- DRAFT - - DRAFT - - DRAFT -

EXTENSIONS DE QUELQUES HYPOTHÈSES APPLICATIVES

TSI-00-30

23 octobre 2000

Nicolas MUSEUX

École Nationale Supérieure des Mines de Paris
Centre de Recherche en Informatique

Fontainebleau 77305
museux@cri.ensmp.fr

Thomson-CSF / Laboratoire Central de Recherches
Groupe Informatique
Laboratoire Architectures et Technologies Systèmes
Orsay 91404
nicolas.museux@lcr.thomson-csf.com

Chapitre 1

Introduction

La thèse de Christophe Guettier ([Gue97]) avait pour objectif principal de montrer la faisabilité du placement [1] d'applications de traitement de signal systématique, sur des architectures multi-processeurs homogènes, en utilisant la programmation par contraintes et la modélisation concurrente. De ce fait, les applications traitées et les architectures considérées étaient simples et idéales [2].

La thèse en cours ayant pour ambition de repousser les limites de ce travail, les applications et les architectures prises en compte sont un peu plus complexes. Le benchmark utilisé, permettant d'aller plus loin, est le mode radar MFR [Aul99] [Mus00]. L'architecture cible est quand à elle un PSP-RBE2 NG simplifié. Ces deux éléments sont fourmis par Thomson-CSF/DETEXIS.

Les différences entre le mode MFR et les applications traitées dans [Gue97] (comme, par exemple, la **V**eille **B**ande **L**arge) sont remarquables. En effet, la VBL est une chaîne (*i.e.*, une séquence) de tâches alors que l'algorithme du mode MFR est un graphe [3]. Une autre différence est que les fonctions d'accès aux données deviennent affines et ne sont plus seulement linéaires. Par ailleurs, certaines tâches du mode MFR font des accès distincts à un même tableau. C'est-à-dire qu'un tableau possède plusieurs fonctions d'accès pour un même calcul. Il y a aussi l'apparition de traitements conditionnels dans les tâches élémentaires. Enfin, certains nids de boucles ont des bornes inférieures non nulles, ou encore des bornes paramétrées.

Certaines de ces différences ont fait l'objet d'une étude poussée afin d'être intégrées dans les modèles (passage d'une séquence à un graphe de tâche, fonction d'accès affines). Les autres font l'objet de pré-traitements (*i.e.*, réécritures aboutissant à des approximations) afin d'obtenir une première série d'exprérimentations.

Ce document regroupe les différentes réfexions que nous avons pu avoir sur ces sujets. Ainsi le chapitre 2 est consacré au passage d'une chaîne à un graphe de tâches. Le chapitre 3 reprend le problème de la fonction d'accès affine aux données. Enfin, le chapitre 4 est dédié aux pré-traitements effectués sur la spécification de l'application, permettant ainsi de retrouver un cadre hypothétique favorable.

1. Au sens de Thomson-CSF, c'est-à-dire en considérant la distribution des données, le partitionnement des espaces d'itérations des calculs et leur ordonnancement dans le temps sous contraintes de ressources.
2. Ce qui ne veut pas dire pour autant que les problèmes à résoudre l'étaient également, bien au contraire.
3. Le mode MFR ne compte pas moins d'une trentaine de tâches contre *seulement* sept pour la VBL. Ce surplus de complexité va permettre également de tester la robustesse de l'approche contrainte.

Chapitre 2

D'une séquence de tâches vers un graphe de tâches acyclique

Sommaire

2.1 Introduction **6**
2.2 Définitions **6**
2.3 D'une chaîne à un graphe : l'ordonnancement **7**
2.4 Conditions sur la fonction de numérotation **8**
2.5 Quel ordre choisir ? **8**
2.5.1 Parcours de graphe *(en profondeur, en largeur)* 9
2.5.2 Selon la position d'une tâche dans le graphe (*tri topologique 1*) 9
2.5.3 Version relachée (*tri topologique 2*) 9
2.5.4 Le choix 10
2.6 Et le modèle de latence ? **11**
2.7 Conclusion **12**

2.1 Introduction

Dans [Gue97], la première hypothèse posée concernant le graphe de tâches d'une application de traitement de signal systématique est qu'il soit acyclique. Cependant, afin de simplifier l'ensemble de la modélisation (qui n'était alors pas encore définie), l'ensemble des graphes acycliques considérés a été réduit aux chaînes. Dès lors, une hypothèse forte est de dire que *deux tâches dépendantes sont consécutives*. Cette hypothèse intervient dans la linéarisation des nids de boucles partitionnés, l'ordonnancement, les communications, la mémoire ou encore certains pré-traitements tels que la fusion triviale de nids de boucles. Cependant, l'impact n'est pas direct pour tous les modèles évoqués à l'instant. Seuls l'ordonnancement (dans un contexte SIMD) et la latence sont impliqués directement. On pourrait croire qu'il en va de même pour les dépendances, mais il suffit de changer le point de vue du problème. C'est-à-dire qu'il ne faut pas aborder le problème en parlant de *tâches dépendantes* mais de *tableaux sur lesquels s'éxerce une dépendance*. La notion de graphe ou de chaîne disparait alors.

Nous ne considérerons ici que les modèles d'ordonnancement et de latence. Nous proposons un principe permettant de supprimer la restriction aux séquences de tâches, à moindre coût pour le modèle d'ordonnancement établi. Ensuite nous expliquerons en quoi cela intervient dans le modèle de la latence.

2.2 Définitions

L'ensemble des tâches appartenant au graphe est divisé en trois sous-ensembles. Chacun des ces sous-ensembles est défini ci-dessous.

Définition 2.2.1 : L'ensemble des tâches appartenant au graphe de tâches est noté Γ.

Définition 2.2.2 : Soient deux tâches $T^w \in \Gamma$ et $T^r \in \Gamma$. L'existence d'une dépendance entre T^w et T^r telle que T^w produit des données consommées par T^r sera notée :

$$D(T^w, T^r)$$

Définition 2.2.3 : On dira d'une tâche $T \in \Gamma$ qu'elle est **initiale** si et seulement s'il n'existe pas de tâche T^w produisant des données consommées par T dans le graphe de tâches.

$$(T \text{ est initiale }) \Leftrightarrow (\nexists\, T^w \in \Gamma, D(T^w, T))$$

L'ensemble des tâches initiales est noté Γ^s et on a $\Gamma^s \subset \Gamma$.

Définition 2.2.4 : On dira d'une tâche $T \in \Gamma$ qu'elle est **terminale** si et seulement s'il n'existe pas de tâche T^r consommant des données produites par T dans le graphe de tâches.

$$(T \text{ est terminale }) \Leftrightarrow (\nexists T^r \in \Gamma, D(T, T^r))$$

L'ensemble des tâches terminales est noté Γ^e et on a $\Gamma^e \subset \Gamma$.

Remarque :

- De ces définitions nous déduisons[1] aisément que :

$$(card(\Gamma) > 1) \Leftrightarrow (\Gamma^s \cap \Gamma^e = \emptyset)$$

- Dans le cas où le graphe de tâches se réduit à une et une seule tâche, on a :

$$\Gamma = \Gamma^s = \Gamma^e$$

2.3 D'une chaîne à un graphe : l'ordonnancement

La fonction d'ordonnancement d'une tâche $T \in \Gamma$ ([Gue97, Chap. 7]) est une fonction affine du paramètre temporel c^T issu du partitionnement 3D de l'espace d'itérations de la dite tâche ([Gue97, Chap. 4]). C'est-à-dire $\alpha^T.c^T + \beta^T$. Les paramètres α^T et β^T associés à la tâche T sont également des inconnues du problème (c^T et α^T sont des vecteurs dont la dimension est égale à la profondeur du nid de boucles correspondant, et β^T est un scalaire).

