcs $(u\prime, v)$ et $(w, u\prime)$ de $G\prime$. Le choix effectué lors de cette phase de fusion influence directement la structure du graphe réduit $G\prime$. Ici, les fusions sont effectuées de manière à obtenir un graphe régulier à dépendances locales. Plus précisément, la fusion de deux noeuds $(i, j, k)$ et $(i, j, h)$ tels que $i \leq k < h < j$, et $k - i = j - h$, est obtenue en supprimant le noeud le plus bas $(i, j, k)$ et en gardant le noeud le plus haut $(i, j, k)$. Cette opération conduit à un graphe réduit $G\prime$ dont tous les arcs sont locaux, i.e. tous les arcs connectent uniquement des noeuds voisins. Plus précisément :

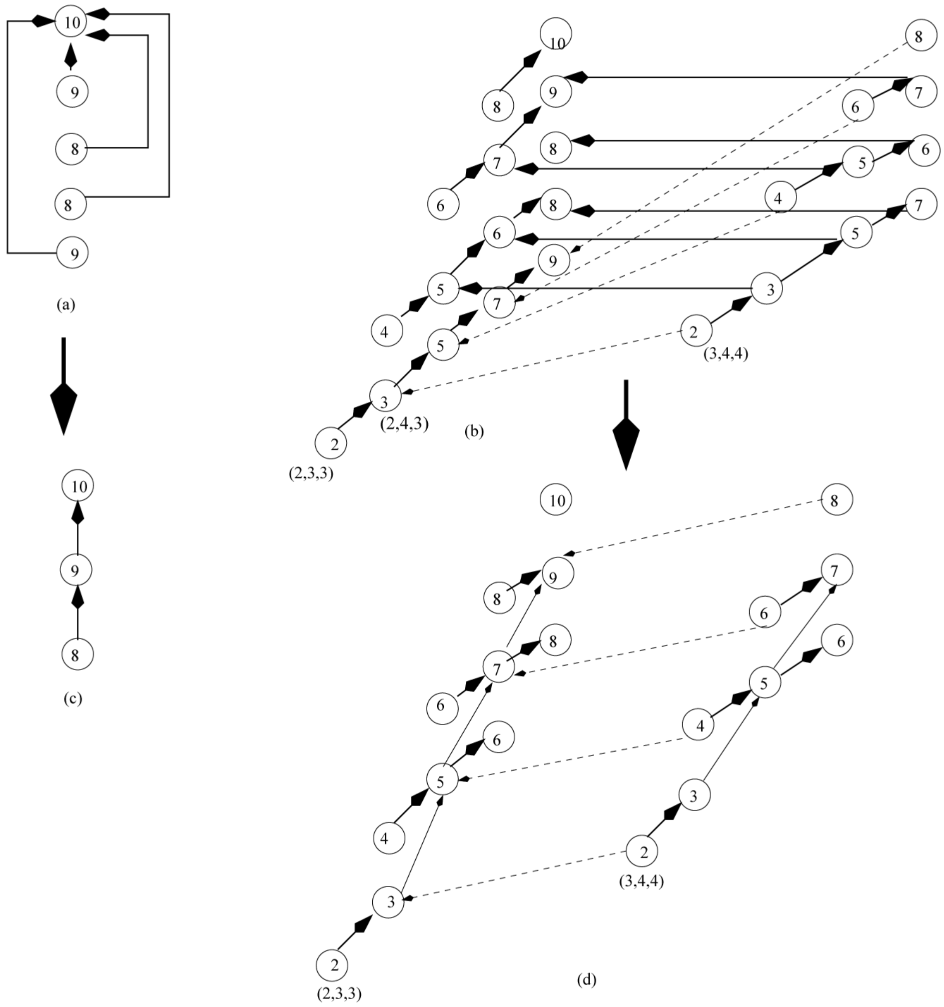


**Figure 3.** *Graphe reduit de la PD pour* $N = 7$

1) Les dépendances non-locales de la forme $[(i,j,j),(i-1,j,i)]$, représentées par des flèches en pointillées dans les figures 2.b et 3 deviennent des dépendances locales de la forme $[(i,j,j),(i-1,j,j-1)]$ (cf. flèches en pointillées de la figure 3.d).

2) Toute dépendance horizontale dirigée de la droite vers la gauche de la figure 2 qui est de la forme $[(i,j,k)],(i-1,j,k)]$ est soit gardée, soit transformée en une dépendance de la forme $[(i,j,k\prime),(i-1,j,k\prime-1)]$ (cf. flèches en gras de la figure 3.d).

3) Toute dépendance horizontale de la forme $[(i,j,k),(i,j+1,k)]$ est soit gardée, soit transformée en une dépendance de la forme $[(i,j,k\prime),(i,j+1,k\prime+1)]$.

