Une méthode de transformation de programmes applicable à la systolisation

Raymond Durand, Brigitte Joinnault, Martine Vergne

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 741

UNE METHODE DE TRANSFORMATION DE PROGRAMMES APPLICABLE A LA SYSTOLISATION

Raymond DURAND
Brigitte JOINNAULT
Martine VERGNE

OCTOBRE 1987

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

Une méthode de transformation de programmes applicable à la systolisation *

Raymond Durand† Brigitte Joinnault‡
Martine Vergne‡

A PROGRAM TRANSFORMATION METHOD APPLIED TO SYSTOLIC DESIGN

Résumé

Nous présentons une méthodologie de transformation de systèmes d'équations dont l'objectif est la dérivation de systèmes d'équations récurrentes linéaires (SERL). Ces systèmes SERL, moyennant une étape de transformation supplémentaire (l'uniformisation [Joi87]), peuvent notamment être utilisés comme point d'entrée de méthodes de systolisation automatique comme la méthode de projection des dépendances [Qui84]. Dans cet article nous décrivons les principes du système de transformation (fonction de généralisation, schémas de calcul fonctionnels et représentation de type abstrait). Puis nous illustrons le fonctionnement de cette méthode par la systolisation d'un cas concret simple : le produit de convolution.

Abstract

A transformation methodology aiming at the derivation of linear recurrence equations (LRE) is presented. After an additional transformation step (the uniformisation [Joi87]), these LRE can be used, in

*Publication soumise au 5ième symposium sur les Aspects Théoriques de l'Informatique, STACS 88
Ce travail a été effectué dans le cadre du contrat 1-86-6001-00-312000-06-2 entre l'INRIA et la société SOREP de Chateaubourg
†LIF, Université Paris VI, 4, Place Jussieu, Paris
‡IRISA, Campus de Beaulieu, 35042 Rennes Cédex

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L.A. 227)
UNIVERSITÉ DE RENNES 1 I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

particular, as entry points for the automatic design of systolic arrays [Qui84]. In this paper the principles of this transformation methodology (generalization function, functionnal computation schemes, abstract data type representation) are first described. Then the method is applied to the systolic design of a well-knowed problem : the convolution product.

Introduction

Nous présentons une méthode de transformation de programmes appliquée à la conception automatique d'architectures systoliques. C'est une méthode de dérivation de systèmes d'équations récurrentes, étape importante de la recherche d'algorithmes et d'architectures parallèles. En effet un grand nombre de systèmes de conception d'algorithmes ou d'architectures parallèles partent de systèmes d'équations récurrentes (travaux de Karp, Miller et Winograd sur la détection du parallélisme [KMW67], travaux de Kogge sur la parallélisation d'algorithmes [Kog74], méthodes de systolisation par analyse des dépendances [Mol82,DI85,Mon85] [MW84,Qui84,Gue86], ...).

Les architectures systoliques sont des machines parallèles spécialisées et synchrones [Kun82]. Elles sont constituées par un réseau de processeurs élémentaires (cellules) connectés régulièrement et localement. Les cellules implémentent un algorithme en se transmettant des flots de données et en effectuant des calculs sur les données qui les traversent. Les architectures systoliques sont généralement utilisées comme périphériques d'un calculateur hôte pour accélérer des traitements nécessitant une grande puissance de calcul et absorbant un flot continu de données. Les domaines d'application dans lesquels elles sont le plus souvent utilisées sont le traitement de signal et d'image, et l'analyse numérique. Cependant, il existe aussi des applications dans le domaine des algorithmes non numériques (algorithmes sur les graphes, programmation dynamique, tri...).