Dans le cadre d'une architecture SIMD, cette fonction d'ordonnancement doit être injective. En effet, dans un tel contexte, deux calculs différents ne peuvent pas être effectués en même temps. L'artifice utilisé pour s'en assurer est de définir pour chaque nid de boucles partitionné une fonction d'ordonnancement locale en γ/δ tels que :

$$\begin{array}{rcl} \alpha^T & = & card\,(\Gamma)\,.\gamma^T \\ \beta^T & = & card\,(\Gamma)\,.\delta^T + k^T \end{array}$$

k^T est le numéro attribué à la tâche T.

L'attribution d'un numéro à une tâche est arbitraire. De manière générale (et naturelle) cette numérotation est dictée par la succession des tâches les unes par rapport aux autres en suivant le sens des flêches.

Dans le cas où le graphe de tâches est une chaîne, la numérotation correspond exactement à la séquence. Ainsi le paramètre k^T est déterminé dès le début. Cependant, dans le cas d'un graphe acyclique quelconque, il arrive que deux chemins dans le graphe soient parallèles[2]. La décision de l'attribution d'un numéro à une tâche devient alors totalement arbitraire (cf. Fig.2.1). Comme ce numéro k^T intervient dans la fonction d'ordonnancement, il n'est pas possible de décider de sa valeur sans favoriser du même coup la recherche vers une partie de l'espace des solutions. Il convient donc de rendre ce paramètre variable et de laisser le système décider de sa valeur.

Cependant, le fait de laisser le choix au système sans autres indications ne peut qu'augmenter la combinatoire. Il faut alors la couper au mieux. C'est dans cet optique que nous avons cherché des méthodes de numérotation (largement inspirées par la théorie des graphes [BBC92]).

1. Rappelons que le graphe est orienté et sans circuit (par abus de langage nous le dirons *acyclique*).
2. Nous dirons que deux chemins dans un graphe orienté sont parallèles s'ils ne partagent au plus que deux sommets : le sommet de départ et/ou d'arrivée des dits chemins.

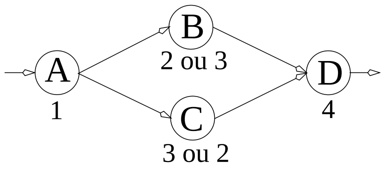


FIG. 2.1 – *Chemins parallèles dans un graphe : plusieurs possibilités de numérotation.*

2.4 Conditions sur la fonction de numérotation

Tout d'abord, la fonction de numérotation se doit d'être bijective, c'est-à-dire que toutes les tâches sont numérotées, à chaque numéro est associée une unique tâche et deux tâches différentes ont des numéros différents. Ainsi, si l'on note $\mathcal{N}$ cette fonction et $\mathcal{D}_k$ le domaine de numérotation, on a :

$$\forall T \in \Gamma, \exists k^T \in \mathcal{D}_k, \mathcal{N}(T) = k^T \tag{2.1}$$

$$\forall k^T \in \mathcal{D}_k, \exists T \in \Gamma, \mathcal{N}^{-1}(k^T) = T \tag{2.2}$$

$$\forall \left(T^1, T^2\right) \in \Gamma \times \Gamma, \exists \left(k^{T^1}, k^{T^2}\right) \in \mathcal{D}_k^2 \ / \ \left(k^{T^1} \neq k^{T^2}\right) \Leftrightarrow \left(T^1 \neq T^2\right) \tag{2.3}$$

Il reste encore à définir $\mathcal{D}_k$.

Le premier calcul de la première tâche marque l'origine de l'axe temporel d'exécution. Il est, par conséquent, effectué à l'instant logique 0. La fonction d'ordonnancement étant $\alpha^T.c^T + \beta^T = card\,(\Gamma) \times \left(\gamma^T.c^T + \delta^T\right) + k^T$, cela signifie que le numéro de la première tâche doit être 0. En effet, le premier calcul d'une tâche T a lieu au temps logique correspondant à $c^T = 0$, c'est-à-dire à $\beta^T = card\,(\Gamma) \times \delta^T + k^T$. Si T est la première tâche, on a alors $\beta^T = card\,(\Gamma) \times \delta^T + k^T = 0$. Or δ^T et k^T étant par définition des entiers positifs et $card\,(\Gamma)$ étant strictement positif, on en déduit que :

$$\left(card\,(\Gamma) \times \delta^T + k^T = 0\right) \Leftrightarrow \left\{ \begin{array}{ccc} \delta^T & = & 0 \\ & \wedge & \\ k^T & = & 0 \end{array} \right.$$

Ainsi nous définissons l'intervalle de numérotation $\mathcal{D}_k$:

$$\mathcal{D}_k = [0, card\,(\Gamma) - 1]$$

La fonction bijective de numérotation est ainsi définie par :

$$\begin{array}{rccl} \mathcal{N}: & \Gamma & \longrightarrow & \mathcal{D}_k = [0, card\,(\Gamma) - 1] \\ & T & \longmapsto & k^T \end{array}$$

Cependant, nous n'avons toujours pas choisi **comment** numéroter les tâches du graphes.

2.5 Quel ordre choisir ?

Dans la suite nous présentons trois façons d'*ordonner* le graphe :

1. parcours de graphe (*en profondeur et en largeur*).
2. en fonction de la position respective des tâches dans le graphe (*tri topologique 1*) ;
3. et une version relachée de la précédente (*tri topologique 2*);

2.5.1 Parcours de graphe *(en profondeur, en largeur)*

La première idée que l'on a pour numéroter les tâches du graphe est de suivre un chemin et de numéroter les tâches qui ne le sont pas déjà. En fait, il s'agit des algorithmes de parcours de graphe traditionnels dits *en largeur* ou *en profondeur*. La figure 2.2.c illustre un parcours en profondeur d'un graphe.

Pour que ce soit le système qui décide de la numérotation et, surtout, pour qu'il soit capable de remettre en cause sa décision, il est nécessaire que l'algorithme de parcours soit exprimé à l'aide de contraintes. Or, cette mise en contraintes n'est pas triviale et demande beaucoup (trop) d'efforts.

De plus, le résultat obtenu après ce type de parcours n'est pas très intéressant et ne préfigure pas l'ordre donné par les précédences.

2.5.2 Selon la position d'une tâche dans le graphe (*tri topologique 1*)

Le tri topologique des nœuds d'un graphe orienté sans circuit consiste à déterminer une extension linéaire d'un ordre partiel établi sur ces nœuds.

Dans le cadre de notre graphe de tâches, l'ordre partiel découle des précédences.

$$\forall (T^w, T^r) \in \Gamma \times \Gamma, \exists \left(k^{T^w}, k^{T^r}\right) \in \mathcal{D}_k \times \mathcal{D}_k, \mathcal{N}(T^w) = k^{T^w}, \mathcal{N}(T^r) = k^{T^r} / \ (D(T^w, T^r)) \Rightarrow \left(k^{T^w} < k^{T^r}\right)$$

L'extension linéaire définissant le tri topologique est laissé au choix du système. Cependant, afin de limiter le choix des valeurs, nous définissons un intervalle de valeurs possibles pour chacune des tâches du graphe respectant l'ensemble des conditions présentées ci-dessus.

Appelons $\mathcal{A}^T$ l'ensemble des tâches placées dans le graphe avant la tâche T et $\mathcal{P}^T$ l'ensemble des tâches situées sur des chemins parallèles à celui sur lequel est T et n'étant pas dans $\mathcal{A}^T$. Le numéro de T sera alors dans l'intervalle $\left[Card(\mathcal{A}^T), Card(\mathcal{A}^T) + Card(\mathcal{P}^T)\right]$.