En revanche, les dépendances verticales sont maintenues. Ici, nous choisissons d'effectuer les calculs correspondant à la recherche du minimum de (2) de bas en haut, i.e. dans l'ordre croissant des index $k$. Ce choix induit des arcs supplémentaires de la forme $[(i, j, k], (i, j, k + 1)]$ dans le graphe réduit (voir les arcs en pointillés de la figure 3). En conséquence, tout segment vertical est transformé en un chemin orienté de bas en haut comme illustré à la figure 3.

En résumé, le graphe réduit $G\prime = (V\prime, U\prime)$ est tel que :

1) $V\prime = \{(i, j, k) \in Z^3 : 1 \leq i \leq k < j \leq n$ et $k - i \geq j - k\}$

2) Tout arc de $U\prime$ est de l'une des formes suivantes : $[(i, j, k), (i - 1, j, k)]$, $[(i, j, k), (i - 1, j, k - 1)]$, $[(i, j, k), (i, j + 1, k)]$, $[(i, j, k), (i, j + 1, k + 1)]$, $[(i, j, k), (i, j, k + 1)]$.

# 4. Un réseau optimal

## 4.1. Construction de la fonction d'allocation

Dans cette section, nous dérivons un réseau optimal en surface et en temps qui nécessite $n^2/14 + O(n)$ PEs et $T = 2n - 1$ unités de temps. Cette solution meilleure est obtenue en appliquant la méthode d'allocation proposée dans [9, 11, 12]. La dérivation du nouveau réseau se fait en quatre phases.

**Phase 1** Nous appliquons à $D_0$ une transformation unimodulaire $q$ qui transforme la fonction de temps affine et initial $T(i, j, k) = -2i + j + k$ en une nouvelle fonction de temps $T\prime(i, j, k) = k$. Pour ce faire, nous posons $q_0(i, j, k) = (i, j, -2i + j + k)$. La fonction de re-indexation $q_0$ conduit à un nouveau domaine d'indexation $D_1 = q_0(D_0) = \{(i, j, k) \in Z^3 \mid -i + 1 \leq 0,\ 2i - 2j + k \leq 0,\ j - n \leq 0,\ -3i + 3j - 2k \leq 0\}$.

**Phase 2** Nous appliquons maintenant à $D_1$ une fonction de re-indexation qui localise les points de $\{(i, j, k) \in D_1 \mid 2i - 2j + k = 0\}$ sur le plan d'équation cartésienne $i = 0$. Pour ce faire, nous utilisons la fonction de re-indexation suivante :

$$q_1(i, j, k) = \left(i, \left\lfloor \frac{-2i + 2j - k}{2} \right\rfloor, k\right)$$

Nous avons $D_2 = q_1(D_1) = \{-i \leq -1,\ -j \leq 0,\ 2i + 2j + k \leq 2n - 1,\ 6j - k \leq -3\}$. La projection de $D_2 \subset \{-i \leq -1,\ -j \leq 0,\ , i + 4j \leq n - 2\}$ suivant la direction $\vec{k}$ conduit à un réseau de $n^2/8 + O(n)$ PEs.

**Phase 3** Le réseau obtenu à la phase 2 peut encore être optimisé. L'idée de base de cette optimisation est de compresser le domaine d'indexation $D_2$ suivant la direction

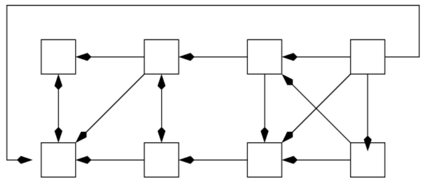

**Figure 4.** *Programmation Dynamique : réseau obtenu pour* $N = 7$

$-\vec{i} + 4\vec{j}$. Pour ce faire, nous effectuons d'abord un changement de base, $q_2(i,j,k) = (i+3j, i+4j, k)$. Nous avons $q_2(D_2) \subset D_3 = \{(i,j,k) \in Z^3 \mid -4i+3j \leq 0,\ i-j \leq 0,\ 6i-4j+k \leq 2n,\ -6i+6j-k \leq 0\}$

**Phase 4** Pour obtenir un réseau optimal en surface et en temps, nous ré-indexons les points de $D_3$ comme suit :

$$q_3(i,j,k) = \begin{cases} (\lfloor \frac{4i-3j}{4} \rfloor, j, k) & \text{si } (i,j,k) \in D_3^{(1)} \\ (\lfloor \frac{6i-6j+k}{6} \rfloor, j, k) & \text{si } (i,j,k) \in D_3^{(2)} \end{cases} \qquad [9]$$