Un grand nombre de ces problèmes systolisables s'expriment à l'aide de fonctions que nous appelons fonctions de généralisation; par exemple $\sum$ est la fonction de généralisation de la fonction +; c'est à cette classe de problèmes que s'applique notre méthode. Les premières architectures systoliques ont été obtenues par "euréka", depuis plusieurs travaux ont été faits concernant des méthodologies de dérivation

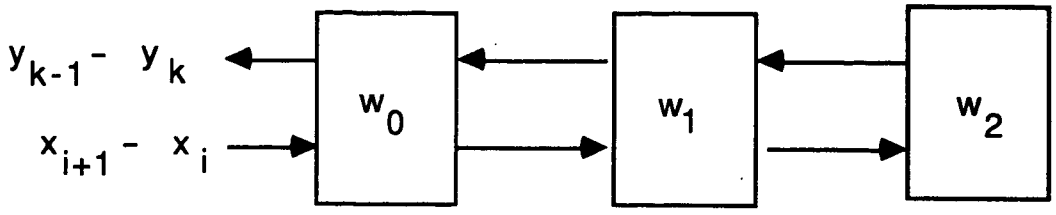

Figure 1 : Une architecture systolique pour le produit de convolution

systématique [LRS83,Coh78,FFW85], [Qui84,DI86]; la plupart n'utilise pas la spécification initiale du problème (la forme mathématique ou logique) mais une spécification (ou un programme itératif) déjà adaptée à la méthode de systolisation. Par exemple l'architecture systolique bien connue (figure 1) calculant le produit de convolution

$$\forall i \geq 0,\ y_i = \sum_{k \in [0,K]} x_{i-k} \times w_k \tag{1}$$

peut être obtenue automatiquement à l'aide du système DIASTOL [QG84] -de conception d'architectures systoliques- à partir du système d'équations récurrentes uniformes (SERU) suivant :

$$\begin{array}{l} \forall i,k,\ i > 0,\ 0 \leq k \leq K \\ y(i) \equiv Y(i,0) \\ Y(i,k) = \left\{ \begin{array}{l} \textbf{si } k = K \textbf{ alors } X(i,k) \times W(i,k) \\ \textbf{sinon } X(i,k) \times W(i,k) + Y(i,k+1) \end{array} \right. \\ X(i,k) = \left\{ \begin{array}{l} \textbf{si } i < 0 \textbf{ alors } 0 \\ \textbf{si } k = 0 \textbf{ alors } x(i-k) \\ \textbf{sinon } X(i-1,k-1) \end{array} \right. \\ W(i,k) = \left\{ \begin{array}{l} \textbf{si } i = 0 \textbf{ alors } w(k) \\ \textbf{sinon } W(i-1,k) \end{array} \right. \end{array} \tag{2}$$

Le problème qui subsiste pour l'utilisateur de telles méthodes de systolisation est la spécification de son problème sous forme d'un SERU. Le travail d'écriture peut parfois être très difficile à réaliser à la main sans

erreur et de façon pertinente. Comme il existe pour un même problème un grand nombre de SERU qui conduisent à des architectures différentes, il est important de maitriser cette étape d'écriture des SERU pour pouvoir trouver différentes solutions et les comparer. La technique présentée ici est une technique de transformation de programmes à l'aide de schémas abstraits de transformation, c'est une technique sûre (les transformations sont prouvées) et automatisable. Elle permet de transformer une spécification initiale du problème exprimée à l'aide de fonctions de généralisation (spécification mathématique) en une formulation sous forme de système d'équations récurrentes linéaires (SERL). On trouvera des définitions précises des SERL et des SERU dans [Joi87]. D'autres techniques dites d'uniformisation (pipeline, routage) développées dans [Joi87] permettent ensuite de transformer un SERL en SERU pour obtenir une forme systolisable.

Dans la première partie de cet article nous définissons les fonctions généralisables et nous donnons des programmes fonctionnels permettant leur calcul. Ces programmes fonctionnels sont utilisés comme des schémas de SERL des fonctions de généralisation. Dans la seconde partie, nous donnons un exemple d'application à la systolisation de problème (produit de convolution).

1 Fonctions généralisables et schémas de calcul

Pour calculer $\sum_{k=0}^{K} x_k$, on peut calculer $x_0+(x_1+(x_2+(\ldots+x_n)\ldots)$ ou $x_0+(x_n+(x_2+(x_{n-1}+\ldots)\ldots)$ ou bien choisir un autre ordre s'il nous convient mieux pour l'implémentation. C'est cette notion d'ordre quelconque dans les calculs qui caractérise les fonctions généralisables.

