Synthèse d'ordonnancements parallèles par reproduction canonique

Claude Tadonki

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Synthèse d'Ordonnancements Parallèles par Reproduction Canonique.

Claude Tadonki

N°3996

Septembre 2000

THÈMES 1 et 4

ISSN 0249-6399 ISRN INRIA/RR--3996--FR

INRIA
RENNES

Synthèse d'Ordonnancements Parallèles par Reproduction Canonique.

Claude Tadonki *

Thèmes 1 et 4 — Réseaux et systèmes — Simulation et optimisation de systèmes complexes
Projets Aladin et Cosi

Rapport de recherche n ° 3996 — Septembre 2000 — 31 pages

Résumé : Dans cet article, nous proposons une méthodologie de conception d'ordonnancements parallèles réguliers. La technique s'applique aussi bien sur un système d'équations de récurrence que sur un graphe de dépendance. Le principe de base repose sur la possibilité de construire l'espace global de calcul à partir d'un sous-espace générique. Techniquement, il s'agit de partir d'un ordonnancement local de la struture générique pour construire un ordonnancement complet par des reproductions successives. De plus, cette construction se fait systématiquement dès lors que tous les paramètres ont pu être identifiés. Les exemples du *chemin algébrique*, de la *factorisation de Cholesky*, et du *produit de Kronecker* sont prsentés en guise d'illustration.

Mots-clé : programmation dynamique, équations de récurrence, ordonnancement, graphe, complexité.

(Abstract: pto)

* IRISA, Campus de Beaulieu, 35042, Rennes cedex, ctadonki@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Synthesis of Parallel Schedules by Canonical Reproduction.

Abstract: The paper provides an approach to design regular parallel schedules from a static specifications containing an internal geometric similarity. The method can be applied either on a system of recurrence equations under a syntaxical criterion, or on a directed graph after performing a canonical decomposition of its vertices. The key behind the idea is to focus on a basic substructure from which the overall calculation space can be constructed by its successive reproduction. The most technical advantage of the method is that it automatically produces a global schedule when given a local schedule of the basic substructure and required parameters for the reproduction. Some case studies are presented as an illustration.

Key-words: dynamical programming, recurrence equations, schedule, graph, complexity.

1 Introduction

D'une manière générale, trouver un ordre d'exécution des tâches et une répartition de celles-ci sur plusieurs processeurs constitue un des problèmes fondamentaux du parallélisme. A la base, le modèle considéré est celui des graphes orientés (éventuellement valués et pondérés). A la suite, le mode d'expression des calculs sous forme de systèmes d'équations récurrentes [8] a conduit à la naissance des approches analytiques, pouvant par conséquent faire l'objet de manipulations syntaxiques systématiques ou semi-automatiques [7, 14, 16, 18]. Toutefois, ces méthodologies fournissent en général une solution dont le partitionnement (en cas de ressources matérielles insuffisantes) est une opération supplémentaire, pouvant nécessiter une importante modification de la logique des processeurs, et même parfois conduire à une perte d'efficacité [17]. Ceci est sans doute dû au fait que dans la majeure partie des cas, l'on ne s'intéresse qu'à la nature des dépendances, et par ailleurs le problème de ressources quant à lui n'est analysée qu'à posteriori.

Nous proposons une méthode qui s'appuie le moins possible sur la nature des dépendances dans le graphe, et dérive systématiquement une ordonnancement dès lors que les hypothèses sur la structure globale du graphe sont satisfaites. De plus, la quantité de resources disponible peût être prise en compte plus tôt, et à défaut, l'algorithme obtenue peût être trivialement adaptatée pour fonctionner sur une surface réduite.

En effet, certains schémas de calculs peuvent se décomposer en une cascade de sous-structures similaires et localement dépendantes. Un paradigme naturel pour paralléliser ce type d'algorithme est le pipeline. Dans cette optique, le pipeline peut se faire soit au à l'intérieur de chaque subdivision, soit entre les subdivisions elles-mêmes. Dans cette deuxième approche qui correspond à celle qui nous proposons dans ce travail, l'ordonancement parallèle global est obtenu en appliquant une composition de transformations locales à partir d'une solution partielle. Il apparait alors que la qualité du résultat obtenu dépendra de l'adéquation entre l'ordonnancement générique et les paramètres de sa reproduction sur le graphe tout entier. Dans le cadre séquentiel, cette idée est bien développée dans le paradigme de la programmation dynamique. Par contre, dans le cas parallèle, à l'exception des schémas intrinsèquement réguliers (schémas uniformes ou affines), la synthèse d'algorithme se fait au cas par cas selon le problème considéré et par rapport au type d'architecture recherché ou encore en fonction de la complexité à atteindre.

Notre approche est donc générale et systématique. Par ailleurs, la nature de la solution obtenue met en évidence deux principaux avantages qui sont la tolérence aux pannes (par simple décalage), et le partitionnement direct (par des passes multiples).

Ajoutons à cele le fait que, la nature des dépendances n'étant pas un des facteurs de nos hypothèses, le champs d'action de notre méthode est est à priori beaucoup moins restrictif que d'habitude.

Le reste du papier est organisé comme suit. La section 2 présente la problème général de l'ordonnancement et quelques résultats fondamentaux. Dans la section 3, nous présentons le concept de reproduction canonique sous sa forme syntaxique, suivie en section par une description de la technique de synthèse à partir des équations récurrentes. Dans la section 3, nous définissions le critère de la reproduction canonique sous sa forme algébrique et décrivons la technique de synthse à partir d'un graphe de dépendance. La section 5 est consacrée à l'étude complète de quelques exemples et nous concluons en section 6.

2 Notions préliminaires sur l'ordonnancement

Il s'agit essentiellement des techniques qui permettent de construire des fonctions de temps et des stratégies d'allocation. Pour chacun de ses paramètres, il existe d'une part des résultats généraux qui donnent une idée de leur importance et de leur compexité, et d'autres parts des méthodes de détermination qui se distinguent les unes des autres par le contexte d'application et la qualité des résulats (Voir [11, 5])

Fonction de temps.
Considérons une représentation des tâches sous forme de graphe orienté $G = (X,A)$, où A correspond à la relation de précédence.

Définition 2.1 *Une fonction $t : X \to N$ définie une fonction de temps compatible avec le $G = (X,A)$ si elle vérifie la proprété*

$$\forall x,y \in X : [(x,y) \in A] \Longrightarrow [t(y) > t(x)]. \tag{1}$$

Avec une telle fonction, une tâche x est réalisée à l'instant $t(x)$. L'algorithme sera séquentiel si la fonction t est *injective*. Dans tous les cas, le temps global d'exécution, notée $\Theta(t,G)$, est donnée par

$$\Theta(t,G) = \max_{x \in X} t(x) - \min_{x \in X} t(x) + 1. \tag{2}$$

Etant donné qu'un graphe de précédence G n'a pas de cycle (*acyclique*), son *diamètre*, noté $\gamma(G)$, est bien définie et est égal à la longueur d'un plus long chemin dans G. On a alors le résultat suivant.

Théorème 2.1 *Si t est une fonction de temps définie sur un graphe $G = (X,A)$, on a*

$$\theta(t,G) \leq \gamma(G). \tag{3}$$

Cette borne est générale c'est à dire qu'elle reste valable même dans le cas des graphes valués et pondérés. Il existe des techniques bien connues qui permettent de construire des fonctions de temps qui atteignent cette borne, à l'exemple des méthodes du *chemin critique* [5].

Plusieurs méthodes ont été proposées dans la littérature pour construire des fonctions de temps. Pour la plupart, elles reposent sur des transformations analytiques directes (parfois à support géométrique), suivies éventuellement de la résolution des problèmes de programmation linéaire [8, 14, 12, 15] (voir [3, 19, 6] pour des notions sur la programmtion linéaire). Il existe aussi des méthodes combinatoires [5, 2].

Stratégie d'allocation.

Définition 2.2 *Soit $G = (X,V)$ un graphe orienté et t une fonction de temps comptaible avec G. Une fonction $A : X \rightarrow E$, où E est un ensemble discret quelconque, définie une fonction d'allocation sur G compatible avec la fonction de temps t, si elle vérifie la condition*

$$\forall x,y \in X,[t(x) = t(y)] \Longrightarrow [A(x) \neq A(y)]. \tag{4}$$

Le lecteur aura remarqué le lien entre la fonction d'allocation et la fonction de temps. En fait, ce lien traduit celui de la fonction d'allocation avec les dépendences entre les tâches dans une approche purement combinatoire. Bien que la condition (4) soit théoriquement suffisante, d'autres critères sont en prendre en compte de part leur implication sur la *localité*, la *régularité*, le coût global des *communications*, l'*équilibre des charges*, et la *modularité* de l'architecture induite.

Muni d'une fonction d'allocation A, la *surface* de l'algorithme, notée $S(A,G)$, est trivialement donnée par

$$S(A,G) = |A(X)|, \tag{5}$$

où $|.|$ est l'opérateur du cardinal d'un ensemble. Pour une fonction de temps donnée, il existe en général plusieurs fonctions d'allocation possibles. Un des problèmes classiques consiste donc à en rechercher une qui implique une surface minimale.

Pour ce qui est des communications, remarquons qu'elles interviennent chaque fois que l'on a deux tâches $x,y \in X$ telles que

$$[(x,y) \in V] \wedge [A(x) \neq A(y)]. \tag{6}$$

Dans ce cas, une communication est nécessaire pour transférer les données liées à x, de $A(x)$ vers $A(y)$. Cette communication doit intervenir à un instant situé $t(x)$ et $t(y)-1$, le choix du bon moment étant lui aussi un problème pratique important (parfois superflu dans un cadre théorique). Dans tous les cas, on peut consiérer que le volume des communications est donnée par

$$C(t,A,G) = |(x,y) \in V : [(x,y) \in V] \wedge [A(x) \neq A(y)]|. \tag{7}$$

Un autre problème combinatoire qui en découle est celui de la recherche d'une allocation A qui minimise $C(t,A,G)$.

