Une expérience d'implantation d'un algorithme systolique sur hypercube

S. Genaud, G.-R. Perrin
Laboratoire d'Informatique de Besançon
Université de Franche-Comté
F-25030 Besançon cedex

Introduction

Les nouvelles générations de multiprocesseurs à mémoire distribuée proposent des tableaux de processeurs, à connexions locales et régulières (grilles, hypercubes, tores, etc.). On peut noter, et profiter de, l'analogie entre de telles architectures et des topologies systoliques dont le design a fait l'objet de nombreuses recherches dans la dernière décade.

Dans cet esprit, les systèmes d'équations récurrentes affines sont de bons candidats, de par leurs caractéristiques de régularité et de localité, à l'expression des problèmes et des algorithmes sous la forme de solutions systoliques ou de solutions parallèles synchrones plus générales. L'implantation de ces solutions peut alors être réalisée sur les multiprocesseurs à mémoire distribuée. Les problèmes posés sont des problèmes d'adaptation de l'algorithme à l'architecture : nombre de processeurs, topologie du réseau de connexions, etc. Les solutions passent par des techniques d'allocation par optimisation et partitionnement.

De nombreux travaux proposent des méthodes de synthèse de solutions systoliques à partir de systèmes d'équations récurrentes (voir entre autres [Mol84], [Mon85], [Qui84] et [RF87]). Les différentes étapes de ce type de méthodes sont l'analyse des dépendances entre les calculs du problème, la détermination d'une fonction de temps qui définit un ordonnancement des calculs, et la détermination d'une allocation de ces calculs à une surface de processeurs.

Les solutions parallèles ainsi synthétisées peuvent être, soit des réseaux systoliques, soit des réseaux semi-systoliques (i.e. avec diffusion), soit des réseaux pseudo-systoliques (i.e. avec processeurs programmables). A partir de ces solutions initiales, des techniques d'optimisation et de partitionnement permettent d'obtenir de nouvelles solutions régulières, caractérisées par un nombre minimal de processeurs.

Nous rapportons ici l'expérience de portage d'une telle solution sur un multiprocesseur à mémoire distribuée, en l'occurrence un hypercube iPSC/2 de Intel. La présentation qui suit est illustrée avec l'exemple du chemin algébrique (APP).

1 Les systèmes d'équations récurrentes affines

Nous considérons donc l'expression des algorithmes sous la forme de systèmes d'équations récurrentes affines. Une équation récurrente affine est de la forme :

$$V(z) = f_V(\ldots, U(\rho(z)), \ldots) \quad z \in Dom(V) \qquad (E)$$

où $Dom(V)$, *domaine de* V, est un polyèdre convexe borné de Z^n et la fonction ρ est une fonction affine de Z^n dans $Z^{n'}$, $(n' \leq n)$.

Exemple 1 *Nous considérons l'exemple classique du chemin algébrique. Cet exemple constitue un exercice favori dans la littérature traitant de méthodes de synthèse. Il est présenté dans [Rot85] comme suit.*

Considérons un graphe orienté $G = (N, L, w)$ où N est un ensemble de n nœuds, L l'ensemble des arcs et w la fonction qui associe un poids à chaque arc. Ces poids définissent la matrice A de dimension $n \times n$.

Le problème du chemin algébrique consiste à déterminer, pour tout couple de nœuds (i, j), la valeur $d_{ij} = \oplus_{p \in M_{ij}} w(p)$, où M_{ij} est l'ensemble des chemins de i à j. Le poids d'un chemin $p \in M_{ij}$ est défini par $w(p) = \otimes_{e \in p} w(e)$.

Notons $M_{ij}^{(k)}$ l'ensemble des chemins de i à j dont les nœuds intermédiaires ont un numéro inférieur ou égal à k. Les suites a_{ij} définies par $a_{ij}^{(k)} = \oplus_{p \in M_{ij}^k} w(p)$ sont telles que $d_{ij} = a_{ij}^{(n)}$. Ces suites sont définies par les récurrences suivantes :

$$\left\{ \begin{array}{lllr} a(k,k,k) & = a(k,k,k-1)^* & 1 \leq k \leq n & (1) \\ a(i,j,k) & = a(i,j,k-1) \otimes a(k,j,k) & 1 \leq i,k \leq n,\ i \neq k,\ j = k & (2) \\ a(i,j,k) & = a(i,j,k-1) \otimes a(i,k,k) & 1 \leq j,k \leq n,\ j \neq k,\ i = k & (3) \\ a(i,j,k) & = a(i,j,k-1) \oplus a(i,k,k) \otimes a(k,j,k-1) & & (4) \\ & & 1 \leq i,j,k \leq n,\ i \neq k,\ j \neq k & \end{array} \right.$$