1.1 Définitions

Etant donné une fonction $\gamma : Z \times X \to X$ (resp. $X \times X \to X$), nous dirons que γ est généralisable (sous entendu à $\mathcal{P}(Z) \times X \to X$ (resp. $\mathcal{P}(X) \to X$)) si et seulement si le résultat du calcul de $\gamma(z_1, \gamma(z_2, \ldots, \gamma(z_n, x), \ldots))$ est indépendant de l'ordre des calculs.

Plus formellement, étant donné $\gamma : Z \times X \to X$, $ch : \mathcal{P}(Z) \to Z$ une fonction de choix et $m\_ch : \mathcal{P}(Z) \to \mathcal{P}(Z)$ telle que $m\_ch(Z') = Z' - ch(Z')$, soit

$$\Gamma_{ch} : \mathcal{P}(Z) \times X \to X$$

$$\Gamma_{ch}(Z', x) = \begin{cases} \textbf{si } vide(Z') \textbf{ alors } e \\ \textbf{sinon } \gamma(ch(Z'), \Gamma_{ch}(m\_ch(Z'), x)) \end{cases}$$

γ est généralisable (sous entendu sur $\mathcal{P}(Z)$) si et seulement si Γ_{ch} ne dépend pas de la fonction de choix. Autrement dit $\forall ch, ch', \Gamma_{ch} = \Gamma_{ch'}$. Dans la suite, lorsque γ est généralisable, nous noterons Γ sa généralisation.

Proposition 1.1 *γ est généralisable si et seulement si :*

$$\forall z_1,\ z_2 \in Z,\ \forall x \in X$$

$$\gamma(z_1, \gamma(z_2, x)) = \gamma(z_2, \gamma(z_1, x))$$

Preuve Par récurrence sur la cardinalité de Z.

Nous allons nous limiter par la suite aux fonctions associatives et commutatives qui satisfont la condition ci-dessus.

1.2 Schémas de calcul

1.2.1 Schéma de base

Dans tout ce paragraphe $\gamma : X \times X \to X$ est une fonction associative et commutative, $\Gamma : \mathcal{P}(X) \to X$ est sa généralisation calculée par :

Schéma 1.1

$$\Gamma(X) = \begin{cases} \textbf{si } element(1, X) \textbf{ alors } ch(X) \\ \textbf{sinon } \gamma(ch(X), \Gamma(m\_ch(X))) \end{cases}$$

Et si γ a un élément neutre, e, la définition précédente est équivalente à :

Schéma 1.2

$$\Gamma'(X) = \begin{cases} \textbf{si } vide(X) \textbf{ alors } e \\ \textbf{sinon } \gamma(ch(X), \Gamma'(m\_ch(X))) \end{cases}$$

où $element(i, X)$ est vrai si et seulement si X a i éléments et $vide(X)$ est vrai si X est vide.

Remarques

- la notation habituelle de la fonction $\Gamma(X)$ est $\Gamma_{x\in X}(x)$ (par exemple $\sum_{x\in[0,K]}(x)$). Nous utiliserons l'une ou l'autre de ces notations,
- l'introduction de la fonction abstraite de choix permet de retarder la décision concrète de l'ordre d'évaluation de la fonction et de garder ainsi accessible un large éventail de solutions, la décision étant prise en tenant compte des impératifs de calcul (dans notre cas l'impératif est la systolisation, mais on peut envisager d'autres applications).