avec $D_3^{(1)} = \{(i,j,k) \in D_3 \mid 6j-4k \leq 0\}$ et $D_3^{(2)} = \{(i,j,k) \in D_3 \mid 6j-4k > 0\}$. Nous avons $D_4^{(2)} = q_3(D_3^{(2)}) = \{(i,j,k) \in Z^3 \mid -6j+4k \leq 0,\ -12i-3j+2k \leq 0,\ 6i-k \leq 0,\ 3i+j \leq n,\ -i \leq 0\}$ et $D_4^{(1)} = q_3(D_3^{(1)}) = \{(i,j,k) \in Z^3 \mid 6j-4k \leq 0,\ -i \leq 0,\ 4i-j \leq 0,\ 12i+j+2k \leq 4n,\ -12i+3j-2k \leq 0\}$. Notons $D_4 = q_3(D_3)$. Il est facile de voir que $D_4^{(1)} \subset D_5 = \{(i,j,k) \in Z^3 \mid -i \leq 0,\ 3i+j \leq n,\ 4i-j \leq 0\}$ et $D_4^{(2)} \subset D_5$. Ainsi $D_4 \subset D_5$. La projection de $D_5$ suivant la direction $\vec{k}$ conduit à un réseau de $n^2/14 + O(n)$ PEs. Cette nouvelle solution est optimale [2].

## 5. Conclusion

Dans ce papier, nous avons étudié la résolution du problème de programmation dynamique sur les architectures régulières. Le résultat obtenu améliore, en terme de nombre de PEs, la meilleure borne précédemment connue. Ce meilleur réseau est obtenu en transformant le graphe initial de tâches par ré-indexation avant d'appliquer la méthode d'allocation dite affine. Le pré-traitement par ré-indexation améliore le parallélisme de la méthode de projection par rapport au domaine d'indexation initial. Cette méthode d'allocation [12, 13, 9, 11] exploite des propriétés de convexité du graphe de dépendances.

## 6. Bibliographie

[1] L.J. Guibas, H.T. Kung, C.D. Thompson, Direct VLSI Implementation of Combinatorial Algorithms, *Proc. Conf. Very Large Scale Integration : Architecture, Design, Fabrication*, California Institute of Technology, (Jan. 1979) 509-525.

[2] B. Louka, M. Tchuenté, Dynamic programming on two-dimensional systolic arrays, *Information Processing Letters*, 29 (2) (September 1988) 97-104

[3] B. Louka, M. Tchuenté, Dynamic programming on 2D arrays, Dans *Parallel Processing*, M. Cosnard, M.H. Barton and Vanneschi eds., North Holland, (1988) 265-274.

[4] J.F. Myoupo, Mapping Dynamic Programming Onto Modular Linear Systolic Arrays, *Distributed Computing* 6(3) : 165-179 (1993)

[5] Thierry Lecroq, Guillaume Luce, Jean Frédéric Myoupo, A faster linear systolic algorithm for recovering a longest common subsequence *Information Processing Letters*, 61 (3) (14 February 1997) 129-136.

[6] Yves Robert, Maurice Tchuente, A systolic array for the longest common subsequence problem, *Information Processing Letters*, 21 (4) (7 October 1985) 191-198

[7] P. Gachet, B. Joinnault, P. Quinton, Synthesizing systolic arrays using DIASTOL, In W. Moore, A. McCabe and R. Urquhart, eds., Proc. *International Workshop on Systolic Arrays*, Adam Hilger, Bristol, U.K., (1986).

[8] C. T. Djamegni, M. Tchuenté, A New Dynamic Programming Algorithm on two dimensional arrays, *Parallel Processing Letters*, 10 (1) (2000) 15-27.

[9] C. T. Djamegni, Mapping rectangular mesh algorithms onto asymptotically space-optimal arrays, *Journal of Parallel and Distributed Computing*, 64 (2004) 345-359.

[10] C.T. Djamegni, Complexity of matrix product on modular linear systolic arrays for algorithms with affi ne schedules, *Journal of Parallel and Distributed Computing*,66 (2006) 323-333.

[11] C. Tayou Djamegni, P. Quinton, S. Rajopadhye, T. Risset, M. Tchuenté, A re-indexing based approach towards mapping of DAG with affi ne schedule onto parallel embedded systems. Accepté à *Journal of Parallel and Distributed Computing* (2008) (34 pages).

[12] C. Tayou Djamegni, Méthode d'optimisation pour la synthèse d'architectures régulières, *Thèse de Doctorat d'Etat*, Département d'Informatique, Université de Yaoundé I, (Dec. 2005).

[13] C. T. Djamegni, Synthesis of space-time optimal systolic algorithms for the Cholesky factorization, *Discrete Mathematics and Theoritical Computer Science*, 5 (2002) 109-120.

[14] C.Tayou, P. Quinton, S. Rajopadhye, T. Risset, Derivation of systolic algorithms for the algebraic path problem by recurrence transformations, *Parallel Comput.*, 26 (11) (2000)1429-1445

[15] M. Tchuente, Parallel Computation on regular arrays, *Manchester University Press and John Wiley Sons*, (1992).