Ce problème définit les 4 sous-domaines suivants du domaine de a, associés respectivement à chacune de ces équations :

$$\begin{aligned}
D_1 &= \{(k,k,k) \in Z^3 \mid 1 \leq k \leq n\} \\
D_2 &= \{(i,k,k) \in Z^3 \mid 1 \leq i,k \leq n,\ i \neq k\} \\
D_3 &= \{(k,j,k) \in Z^3 \mid 1 \leq j,k \leq n,\ j \neq k\} \\
D_4 &= \{(i,j,k) \in Z^3 \mid 1 \leq i,j,k \leq n,\ i \neq k,\ j \neq k\}
\end{aligned}$$

□

2 Synthèse de solutions systoliques

La définition d'une solution algorithmique à partir d'un système d'équations récurrentes consiste en une *allocation dans l'espace et dans le temps* des calculs spécifiés, c'est-à-dire en une transformation de ce système pour substituer un domaine de la forme $S \times T$ à tout domaine $Dom(V)$. S est un domaine de dimension r, $r \leq n-1$, qui représente l'architecture sur laquelle sont implantés les calculs, et $T \subset Z$ est la référence selon laquelle les calculs alloués au même processeur sont cadencés.

Dans la pratique, les méthodes de synthèse opèrent lorsque $r = n-1$. L'allocation dans l'espace est alors réalisée par *projection* parallèlement à un vecteur ξ de Z^n, dit *vecteur d'allocation*. Ces méthodes reposent sur une modélisation des dépendances de données, explicitées par le système d'équations.

2.1 Analyse des dépendances

La première étape consiste à extraire les informations caractérisant les *dépendances causales* entre les calculs élémentaires du problème.

Pour une équation (E), on caractérise l'ensemble des points qui utilisent une occurrence donnée $U(\zeta_0)$, $\zeta_0 \in Z^{n'}$. Nous appelons cet ensemble l'*ensemble d'utilisation* (cf. [MCP92]). Il est défini ainsi :

Définition 1 *L'*ensemble d'utilisation *associé à une occurrence $U(\zeta_0)$ dans l'équation (E) est $Util_U(\zeta_0) = \{z \in Dom(V) \mid \rho(z) = \zeta_0\}$.*

Puisque la fonction ρ est affine, les solutions de $\rho(z) = \zeta_0$ définissent un sous-espace affine de dimension m de Z^n. $Util_U(\zeta_0)$ est donc l'intersection de ce sous-espace affine et d'un polyèdre convexe borné. D'où la propriété suivante :

Propriété 1 *Tout ensemble d'utilisation $Util_U(\zeta_0)$ est un polyèdre convexe borné.*

Définition 2 *On appelle* vecteur générateur *associé à la variable U dans l'équation (E) tout vecteur d'une base de $Util_U(\zeta_0)$, pour tout $\zeta_0 \in Z^{n'}$. On note $\phi_{U,i}$, $i = 1, \ldots, m$, de tels vecteurs.*

Notons que les seules dépendances causales imposées par un problème sont liées aux suites résultats. Une donnée U ne définit en effet pas de dépendance *a priori* entre les calculs. Cependant comme l'objectif ultime est de synthétiser une architecture parallèle, la distribution des flots de données doit être définie : pour chaque donnée U ceci est réalisé à partir l'ensemble $Util_U(\zeta_0)$ et des vecteurs générateurs.

Pour ce qui concerne le cas où la variable U qui apparait dans l'équation (E) est un résultat, on suppose dans la suite que $n' = n$. Pour toute occurrence $U(\zeta_0)$ l'ensemble des points z qui dépendent causalement du point ζ_0 est $Util_U(\zeta_0)$. Pour représenter la dépendance causale associée à cet ensemble d'utilisation, nous introduisons la notion de *vecteur de dépendances* défini de la façon suivante :

Définition 3 *On appelle* vecteur de dépendances *associé à la variable résultat U tout vecteur $\psi_{U,\rho(z)} = z - \rho(z)$.*

Un ensemble d'utilisation associé à une variable résultat est donc caractérisé par sa base de vecteurs générateurs ϕ et par les vecteurs de dépendances ψ. Ces derniers connectent le point $\zeta_0 = \rho(z)$ qui calcule le résultat $U(\zeta_0)$ à tous les points z, $z \in Util_U(\zeta_0)$, qui en dépendent. Tous les vecteurs de dépendances ψ_{U,ζ_0} d'un ensemble donné ont donc la même origine ζ_0 et leurs extrémités appartiennent au polyèdre convexe borné $Util_U(\zeta_0)$. On en déduit la propriété suivante :

Propriété 2 *L'ensemble des vecteurs de dépendances ψ_{U,ζ_0} associés au résultat $U(\zeta_0)$ dans l'équation (E) définit un* cône *caractérisé par un ensemble fini de* vecteurs extrémaux*.*

Notons qu'il y a autant de cônes de vecteurs de dépendances qu'il y a de variables résultats dans la partie droite de n'importe quelle équation (E).