1.2.2 Schéma de calcul d'une généralisation portant sur une fonction

Supposons que l'on veuille calculer $\sum_{x\in X} g(x)$; la notation habituelle est $\Gamma_{x\in X} g(x)$ que nous noterons fonctionnellement $\Gamma_g(X)$. On peut démontrer que si Γ est la généralisation de γ, Γ_g est calculée par :

Schéma 1.3

$$\Gamma_g(X) = \begin{cases} \text{si } element(1,Y) \text{ alors } g(ch(X)) \\ \text{sinon } \gamma\,(g(ch(X)), \Gamma_g(m\_ch(X))) \end{cases}$$

Exemple Soit le produit de convolution

$$f(i) = \sum_{k=0}^{K} x(i-k) \times w(k)$$

avec $x(j) = 0$ pour $j < 0$. En posant $g_i(k) = w(k) \times x(i-k)$, ($i$ est une constante pour la généralisation), $f(i)$ est calculée par :

$$f(i) = \Gamma_{g_i}([0,K])$$
$$\Gamma_{g_i}(X) = \begin{cases} \text{si } vide(X) \text{ alors } 0 \\ \text{sinon } x(i - ch(X)) \times w(ch(X)) + \Gamma_{g_i}(m\_ch(X)) \end{cases}$$

Nous verrons au paragraphe 2 comment représenter X et les fonctions de choix possibles.

1.2.3 Schéma de calcul d'une généralisation contenant un appel récursif

Soit à calculer :

$$h(i,j) = \begin{cases} \textbf{si } vide(]i,j[) \textbf{ alors } w_{i,j} \\ \textbf{sinon } Min_{y\in]i,j[}(h(i,j)) \end{cases}$$

(Il s'agit en fait d'un cas particulier du problème de la programmation dynamique). La généralisation contient un appel récursif. La forme générale d'une telle fonction est :

$$f(x) = \begin{cases} \textbf{si } \Pi(x) \\ \textbf{alors } \varphi(x) \\ \textbf{sinon } \Gamma_{y\in\beta(x)}(\theta(\xi(x,y), f(\alpha_1(x,y)), f(\alpha_2(x,y)), \ldots)) \end{cases}$$

avec $(non\ \Pi(x)) \implies (\beta(x)\ non\ vide)$ (sauf si γ a un élément neutre). Nous allons simplifier l'écriture au cas d'un seul appel récursif mais le raisonnement est valide quelque soit le nombre d'appels. Dans ce cas on doit calculer :

$$f(x) = \begin{cases} \textbf{si } \Pi(x) \\ \textbf{alors } \varphi(x) \\ \textbf{sinon } \Gamma_{y\in\beta(x)}(\theta(\xi(x,y), f(\alpha(x,y)))) \end{cases}$$

(dans notre exemple $\Pi = vide, \varphi = w, \xi$ n'apparait pas, $\gamma = min, \Gamma = Min, \theta = Id, \alpha((i,j),y) = (i,j), \beta(i,j) =]i,j[$) En remplaçant $g(x)$ par $g_x(y) = \theta(\xi(x,y), f(\alpha(x,y))$ dans le schéma 1.3, $f(x)$ est calculée par :

Schéma 1.4

$$f(x) = \begin{cases} \textbf{si } \Pi(x) \\ \textbf{alors } \varphi(x) \\ \textbf{sinon } \Gamma_{g_x}(\beta(x)) \end{cases}$$

$$\Gamma_{g_x}(Y) = \begin{cases} \textbf{si } element(1,Y) \textbf{ alors } \theta(\xi(x, ch(Y)), f(\alpha(x, ch(Y))) \\ \textbf{sinon } \gamma(\theta(\xi(x, ch(Y)), f(\alpha(x, ch(Y))), \Gamma_{g_x}(m\_ch(Y))) \end{cases}$$

Exemple La fonction h ci-dessus est calculée par :

$$h(i,j) = \begin{cases} \textbf{si } vide(]i,j[) \textbf{ alors } w_{i,j} \\ \textbf{sinon } \Gamma_{g_{i,j}}(]i,j[) \qquad \textbf{avec } g_{i,j}(k) = h(i,y) \end{cases}$$

$$\Gamma_{g_{i,j}}(Y) = \begin{cases} \textbf{si } element(1,Y) \textbf{ alors } h(i, ch(Y)) \\ \textbf{sinon } min(h(i, ch(Y)), \Gamma_{g_{i,j}}(m\_ch(Y))) \end{cases}$$