Dans l'exemple de la figure 2.2.b, considérons le sommets F. On a alors $\mathcal{A}^F = [A, C, D]$ et $\mathcal{P}^F = [B, E]$. Ainsi l'intervalle des valeurs possibles pour numéroter F est
$\left[Card(\mathcal{A}^F), Card(\mathcal{A}^F) + Card(\mathcal{P}^F)\right] = [3, 5]$.

Si le sommet traité T se trouve sur un chemin pour lequel $\mathcal{P}^T = \emptyset$ alors son numéro est déterminé et correspond à sa position dans le graphe.

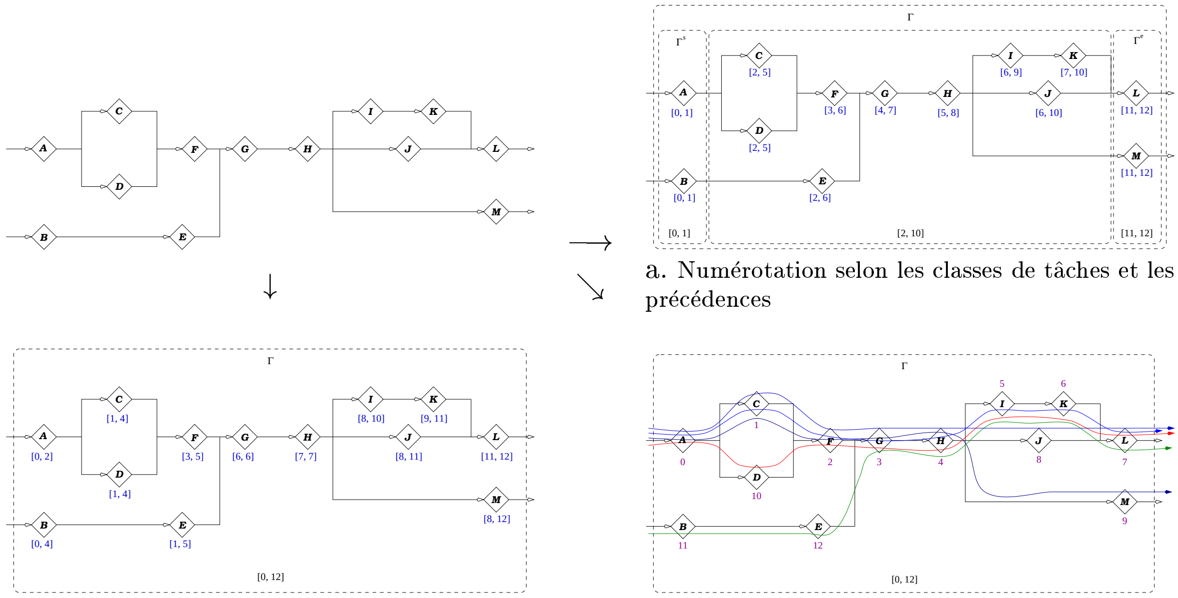
Cependant, là encore, la mise en contraintes est ardue. La difficulté vient des nombreux parcours du graphe à effectuer pour déterminer $\mathcal{P}^T$.

2.5.3 Version relachée (*tri topologique 2*)

Comme précédemment, on garde l'ordre partiel imposé par les précédences.

De plus, afin d'augmenter l'entropie de la quantité d'information transmise, et du fait que nous avons défini trois classes de tâches dans le graphe (les tâches initiales, les tâches terminales et les autres), nous associons à chacune de ces classes des domaines de définition tels que :

$$\begin{array}{ll} \forall T \in \Gamma^s, & k^T \in [0, card(\Gamma^s) - 1] \\ \forall T \in \Gamma^e, & k^T \in [card(\Gamma) - card(\Gamma^e), card(\Gamma) - 1] \\ \forall T \in \Gamma / (\Gamma^s \cup \Gamma^e), & k^T \in [card(\Gamma^s), card(\Gamma) - card(\Gamma^e) - 1] \end{array}$$



a. Numérotation selon les classes de tâches et les précédences

b. Numérotation selon la positions des tâches dans le graphes et les précédences

c. Numérotation sur un chemin : parcours en profondeur

FIG. 2.2 – *Effet des différentes méthodes de numérotation sur un exemple.*

Considérons un graphe acyclique tel qu'il est présenté Fig2.2. On a

$$\begin{aligned}\Gamma &= \{A, B, C, D, E, F, G, H, I, J, K, L, M\} \\ \Gamma^s &= \{A, B\} \\ \Gamma^e &= \{L, M\}\end{aligned}$$

Ainsi, par cette méthode, la tâche A a comme domaine de numérotation $[0, 1]$, tout comme la tâche B. Les tâches L et M se verront attribuer, quant à elles, un numéro appartenant à l'intervalle $[11, 12]$. Les intervalles de définition de la numérotation des autres tâches est déduit des précédences. On obtient alors le résultat de la figure 2.2.a.

2.5.4 Le choix

Pour des raisons d'implémentation, nous avons choisi la dernière méthode présentée. Cependant, elle présente tout de même un inconvénient. Elle suppose que les tâches initiales (resp. terminales) sont situées plutot en début (resp. fin) de graphe. Aussi, les tâches initiales (resp. terminales) auront un petit (resp. grand) numéro.

Les conséquences de cet inconvénient se retrouvent dans les fonctions d'ordonnancement en étirant, dans l'espace de temps logique, les dates d'exécution des différentes tâches. En effet, la date de première exécution est donnée par le paramètre β. Ce dernier étant défini par $\beta = Card(\Gamma).\delta + k$, une tâche initiale située plutot en fin de graphe (et donc qui aura tendance a être exécutée plutot tardivement) aura une date de première exécution tres éloignée des autres du fait de la compensation du paramètre δ sur k.

Illustrons cette situation à l'aide d'un exemple. Considérons l'ordonnancement $ADCFBE$ des 6 premières tâches de notre graphe. Pour obtenir cet ordonnancement, la méthode 2 (§2.5.2) donnera comme première solution :

	A	B	C	D	E	F
k	0	4	2	1	5	3
δ	0	0	0	0	0	0
β	0	4	2	1	5	3

alors que la méthode 3, pour obtenir le même ordre d'exécution, donnera plutot :

	A	B	C	D	E	F
k	0	1	3	2	5	4
δ	0	1	0	0	1	0
β	0	13	3	2	17	4

Comme il ne s'agit que de temps logiques, la dilatation de cet axe temporel n'a pas d'impact sur le reste (calcul de latence...) puisqu'il n'est question que de définir un ordre d'exécution. Les temps logiques de **NOP** sont éliminés dès que l'on s'interesse à des durées.

En fait, l'influence de telle ou telle méthode de numérotation des tâches du graphe est très faible sur la qualité de la solution. En effet, il est toujours possible de trouver des valeurs pour les coefficients δ telles que l'on obtienne des ordonnancements équivalents d'une méthode à l'autre. Ce qui est important est le nombre d'évènements qui apparaissent sur une période et pas la longueur de la période.

Avoir une stratégie de numérotation des tâches permet d'accélérer la recherche de solution dans la mesure où les valeurs sont contraintes et dans des intervalles réduits à quelques valeurs.

2.6 Et le modèle de latence?

La latence est évaluée à partir de la longueur (en temps logique) du chemin *critique* issu du calcul des dépendances. Dans le cas où le graphe est une chaîne (*i.e.*, les tâches sont exécutées séquentiellement les unes par rapport aux autres) La longueur de ce chemin critique correspond à la somme des liens de dépendance les plus longs entre deux tâches (Cf. Fig. 2.3), en supposant que nous soyons dans un contexte de modèle SIMD ou SPMD.

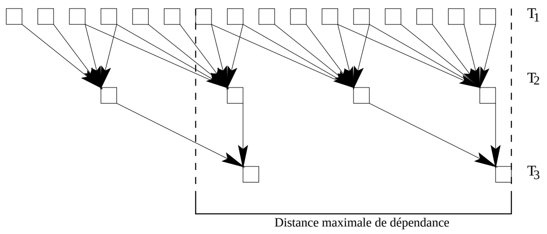


FIG. 2.3 – *Évaluation du chemin critique dans une chaîne.*

Dans le cas d'un graphe autre, mais toujours dans un modèle SIMD ou SPMD, il n'est pas question de faire cette somme pour déterminer le chemin de coût maximum, sous peine de sur-approximer (très grossièrement) ce coût.

Considérons un algorithme de recherche d'un coût maximum dans un graphe et adaptons-le de façon a en extraire des contraintes.

Soit $G = (S, A)$ le graphe orienté sans circuit, représentant le graphe de précédences d'une application de traitement du signal. S est l'ensemble des sommets de G et A l'ensemble des arcs de G. À chaque sommet i, nous associons un poids s_i défini par $s_i = \max_j\{a_{i,j} + s_j\}$. Le terme $a_{i,j}$ désigne le coût de l'arc liant le sommet i au sommet j.

Ainsi, nous avons les contraintes suivantes :

$$\begin{array}{l}
\Lambda = \max\limits_{i \in \Gamma^s}(\beta_i + n_i) + 1 \\
\forall i \in \Gamma \setminus \Gamma^e, \quad n_i = \max\limits_{j}(a_{i,j} + n_j) \\
\forall i \in \Gamma^e, \quad n_i = 0 \\
\forall i, \forall j, a_{i,j} = \max\limits_{k \in \left\{ \substack{\text{sommets du polytope} \\ \text{de dépendance}} \right\}} (d^j(c_{k,j}) - d^i(c_{k,i}))
\end{array}$$

2.7 Conclusion

Ainsi, grâce à ces règles de numérotation des tâches, il est possible de modéliser toutes les applications dont le graphe de tâches est un graphe acyclique quelconque.

De plus, le seul ordre imposé est celui donné par le graphe de précédence. Ce qui laisse le système libre de choisir les valeurs adequates des coefficients, sans pour autant augmenter d'un facteur exponentiel la combinatoire de l'ensemble.

Par ailleurs, nous avons généralisé le calcul du coût du chemin critique dans le graphe partitionné aux graphes acycliques quelconques, à partir duquel nous calculons la latence de l'application placée.

Chapitre 3

Une fonction d'accès affine

Sommaire

3.1 Introduction **14**
3.2 Impact sur les dépendances **14**
3.2.1 Définition d'une dépendance entre deux nids de boucles 14
3.2.2 Effet du paramètre constant sur le polygone des dépendances 15
3.3 La fonction *modulo* dans la fonction d'accès **19**

3.1 Introduction

La fonction d'accès aux données d'un tableau considérée dans [Gue97] est une application du type $\Omega_{pav}.i + \Omega_{fit}.m$ où Ω_{pav} et Ω_{fit} sont, respectivement, les matrice de pavages et d'ajustage du tableau associé, et i et m sont respectivement les vecteurs d'itérations externes et internes.

La question est d'étudier l'impact d'une fonction d'accès affine sur les différents modèles concernés (dépendances et mémoire), *i.e.*, l'impact d'une fonction du type

$$\Omega_{pav}.i + \Omega_{fit}.m + \mathcal{K}$$

où $\mathcal{K}$ est un vecteur de constantes.

L'interêt est de pouvoir :

1. représenter les accès du type `T[3.i + 2.m + 2]`;
2. prendre en compte des boucles dont la borne inférieure n'est pas zéro par le biais d'une translation de l'espace d'itération correspondant.

Nous commencerons par l'impact que peut avoir une telle fonction sur les dépendances. C'est le modèle le plus concerné puisqu'il est nécessaire de connaître précisément la fonction d'accès aux données pour déterminer les dépendances entre nids de boucles. Alors que le modèle mémoire, que nous commentons ensuite, n'est que peu touché par cette extension[1] puisque l'ajout de ce paramètre constant ne modifie pas le motif mais effectue une translation de ce dernier dans le tableau.

Enfin, nous verrons l'effet du modulo sur la fonction d'accès et les dépendances.

3.2 Impact sur les dépendances

3.2.1 Définition d'une dépendance entre deux nids de boucles

Soient deux nids de boucles N^w et N^r dépendant l'un de l'autre. N^w produit des données consommées par N^r. Appelons T le tableau sur lequel repose cette dépendance. Soient $\begin{pmatrix} i^w \\ m^w \end{pmatrix}$ et $\begin{pmatrix} i^r \\ m^r \end{pmatrix}$ les vecteurs d'itérations de chacun des deux nids de boucles, $\begin{pmatrix} \Omega^w_{pav} & \Omega^w_{fit} \end{pmatrix}$ et $\begin{pmatrix} \Omega^r_{pav} & \Omega^r_{fit} \end{pmatrix}$ les matrices d'accès associées, et $\mathcal{K}^w$ et $\mathcal{K}^r$ deux vecteurs de constantes. On défini φ^w et φ^r les vecteurs de références aux données de T tels que $\forall j \in \{w, r\} \quad \varphi^j = \Omega^j_{pav}.i^j + \Omega^j_{fit}.m^j + \mathcal{K}^j$

Par définition du partitionnement 3D, nous avons $\forall j \in \{w, r\} \quad i^j = P^j.L^j.c^j + L^j.p^j + l^j$ pour un partitionnement bloc/cycle. Ainsi, l'ensemble des éléments du tableau T référencés par une partition temporelle du nid de boucles j est :

$$\begin{aligned}\mathcal{R}\left(\mathcal{P}_{c^j}\right) = \Big\{\varphi^j \mid \varphi^j \in \Phi^j, \quad \exists (l^j, p^j), \quad 0 \le {L^j}^{-1}.l^j < 1, \quad 0 \le {P^j}^{-1}.p^j < 1 \quad \exists m^j \in \mathcal{M}^j, \\ \varphi^j = \Omega^j_{pav}\left(P^j.L^j.c^j + L^j.p^j + l^j\right) + \Omega^j_{fit}.m^j + \mathcal{K}^j\Big\}\end{aligned}$$

En posant $s^j = L^j.p^j + l^j$, on a une vue plus simple de la distribution temporelle des données [Gue97, Ch. 4]. Ainsi l'ensemble précédent devient :

$$\begin{aligned}\mathcal{R}\left(\mathcal{P}_{c^j}\right) = \Big\{\varphi^j \mid \varphi^j \in \Phi^j, \quad \exists s^j, \quad 0 \le \left(L^j.P^j\right)^{-1}.s^j < 1, \quad \exists m^j \in \mathcal{M}^j, \\ \varphi^j = \Omega^j_{pav}\left(P^j.L^j.c^j + s^j\right) + \Omega^j_{fit}.m^j + \mathcal{K}^j\Big\}\end{aligned}$$

1. D'autant plus qu'il ne s'agit que d'un modèle capacitif.

Nous dirons qu'il existe une dépendance écriture / lecture entre N^w et N^r pour un couple (c^w, c^r) si et seulement si $\mathcal{R}(\mathcal{P}_{c^w}) \cap \mathcal{R}(\mathcal{P}_{c^r}) \neq \emptyset$ [Gue97, Chap. 5]

C'est-à-dire :

$$\mathcal{D}(c^w, c^r) \Rightarrow \begin{cases} \exists (m^w, m^r) \in \mathcal{M}^w \times \mathcal{M}^r \\ \exists s^w, \quad 0 \leq (L^w.P^w)^{-1}.s^w < 1 \\ \exists s^r, \quad 0 \leq (L^r.P^r)^{-1}.s^r < 1 \\ \Omega^w_{pav}.\left(P^w.L^w.c^w + s^w\right) + \Omega^w_{fit}.m^w + \mathcal{K}^w = \Omega^r_{pav}.\left(P^r.L^r.c^r + s^r\right) + \Omega^r_{fit}.m^r + \mathcal{K}^r \end{cases}$$

Les fonctions d'accès étant linéairement indépendantes et les matrices de partitionnement étant diagonales, on peut aborder le problème dimension de tableau par dimension de tableau. De ce fait, l'espace décrivant les dépendances pour chaque dimension q de T est un polygone caractérisé par le système suivant (pour simplifier la notation, l'indice q dénotant les dimensions a été omis sur chacun des termes) :

$$\begin{cases} 0 \leq c^w \leq c^w_{max} - 1 \\ 0 \leq c^r \leq c^r_{max} - 1 \\ 0 \leq m^w \leq m^w_{max} - 1 \\ 0 \leq m^r \leq m^r_{max} - 1 \\ 0 \leq s^w \leq L^w.P^w - 1 \\ 0 \leq s^r \leq L^r.P^r - 1 \\ \Omega^w_{pav}.\left(P^w.L^w.c^w + s^w\right) + \Omega^w_{fit}.m^w + \mathcal{K}^w = \Omega^r_{pav}.\left(P^r.L^r.c^r + s^r\right) + \Omega^r_{fit}.m^r + \mathcal{K}^r \end{cases}$$

L'élimination des variables s^j et m^j engendre le système de contraintes suivant :

$$\begin{cases} 0 \leq c^w \leq c^w_{max} - 1 \\ 0 \leq c^r \leq c^r_{max} - 1 \\ -\Omega^r_{pav} L^r P^r.c^r + \Omega^w_{pav} L^w P^w.c^w \geq -\Omega^w_{fit}(m^w_{max} - 1) - \Omega^w_{pav}(L^w P^w - 1) + \mathcal{K}^r - \mathcal{K}^w & (D_{inf}) \\ -\Omega^r_{pav} L^r P^r.c^r + \Omega^w_{pav} L^w P^w.c^w \leq \Omega^r_{fit}(m^r_{max} - 1) + \Omega^r_{pav}(L^r P^r - 1) + \mathcal{K}^r - \mathcal{K}^w & (D_{sup}) \end{cases}$$

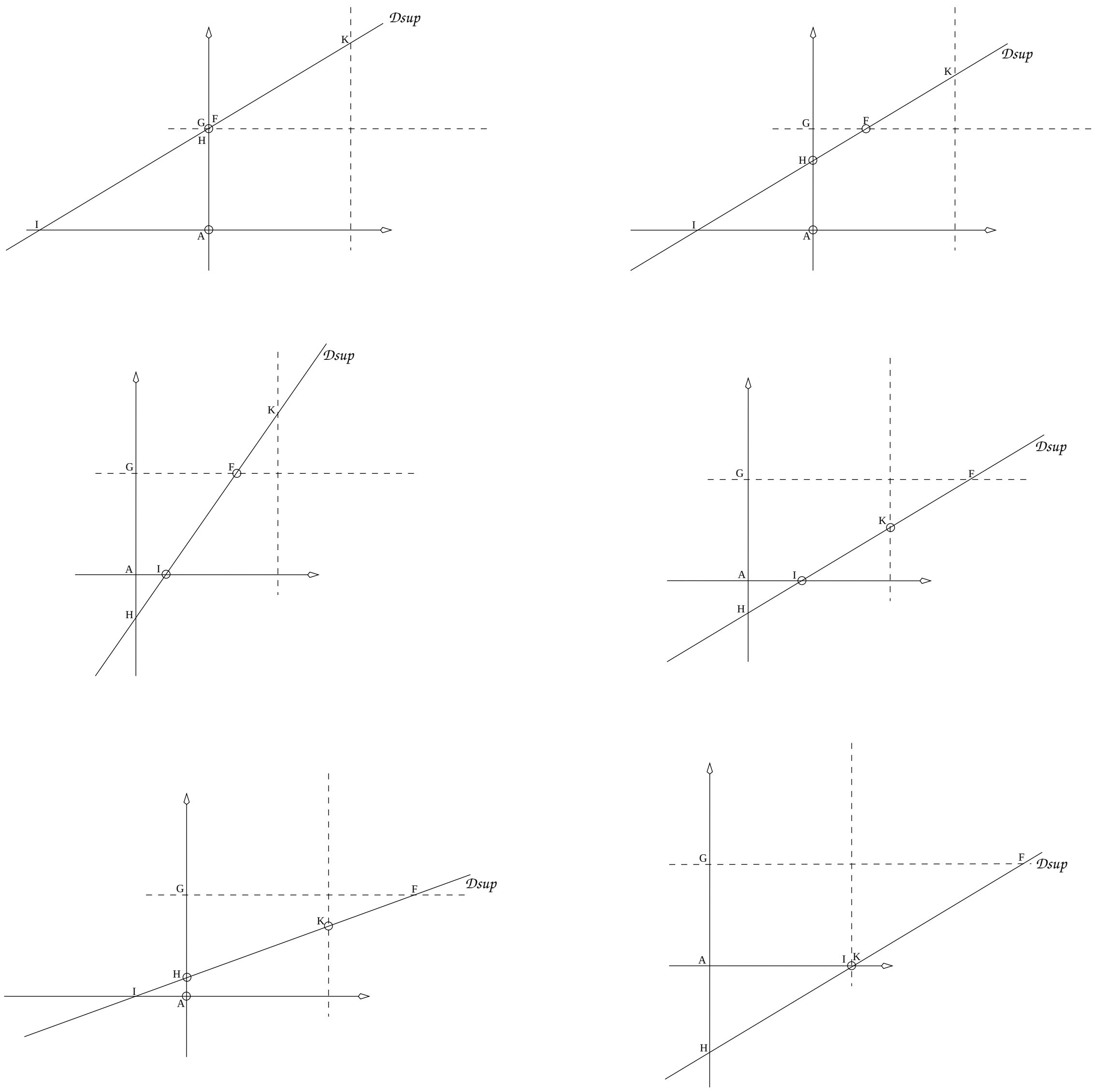
L'espace de dépendance est donc délimité par les droites d'équations $c^j = cst$, D_{inf} et D_{sup}.

3.2.2 Effet du paramètre constant sur le polygone des dépendances

La présence de cet offset $\mathcal{K}^r - \mathcal{K}^w$ donne une plus grande liberté quant à la position des droites D_{inf} et D_{sup}. En effet, si l'on considère D_{sup} (resp. D_{inf}), $\Omega^r_{fit}(m^r_{max}-1)+\Omega^r_{pav}(L^r P^r-1)$ est toujours positif ou nul (resp. $-\Omega^w_{fit}(m^w_{max}-1)-\Omega^w_{pav}(L^w P^w-1)$ est toujours négatif ou nul), par définition des différents coefficients intervenant dans ces termes. Il est possible d'obtenir un partitionnement tel que $\mathcal{K}^r - \mathcal{K}^w$ change le signe du membre droit des équations des droites. La figure 3.1 illustre les différents cas possibles d'intersection de la droite D_{sup} avec l'espace d'itérations partitionné. On obtient la même chose avec la droite D_{inf} sachant que l'on a toujours D_{sup} "au dessus" de D_{inf}.

Par ailleurs, la relaxation dans les entiers des différents points d'intersections oblige à sur-approximer [2] le polygone de dépendance par la plus petite enveloppe convexe dont les sommets générateurs sont à coordonnées entières. On obtient alors les points suivants :

2. L'élimination de cette sur-approximation fait l'objet d'une étude.



Les différentes possibilités de positionnement de la droite D_{inf} sont les mêmes que celles de la droites D_{sup}. Aussi, il y a $6 \times 6 = 36$ polygônes de dépendances probables alors qu'il n'y en a que $3 \times 3 = 9$ si l'on ne prend pas en compte les offsets dans les fonctions d'accès. En effet, sans les offsets, les points I et J ont des coordonnées toujours négatives (ou nulles) ce qui n'est plus le cas selon les valeurs de ces décalages.

FIG. 3.1 – *Énumération des différentes positions possible pour la droite* D_{sup}.

$$\left\{\begin{array}{lcl} x_A & = & 0 \\ y_A & = & 0 \end{array}\right. \qquad \left\{\begin{array}{lcl} x_E & = & c^r_{max} - 1 \\ y_E & = & c^w_{max} - 1 \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_C & = & c^r_{max} - 1 \\ y_C & = & 0 \end{array}\right. \qquad \left\{\begin{array}{lcl} x_G & = & 0 \\ y_G & = & c^w_{max} - 1 \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_I & = & \left\lfloor \frac{-\Omega^r_{fit}(m^r_{max}-1)-\Omega^r_{pav}(L^rP^r-1)+\mathcal{K}^w-\mathcal{K}^r}{\Omega^r_{pav}L^rP^r} \right\rfloor \\ y_I & = & 0 \end{array}\right. \qquad \left\{\begin{array}{lcl} x_L & = & \left\lceil \frac{\Omega^w_{fit}(m^w_{max}-1)+\Omega^w_{pav}(L^wP^w.c^w_{max}-1)-\mathcal{K}^r+\mathcal{K}^w}{\Omega^r_{pav}L^rP^r} \right\rceil \\ y_L & = & c^w_{max} - 1 \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_J & = & 0 \\ y_J & = & \left\lfloor \frac{-\Omega^w_{fit}(m^w_{max}-1)-\Omega^w_{pav}(L^wP^w-1)+\mathcal{K}^r-\mathcal{K}^w}{\Omega^w_{pav}L^wP^w} \right\rfloor \end{array}\right. \qquad \left\{\begin{array}{lcl} x_K & = & c^r_{max} - 1 \\ y_K & = & \left\lceil \frac{\Omega^r_{fit}(m^r_{max}-1)+\Omega^r_{pav}(L^rP^r.c^r_{max}-1)+\mathcal{K}^r-\mathcal{K}^w}{\Omega^w_{pav}L^wP^w} \right\rceil \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_B & = & \left\lceil \frac{\Omega^w_{fit}(m^w_{max}-1)+\Omega^w_{pav}(L^wP^w-1)-\mathcal{K}^r+\mathcal{K}^w}{\Omega^r_{pav}L^rP^r} \right\rceil \\ y_B & = & 0 \end{array}\right. \qquad \left\{\begin{array}{lcl} x_H & = & 0 \\ y_H & = & \left\lceil \frac{\Omega^r_{fit}(m^r_{max}-1)+\Omega^r_{pav}(L^rP^r-1)-\mathcal{K}^w+\mathcal{K}^r}{\Omega^w_{pav}L^wP^w} \right\rceil \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_F & = & \left\lfloor \frac{-\Omega^r_{fit}(m^r_{max}-1)-\Omega^r_{pav}(L^rP^r-1)+\Omega^w_{pav}L^wP^w(c^w_{max}-1)+\mathcal{K}^w-\mathcal{K}^r}{\Omega^r_{pav}L^rP^r} \right\rfloor \\ y_F & = & c^w_{max} - 1 \end{array}\right.$$

$$\left\{\begin{array}{lcl} x_D & = & c^r_{max} - 1 \\ y_D & = & \left\lfloor \frac{-\Omega^w_{fit}(m^w_{max}-1)-\Omega^w_{pav}(L^wP^w-1)+\Omega^r_{pav}L^rP^r(c^r_{max}-1)+\mathcal{K}^r-\mathcal{K}^w}{\Omega^w_{pav}L^wP^w} \right\rfloor \end{array}\right.$$

La suppression des restrictions sur les points I et J confère une propriété de symétrie au polygône de dépendances. Plus précisément, il s'agit d'une symétrie de *configuration* dans laquelle on a les relations suivantes :

$$\begin{array}{ccc|ccc} A & \leftrightarrow & E & D & \leftrightarrow & H \\ B & \leftrightarrow & F & I & \leftrightarrow & L \\ C & \leftrightarrow & G & J & \leftrightarrow & K \end{array}$$

Selon la configuration obtenue, les sommets générateurs de l'enveloppe convexe seront tels ou tels points d'intersections listés ci-dessus. De ce fait, les sommets générateurs du polygone de dépendance sont (comme illustrés par la figure 3.2) :

$$\begin{array}{ll} AIJ & = (\max(\min(x_I, x_C), x_A) \quad , \quad \max(\min(y_J, y_G), y_A)) \\ BCJ & = (\max(\min(x_B, x_C), x_A) \quad , \quad \max(\min(y_J, y_G), y_A)) \\ CDL & = (\max(\min(x_L, x_C), x_A) \quad , \quad \max(\min(y_D, y_G), y_A)) \\ EKL & = (\max(\min(x_L, x_C), x_A) \quad , \quad \max(\min(y_K, y_G), y_A)) \\ FGK & = (\max(\min(x_F, x_C), x_A) \quad , \quad \max(\min(y_K, y_G), y_A)) \\ GHI & = (\max(\min(x_I, x_C), x_A) \quad , \quad \max(\min(y_H, y_G), y_A)) \end{array}$$

L'écriture AIJ signifie que, selon le cas, ce sommet est confondu avec A, I ou J.