2.2 Cadencement et allocation

Les méthodes classiques de synthèse de solutions systoliques considèrent seulement des cadencements des calculs, qui sont des fonctions affines des coordonnées z des points de calcul, et des fonctions d'allocation linéaires qui correspondent, comme nous l'avons dit, à une projection des calculs sur une surface de processeurs. Ces méthodes d'allocation opèrent en se restreignant, de plus, au cas de fonctions de temps indépendantes des variables du problème.

Une fonction de temps définit donc, pour tout point z du domaine, son instant d'exécution $t(z) = \lambda \cdot z + \alpha$ où λ est un vecteur unimodulaire de Z^n et α est une constante entière. Pour satisfaire les dépendances causales, toute fonction de temps affine doit vérifier la propriété suivante :

Propriété 3 *Une fonction de temps affine est valide si et seulement si sa partie* *linéaire λ satisfait $\lambda \cdot \psi_{ex} > 0$ pour tout vecteur de dépendances extrémal ψ_{ex} du problème.*

L'allocation des calculs aux processeurs est réalisée par projection du domaine parallèlement à une *direction d'allocation*, définie par un vecteur unimodulaire $\xi \in Z^n$.

A partir de cette direction, on déduit une *fonction d'allocation* qui projette tout point z du domaine de calcul sur un ensemble de processeurs dans Z^{n-1}. Cette fonction, de la forme $alloc(z) = \sigma \cdot z$, est définie en inversant la base de Z^n, dite *base spatiale*, qui complète le vecteur ξ.

Tous les points du domaine appartenant à une même droite parallèle à ξ sont exécutés sur le même processeur et ne peuvent donc être exécutés au même instant. Cette condition entre les fonctions de temps et d'allocation s'exprime par la propriété suivante :

Propriété 4 *Soit λ la partie linéaire d'une fonction de temps affine. Un vecteur ξ définit une direction d'allocation si et seulement si $\lambda \cdot \xi \neq 0$.*

A partir des fonctions de temps et d'allocation, le design d'un réseau solution (topologie, distribution des flots de données, cadencement) est immédiat.

Exemple 2 *L'exemple du chemin algébrique pose le problème de l'existence d'une fonction de temps affine : un ré-indiçage des équations initiales est nécessaire à la détermination d'un cadencement des calculs (cf. [CMP92b]). Nous considérons par conséquent ce problème après ces transformations :*

$$
\left\{
\begin{array}{lllr}
a(k,k,k) & = a(k,k,k-1)^* & 1 \leq k \leq n & (1) \\
a(i,k,k) & = a(i,k,k-1) \otimes a(k,k,k) & k < i < k+n-1,\ 1 \leq k \leq n & (2.1) \\
a(i,k,k) & = a(i-n,k,k-1) \otimes a(k,k,k) & i = k+n-1,\ 1 \leq k \leq n & (2.2) \\
a(k,j,k) & = a(k,j,k-1) \otimes a(k,k,k) & k < j < k+n-1,\ 1 \leq k \leq n & (3.1) \\
a(k,j,k) & = a(k,j-n,k-1) \otimes a(k,k,k) & j = k+n-1,\ 1 \leq k \leq n & (3.2) \\
a(i,j,k) & = a(i,j,k-1) \oplus a(i,k,k) \otimes a(k,j,k-1) & & (4.1) \\
 & & k < i,j < k+n-1,\ 1 \leq k \leq n & \\
a(i,j,k) & = a(i-n,j,k-1) \oplus a(i,k,k) \otimes a(k,j,k-1) & & (4.2) \\
 & & i = k+n-1,\ k < j < k+n-1,\ 1 \leq k \leq n & \\
a(i,j,k) & = a(i,j-n,k-1) \oplus a(i,k,k) \otimes a(k,j-n,k-1) & & (4.3) \\
 & & k < i < k+n-1,\ j = k+n-1,\ 1 \leq k \leq n & \\
a(i,j,k) & = a(i-n,j-n,k-1) \oplus a(i,k,k) \otimes a(k,j-n,k-1) & & (4.4) \\
 & & i = k+n-1,\ j = k+n-1,\ 1 \leq k \leq n &
\end{array}
\right.
$$

Le domaine de a est défini par

$$Dom(a) = \{(i,j,k) \in Z^3 \mid k \leq i \leq k+n-1,\ k \leq j \leq k+n-1,\ 1 \leq k \leq n\}$$

Une solution systolique peut être synthétisée, caractérisée par un cadencement et une direction d'allocation définis par :

$$\lambda = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad \xi = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Ce cadencement optimal définit une latence $\tau = 5n - 4$. La solution est représentée pour $n = 6$ sur la figure 1.