On peut obtenir d'autres schémas de calcul en dépliant la définition de f dans Γ. Avec un seul dépliage on obtient le schéma (en utilisant la distributivité de la composition par rapport à l'alternative)

Schéma 1.5

$$f(x) = \begin{cases} \textbf{si } \Pi(x) \\ \textbf{alors } \varphi(x) \\ \textbf{sinon } \Gamma_{g_x}(\beta(x)) \end{cases}$$

$$\Gamma_{g_x}(,Y) = \begin{cases} \textbf{si } \mathit{element}(1,Y) \textbf{ alors } \theta(\xi(x,ch(Y)), f(\alpha(x,ch(Y))) \\ \textbf{sinon} \\ \quad \begin{cases} \textbf{si } \mathit{element}(2,Y) \textbf{ alors } \gamma(\theta(\xi(x,ch(Y)), f(\alpha(x,ch(Y))), \\ \quad\quad \theta(\xi(x,ch(m\_ch(Y))), f(\alpha(x,ch(m\_ch(Y)))))) \\ \textbf{sinon } \gamma(\gamma(\theta(\xi(x,ch(Y)), f(\alpha(x,ch(Y))), \\ \quad\quad \theta(\xi(x,ch(m\_ch(Y))), f(\alpha(x,ch(m\_ch(Y))))), \Gamma_{g_x}(,m\_ch^2(Y)))) \end{cases} \end{cases}$$

1.2.4 Schémas obtenus par éclatement

D'autres schémas abstraits utiles pour la systolisation sont obtenus en éclatant le calcul de Γ en p parties (qui pourront être éventuellement calculées en parallèle). Nous donnons un exemple à partir du schéma 1.3 $f(x) = \Gamma_g(X)$ est calculée par :

Schéma 1.6

$$\begin{aligned} f(x) &= \Gamma_{q\in[1,q]} f_q(x) \\ f_q(x) &= \Gamma_{g_x}(\beta_q(x)) \quad \textbf{avec } \forall x \quad X = \cup_{q\in[1,q]}\beta_q(x) \end{aligned}$$

Exemple Le produit de convolution peut être calculé par :

$$\begin{aligned} f(i) &= \textstyle\sum_{q\in[1,2]} f_q(i) \\ f_1(i) &= \textstyle\sum_{k\in pair[0,K]} w(k) \times x(i-k) \\ f_2(i) &= \textstyle\sum_{k\in impair[0,K]} w(k) \times x(i-k) \end{aligned}$$

(on a bien $[0,K] = pair[0,K] \cup impair[0,K]$). On peut calculer f_1 et f_2 avec le schema 1.3, ce qui donne :

$$f(i) = \Gamma_{g_i}(i, pair[0,K]) + \Gamma_{g_i}(i, impair[0,K])$$

$$\Gamma_{g_i}(i,X) = \begin{cases} \textbf{si } vide(X) \textbf{ alors } 0 \\ \textbf{sinon } x(i - ch(X)) \times w(ch(X)) + \Gamma_{g_i}(i, m\_ch(X)) \end{cases}$$

2 Application aux architectures systoliques

Dans cette partie nous allons construire des architectures systoliques de problèmes spécifiés à l'aide de fonctions de généralisation. Notre méthode se situe en aval des systèmes de conception automatique d'architectures systoliques qui utilisent des équations récurrentes (en particulier de la méthode de projection des dépendances et du logiciel DIASTOL). L'automatisation de la méthode comporte une phase de reconnaissance et d'application des schémas abstraits (filtrage, réécriture) et une phase de choix de solutions concrètes permettant une systolisation. En pratique les fonctions de choix concrètes utilisables sont stockées dans un catalogue conçu en tenant compte des impératifs de systolisation (linéarité, localité, uniformité). Aucune des machines systoliques que nous connaissons ne nécessite la conception de solutions concrètes non prévues dans ce catalogue. Nous ne donnons pas l'intégralité du catalogue ni les démonstrations de validité des représentations, mais nous en verrons des extraits à travers les applications. Pour une présentation détaillée des représentations concrètes on se reportera à [Joi87]. On s'y reportera également pour une description détaillée du fonctionnement des différentes architectures mentionnées dans cet article.