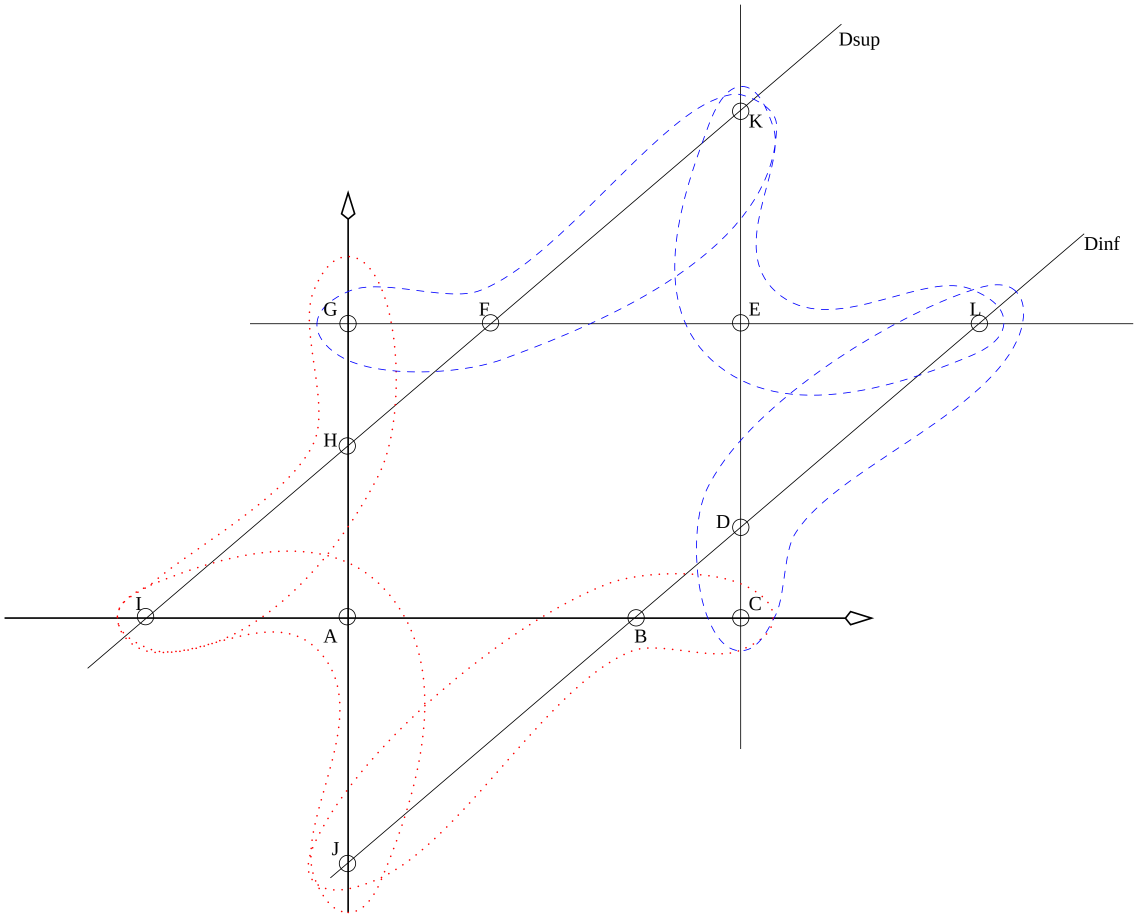


FIG. 3.2 – *Regroupement des points déterminant les sommets du polygone.*

3.3 La fonction *modulo* dans la fonction d'accès

Que se passe-t-il lorsque la fonction d'accès à un tableau fait intervenir un modulo, *i.e.*, est de la forme :

$$(\Omega_{pav}.i + \Omega_{fit}.m + \mathcal{K}) \bmod \phi_{max}$$

où ϕ_{max} est le vecteur des tailles des dimensions du tableau accédé par cette fonction ?

Remarque : De part l'hypothèse de tableaux à assignation unique, les seuls accès *modulo* autorisés sont des accès en lecture.

La figure 3.3 page 20 montre un exemple d'accès avec modulo. Dans cet exemple, le tableau monodimensionnel accédé en écriture et en lecture est noté T. Sa taille ϕ_{max} est de 15 mots. La fonction d'accès en écriture est $3.i^w : 3.i^w + 2$ avec $i^w \in [0, 4]$. Celle en lecture est : $(4.i^r : 4.i^r + 3) \bmod \phi_{max}$ avec $i^r \in [0, 14]$.

En supposant des partitionnements tels que pour chaque espace d'itération on a $P^w = P^r = 1$, $L^w = L^r = 1$ et $c^w_{max} = i^w_{max} = 5, \quad c^r_{max} = i^r_{max} = 15$, on obtient les équations des droites D_{inf} et D_{sup} suivantes :

$$\begin{array}{ll} D_{inf}: & 3.c^w - 4.c^r = -2 \\ D_{sup}: & 3.c^w - 4.c^r = 3 \end{array}$$

Sur la figure, les points noirs représentent les itérations sur lesquelles il y a vraiment une dépendance de données. Les points blancs représentent les points de dépendances que l'on aurait s'il n'y avait pas de modulo.

On constate alors que les *bandes* de dépendances correspondent à des morceaux de la bande virtuelle. Plus précisément, elles sont les morceaux issus de l'application du modulo.

Selon le partitionnement, les équations des droites délimitants le polygone des dépendances changent. Aussi, s'il est possible d'obtenir une représentation formelle de chacune de ces bandes, il est difficile, voire quasi-impossible, d'en extraire une modélisation en contraintes. C'est pourquoi la solution adoptée est de sur-approximer cet espace de dépendances par la plus petite enveloppe convexe recouvrant l'ensemble de ces sous polygones. Dans notre exemple, cette sur-approximation est matérialisée sur la figure par un polygone en trait gras.

De plus, chercher à représenter exactement l'espace de dépendance dans le cas d'un modulo n'est pas interressant, au contraire. Le polygone de dépendance représente l'espace des itérations en écriture et en lecture dépendantes les unes des autres pour l'accès à un tableau donné. Schématiquement, on peut traduire un modulo comme définissant les accès à des données, ou blocs de données, non contigües d'un tableau (par exemple en début et en fin du tableau). Les itérations effectuant les opérations en écriture sont séquentielles : on fait la première, puis la deuxième ainsi de suite. Aussi, si pour un calcul en lecture (*i.e.*, une itération), les dépendances font apparaitre la première et la dernière itération en écriture, cela ne signifie pas que les autres n'ont pas été exécutées. De ce fait, les données produites au cours de ces calculs intermédiaires sont stockées en mémoire attendant d'être utilisées. De ce point de vu, l'approximation donne un meilleur résultat qu'une représentation exacte puisqu'elle prend en compte ce phénomène.

Il est tout de même possible d'améliorée la sur-approximation. En effet, celle effectuée dans [Gue97] n'est pas exactement la plus petite enveloppe convexe recouvrant l'ensemble des dépendances. Pour reprendre notre exemple, lorsque l'itération en lecture $c^r = 11$ est terminée, il est possible de libérer l'espace mémoire occupé par les données produites à $c^w = 0$ car elles ne seront plus utilisées par la