□

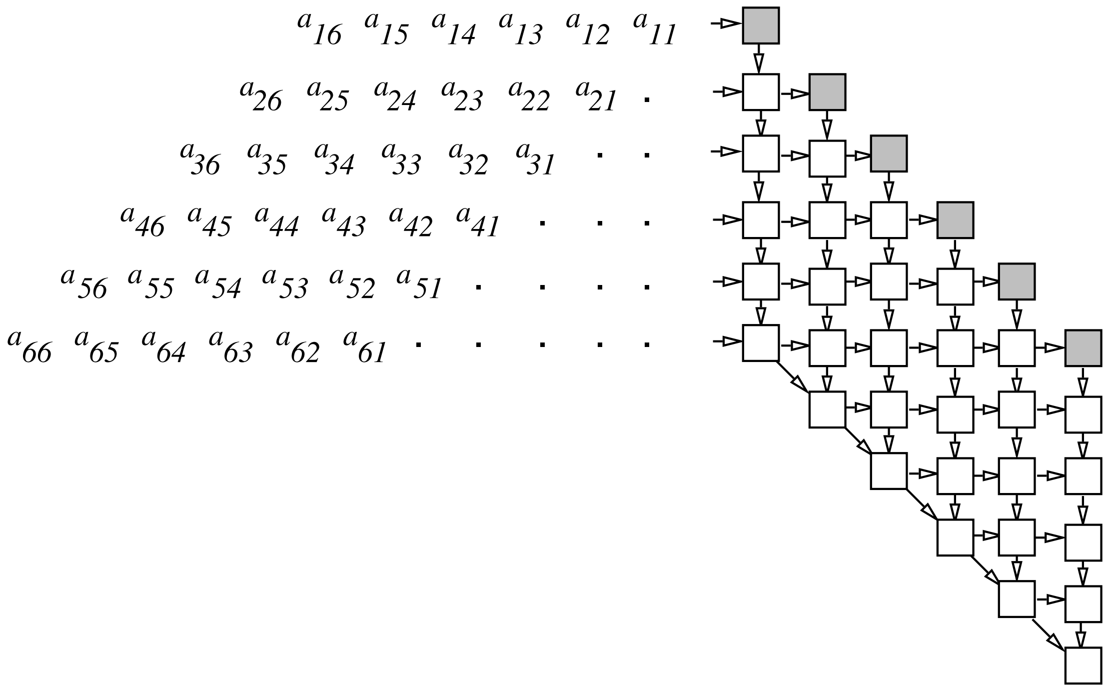


FIG. 1 - *Un réseau systolique pour l'APP*

3 Optimisation du nombre de processeurs

Les réseaux systoliques déterminés par ce type de méthodes ont de bonnes propriétés de régularité et de localité des connexions. Mais leur inconvénient majeur est que le taux de parallélisme de ces solutions est souvent loin d'être optimal : le nombre de processeurs du réseau est beaucoup plus grand que le nombre maximal de processeurs actifs simultanément. Des résultats complets quant à la recherche de l'optimalité en nombre de processeurs sont donnés dans [Cla90] et [CMP92a]. Nous n'en présentons ici que quelques notions de base.

Pour un cadencement donné, deux situations peuvent entrainer un taux de parallélisme non-optimal. Soit les processeurs ne sont pas *efficaces*, c'est-à-dire qu'ils ne sont pas actifs à chaque instant entre leur première et dernière activation : on peut alors appliquer une technique classique de *clustering* (voir par exemple [Cla90], [Dar91] et [BDD90]). Soit les processeurs ont des périodes d'activité disjointes : on peut alors *pipeliner* de tels processeurs en une seule unité fonctionnelle. On montre dans [Cla90] que l'application de ces deux techniques permet toujours de déterminer des implantations optimales en place. Leur définition passe par une modélisation de l'activité des réseaux initiaux.

3.1 Modélisation de l'activité des réseaux systoliques

3.1.1 Rythme d'activité

Le vecteur λ caractérisant toute fonction de temps affine, est orthogonal aux hyperplans de Z^n des points ayant le même instant d'exécution. Ces hyperplans, appelés *surfaces de temps* sont engendrés par $n-1$ vecteurs δ, obtenus en inversant la base de Z^n complétant λ, appelée sa *base de temps*. Le nombre de surfaces de temps coupant le domaine de calcul définit le temps total d'exécution τ, ou *latence*, de l'algorithme correspondant.

Les deux bases ainsi définies, la base spatiale et la base de temps, modélisent le rythme d'activité du réseau systolique associé : le produit scalaire $\lambda \cdot \xi$ (supposé > 0) mesure le *taux d'activité* du réseau, dont les processeurs sont actifs un top sur $\lambda \cdot \xi$. De plus, le produit de matrices $A = \sigma \cdot \delta$, ou *matrice d'activité*, détermine les processeurs du réseau actifs simultanément.

Exemple 3 *Retrouvons l'exemple du chemin algébrique. Pour le cadencement précédent mais une autre direction d'allocation*

$$\lambda = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad \xi = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

les bases de temps et spatiale sont égales à :

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

Le taux d'activité est $\lambda \cdot \xi = 3$: les processeurs sont actifs un top sur 3. La matrice d'activité est égale à :

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

ce qui signifie que si (x, y) est un processeur actif à un instant t_0, tout processeur décalé par rapport à (x, y) d'une combinaison linéaire des vecteurs $\left(\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right)$ et $\left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right)$ est également actif à cet instant t_0.
Au contraire la direction d'allocation précédente définit un réseau efficace, dont les processeurs sont actifs à chaque instant pendant leur période d'activité.
□

3.1.2 Période d'activité

Revenons au cadencement, pour modéliser maintenant les périodes d'activité des processeurs. La projection des points de calculs selon une direction d'allocation ξ définit un ensemble de segments de droites parallèles coupant le domaine de calcul. Ces segments, appelés *segments d'allocation*, intersectent au moins deux faces du polyèdre convexe défini par le domaine de calcul. Les deux extrêmités de chacun des segments définissent respectivement la première et la dernière activation du processeur associé, selon le cadencement choisi. Notons que ces extrémités décrivent des faces du polyèdre, mais que dans certains cas, ces points décrivent également des parties bornées d'hyperplans parallèles à ces faces. On parle alors de *faces épaisses*. Pour simplifier cette présentation, nous ne considérons pas le cas des faces épaisses.

Définition 4 *On appelle* premières faces, *respectivement* dernières faces, *les faces du polyèdre défini par le domaine de calcul, contenant les extrémités des segments d'allocation.*

Pour tout instant t_0, les intersections entre la surface de temps associée et les premières, ou dernières, faces caractérisent les processeurs qui ont leurs premières, ou dernières, activations à cet instant t_0. Ces intersections définissent soit des segments de droite, soit les faces elles-mêmes. Dans la suite, notre présentation se limite aux cas où ces intersections définissent des segments de droite, dont on note respectivement et de façon générique pour l'ensemble des premières et des dernières faces, f et l les vecteurs directeurs.

Pour tout instant t_0 et dans le cas où les processeurs sont actifs à chaque instant, on note f_{t_0+1} une droite de la surface de temps associée à t_0+1, engendrée par f, et l_{t_0} une droite de la surface de temps t_0 engendrée par l. La projection de ces deux droites sur l'architecture initiale, parallèlement à ξ caractérise soit un cône, soit deux droites parallèles, dans le cas $f = l$. Ce cône ou ces droites parallèles caractérisent des processeurs dont la première activation est définie par un point de la première face considérée et dont la dernière activation est définie par un point de la dernière face considérée. Ces processeurs sont donc actifs à chaque instant t_0 pour lequel ils apparaissent à l'intérieur de la zône définie à partir de l_{t_0} et f_{t_0+1}.

Dans le cas où les processeurs ne sont pas actifs à chaque instant, ce résultat se généralise immédiatement en considérant l_{t_0} et $f_{t_0+\lambda\cdot\xi}$.

Exemple 4 *Pour le chemin algébrique, la direction d'allocation*

$$\xi = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

définit une seule première face et une seule dernière face. Les intersections entre ces faces et une surface de temps définissent deux droites parallèles. Par conséquent, les processeurs actifs à un instant t_0 donné sont situés entre les deux droites parallèles l_{t_0} et f_{t_0+1}. Celles-ci sont représentées en grisé sur la figure 2 pour $t_0 = 11$ et $n = 6$.

Remarquons le faible nombre de processeurs actifs par rapport au nombre total de processeurs de l'architecture. Le nombre maximum de processeurs actifs simultanément, c'est-à-dire le parallélisme potentiel de cet algorithme est en effet égal à :

$$pp = \begin{cases} n^2/3 & si\ n \bmod 3 = 0 \\ (n^2+2)/3 & sinon \end{cases}$$

□

3.2 Exemple d'optimisation par pipeline

Dans le cas de réseaux systoliques efficaces, on peut *pipeliner* plusieurs processeurs en une seule unité fonctionnelle, en considérant leurs périodes d'activité respectives. Exprimée dans la modélisation précédente, cette technique alloue à la même unité fonctionnelle les calculs effectués par un processeur associé à une droite l_{t_0} et ceux effectués par un processeur associé à une droite f_{t_0+1}.