2.1 Produit de convolution

2.1.1 Définition du problème

$$f(i) = \sum_{k=0}^{K} x(i-k) \times w(k)$$

2.1.2 Application du schéma 1.3

D'après l'exemple du paragraphe 1.2.2, on obtient en remplaçant la notation $\Gamma_{g_i}(X)$ par $F(i, X)$

$$f(i) = F(i, [0, K])$$
$$F(i, X) = \begin{cases} \text{si } \mathit{vide}(X) \text{ alors } 0 \\ \text{sinon } x(i - \mathit{ch}(X)) \times w(\mathit{ch}(X)) + F(i, \mathit{m\_ch}(X)) \end{cases}$$

où X est un ensemble et les fonctions $vide, ch, m\_ch$ s'appliquent à cet ensemble. Le choix d'une représentation concrète du type ensemble permet de passer d'un schéma fonctionnel abstrait à un SERL concret. Pour une raison de place, nous ne développons pas ici la démarche de représentation ni les différents choix possibles. Cette étape de la méthode est présentée de façon détaillée dans [Joi87]

Solutions concrètes Nous donnons maintenant deux solutions concrètes possibles (il y en a d'autres) pour la solution abstraite ci-dessus.

1. On représente l'ensemble $[k, K]$ par k; en particulier $[0, K]$ est représenté par 0. Les fonctions concrètes sont données par :

$$\begin{cases} ch([k,K]) & == k \\ m\_ch([k,K]) & == k+1 \\ vide([k,K]) & == (k > K) \end{cases}$$

D'où le système :

$$f(i) = F(i,0)$$
$$F(i,k) = \begin{cases} \textbf{si } k > K \textbf{ alors } 0 \\ \textbf{sinon } x(i-k) \times w(k) + F(i,k+1) \end{cases}$$

A partir de ce système SERL, on applique une technique d'uniformisation par pipeline et on obtient le SERU :

$$f(i) \equiv F(i,0)$$
$$F(i,k) = \begin{cases} \textbf{si } k > K \textbf{ alors } 0 \\ \textbf{sinon } X(i,k) \times W(i,k) + F(i,k+1) \end{cases}$$
$$X(i,k) = \begin{cases} \textbf{si } i < 0 \textbf{ alors } 0 \\ \textbf{si } k = 0 \textbf{ alors } x(i-k) \\ \textbf{sinon } X(i-1,k-1) \end{cases}$$
$$W(i,k) = \begin{cases} \textbf{si } i = 0 \textbf{ alors } w(k) \\ \textbf{sinon } W(i-1,k) \end{cases}$$

Par la méthode de projection des dépendances [Qui84] on obtient l'architecture de la figure 2.

D'autres architectures peuvent bien sûr être obtenues à partir de ce système en projetant différemment (voir [Qui84]).

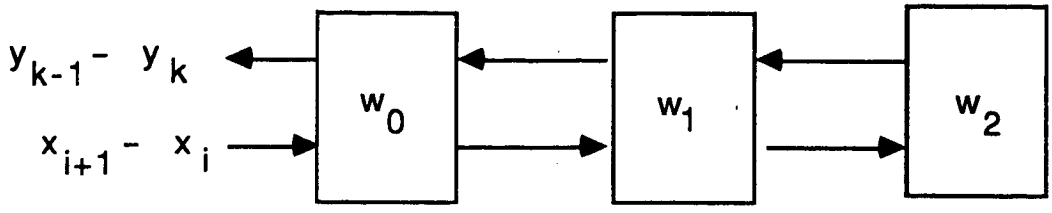

Figure 2 : Une première application du schéma 1.3

2. On représente $[0,k]$ par k, en particulier $[0,K]$ est représenté par K, les fonctions concrètes sont données par :

$$\begin{cases} ch([0,k]) & == k \\ m\_ch([0,k]) & == k-1 \\ vide([0,k]) & == (k<0) \end{cases}$$

D'où le système :

$$f(i) = F(i,K)$$
$$F(i,k) = \begin{cases} \textbf{si } k<0 \textbf{ alors } 0 \\ \textbf{sinon } x(i-k) \times w(k) + F(i,k-1) \end{cases}$$

qui permet d'obtenir après uniformisation par application de la méthode de projection des dépendances, l'architecture de la figure 3.