On comprend donc que la recherche d'une allocation optimale est un problème combinatoire difficile [4, 21]. Le lecteur trouvera aussi des stratégies d'allocation dans les différentes références mentionnées dans la section précédente sur la fonction de temps.

D'une manière générale, le problème de l'ordonnancement parallèle optimal est un problème combinatoire difficile [5, 13]. Il peut être posé comme suit [4]:

(P1) Problème général de l'ordonnancement

Instance: Un ensemble T de n tâches, un ordre partiel $<$ sur T, des poids A_i, $1 \leq i \leq n$, m processeurs, et une borne de temps k.

Question: Peut-on trouver une fonction totale $\tau : T \rightarrow \{1,2,\cdots,k\}$ telle que:

(i) si $i < j$ alors $\tau(i) + A_i \leq \tau(j)$

(ii) $\forall i \in T$, $\tau(i) + A_i \leq k$

(iii) $\forall t, 1 \leq t \leq k$, $|i \in T, \tau(i) \leq t \leq \tau(i) + A_i| \leq m$.

Ce problème a été prouvé NP-complet [8]. Le problème reste NP-complet même dans les cas restreints suivants:

- $A_i = 1$, $i = 1,\cdots,n$ [21]
- $m = 2$, et $A_i \in \{1,2\}$, $i = 1,\cdots,n$ [21]

A coté de cela, il faudrait aussi noter l'existence des allocations dynamiques, techniques parfois intéressantes lorsqu'il faut faire face à des problèmes de pannes, ou lorsqu'on souhaiterait maintenir un certain équilibre des charges dans un contexte où très peu d'informations sont statiquement disponibles.

La technique que nous proposons repose sur le principe de la *reproduction* d'un ordonnancement partiel. Cette *reproduction* suppose la satisfaction d'une hypothèse qui se confond avec sa définition.

3 Equations récurrentes et reproduction canonique

3.1 Equations récurrentes

Définition 3.1 *Une **équation récurrente** définissant une fonction (ou une variable) X en tous les points z d'un domaine D, est une équation de la forme*

$$X(z) = D^X \quad : \quad g(\ldots X(f(z))\ldots) \tag{8}$$

où

- *z est une **variable d'index** de dimension n.*
- *X est une **variable de donnée** correspondant à une fonction de n arguments entiers; on parle de variable de dimension n.*
- *$f(z)$ est une **fonction de dépendance**, $f : \mathcal{Z}^n \rightarrow \mathcal{Z}^n$;*
- *g est une fonction scalaire; elle apparait souvent de manière implicite sous la forme d'une expression impliquant des opérandes de la forme $X(f(z))$, combinée avec des opérateurs de base et des parenthèses.*
- *D^X est une ensemble de points de $\mathcal{Z}^n$ et est appelé le **domaine** de l'équation. Les domaines sont très souvent des espaces d'index polyhédriques avec un ou plusieurs (notons ℓ) paramètres, $p \in \mathcal{Z}^\ell$.*

Une variable pourait être définie par plusieurs équations. Dans ce cas, on utilise la syntaxe ci-dessous:

$$X(z) = \left\{ \begin{array}{ccc} & \vdots & \\ D_i^X & : & g_i(\ldots X(f(z))\ldots) \\ & \vdots & \end{array} \right. \tag{9}$$

Chaque ligne correspond à un **cas**, et le domaine de X est une union des domaines disjoints de tous les cas, $D^X = \bigcup_i D_i^X$. On dire par ailleurs que la fonction de dépendance f est valable dans le (sous) domaine D_i^X.

Définition 3.2 *Une équation récurrente telle que définie par (8) est dite **affine** si chaque fonction de dépendance est de la forme $f(z) = Az + Bp + a$, où A (respectivement B) est une matrice constante $n \times n$ (respectivement $n \times l$) et a est une vecteur constant de dimensione n. L'équation récurrente est dite **uniforme** si chaque fonction de dépendance est de la forme $f(z) = z + a$, où a est un vecteur constant de dimension n appelé vecteur de dépendance.*

Définition 3.3 *Un* ***système*** *d'équations récurrentes* (SER) *est un ensemble de m équations récurrentes, définissant des variables de données $X_1, \cdots, X_m$, où chaque variable X_i est de dimension n_i. Puisque les équations du système sont mutuellement récursives, la fonction de dépendance f sera de type approprié.*

Domaine. Un important aspect du formalisme des équations récurrentes est la notion de *domaine*, ensemble d'indices sur lesquels des calculs particuliers sont définis. Ce domaine est d'habitude bien spécifié dans l'écriture d'un SER. Le domaine le plus souvent utilisé est un *polyhèdre* (ensemble de points entiers satisfaisant une nombre fini de contraintes linéaires d'(in)égalités), ou une réunion finie de polyhèdres.

Transformations de SER. Une des plus importantes manipulations que l'on peut effectuer sur une SER est l'opération de *réindexation* (aussi appelée *changement de base* ou *réarrangement spatio-temporelle*) de ses variables. La tranformation, notée $\mathcal{T}$, devrait admettre une *réciproque à gauche* pour tous les points du domaine de la variable. Lorsqu'elle appliquée à la variable X d'un SER défini comme suit:

$$X(z) \quad = \quad \left\{ \begin{array}{ccc} & \vdots & \\ D_i^X & : & g_i(\ldots Y(f(z))\ldots), \\ & \vdots & \end{array} \right.$$

le SER obtenu en appliquant les règles suivantes est équivalent au SER original:

- Remplacer chaque D_i^X par $\mathcal{T}(D_i^X)$, c'est à dire son image par $\mathcal{T}$.
- Dans la partie droite de l'équation définissant X, remplacer chaque dépendance f par $f \circ \mathcal{T}^{-1}$, c'est à dire la composition de f et $\mathcal{T}^{-1}$.
- Dans toutes les occurrences $X(g(z))$ de la partie droite de *chaque* équation, replacer la dépendance g par $\mathcal{T} \circ g$.

Pour le cas spécial des occurrences de X apparaissant dans la partie droite de l'équation définissant X, remplacer la dépendance f par $\mathcal{T} \circ f \circ \mathcal{T}^{-1}$.

3.2 Reproduction canonique

Définition 3.4 *Un système d'équations récurrentes sur $\mathcal{Z}^n$ est dit* **canoniquement reproductible** *s'il existe un sous-ensemble strict I de $\{1,...,n\}$, tel que pour toute fonction de dépendance $f = (f_1, \cdots, f_n)$, chacune des projections $f_i, i \in I$, ne dépend éventuellement que des composantes z_i, $i \in I$. Le sous-ensemble I est appelé*

direction *de reproduction, et son complémentaire* $\bar{I}$ *est quant à lui appelé* ***base*** *de reproduction .*

En d'autres termes, restreint au sous-espace de $\mathcal{Z}^n$ engendré par la famille $\{e_i, i \in I\}$ de la base canonique de $\mathcal{Z}^n$, le SER est *auto-dependant.*

Définition 3.5 *Une reproduction de direction* I *sera dite* **régulière**, *si on a aussi une reproduction de direction* $\bar{I}$. *En d'autres termes, pour toute fonction de dépendance* $f = (f_1, \cdots, f_n)$, *chacune des projections* $f_i, i \in I$ *(resp.* $i \in \bar{I}$*), ne dépend éventuellement que des composantes* z_i, $i \in I$ *(resp.* $i \in \bar{I}$*). Globalement, cela signifie que le* SER *considéré est auto-dependant dans chacune de ses restrictions aux sous-espaces supplémentaires engendrés respectivement par* $\{e_i, i \in I\}$ *et par* $\{e_i, i \in \bar{I}\}$.

Dans ces définitions, Il est à noter l'importance du fait que le sous-ensemble I doit être strict (c'est à dire $I \neq \emptyset \wedge I \neq \{1, \cdots, n\}$), autrement il repondrait trivialement au critère de reproductibilité sans être d'un quelconque intérêt.

Pour un SER *reproductible*, il peut exister plusieurs directions de reproduction, et il est facile de vérifier que l'ensemble des directions possibles est d'une certaine manière stable pour les opérations ensemblistes de base.

Propriétés 3.1.

(*i*) Toute intersection non vide de directions de reproduction est une direction de reproduction.

(*ii*) Toute réunion non pleine de directions de reproduction est une direction de reproduction.

Exemple 3.1 *Consiérons le cas des récurrences fictives suivantes :*

$$X(i,j,k) = g(\cdots X(i-j,k,k)\cdots) \quad (10)$$

$$X(i,j,k) = g(\cdots X(i-k,j-i,k-j)\cdots) \quad (11)$$

La relation (10) implique une reproduction de direction $I = \{2,3\}$, *car en posant* $\varphi(z) = z - f(z)$, *on a* $\varphi(i,j,k) = (j,j-k,0)$, *ce qui montre une auto-dependance du plan* (j,k). *Par contre la relation (11), qui correspond à* $\varphi(i,j,k) = (k,i,j)$, *est bloquante pour une reproduction canonique.*

Il serait naturel de se poser la question de savoir s'il existe des systèmes qui soient non canoniquement reproductibles et qui soient *calculables.* Le reflexe ayant conduit à cette question découle de l'entrelassement cyclique qu'on peut observer dans les

dépendances *bloquantes*. Nous n'étudierons pas cet aspect du problème qui peut aussi bien être évident qu'extrêmement complexe.