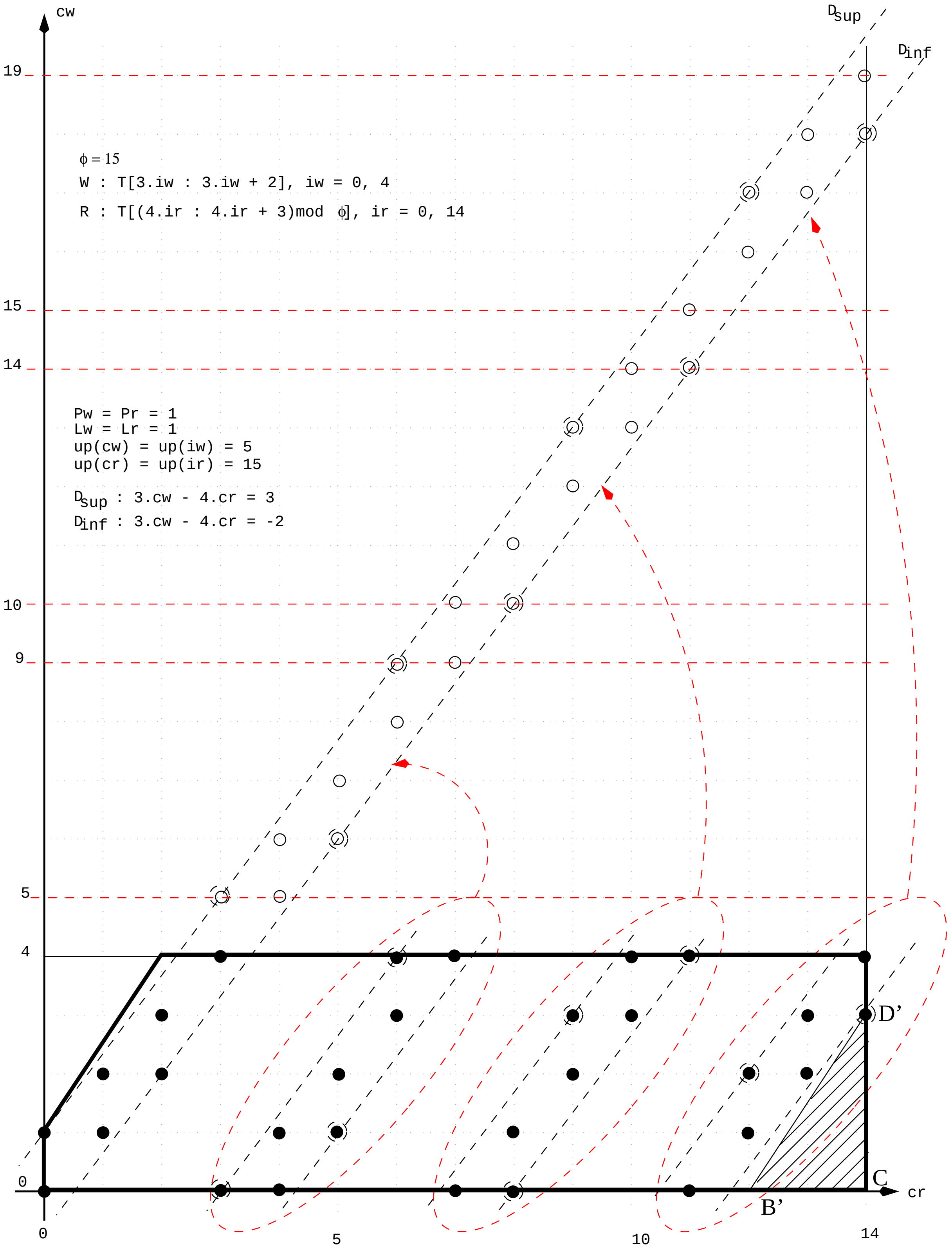


Fig. 3.3 – *Effet d'un modulo dans la fonction d'accès*

suite. Cela correspond à la partie hachurée de la figure 3.3 (le triangle B'CD'). Les points B' et D' sont définis par :

$$B' = \left(\left\lceil \frac{-a.x_{D'} + b.y_{D'}}{-a} \right\rceil, 0 \right) \text{ et } D' = (c^r_{max} - 1, y_D \bmod c^w_{max})$$

Ainsi, on défini deux sommets B'C et CD' tels que :

$$\begin{array}{rl} B'C = & (\min(x_{B'}, c^r_{max} - 1), 0) \\ D'C = & (c^r_{max} - 1, \max(0, y_{D'})) \end{array}$$

Remarque : si recouvrement des bandes, alors signifie besoin de plus de donnees que disponibles ==> non sens

Chapitre 4

Les pré-traitements

Sommaire

4.1 Introduction **24**
4.2 Accès distincts à un même tableau dans une TE **24**
4.3 Conditionnement des tâches élémentaires **24**
4.3.1 TE conditionnée par un paramètre constant 24
4.3.2 TE conditionnée par un paramètre variable 24
4.4 Bornes de boucles paramétrées **24**
4.4.1 Paramètre constant 25
4.4.2 Paramètre variable 25

4.1 Introduction

Ce chapitre décrit succintement les différentes manipulations à effectuer sur l'ensemble des nids de boucles (*i.e.*, l'application) à placer afin que l'outil de placement puisse aider au mieux l'expert.

4.2 Accès distincts à un même tableau dans une TE

Les accès multiples à un même tableau peuvent être pris en compte :

1. en augmentant le modèle de dépendance ;
2. en ajoutant une opération de copie pour la deuxième référence et les suivantes ;
3. en ajoutant une boucle interne, i.e. un motif, pour regrouper par fermeture hyper-rectangulaire toutes les références.

Le probème : Surcout mémoire et calcul des dépendances incorrect.

Solution adoptée : Ajout d'une tâche virtuelle qui effectue la copie du tableau en question. Laisse le surcoût mémoire mais donne des dépendances justes (impact directe sur l'ordonnancement et donc la latence). En mettant une durée d'exécution de la copie nulle, la latence n'est pas influencée.

4.3 Conditionnement des tâches élémentaires

4.3.1 TE conditionnée par un paramètre constant

Dérivation de la TE en autant de valeur du paramètre possible.

4.3.2 TE conditionnée par un paramètre variable

De manière générale le paramètre variable est une donnée calculée préalablement. Les modèles concernés sont :

la latence : selon la valeur du paramètre, on fait tel ou tel calcul ;

la mémoire : pour tel ou tel calcul on a besoin de telle quantité de mémoire ;

les dépendances : selon le calcul, les fonctions d'accès diffèrent

On se base alors sur l'expertise domaine afin d'en tirer des probabilités de calcul. Cela agit directement sur la latence.

4.4 Bornes de boucles paramétrées

Le problème des bornes de boucles paramétrées est le caractère dynamique que cela donne à la tâche. En effet, les modèles posés bénéficient de l'aspect statique du traitement de signal systématique. S'il l'on considère par exemple le partitionnement, il n'est pas possible de déterminer un découpage de l'espace d'itération dans un contexte de programmation par contrainte s'il n'est pas fixe. À moins de reprendre totalement le modèle.

4.4.1 Paramètre constant

En première approximation, on affecte au paramètre une valeur au pire cas parmi celles possibles. L'idéal serait de prendre en compte toutes les valeurs possibles.

4.4.2 Paramètre variable

Réécriture de façon a rendre constante les bornes de boucles. Inconvénient : perte de la sémantique du calcul car bien souvent la réécriture ne permet pas d'obtenir un algorithme équivalent. De plus, cette réécriture doit faire en sorte de sur-approximer les volumes mémoire utiles et les dépendances afin d'assurer la validité de la solution donnée. En effet, si le système trouve une solution pour un problème plus contraint (au sens où les ressources machines sont plus sollicités), il est raisonnable de penser qu'il existe une solution pour le problème réelle.

Bibliographie

[Aul99] D. Aulagnier. Benchmark - radar aéroporté à forme d'onde mfr. Technical Report SBU.IS/DSTN/ELA99/NT/P1, Thomson-CSF / Detexis, 1999.

[BBC92] D. Beauquier, J. Berstel, and Ph. Chrétienne. *Élément d'algorithmique*. Masson, 1992.

[Gue97] C. Guettier. *Optimisation globale et placement d'applications de traitement du signal sur architectures parallèles utilisant la programmation par contraintes*. PhD thesis, École Nationale Supérieure des Mines de Paris, November 1997.

[Mus00] N. Museux. Radar aéroporté à forme d'onde mfr - transcription. Technical Report E/226/CRI - TSI/99/906, Thomson-CSF / LCR - ENSMP/CRI, 2000.

Table des matières

1 Introduction **3**

2 D'une séquence de tâches vers un graphe de tâches acyclique **5**
2.1 Introduction 6
2.2 Définitions 6
2.3 D'une chaîne à un graphe : l'ordonnancement 7
2.4 Conditions sur la fonction de numérotation 8
2.5 Quel ordre choisir ? 8
2.5.1 Parcours de graphe *(en profondeur, en largeur)* 9
2.5.2 Selon la position d'une tâche dans le graphe (*tri topologique 1*) 9
2.5.3 Version relachée (*tri topologique 2*) 9
2.5.4 Le choix 10
2.6 Et le modèle de latence? 11
2.7 Conclusion 12

3 Une fonction d'accès affine **13**
3.1 Introduction 14
3.2 Impact sur les dépendances 14
3.2.1 Définition d'une dépendance entre deux nids de boucles 14
3.2.2 Effet du paramètre constant sur le polygone des dépendances 15
3.3 La fonction *modulo* dans la fonction d'accès 19

4 Les pré-traitements **23**
4.1 Introduction 24
4.2 Accès distincts à un même tableau dans une TE 24
4.3 Conditionnement des tâches élémentaires 24
4.3.1 TE conditionnée par un paramètre constant 24
4.3.2 TE conditionnée par un paramètre variable 24
4.4 Bornes de boucles paramétrées 24
4.4.1 Paramètre constant 25
4.4.2 Paramètre variable 25

Références **25**