2.1.3 Application du schéma 1.6

D'après l'exemple du paragraphe 1.2.4 et en remplaçant la notation $\Gamma_{g_i}(X)$ par $F(i,X)$ on obtient :

$$f(i) = F(i, pair[0,K]) + F(i, impair[0,K])$$
$$F(i,X) = \begin{cases} \textbf{si } vide(X) \textbf{ alors } 0 \\ \textbf{sinon } x(i-ch(X)) \times w(ch(X)) + F(i, m\_ch(X)) \end{cases}$$

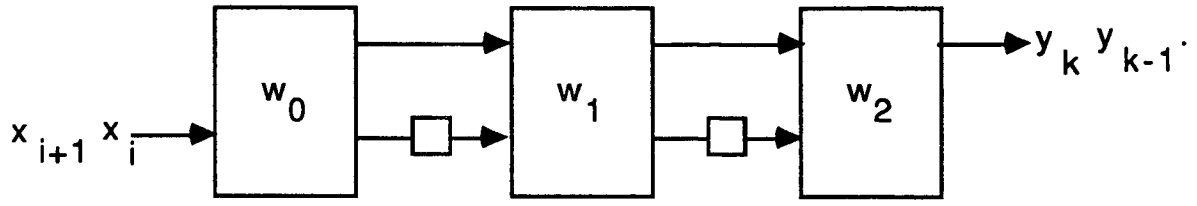

Figure 3 : Une seconde application du schéma 1.3

On représente l'ensemble $X = \{k+2l \mid 0 \leq k+2l \leq K\}$ par k, en particulier $pair([0,K])$ est représenté par 0 et $impair([0,K])$ est représenté par 1. Les fonctions concrètes sont données par :

$$\begin{cases} ch(X) & == k \\ m\_ch(X) & == k+2 \\ vide(X) & == (k > K) \end{cases}$$

D'où le système :

$$f(i) = F(i,0) + F(i,1)$$
$$F(i,k) = \begin{cases} \textbf{si } k > K \textbf{ alors } 0 \\ \textbf{sinon } x(i-k) \times w(k) + F(i,k+2) \end{cases}$$

Par uniformisation et application de la méthode de projection des dépendances on peut ainsi obtenir l'architecture de la figure 4.

De nombreuses autres architectures peuvent être obtenues pour le produit de convolution (on peut utiliser la technique de dépliage,...), et évidemment faire d'autres choix d'uniformisation et de projection des dépendances [Qui84,Joi87]. Le choix final dépend des contraites techniques de l'utilisateur.

Conclusion

L'utilisation d'un langage fonctionnel et de méthodes connues de transformation [Bau79,BD77] nous ont permis de trouver les schémas présentés