Exemple 3.2 *Considérons maintenant l'exemple du problème du chemin algébrique. On a le système d'équations récurrentes suivant [17].*

$$F(i,j,k) = \begin{cases} \{i,j,k \mid k=0\} & : a_{i,j} \\ \{i,j,k \mid i=j=k\} & : F(i,j,k-1)* \\ \{i,j,k \mid i=k\neq j\} & : F(k,k,k)\otimes F(i,j,k-1) \\ \{i,j,k \mid j=k\neq i\} & : F(i,j,k-1)\otimes F(k,k,k) \\ \{i,j,k \mid i\neq k; j\neq k\} & : F(i,j,k-1)\oplus \\ & \qquad (F(i,k,k)\otimes F(k,j,k-1)) \end{cases} \tag{12}$$

On a une reproduction unique de direction $I=\{3\}$. Une réindexation appropriée [10] permet d'obtenir une version dans laquelle toutes les occurrences de k ne se trouvant pas en troisième position sont remplacées par une constante. Ainsi, tout sous-ensemble strict de $\{1,2,3\}$ devient éligible pour une direction de reproduction. Ceci illustre le fait que la réindexation est une opération permettant d'augmenter le pouvoir de reproduction.

Définition 3.6 *Considérant une reproduction de direction I d'un* SER *de dimension n, l'entier $p=|I|$ (resp. $\frac{p}{n}$) sera appelé* ***indice absolu*** *(resp.* ***relatif****) de reproduction. Pour $p=1$, $p=2$, $p=3$, et $p>3$, on parlera respectivement de reproduction* ***linéaire****,* ***planaire****,* ***spatiale****, et* ***étendue****.*

Dans l'exemple 3.2, on a une reproduction linéaire. Les systèmes d'équations récurrentes affines ont un forte capacité de reproduction car tout sous-ensemble strict de $\{1,\cdots,n\}$ est une potentielle direction de reproduction.

L'intuition qui soutend le concept de *reproduction canonique* est la suivante. Partant d'un système d'équations récurrentes sur $\mathcal{Z}^n$ *canoniquement reproductible* et considérant une direction de reproduction I, si on a un ordonnancement du sous-systeme obtenu par projection du systeme entier sur le sous-espace de $\mathcal{Z}^n$ engendré par $\{e_i, i\in\bar{I}\}$, appelé *espace de base* (les composantes de rang dans I étant considérés ici comme des paramètres), alors on peut dériver un ordonnancement complet en reproduisant cet ordonnancement partiel le long du sous-espace de $\mathcal{Z}^n$ engendré par $\{e_i, i\in I\}$ (*espace de direction*). Selon que la reproductibilité est *régulière* ou non, la réplication sera *exacte* ou *biaisée*.

4 Synthèse à partir des équations récurrentes

4.1 Méthodologie

4.1.1 Fonction de temps

Considérons un SER canoniquement reproductible, et soit I une direction de reproduction. Soit $\mathcal{F}$ l'ensemble de ses fonctions de dépendances. On procède comme suit:

Etape 1. Partitionner $\mathcal{F}$ en deux classes $\mathcal{F}_b$ et $\mathcal{F}_d$ définies par

$$\mathcal{F}_b = \{f \in \mathcal{F} : f_{/<e_i, i\in I>} = id_{/<e_i, i\in I>}\} \quad (13)$$

$$\mathcal{F}_d = \mathcal{F} - \mathcal{F}_b \quad (14)$$

En d'autres termes, $\mathcal{F}_b$ représente le sous-ensemble des fonctions de dépendance dont la projection sur le sous-espace $< e_i, i \in I >$ se réduit à l'identité. Ce qui revient à dire que les dépendances de $\mathcal{F}_b$ n'ont d'influence que sur l'*espace de base*, tandis que celles de $\mathcal{F}_d$ régissent la précédence dans l'*espace de direction*.

Etape 2. Déterminer une fonction de temps valide, notée u, du SER projeté sur l'*espace de base*, sous les seules contraintes des dépendances de $\mathcal{F}_b$. Notons qu'à ce niveau, les occurences de composantes $z_i, i \in I$, peuvent intervenir mais plutôt comme des paramètres du sous-système. Ensuite, on en fait de même avec la projection du SER sur l'*espace de direction*, ce qui donne une fonction de temps que nous notons v (cette dernière fonction peut être constante dans le cas où $\mathcal{F}_d = \emptyset$).

Etape 3. Considérer une fonction de temps sur SER global sous la forme $t = u + \alpha v + \beta$, où β est une constante de réajustement, et α un coefficient strictement positif (pouvant dépendre des paramètres statiques du système global) qui permet d'introduire un délai entre une instance et sa reproduction. Ce délai à pour rôle de ne faire débuter une reproduction que lorsque les termes impliquées de l'instance précédente sont déjà mis à jour. Dans le cas d'un ordonnancement parallèle, $\alpha = 1$ dans le cas idéal, et $\alpha = \max u$ au pire.

Illustrons cette technique sur l'exemple 3.2, avec la direction $I = \{3\}$. On a $\mathcal{F}_b = \{(i,j) \leftarrow (i,k), (i,j) \leftarrow (k,k)\}$ et $\mathcal{F}_d = \{k \leftarrow k-1\}$. On peut prendre $u(i,j) = |i-k| + |j-k| + (n-k)\delta(i<k)$, où $\delta(i<k) = 1$ si $i < k$ et 0 sinon, et $v(k) = k$. La fonction globale sera alors de la forme $t(i,j,k) = |i-k| + |j-k| + (n-k)\delta(i <$

$k)+\alpha k+\beta$. Lorsque $i \geq k$ et $j \geq k$, la condition $t(i,j,k) \geq t(i,j,k-1)+1$ provenant de la dépendence $(i,j,k) \leftarrow (i,j,k-1)$ impose $\alpha \geq 3$. On vérifie aisément que la fonction $t(i,j,k) = |i-k| + |j-k| + (n-k)\delta(i<k) + 3k - 2$ convient. On obtient donc un algorithme qui s'exécute en $5n-4$ cycles de calcul sur n^2 processeurs, complexité caractérisant la majeur partie des solutions rencontrées dans la littérature (voir l'introduction de [17]).

4.1.2 Fonction d'allocation

Etant donné p processeurs, on procède comme suit. On considère une factorisation non triviale $p = ab$. Ensuite, on alloue les points de la base à b processeurs, et ceux de la direction à a processeurs. Ces allocations partielles A_b et A_d sont supposées valides par rapport aux fonctions de temps u et v, et de plus, l'allocation sur la direction A_d doit être **injective**. L'allocation globale est alors donnée par

$$A = (A_b, A_d). \tag{15}$$

L'ordonnancement partiel défini par (u,A_b) est appelé *ordonnancement générique*, et celui défini par (v,A_d) est appelé *ordonnancement de progression*. Dans l'exemple 2.1, on peut prendre $A_b(i,j) = j$ et $A_d(k) = k$, soit $A(i,j,k) = (j,k)$. Le résultat qui suit établit la validité de l'ordonnancement obtenu par notre approche.

Théorème 4.1 *Soit (u,A_b) est un ordonnancement générique et (v,A_d) un ordonnancement de progression. Soient ensuite α et β, deux scalaires tels que $t = u+\alpha v+\beta$ définit une fonction de temps globale valide. Alors $(t,A = (A_b,A_a))$ est un ordonnancement valide pour le* SER *considéré.*

Preuve. Soient x et y, deux points de $\mathcal{Z}^n$ tels que $t(x) = t(y)$. Deux cas sont envisageables.

• x et y ont la même projection sur l'*espace de direction* ($x_i = y_i$, $i \in I$). Ceci implique qu'on a $v(x) = v(y)$, et par suite $u(x) = u(y)$. Etant donné que x et y ont des projections distinctes sur l'*espace de base* (autrement on aurait $x = y$), $u(x) = u(y)$ implique $A_b(x) \neq A_b(y)$ et par suite $A(x) \neq A(y)$.

• Les projections de x et y sur l'*espace de direction* sont distinctes. Du fait que A_d est injective, on a $A_d(x) \neq A_d(y)$ et par suite $A(x) \neq A(y)$. □

A propos du partitionnement

Puisqu'on a imposé que A_d soit injective, il faudrait donc que a (nombre de processeurs sur une direction) soit égal au volume w de l'espace de direction. Si tel n'est pas le cas et si a est un diviseur de w, partant de l'allocation A_d définie en supposant une quantité non bornée de processeurs, on dérive l'adaptation $\widetilde{A}_d$ ainsi qu'il suit. Considérons sans nuire à la généralité que la direction est $I = \{1, \cdots, q\}$. On factorise a en $a = a_1 \cdots a_q$ tel que a_i divise $w_i, i = 1, \cdots, q$, où w_i est le volume de la projection sur $\mathcal{Z} < e_i >$. L'allocation $\widetilde{A}_d$ est donc donnée par

$$\widetilde{A}_d(x_1, \cdots, x_q) = A_d(x_1 \bmod a_1, \cdots, x_q \bmod a_q) \tag{16}$$

Cette adaptation triviale constitue un très grand avantage des ordonnancements par reproduction (surtout lorsque la reproduction est régulière), car on a une bonne souplesse sur l'exigence en ressources matérielles. Toutefois, elle nécessite un aménagement de la fonction de temps afin d'éviter des conflits dûs à la réutilisation de l'architecture. Une façon directe de résoudre ce problème consiste à achever un passage (mise à jour de a espaces de base consécutifs suivant la direction), avant d'entamer le passage suivant. Mais, dans certains cas, un enchaînement immédiat ou légèrement retardée est possible, avec pour conséquence une importante amélioration du temps total d'exécution. Dans l'exemple 3.2, on aurait besoin de $n \times n = n^2$ processeurs, soit n dans le sens de la direction. Si on avait $n \times m$ processeurs, où m est un diviseur de n, on aurait considéré l'allocation de direction $A_d(k) = k \bmod m$ $(0 \leq k \leq n-1)$, et la fonction de temps globale deviendrait $\widetilde{t} = t + 2n(kdivm)$ ($2n$ est le le nombre de cycles nécessaires à la mise à jour d'un espace de base, et k *div* m représente le rang du passage courant). Dans ce cas précis, un enchaînement retardé d'une constante est possible, cette étude est laissée au soin du lecteur motivé. Précisons tout de même qu'avec $m = n/3$ (soit $n^2/3$ processeurs), la fonction de temps n'a pas besoin d'être modifiée car l'exécution de $n/3$ reproductions consécutives nécessite $2n + 3 \times n/3 = 3n$ cycles, délai suffisant pour entamer les prochains calculs.

4.2 Complexité

Après avoir présenté le concept de *reproduction canonique* et décrit une méthodologie de synthèse, nous allons maintenant analyser la complexité des ordonnancements et faire ressortir les facteurs importants. Notons premièrement que l'obtention des ordonnancements de base et de direction peut se faire de manière ad hoc ou par l'usage des méthodologies appropriées de synthèse automatique. Toutefois, il faut faire des choix adéquats si l'on veut avoir une bonne complexité car, comme

nous allons le préciser, l'efficacité globale dépend d'une bonne coopération entre les fonctions de temps partielles u et v.

Définition 4.1 *Considérons un* SER *canoniquement reproductible, I une direction de reproduction, et v une fonction de temps de progression; pour $x \in \mathcal{Z}^n$, on considère $\mathcal{P}(x) = \{z \in Zn : v(z) = v(x)$. Pour u une fonction de temps générique vérifiant $\min u = 1$, et α un coefficient tel que $t = u + \alpha v$ est une fonction de temps valide, on définit la charge d'un point $x \in \mathcal{Z}^n$, notée $c(x)$ par*

$$c(x) = \max_{z \in \mathcal{P}(x)} u(z), \tag{17}$$

et pour deux points x et y tels que $v(y) - v(x) = 1$, on définit la latence de reproduction $\lambda(x,y)$ par

$$\lambda(x,y) = (\alpha + c(y) - c(x))\delta(\alpha + c(y) - c(x) > 0). \tag{18}$$

Enfin, on définit le ratio de reproduction $\rho(x,y)$ par

$$\rho(x,y) = \frac{\lambda(x,y)}{c(x)} \tag{19}$$

Ce ratio permet d'apprécier l'efficacité du pipeline. Plus il est petit, plus le pipeline est quantitativement bénéfique. Ceci vient du fait que, la fonction d'allocation de progression étant initialement injective, il vaut mieux qu'un volume considérable de calculs sur une base restent à être effectués lorsque ceux de la base suivante sont entamés. De plus, les valeurs d'une base doivent être calculées de manière à ravitailler le plus immediatement possible les calculs de la base suivante (à défaut, il y aura soit des cycles d'inactivité pour synchroniser, soit un besoin de mémoire pour les stockages intermédiaires). Notons par ailleurs que c'est cet aspect qui implique éventuellement un certain délai entre deux passes consécutives dans la version partitionnée, mais dans ce cas, un gain d'efficacité est envisageable à cause d'une réduction des cycles d'inactivité intervenant lors du premier et du dernier pipeline. Lorsque $\rho(x,y) = 0$, le coût des calculs dans $\mathcal{P}(y)$ est recouverts par celui dans $\mathcal{P}(x)$. Dans l'exemple 3.2, notre ordonnancement fournit un ratio égal à $\rho = \frac{3}{2n}$, ratio qui est évidemment satisfaisant. Dans le cas d'une reproduction non régulière, ce facteur peut varier en fonction des points choisit, on définit alors naturellement le *ratio minimal* ρ_{min}, le *ratio maximal* ρ_{max} et le *ratio moyen* ρ_{moy}. En supposant que v est réajusté de sorte que $\min v = 1$, on définit la *longueur de reproduction* γ par $\gamma = \max v$. De plus, pour

k, $1 \leq k \leq \gamma$, on définit $\mathcal{P}_k = \{x : v(x) = k\}$ et $c_k = \max_{x \in \mathcal{P}_k} u(x)$. Le résultat suivant donne un encadrement du temps total d'exécution de l'ordonnancement.

Théorème 4.2 *Si $c_1 \leq c_2 \leq \cdots \leq c_\gamma$, alors le temps total d'exécution $T_{//}$ de l'ordonnancement canonique vérifie,*

$$c_1(1 + \rho_{min}\gamma) \leq T_{//} \leq c_\gamma(1 + \rho_{max}\gamma) \tag{20}$$

Preuve. On a

$$T_{//} = c_\gamma + \sum_{k=1}^{\gamma-1} \lambda(x_k,x_{k+1}), \quad x_k \in \mathcal{P}_k \tag{21}$$

Du fait que $\lambda(x_k,x_{k+1}) = c(x_k)\rho(x_k,x_{k+1})$, on a

$$T_{//} = c_\gamma + \sum_{k=1}^{\gamma-1} c_k\rho(x_k,x_{k+1}), \quad x_k \in \mathcal{P}_k \tag{22}$$

On obtient le résultat en appliquant à (22) les encadrements $c_1 \leq c_k \leq c_\gamma$, $k = 1, \cdots, \gamma$ et $\rho_{min} \leq \rho(x_k,x_{k+1}) \leq \rho_{max}$. □

Remarquons que si on consière la condition duale $c_1 \geq c_2 \geq \cdots \geq c_\gamma$, on aura l'encadrement

$$c_\gamma(1 + \rho_{min}\gamma) \leq T_{//} \leq c_1(1 + \rho_{max}\gamma)$$

Dans notre exemple d'accompagnement, on a les ordres de grandeur suivants: $\gamma = n$, $c_1 = c_n = 2n$, $\rho_{min} = \frac{2}{2n}$, $\rho_{max} = \frac{4}{2n}$, ce qui donne $4n \leq T_{//} \leq 6n$, soit $5n$ en moyenne. Notons que $4n$ est la meilleure complexité actuelle des ordonnancements pipelinés pour le problème du *chemin algébrique*.

Notons que la condition $c_1 \leq c_2 \leq \cdots \leq c_\gamma$ ou $c_1 \geq c_2 \geq \cdots \geq c_\gamma$ correspond au mieux à la situation $c_1 = c_2 = \cdots = c_\gamma$, à une constante additive près. Un tel équilibre de charge assurerait une bonne régularité de l'ordonnancement. Dans tous les cas, il est clair que minimiser ρ_{max} conduirait à un temps total d'exécution meilleur. Ceci revient à rechercher parmi les fonctions temps u impliquant des charges équivalentes, celles pour lesquelles le coefficient α est minimal, ou celles qui réalisent un entrelassement dont l'effet de retardement est moins important.

4.3 Construction d'un ordonnancement de base efficace

Soit $\mathcal{B}$ (resp. $\mathcal{D}$) l'*espace de base* (resp. *de direction*) d'un SER. Etant donné une fonction de temps de progression v supposée injective, on procède comme suit pour lui associer une fonction de temps u de manière à obtenir un enchainement efficace.

• Définir une fonction $\phi : \mathcal{D} \rightarrow \mathcal{D}$ telle que $v(x) = v(\phi(x)) + 1$, $x \in \mathcal{D}$. Cette fonction existera toujours dans la mesure où v est injective et atteint toutes les valeurs de l'intervalle [1,γ], où $\gamma = \max_{x \in \mathcal{D}} v(x)$. En effet on $\phi(x) = v^{-1}(v(x) - 1)$. Dans l'exemple 3.2, on a $\phi(k) = k - 1$. La fonction ϕ modélise le mécanisme de progression. Pour $z \in \mathcal{B}$, on définit $\phi(z)$ comme étant l'ensemble des points obtenus de z en faisant subir respectivement à chacune de ses composantes liée à $\mathcal{D}$, l'action induite par ϕ. Dans l'exemple 3.2, on a $\phi(i,j) = \{(i-1,j),(i,j-1)\}$ à cause des influences $(i,j) \leftarrow (i,k)$ et $(i,j) \leftarrow (k,k)$, et de $\phi(k) = k - 1$. On obtient donc un système d'équations récurrentes internes àLa projection du SER sur $\mathcal{B}$ devient donc auto-dépendante.

• Ordonnancer le système ainsi obtenu avec les dépendances crées par la manipulation précédente ($(i,j) \leftarrow (i-1,j)$ et $(i,j) \leftarrow (i,j-1)$ dans l'exemple). Notons que si on a p processeurs, on doit tenir compte du fait qu'on n'en aura que $\frac{p}{\gamma}$ pour effectuer l'ordonnancement dans $\mathcal{D}$. Si on ne tient pas compte des ressources matérielles, on pourrait alors considérer les fonctions de temps au plus tôt ou au plus tard, ou toute autre fonction valide. Il convient tout de même de noter que, même si les variables de $\mathcal{D}$ n'apparaissent plus dans les dépendances, elles peuvent agir comme paramètres d'espace. Dans notre exemple, on aura:

$$\left\{ \begin{array}{lcl} (i,j) \leftarrow (i-1,j) & : & (i \neq k) \\ (i,j) \leftarrow (i,j-1) & : & (j \neq k) \end{array} \right. \tag{23}$$

où les opérations sont faites modulo n (i.e $(1,j) \leftarrow (n,j)$ et $(i,1) \leftarrow (i,n)$). On pourrait donc prendre pour ce cas la fonction

$$u(i,j) = |i-k| + |j-k| + [2(i-k)+n]\delta(i<k) + [2(j-k)+n]\delta(j<k), \tag{24}$$

pour laquelle on a $\rho_{max} = \frac{3}{2n}$. En prenant en compte la combinaison $v(k) = k$ et $\alpha = 3$, on a un temps d'exécution global de l'ordre de $5n$ avec n^2 processeurs.

L'idée dans sous-jacente à cette construction est de reproduire le mécanisme de progression sur la base, de façon à ravitailler au mieux le pipeline. Ce mécanisme peut être totalement ou partiellement pris en compte en fonction du type d'architecture recherchée. En effet, plus on prend en compte des dépendances, plus les calculs sont sérialisés et on tend presque vers une situation où l'architecture de base découle trivialement du graphe de dépendances. Par contre, lorsqu'on omet certaines des dépendances créées (au profit bien sûr d'une plus grande liberté temporelle), une analyse spéciale devient indispensable, car une vraie dépendance pourrait être perdue. Dans (23), si on omet la dépendance $(i,j) \leftarrow (i-1,j)$, on perd la vraie dépendance $(i,j) \leftarrow (k,j) : i \neq k$, et il faudra par conséquent faire attention dans la construction de la fonction de temps générique.

4.4 Dérivation de la version par bloc

Une application directe de notre méthode conduit dans un premier temps à une solution à grains fins, ce qui implique des communications portant sur des données élémentaires. Or, dans le cadre des machines à mémoire distribuée, de tels ordonnancements conduiraient à des algorithmes inefficaces, à cause de la latence qui intervient à chaque appel d'une routine de communication. Une méthode bien connue pour venir à bout de ce problème est le regroupement en blocs des points de calculs [9]. De la sorte, toutes opérations de calcul et de communication s'effectuent sur des blocs de données, limitant ainsi l'effet pénalisant de la latence. Toutefois, cette adaptation est une activité combinatoire généralement difficile car un regroupement des données créé des dépendances d'ensemble pouvant agir sur l'enchainement des tâches.

Dans le cas des ordonnancements canoniques, il suffit d'effectuer cette adaptation sur la base $\mathcal{B}$ par une approche de type LSGP (localement séquentiel et globalement parallèle). La nature de l'ordonnancement global reste donc conservée. Toutefois, il convient de noter le besoin d'une mémoire nécessaire au stockage et à la manipulation des groupes de données (ce qui n'est pas un problème puisqu'on est dans la cas des machines à mémoire locale).

5 Décomposition canonique et Synthèse à partir du graphe de dépendances

Considérons un graphe de tâche $G = (T,P)$, où T est l'ensemble des tâches et P la relation de précédence.

Définition 5.1 *Un graphe de tâche $G = (T,P)$ sera dit canoniquement reproductible s'il existe une partition $T = T_1 \cup T_2 \cup \cdots \cup T_\gamma$ telle que les sous-graphes induits par T_k et T_{k+1}, $1 \leq k < \gamma$, sont isomorphes (i.e. il existe une bijection $\phi_k : T_{k+1} \rightarrow T_k$ telle que pour tout $x,y \in T_{k+1}$, $(x,y) \in P \Longleftrightarrow (\phi_k(x),\phi_k(y)) \in P$), et de plus, toutes les dépendances directes sont internes aux T_k ou localisées entre T_k et T_{k+1} dans le sens de T_k vers T_{k+1} (i.e. $\forall (x,y) \in T^2 : (x,y) \in P \Rightarrow [x,y \in T_k) \vee (x \in T_k \wedge y \in T_{k+1})$.*

Dans notre exemple du chemin algébrique, on peut considérer

$$T_k = \{(i,j,k) : 1 \leq i,j \leq n\}, k = 1, \cdots, n, \tag{25}$$

avec les isomorphismes ϕ_k définies par:

$$\phi_k(i,j,k) = \begin{cases} (1,1,k+1) & : \ i = n \wedge j = n \\ (1,j+1,k+1) & : \ i = n \wedge j < n \\ (i+1,1,k+1) & : \ i < n \wedge j = n \\ (i+1,j+1,k+1) & : \ i,j < n \end{cases} \tag{26}$$

Notons au passage que de tels isomorphismes peuvent guider la recherche d'une *réindexation* du SER, de manière à le rendre par exemple affine. Par ailleurs, il est clair que pour un graphe donné, il peut exister plusieurs décompositions possibles. De plus, le fait d'impliquer des isomorphismes découle d'un souci de régularité et d'équilibre dans les calculs. On comprend alors qu'on puisse envisager des ordonnacements des T_k ayant des complexités équivalantes à une constante additive près. Toutefois, on pourrait dans certains cas se contenter des applications injectives (si $|T_{k+1}| < |T_k|$). Revennons sur la construction de l'ordonnancement.

On définit $\pi_k : T_k \to T_1$ par $\pi_k = \phi_1 o \phi_2 o \cdots o \phi_{k-2} o \phi_{k-1}$. Soit (u,A_b) un ordonnancement de T_1, où u est une fonction de temps et A_b une fonction d'allocation. On considère sur chaque $T_k, k > 1$, la fonction de temps relative $u_k = u o \pi_k$. Soit maintenant $\{v_k, k = 1, \cdots, k-1\}$, une famille de termes tels que v_k de dépend que de k et éventuellement des paramètres statiques du graphes, qui vérifie

$$\forall (x,y) \in T_k \times T_{k+1} : (x,y) \in P \Rightarrow u_{k+1}(y) - u_k(x) + v_{k+1} > 0. \tag{27}$$

On définit alors une fonction de temps globale $t : T \to \mathcal{N}$ comme suit:

$$T_k : t(z) = u(\pi_k(z)) + v(k), \tag{28}$$

où $v(k) = v_k + v_{k-1} + \cdots + v_1$.

Théorème 5.1 *L'expression (28) définit un fonction de temps valide pour le graphe* $G = (T,P)$.

Preuve. Soient $x,y \in T$ tels que $(x,y) \in P$. Montrons que $t(y) > t(x)$. En effet, d'après les propriétés du graphe G, on a deux cas à envisager:

- $x,y \in T_k$, $1 \leq k \leq \gamma$.

D'après la définition de π_k, on a $(\pi_k(x),\pi_k(y)) \in P$ et par conséquent $u(\pi_k(y)) > u(\pi_k(x))$ car $\pi_k(x),\pi_k(y) \in T_1$ et u est un timing valide de T_1. Puisque $x,y \in T_k$, cette dernière inégalité conduit au résultat en ajoutant l'expression $v(k)$ à chacun de ses membres.

- $x \in T_k$ et $y \in T_{k+1}$. D'après (28), on a

$$\begin{aligned} t(y) - t(x) &= (u_{k+1}(y) + v(k+1)) - (u_k(x) + v(k)) \\ &= u_{k+1}(y) - u_k(x) + (v(k+1) - v(k)) \\ &= u_{k+1}(y) - u_k(x) + v_{k+1} > 0 \end{aligned} \tag{29}$$

Ceci achève la preuve du résultat. □

La fonction d'allocation globale $A : T \to T$ quant à elle donnée par

$$T_k : A(x) = (k,A_b(\pi_k(x))). \tag{30}$$

Théorème 5.2 *Les fonctions de temps et d'allocation définies respectivement par (28) et (30) forment un ordonnancement global valide du graphe $G = (T,P)$.*

Preuve. Soient $x,y \in T$ tels que $A(x) = A(y)$. Montrons que $t(x) \neq t(y)$. En effet, $A(x) = A(y)$ implique qu'il existe $k \in \{1,\cdots,\gamma\}$ tel que $x,y \in T_k$. On a ensuite $A_b(\pi_k(x)) = A_b(\pi_k(y))$, qui implique $u(\pi_k(x)) \neq u(\pi_k(x))$ car $\pi_k(x),\pi_k(y) \in T_1$ et (u,A_b) est un ordonnancement valide de T_1. En ajoutant $v(k)$ dans les deux membres de cette dernière inégalité, on obtient $t(x) \neq t(y)$, ce qui achève la démonstration. □

Dans notre, exemple, l'isomorphisme défini par (26) donne

$$\pi_k^{-1}(i,j,k) = \begin{cases} (i-k+1+n,j-k+1+n,1) & : \; i-k<0 \wedge j-k<0 \\ (i-k+1+n,j-k+1,1) & : \; i-k<0 \wedge j-k\geq 0 \\ (i-k,j-k+n,1) & : \; i-k\geq 0 \wedge j-k<0 \\ (i-k,j-k,1) & : \; i-k\geq 0 \wedge j-k\geq 0 \end{cases} \tag{31}$$

Reste maintenant à trouver un ordonnancement générique (u,A_b) de T_1. Pour cela, on considère l'ordonnancement suivant:

$$u(i,j,1) = i + j - 1 \tag{32}$$

$$A_b(i,j,1) = j \tag{33}$$

La fonction d'allocation globale sera

$$A(i,j,k) = (k,j). \tag{34}$$

Pour la fonction de temps, on doit d'abord déterminer u_k et v_k. D'après la définition $u_k = uo\pi_k$, on a

$$u_k(i,j,k) = \begin{cases} u(i-k+1+n,j-k+1+n,1) & : \; i-k<0 \wedge j-k<0 \\ u(i-k+1+n,j-k+1,1) & : \; i-k<0 \wedge j-k\geq 0 \\ u(i-k+1,j-k+1+n,1) & : \; i-k\geq 0 \wedge j-k<0 \\ u(i-k+1,j-k+1,1) & : \; i-k\geq 0 \wedge j-k\geq 0 \end{cases} \tag{35}$$

soit

$$u_k(i,j,k) = \begin{cases} (i-k+1+n)+(j-k+1+n)-1 & : \; i-k<0 \wedge j-k<0 \\ (i-k+1+n)+(j-k+1)-1 & : \; i-k<0 \wedge j-k\geq 0 \\ (i-k+1)+(j-k+1+n)-1 & : \; i-k\geq 0 \wedge j-k<0 \\ (i-k+1)+(j-k+1)-1 & : \; i-k,j-k\geq 0 \end{cases} \tag{36}$$

Cette expression à la forme compact suivante:

$$u_k(i,j,k) = i+j-2k+1+n[\delta(i-k<0)+\delta(j-k<0)]. \tag{37}$$

En remarquant que $u_{k+1}(i,j,k+1) - u_k(i,j,k) \geq 2$, il vient que que $(v_1 = 0) \wedge (v_k = 3 : k > 1)$ satisfait la condition (27). Tout ceci conduit à la fonction de temps globale donnée par

$$t(i,j,k) = i+j-2k+1+n[\delta(i-k<0)+\delta(j-k<0)]+3(k-1), \tag{38}$$

soit tout simplement

$$t(i,j,k) = i+j+k-2+n[\delta(i-k<0)+\delta(j-k<0)]. \tag{39}$$

On obtient un algorithme qui s'exécute en $t(n-1,n-1,n) = 5n-4$ étapes sur n^2 processeurs.

Dans les cas particuliers de la fermeture transitive et des plus courts chemins, du fait qu'on est affranchi du traitement des lignes $i = k$ et $j = k$, on peut prendre l'ordonnancemnt générique suivant:

$$u'(i,j,1) = \begin{cases} n+2-(i+j) : i+j \leq n+1 \\ 2n-(i+j)+2 : i+j > n+1 \end{cases} \tag{40}$$

$$A'_b(i,j,1) = j \tag{41}$$

Remarquons qu'on peut écrire

$$u'(i,j,1) = n+2-(i+j)+n\delta(i+j>n+1), \tag{42}$$

et par suite $u'(i,j,1) = n+3-u(i,j,1)+n\delta(u(i,j,1) > n)$. On obtient donc $u'_k(i,j,k) = n+3-u_k(i,j,k)+n\delta(u_k(i,j,k) > n)$, et l'ordonnancement recherché est donné par

$$t'(i,j,k) = n + 3k - u_k(i,j,k) + n\delta(u_k(i,j,k) > n) \quad (43)$$

$$A'(i,j,k) = (k,j) \quad (44)$$

On obtient un algorithme qui s'exécute en $t(1,1,n) = 4n-3$ étapes sur n^2 processeurs. Ceci illustre l'importance de l'ordonnancement générique sur la performance globale.

Mieux encore, lorsqu'on observe que les calculs de T_1 sont achevés après n cycles (soit $3n$ cycles d'inactivités par la suite), il vient qu'on pourait avoir le même temps d'exécution en utilisant $n/3$ groupes de processeurs pour le calculs des T_k. Cette nouvelle solution, qui n'est qu'une application du partitionnement par passes multiples telle que expliqué précédemment, conduit à un ordonnancement qui s'exécute en $4n-3$ étapes sur $n \times n/3 = n^2/3$ processeurs. Cette dernière solution est spécifiée par les fonctions suivantes:

$$t'(i,j,k) = n + 3k - u_k(i,j,k) + n\delta(u_k(i,j,k) > n) \quad (45)$$

$$A'(i,j,k) = ([(k-1) \bmod \frac{n}{3}] + 1, j) \quad (46)$$

5.1 Etude combinatoire de la décomposition canonique du graphe

Remarquons que la méthode précédente n'a pas nécessairement besoin d'un isomorphisme entre T_{k+1} et T_k. En effet, il nous suffit d'avoir

$$\text{Pour } x,y \in T_{k+1}, (x,y) \in P \Rightarrow (\phi_k(x),\phi_k(y)) \in P. \quad (47)$$

Ceci allège les critères à vérifier par la décomposition du graphe sans pour autant influer sur la technique de construction de l'ordonnancemment. Toutefois, l'isomorphisme garantit que la reproduction de l'ordonnancement de T_1 sur les autres classes $T_k : k > 1$ aura une complexité relative aussi liée que possible aux dépendances dans T_k qu'elle ne l'aura été dans T_1. En effet, on pourrait avoir moins de dépendances dans T_{k+1} que dans T_k auquel cas, reproduire le timing de T_k sur T_{k+1} reviendrait en quelque sorte à prendre compte dans T_{k+1} des dépendances virtuelles (transportées de T_k vers T_{k+1} par la correspondance). Corriger une telle situation conduirait à une perte de modularité de l'ordonnancement global. Aussi, nous allons poursuivre notre analyse avec la condition (47), avec en prime la liberté d'avoir $|T_k| \geq |T_{k+1}|$ au lieu de l'égalité. Enfin, notons aussi que si on a des dépendances (x,y) entre T_k

et $T_{k+\lambda}$, $\lambda > 1$, on peut tout simplement la supprimer et la remplacer par la cascade de dépendances

$$(x,x_{k+1}),\{(x_t,x_{t+1}) : t = k+1,\cdots,k+\lambda-1\},(x_{k+\lambda-1},y), \tag{48}$$

où $x_{k+1},\cdots,x_{k+\lambda-1}$ sont des sommets de $T_{k+1},\cdots,T_{k+\lambda-1}$ respectivements, choisis de façon judicieuse.

Nous aimerions rappeler une fois de plus que les remarques précédentes visent à simplifier l'étude de l'existence et la construction d'une partition pouvant servir de point de départ de la méthode. Toutefois, le cas idéal reste celui de la définition d'origine avec ses avantages de régularité, d'équilibre et de modularité.

La méthode requiert donc au minimum une partition $T = T_1 \cup T_2 \cup \cdots \cup T_\gamma$ avec des applications injectives $\varphi_k : T_{k+1} \to T_k$ telles que:
(i) Pour $x,y \in T_{k+1}, (x,y) \in P \Rightarrow (\varphi_k(x),\varphi_k(y)) \in P$,
(ii) $(x,y) \in P \Rightarrow \exists i,j; j \geq i : (x \in T_i) \wedge (y \in T_j)$.

Le résultat suivant montre la difficulté générale de la décomposition dans sa forme originale.

Théorème 5.3 *Soient $G_1 = (X_1,\Gamma_1)$ et $G_2 = (X_2,\Gamma_2)$ deux graphes orientés tels que $|X_1| = |X_2|$. G_1 et G_2 sont isomorphes si et seulement si le graphe orienté défini par $G = (X_1 \cup X_2, \Gamma_1 \cup \Gamma_2 \cup (X_1 \times X_2))$ admet une décomposition canonique de longueur 2.*

Preuve.
• G_1 et G_2 sont isomorphes. Il suffit de prendre $T_1 = X_1$ et $T_2 = X_2$ pour avoir la décomposition canonique d'ordre 2.
• $G = (X_1 \cup X_2, \Gamma_1 \cup \Gamma_2 \cup (X_1 \times X_2))$ admet une décomposition canonique T_1 et T_2. Supposons que T_1 soit différent de X_1. Ceci implique qu'on a $T_1 \cap X_2 \neq \emptyset$ et $T_2 \cap X_1 \neq \emptyset$. $X_1 \times X_2 \subset \Gamma$ implique qu'on a un arc de T_2 vers T_1, ce qui n'est pas conforme à la définition de la décomposition canonique. □

Bien que la décomposition canonique est en général un problème combinatoire apparament difficile comme le montre le théorème 5.3, la donnée du graphe sous forme de système d'équations récurrentes sur des domaines polyhédriques simplifie les mécanismes. En effet, la décomposition recherchée devient le résultat de transformations analytiques comme nous allons le voir dans les exemples qui vont suivre.

6 Applications

6.1 La factorisation de Cholesky

Considérons le problème de la factorisation de Cholesky qui, pour une matrice carrée A d'ordre n symétrique définie positive, consiste à construire une matrice triangulaire inférieure $L = (l_{ij}),1 \leq j \leq i \leq n$ telle que $A = LL^T$. Les équations récurrentes correspondantes [1] sont données par:

$$l(i,j,k) = \begin{cases} D \cap \{(k=-1) \wedge (j \leq i)\} & : a_{i,j} \\ D \cap \{i=j=k+1\} & : l(i,j,k-1)^{\frac{1}{2}} \\ D \cap \{(j=k+1) \wedge (i>j)\} & : l(i,j,k-1)/l(k+1,k+1,k) \\ D \cap \{k \leq j-2\} & : l(i,j,k-1) - \\ & \quad l(i,k+1,k) \times l(j,k+1,k) \end{cases} \quad (49)$$

où $D = \{(i,j,k) : -1 \leq k < j \leq i \leq n\}$.

a) Synthèse directe

On a une direction de reproduction donnée par $I = \{3\}$, ce qui nous donne les classes de dépendances $\mathcal{F}_b = \{(i,j) \leftarrow (i,k+1),(i,j) \leftarrow (j,k+1),(i,j) \leftarrow (k+1,k+1)\}$ et $\mathcal{F}_d = \{k \leftarrow k-1\}$. En prenant $v(k) = k$ et en appliquant la technique décrite dans la section 4.3, on obtient $\phi(i,j) = \{(i,j-1),(k+1,k+1)\}$ ($\phi(k) = k-1$). Ce qui produit les dépendances suivantes:

$$\begin{cases} (i,j) \leftarrow (k+1,k+1) & : \quad j \geq k+1 \\ (i,j) \leftarrow (i,j-1) & : \quad j \geq k+2 \end{cases} \quad (50)$$

On peut prendre $u(i,j) = i + j - 2k - 1$, et on a alors une fonction de temps de la forme $t(i,j,k) = i + j + \alpha k + \beta$ (les termes $2k$ et -1 ont été immergés dans la partie $\alpha k + \beta$). La prise en compte de la dépendance $(i,j,k) \leftarrow (i,j,k-1)$ conduit à la condition $\alpha \geq 1$. En prenant donc $\alpha = 1$, on a la fonction de temps globale suivante

$$t(i,j,k) = i + j + k - 1. \quad (51)$$

Pour les allocations, on propose $A_b(i,j) = (j-k)\delta(i+j \leq n) + (i-k)\delta(i+j > n)$ et $A_d(k) = k$, ce qui donne

$$A(i,j,k) = ((j-k)\delta(i+j \leq n) + (i-k)\delta(i+j > n), \quad k). \quad (52)$$

On a donc un ordonnananancement qui fonctionne en $t(n,n,n-1) = 3n-2$ étapes sur $n(n-1)/4$ processeurs, soit un travail dans l'ordre de $\frac{3}{4}n^3$.

b) Synthèse à partir du graphe de dépendance
Par rapport à la précédente description, on a

$$T_k = \{(i,j,k) \mid k < j \leq i \leq n\} : 0 \leq k \leq n-1 \tag{53}$$
$$\varphi_k(i,j,k) = (i-1,j-1,k-1) \tag{54}$$

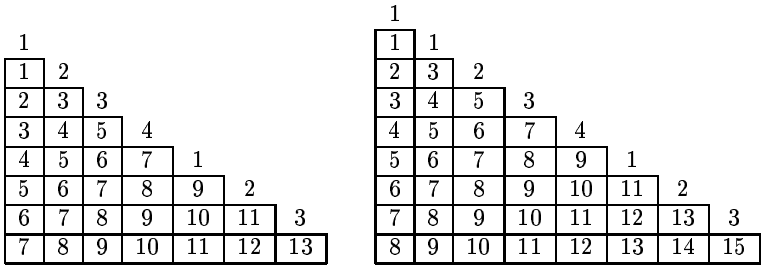
Cherchons un ordonnancement de T_0. Ceci revient à considérer le schéma de récurrence suivant sur le domaine $D = \{(i,j) : 1 \leq j \leq i \leq n\}$:

$$l(i,j) \quad = \quad \begin{cases} D \cap \{i = j = 1\} & : \sqrt{a_{ij}} \\ D \cap \{(i \geq 2) \wedge (j = 1)\} & : a_{ij}/l(j,j) \\ D \cap \{j \geq 2\} & : a_{ij} - l(i,1) \times l(j,1) \end{cases} \tag{55}$$

Un ordonnancement possible est donné par

$$\begin{cases} u(i,j) = i + j - 1 \\ A_b(i,j) = j\delta(j \leq \lceil \frac{n}{2} \rceil) + (j - \lceil \frac{n}{2} \rceil)\delta(j > \lfloor \frac{n}{2} \rfloor) \end{cases} \tag{56}$$

La figure Fig. 1 illustre l'ordonnancement générique pour les cas $n = 7$ et $n = 8$. Les charges sont réparties en colonnes de manière cyclique sur $n/2$ processeurs. Continuons notre synthèse en déterminant $u_k = uo\pi_k$ et v_k. Sachant que $\pi_k =$



Note: La valeur au dessus de chaque colonne représente le processeur concerné.

FIG. 1 – *Ordonnancement de T_0 pour les cas $n = 7$ et $n = 8$.*

$\varphi_0 o \cdots o \varphi_{k-1}$, on a

$$\pi_k(i,j,k) = (i-k,j-k,0). \tag{57}$$

La relation $u_k(i,j,k) = u(i-k,j-k,0)$ nous donne

$$u_k(i,j,k) = (i-k) + (j-k) - 1 = i + j - 2k - 1 \tag{58}$$

En étudiant la quantité $u_k(i,j,k) - u_{k-1}(i,j,k-1)$, on obtient la condition $v_k \geq 3$. Nous prendrons donc $v_k = 3$ ($v_0 = 0$), soit $v(k) = 3k$. En posant $i_k = i-k$, $j_k = j-k$ et $n_k = n-k$, on a l'ordonnancement global donné par

$$\begin{cases} t(i,j,k) = i + j + k - 1 \\ A(i,j,k) = (\ j_k \delta(j_k \leq \lceil \frac{n_k}{2} \rceil) + (j_k - \lceil \frac{n_k}{2} \rceil)\delta(j_k > \lceil \frac{n_k}{2} \rceil),\ k+1\ ) \end{cases} \tag{59}$$

L'algorithme s'exécute en $t(n,n,n-1) = 3n-2$ cycles avec $\frac{n^2}{4} + O(n)$ processeurs. Le timing est le même celui obtenu dans la version précédente, mais l'allocation est légèrement modifée. La figure Fig. 2 Ordonnancement global pour le cas $n = 8$: 22 cycles avec 20 processeurs. 2 illustre l'ordonnancement complet pour le cas d'une matrice d'ordre $n = 8$. En observant cet ordonnancement, on constate que le premier processeur (1,1) est libre au moment où débutent les calculs de $T_{\frac{n}{2}}$ (on peut le montrer facilement par les formules). Ceci nous amène à consdérer l'allocation adaptée

$$A'(i,j,k) = (\ j_k \delta(j_k \leq \lceil \frac{n_k}{2} \rceil) + (j_k - \lceil \frac{n_k}{2} \rceil)\delta(j_k > \lceil \frac{n_k}{2} \rceil),\ k \bmod [\frac{n}{2}] + 1\ ) \tag{60}$$

grâce à laquelle on n'aura besoin que d'environ $\frac{n^2}{8} + O(n)$ processeurs, soit un travail de l'ordre de $\frac{3}{8}n^3$ et par suite une efficaité égale à $\frac{4}{9}$ (La complexité séquentielle est de l'ordre de $\frac{n^3}{6}$). La figure Fig. 3 illustre l'architecture basée sur l'allocation originale pour $n = 8$.

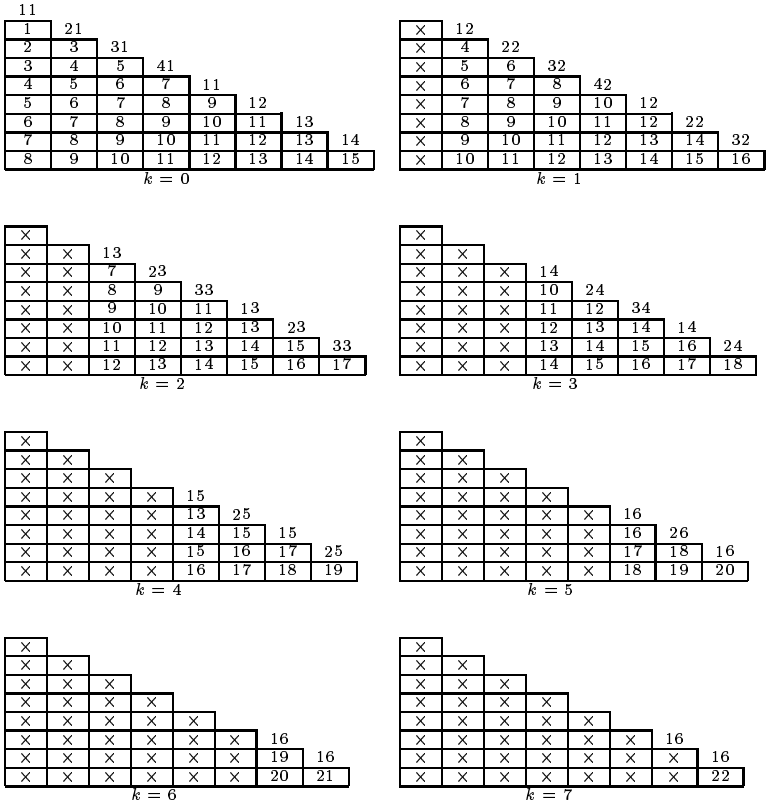
Cette figure montre la modularité du réseau pour des valeurs de n ayant la même parité. Au besoin, on pourrait factoriser la matrice augmentée exprimée par

$$\begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}.$$

6.2 Le produit de kronecker

Définition 6.1 *Pour $A \in R^{n_A \times m_A}$ et $B \in R^{n_B \times m_B}$, le produit de Kronecker (aussi appelé produit tensoriel) de A par B, noté $A \otimes B$, appartient à $R^{n_A n_B \times m_A m_B}$ et est défini par la structure en bloc suivante:*

$$(A \otimes B)_{ij} = a_{ij} B \in R^{n_B \times m_B}$$



ij=processeur de rang i du groupe j.

FIG. 2 – *Ordonnancement global pour le cas* $n = 8$*:* 22 *cycles avec* 20 *processeurs.*

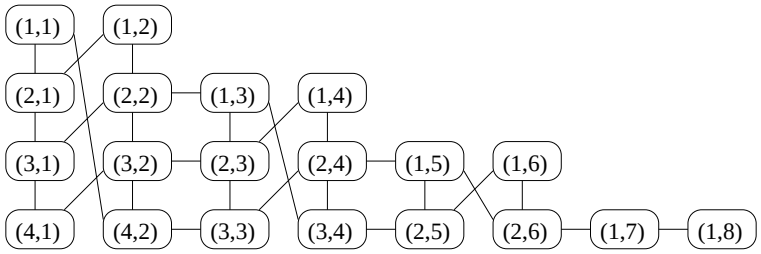


FIG. 3 – *Topologie pour la factorisation de Cholesky avec* $n = 8$.

Exemple 6.1

$$\mathcal{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad \mathcal{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Le produit de Kronecker $C = A \otimes B$ *est donné par*

$$\mathcal{C} = \left(\begin{array}{cc|cc|cc} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \\ \hline a_{31}b_{11} & a_{31}b_{12} & a_{32}b_{11} & a_{32}b_{12} & a_{33}b_{11} & a_{33}b_{12} \\ a_{31}b_{21} & a_{31}b_{22} & a_{32}b_{21} & a_{32}b_{22} & a_{33}b_{21} & a_{33}b_{22} \end{array}\right)$$

Ce produit étant associatif, on définit naturellement le *produit de Kronecker* N matrices $A^{(i)}$, $1 \leq i \leq N$, noté $\otimes_{i=1}^{N} A^{(i)}$.

Nous allons nous intéresser au produit d'un vecteur par une matrice implicitement définie comme étant le produit de Kronecker de plusieurs matrices [20]. Formellement, nous voulons calculer

$$z = x \otimes_{i=1}^{N} A^{(i)}. \tag{61}$$

Il est connu que ce produit peut sa calculer avec une complexité donnée par

$$\rho_N = n_N \times (\rho_{N-1} + \prod_{i=1}^{N} n_i) = (\prod_{i=1}^{N} n_i)(\sum_{i=1}^{N} n_i). \tag{62}$$

Cette complexité (en terme de multiplications flottantes) est obtenue en utilisant la factorisation canonique de la matrice principale, de sorte qu'on évite aussi de la former[20]. Par ailleurs, si les matrices $A^{(i)}$ sont carrées de taille $n_i : i = 1,\cdots,N$, les vecteurs à manipuler seront de longueur $n_1 \times n_2 \times \cdots \times n_N$, qu'on peut indexer par des séquences de la forme $(i_1,i_2,\cdots,i_N)$, avec $1 \leq i_s \leq n_s$, $s = 1,\cdots,N$. Sous ce formalisme, il est établi que notre calcul peut être effectué par l'algorithme suivant [20].

$v^{(N+1)} := x$
for $s := N$ **downto** 1 **do**
  for $(i_1,...,i_N) := (1,...,1)$ **to** $(n_1,...,n_N)$ **do**
    $v^{(s)}(i_1,...,i_{s-1},i_s,i_{s+1},...,i_N) := \sum_{t=1}^{n_s} A^{(s)}(t,i_s)v^{(s+1)}(i_1,...,i_{s-1},t,i_{s+1},...,i_N)$
  end do
end do
$z = v^{(1)}$

Algo 1: Algorithme du produit de Kronecker

Si on suppose (par des soucis de régularité) que les matrices $A^{(i)}$ ont toutes la même taille n (ce cas est fréquent dans les problèmes qui se modélisent avec le produit de Kronecker), on dérive le système d'équations récurrentes suivant sur le domaine $D = \{(i_1,...,i_N,s,t),\ 1 \leq s \leq N+1,\ 0 \leq t \leq n,\ 1 \leq i_p \leq n\}$:

$$v(i_1,\cdots,i_N,s,t) = \begin{cases} D \cup \{s = N+1\} & : x(i_1,...,i_N) \\ D \cup \{s \leq N \wedge t = 0\} & : 0 \\ D \cup \{s \leq N \wedge t > 0\} & : v(i_1,...,i_N,s,t-1)+ \\ & \quad + A^{(s)}(t,i_s)v(i_1,...,i_{s-1},t,i_{s+1},...,i_N,s+1,n) \end{cases} \tag{63}$$

a) Synthèse directe

On a une direction de reproduction donnée par $I = \{N+1,N+2\}$ (les deux dernières composantes). Ce qui donne $\mathcal{F}_b = \emptyset$ et $\mathcal{F}_d = \{(s,t) \leftarrow (s,t-1),(s,t) \leftarrow (s+1,n)\}$. Pour l'ordonnancement de direction, on peut prendre $v(s,t) = (N-s)n+t$ et $A_d(s,t) = 1$. Pour l'ordonnancement de base, étant donné qu'il n'y a pas de dépendance, on peut prendre $u(i_1,\cdots,i_N) = 1$ et $A_b(i_1,\cdots,i_N) = (i_1,\cdots,i_N)$. On a alors l'ordonnancement suivant

$$\begin{cases} t(i_1,\cdots,i_N,s,t) = (N-s)n+t \\ A(i_1,\cdots,i_N,s,t) = (i_1,\cdots,i_N) \end{cases} \tag{64}$$

qui calcule la solution en Nn cycles avec n^N processeurs, soit un travail optimal égal à Nn^{N+1}. Rappelons un fois de plus la souplesse du partitionnement. En effet, si on

dispose d'un nombre de processeurs p, diviseur de n^N, il suffit d'effectuer la décomposition $p = p_1 p_2 \cdots p_N$ telle que chaque p_i divise n, et de considérer l'ordonnancement (toujours de travail optimal) donné par

$$\begin{cases} t(i_1,\cdots,i_N,s,t) = [(N-s)n+t]\frac{n^N}{p} \\ A(i_1,\cdots,i_N,s,t) = ((i_1-1) \bmod p_1 + 1,\cdots,(i_N-1) \bmod p_N + 1) \end{cases} \quad (65)$$

Cette solution correspond à un réseau en hypercube.

b) Synthèse à partir du graphe de précédence
On considère $T_k = \{(i_1,\cdots,i_N,N-k+1,t) : 1 \leq i_p,t \leq N\}$ et $\varphi_k(i_1,\cdots,i_N,s,t) = (i_1,\cdots,i_N,s+1,t)$, $s = N-k+1$. On a donc $\pi_k(i_1,\cdots,i_N,s,t) = (i_1,\cdots,i_N,N,t)$. Cherchons un ordonnancement générique de $T_1 = \{(i_1,\cdots,i_N,N,t) : 1 \leq i_p,t \leq N\}$. On peut prendre $u(i_1,\cdots,i_N,N,t) = (i_1-1)w_1 + \cdots + (i_N-1)w_N + 1$ avec $w_k = n^{N-k}$ (u correspond à ordre lexicographique) et $A_b(i_1,\cdots,i_N,N,t) = t$. On a $u_k(i_1,\cdots,i_N,s,t) = (i_1-1)w_1 + \cdots + (i_N-1)w_N + 1$ et $v_k = w_{s-1} - w_s + 1$, soit $v(k) = w_{s-1} + k - 1$ ($s = N-k+1$). L'ordonnancement obtenu est

$$\begin{cases} t(i_1,\cdots,i_N,s,t) = (i_1-1)w_1 + \cdots + (i_N-1)w_N + w_{N-k} + N - s + 1 \\ A(i_1,\cdots,i_N,s,t) = (s,t) \end{cases} \quad (66)$$

qui calcule la solution en $2n^N+1$ cycles avec Nn processeurs, soit un travail de l'ordre de $2Nn^{N+1}$ (un facteur 2 de l'optimal). Le réseau obtenu est une réseau orthogonal.

7 Conclusion

Nous avons proposé, illustré et analysé une méthodologie de conception d'ordonnancements réguliers. Cette méthodologie s'avère assez efficace pour des schémas de calculs de type programmation dynamique. Les solutions synthétisées ont une bonne modularité, aussi bien au niveau de l'architecture qu'au niveau du flot de données. Toutefois, il reste à approfondir l'aspect physique des réseaux dérivés, principalement l'interconnexion entre les groupes de processeurs dans les enchaînements, et la complexité de la mémoire dans la version partitionnée.

Remerciements. Ce travail a bénéficié du soutien de l'agence française *Aire Développement* dans le cadre du projet *Calcul Parallèle*, et des projets ALADIN et COSI de l'IRISA (Rennes).

Références

[1] P. Clauss, *Synthèse d'Algorithmes Systoliques et Implantation Optimale en place sur Réseaux de Processeurs Synchrones*, Thèse de Doctorat de l'Université de Franche-Comte, 1990.

[2] P. Chretienne and C. Picouleau, *The Basic Scheduling Problem with Interprocessor Communication Delays*, RR MASI, 91-06, 1991.

[3] G. B. Dantzig, *Linear Programmimg and Extensions*, Princeton University Press, 1963.

[4] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice Hall, 1994.

[5] R. Faure, C. Roucairol, et P. Tolla, *Chemins et flots, ordonnancements*, Gauthier-Villard, 1976.

[6] P. Feautrier, *Parametric integer Programming*, RAIRO: Recherche Opérationnelle, 22:243-268, , 1988.

[7] J. A. B. Fortes, K. S. Fu, and B. W. Wah, *Systematic approaches to the design of algorithmic specified systolic arrays*, in Proc. IEEE/ICASSP, Tampa,Mar. 26-29, 1985.

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

[9] K. Högstedt, L. Carter, and J. Ferrante, *Selecting tile shape for minimal execution time*, Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'99, pp. 201-211, May 1987.

[10] S. Y. Kung, S. C. Lo, and P. S. Lewis, An optimal systolic design for the transitive closure and the shortest path problems. *IEEE Transactions on Computers*, C-36(5):603–614, May 1987.

[11] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys, *Séquencing and Scheduling Algorithms and Complexity*, Report BS-R8909 CWI, K.M. Van Hee, and H.G. Sol, Eindhoven University of Technology, 1989.

[12] C. Mongenet, P. Clauss, and G. R. Perrin, *Geometrical tools to map systems of affine recurrence equations on regular arrays*, LIB Report, pp. 30-90, 1990.

[13] C. Picouleau, *Two new NP-Complete Scheduling Problems with Communication Delays and Unlimited Number of Processors*, RR IBP-MASI,91-24, 1991.

[14] Patrice Quinton, *Automatic synthesis of systolic arrays from uniform recurrent equations*, in Proc. 11th Annu. Symp. Comput. Architecture, pp. 208-214, 1984.

[15] S. V. Rajopadhye, *Synthesis, Verification and Optimization of Systolic Arrays*, PhD Thesis, The University of Utah, Deptartment of Computer Science, December 1986.

[16] Sanjay V. Rajopadhye and Richard M. Fujimoto, *Synthetizing systolic arrays from recurrence equations*, Parallel Computing **14** pp. 163-189, June 1990.

[17] Sanjay Rajopadhye, Claude Tadonki, and Tanguy Risset, The algebraic path problem revisited. In *Lecture Note in Computer Science: EuroPar'99 Parallel Processing*, Springer-Verlag, No. 1685, pages 698–707, August 1999.

[18] Ibrahima Sakho, and Maurice Tchuente, *Une méthodologie de conception d'algorithmes systoliques pour réseaux réguliers.*, Technique et Science Informatique, 1989.

[19] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience series in Discrete Mathematics, John Wiley and Sons, 1986.

[20] C. Tadonki and B. Philippe, *Parallel Multiplication of a Vector by a Kronecker Product of Matrices*, Journal of Parallel and Distributed Computing and Practices, To appear, 1999.

[21] J. Ullman, *NP-complete scheduling problems*, Journal of Computer Sciences, 10, 384-393, 1975.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399