Exemple 5 *Dans la solution précédente du chemin algébrique, les positions successives des deux droites parallèles de processeurs actifs permettent de pipeliner les processeurs selon un vecteur constant $(\lceil n/3 \rceil, \lceil n/3 \rceil)$. Ces pipelines de processeurs et l'implantation résultante de forme toroidale sont visualisés figure 3 pour $n = 6$.*

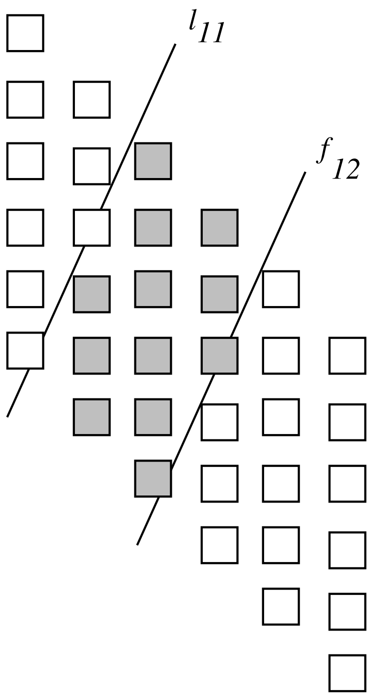


FIG. 2 - *Zône des processeurs actifs à* $t_0 = 11$

Le nombre de processeurs obtenu par cette optimisation est :

$$nbp = \begin{cases} n^2/3 & si\ n \bmod 3 = 0 \\ n(n+2)/3 & si\ n \bmod 3 = 1 \\ n(n+1)/3 & si\ n \bmod 3 = 2 \end{cases}$$

Remarquons que les implantations ainsi déterminées sont optimales en place lorsque $n \bmod 3 = 0$ *et très proches de l'optimalité dans les autres cas.*
□

Cette technique permet d'optimiser directement des solutions efficaces. Dans les autres cas, où $\lambda \cdot \xi > 1$, des techniques classiques de *clustering* peuvent compléter ces transformations pour optimiser les réseaux. C'est le cas sur l'exemple du chemin algébrique, en considérant la seconde direction d'allocation (cf. [CMP92b]).

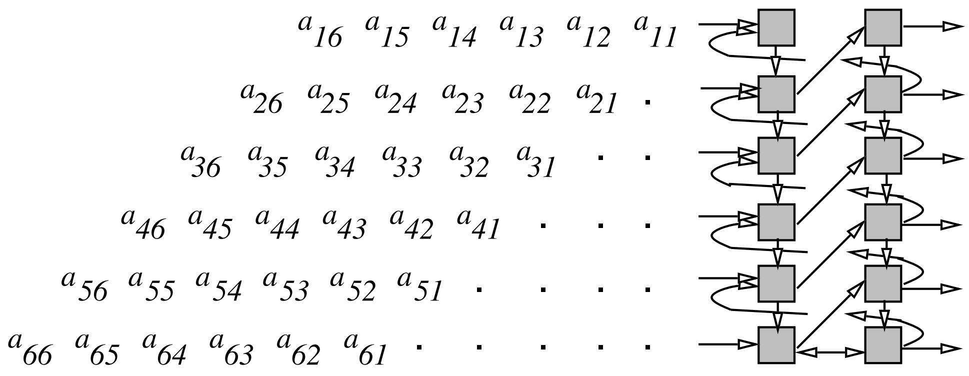


FIG. 3 - *Réseau toroidal pour l'APP*

4 Implantation sur un multiprocesseur à mémoire distribuée

L'implantation de ce type d'algorithmes sur un multiprocesseur à mémoire distribuée suppose leur adaptation aux caractéristiques propres de l'architecture considérée, c'est-à-dire essentiellement la topologie physique du réseau de connexions et le nombre de processeurs, ainsi que la gestion système de ces ressources : placement des processus et routage des messages. Dans les cas les plus favorables, la définition de *topologies virtuelles* implantant des routages réguliers efficaces, par exemple sur des architectures SIMD *data-parallèles* (MasPar, Connection Machine 2, etc.) permet de s'abstraire en partie de ces contraintes.

Du point de vue du modèle d'exécution, l'expression de solutions algorithmiques du type précédent, qui précise le placement et le cadencement des calculs dans un domaine $S \times T$, permet d'introduire

- un schéma SIMD, en considérant d'abord l'itération sur T. Dans ce modèle, appelé *parallélisme de données*, l'algorithme apparaît comme une séquence de traitements appliqués en parallèle sur l'ensemble des processeurs de S, cette séquence étant ponctuée de phases de distribution des données. L'implantation associée est déterministe, et connaissant la durée, bornée, de chaque opération, on peut pré-déterminer les performances de l'algorithme.
- un schéma MIMD, en considérant d'abord l'itération sur S. Dans ce modèle, dit à *parallélisme de contrôle*, l'algorithme apparaît comme une itération parallèle de traitements séquentiels interagissant par des commu-

nications. C'est cette forme d'algorithme que nous avons implantée sur l'iPSC/2 de Intel.

Les options prises dans cette expérimentation sont d'implanter strictement chaque processus de la solution sur un nœud de l'hypercube, après une éventuelle adaptation classique par *partitionnement* [MF86] que nous ne rappelons pas ici, et de distribuer les données par des échanges asynchrones de messages : l'attente sur la réception bloquante de ces messages définit des *phases* d'exécutions successives qui suivent le rythme synchrone de circulation des flots de données, sans pour autant garantir la régularité de la chronologie des calculs.

L'algorithme implanté est la solution optimisée du chemin algébrique présentée ci-dessus, pour $n = 4$, en utilisant un cube à 8 nœuds. Un fichier de traces produit par la bibliothèque PICL permet de reconstituer l'exécution du programme post-mortem. Ces traces peuvent être visualisées sous forme d'animation graphique en utilisant ParaGraph.

4.1 Programmation de l'exemple

Pour la direction d'allocation considérée, les segments d'allocation traversent différents sous-domaines de $Dom(a)$. La solution pseudo-systolique correspondante définit donc des processeurs programmables, exécutant périodiquement

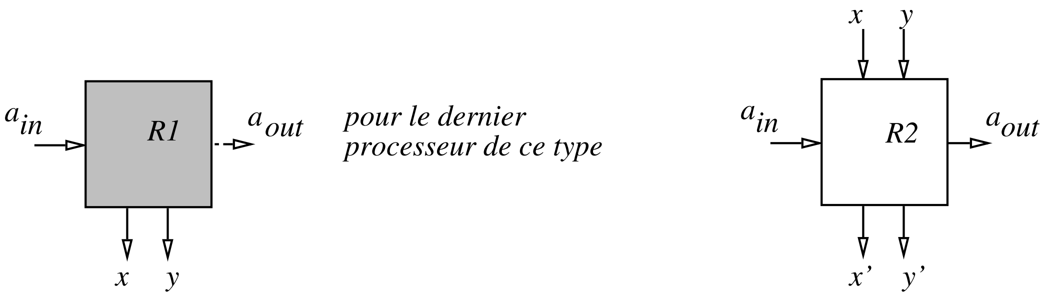
- des calculs correspondant aux équations (1) puis (3), pour les processeurs de la première diagonale du réseau, en grisé sur la figure 1,
- des calculs correspondant aux équations (2) puis (4), pour les autres processeurs.

Ces processeurs sont explicités sur la figure 4 en instantiant le problème du chemin algébrique par le calcul de la matrice des coûts minima dans un graphe. Le programme correspondant est chargé sur tous les nœuds, qui exécutent un des deux types de calcul selon leur position dans la topologie : sur l'exemple traité, les nœuds 7 et 0 exécutent le premier type de traitement, les nœuds 6, 5, 4 et 1, 2, 3 exécutent le second, correspondant respectivement à la première et à la seconde colonne du tore présenté sur la figure 3.

Les données formant la matrice initiale sont transmises aux nœuds 7, 6, 5 et 4 par le processeur hôte, qui reçoit finalement les résultats pour l'affichage. Les nœuds 0, 1, 2 et 3, selon la phase de l'algorithme, routent leur résultat vers les nœuds 7, 6, 5 et 4, ou vers le processeur hôte.

4.2 Résultats et enseignements

L'animation graphique des traces d'exécution sous ParaGraph (cf. figure 5) visualise les phases successives d'exécutions au rythme des échanges de messages



x := ain ; y := ain ; R1 := x ;
faire (n−1) fois
  x := ain + R1 ; {aout := x ; pour le dernier processeur}
  y := ain ;
{aout := R1 pour le dernier processeur}

x' := x ; y' := y ; R2 := ain + x ;
faire (n−1) fois
  aout := min (ain, R2 + y) ;
  x' := x ; y' := y ;
aout := R2

FIG. 4 - *Les deux types de processeurs pour l'APP*

entre les nœuds. Ces phases sont des *dilatations* plus ou moins importantes dans le temps et de façon imprévisible selon l'engorgement des communications, des opérations synchrones définies dans la solution systolique.

L'objectif d'une telle implantation n'est pas une étude en matière de performance. Le faible grain de parallélisme de ce type d'algorithme MIMD se prête évidemment peu à une implantation *en l'état* sur une architecture de ce type. Les temps obtenus sont donc peu significatifs et révèlent une importance exagérée des communications. A cet égard, un des enseignements à tirer est que les optimisations proposées par rapport aux solutions systoliques, par des techniques de clustering et pipeline de processeurs permettent

- de modifier le grain de parallélisme, en améliorant sensiblement l'efficacité de l'implanta-tion, toutes choses étant égales par ailleurs,
- d'adapter un algorithme initial au nombre de processeurs d'une architecture donnée, selon le choix de cadencement, induisant le parallélisme potentiel de la solution, et le choix de direction d'allocation, complétant ainsi les techniques de partitionnement.

Ces techniques, en grande partie maîtrisées, sont toutes des techniques qui viennent en amont de la phase d'implantation elle-même, en raisonnant par transformations et adaptations de la solution, indépendamment du modèle d'exécution retenu. Ces mêmes techniques permettent d'adapter une solution selon les facilités spécifiques d'une architecture donnée, en privilégiant par exemple

Fig. 5 - *Une étape de visualisation sous ParaGraph*

des routages de proximité ou des diffusions partielles ou totales des données. Il demeure que le caractère spécifique des solutions construites par ce type d'approches trouverait bien sûr une réponse plus satisfaisante sur des architectures à parallélisme de données, qui privilégient le niveau d'abstraction et les performances en matière de communications, ou sur des structures quasi dédiées comme un tableau multiprocesseur iWARP, par exemple.

Notons cependant l'intérêt fondamental, et conforté par cette expérimentation, qui réside dans l'aspect systématique, voire automatique, de ce type d'approches pour la construction et l'implantation d'algorithmes parallèles, depuis des énoncés en termes d'équations récurrentes affines, jusqùà des mises en œuvre impliquant des modèles d'exécutions différents. Le savoir-faire validé par cette expérimentation est la base des travaux que nous développons actuellement, pour réaliser un environnement de programmation (cf. à ce sujet les résultats prometteurs de [BL92] pour des travaux du même type), intégrant les étapes de transformations des systèmes d'équations récurrentes, le design des solutions (topologie des communications, calculs des processeurs, etc.) leur optimisation, et leur expression sous la forme de programmes parallèles concrets, selon différents modèles d'exécution SIMD ou MIMD.

Références

[BDD90] J. Bu, E.F. Depreterre, and P. Dewilde. A design methodology for fixed-size systolic arrays. *IEEE Int. Conf. on Application Specific Array Processing*, pages 591–602, 1990.

[BL92] M. Barnett and C. Lengauer. *Loop parallelization and unimodularity.* Algorithmique Parallèle, ERI. Masson, Paris, 1992.

[Cla90] P. Clauss. *Synthèse d'algorithmes systoliques et implantation optimale en place sur réseaux de processeurs synchrones.* PhD thesis, Université de Franche-Comté, Mai 1990.

[CMP92a] P. Clauss, C. Mongenet, and G.-R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. *J. of VLSI Signal Processing*, 4:27–36, 1992.

[CMP92b] P. Clauss, C. Mongenet, and G.-R. Perrin. Synthesis of size-optimal toroidal arrays for the algebraic path problem : A new contribution. *Parallel Computing*, 18:185–194, 1992.

[Dar91] A. Darte. Regular partitioning for synthesizing fixed-size systolic arrays. *VLSI Journal*, 12:293–304, 1991.

[MCP92] C. Mongenet, P. Clauss, and G.-R. Perrin. *Implantations optimales d'équations récurrentes affines.* Algorithmique Parallèle, ERI. Masson, Paris, 1992.

[MF86] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transactions on Computers*, c-35(1):1–12, 1986.

[Mol84] D.I. Moldovan. Advis : a software package for the design of systolic arrays. *IEEE ICCD : VLSI in Computers*, pages 158–164, 1984.

[Mon85] C. Mongenet. *Une méthode de conception d'algorithmes systoliques : résultats théoriques et réalisation.* PhD thesis, INPL, Mai 1985.

[Qui84] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrence equations. *IEEE, Int. Symp. on Computer Architecture*, pages 208–214, 1984.

[RF87] S.V. Rajopadhye and R.M. Fujimoto. Systolic array synthesis by static analysis of program dependencies. *Lect. Notes in Computing Science*, 259:295–310, 1987.

[Rot85] G. Rote. A systolic array algorithm for the algebraic path problem. *Computing*, 34:191–219, 1985.