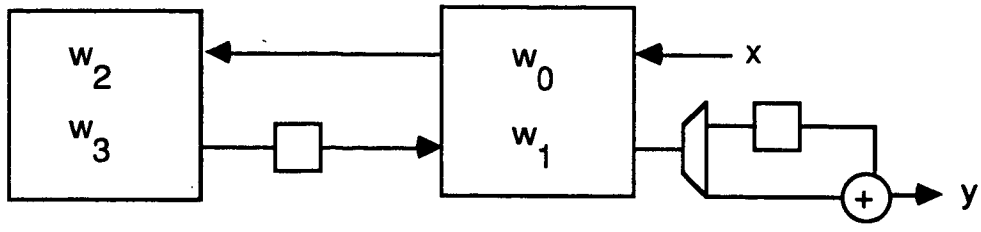

Figure 4 : Une application du schéma 1.6

dans cet article. Nous avons ensuite obtenu plusieurs architectures systoliques pour différents problèmes par l'application de ces schémas (produit de convolution présenté ici, programmation dynamique [GJQ86]). Ces transformations (ainsi que d'autres : schéma de Horner,...) seront intégrées dans la prochaine version du logiciel DIASTOL [Gac87] qui permet déjà de produire automatiquement des architectures systoliques à partir de spécifications sous forme de systèmes d'équations récurrentes uniformes. Le but recherché est de permettre à l'utilisateur de donner sa spécification sous une forme mathématique usuelle. Nous avons appliqué les schémas aux architectures systoliques mais nous envisageons maintenant l'application à d'autres architectures (hypercube par exemple). Nous sommes persuadés que la démarche fonctionnelle de transformation qui permet d'obtenir des algorithmes prouvés est particulièrement bien adaptée à la conception d'architectures parallèles fiables.

Bibliographie

[Bau79] F.L. Bauer. Program development by stepwise transformation - the project cip. *Notes in Computer Science*, 69, 1979.

[BD77] R. Burstall and J. Darlington. A transformation system for developping recursive programs. *J. Assoc. Comput. Mach.*, 44–

67, 1977.

[Coh78] D. Cohen. Mathematical approach to iterative computational networks. In *Proc. Fourth Symp. Computer Arithmetic*, pages 226–238, Oct. 1978.

[DI85] J.M. Delosme and I.C.F. Ipsen. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *International Symposium on VLSI Technology, Systems and Applications, Taipei, Taiwan, R.O.C.*, pages 268–273, May 1985.

[DI86] J.M. Delosme and I.C.F. Ipsen. Efficient systolic arrays for the solution of Toeplitz systems: an illustration of a methodology for the construction of systolic architectures for VLSI. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 37–46, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[FFW85] J.A.B. Fortes, K.S. Fu, and B.W. Wah. Systematic approaches to the design of algorithmically specified systolic arrays. In *ICASSP 85*, pages 300–303, 1985.

[Gac87] P. Gachet. Conception d'algorithmes et d'architectures systoliques. synthèse automatique de circuits. Thèse de l'Université de Rennes, Septembre 1987.

[GJQ86] P. Gachet, B. Joinnault, and P. Quinton. Synthesizing systolic arrays using diastol. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 25–36, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[Gue86] C. Guerra. A unifying framework for systolic designs. In *VLSI Algorithms and Architectures, G.Goos and J. Hartmanis eds, Springer-Verlag*, pages 46–56, July 1986.

[Joi87] B. Joinnault. Conception d'algorithmes et d'architectures systoliques. synthèse de systèmes d'équations récurrentes uniformes (seru). Thèse de l'Université de Rennes, Septembre 1987.

[KMW67] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, 14(3):563–590, July 1967.

[Kog74] P.M. Kogge. Parallel solution of recurrence problems. *IBM Journal of research and development*, 18(2):138–148, March 1974.

[Kun82] H.T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, January 1982.

[LRS83] C.E. Leiserson, F.M. Rose, and J.B. Saxe. Optimizing synchronous circuitry by retiming (preliminary version). In R. Bryant, editor, *Proc. 3d Caltech Conf. on VLSI*, pages 87–116, Computer Science Press, 1983.

[Mol82] D.I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Transactions on Computers*, C-31(11), November 1982.

[Mon85] C. Mongenet. Une méthode de conception d'algorithmes systoliques, résultats théoriques et réalisation. Thèse de 3ieme cycle, 1985.

[MW84] W.L. Miranker and A. Winkler. Space-time representations of systolic computational structures. *Computing*, 32:93–114, 1984.

[QG84] P. Quinton and P. Gachet. *Manuel d'Utilisation de DIASTOL. Version Préliminaire.* Technical Report 41, Rapports de DeRecherche INRIA, Centre de Rennes IRISA, Rocquencourt, France, Octobre 1984.

[Qui84] P. Quinton. Automatic synthesis of systolic arrays from recurrent uniform equations. In *11th Annual Int. Symp. Computer Arch., Ann Arbor*, pages 208–214, June 1984.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique