Mémoire de DEA d'Informatique

Présenté par
Ridha KOUACHE

Durées de vie et compression mémoire

26 Juin 2002

ICPS/LSIIT, Université Louis Pasteur
Pôle API, Bd Sébastien Brant
67400 ILLKIRCH

Responsables de Stage : Philippe Clauss
Frédéric Vivien

Rapporteur : Cédric Wemmert

Table des matières

Remerciements

Les deux premier es personnes auxquelles je voudrais m'adresser mes remerciements sont évidement Philippe Clauss et Frédéric Vivien mes encadreurs de stage. Leurs patiences, leurs curiosités, leurs vivacités d'esprit sont autant de qualité qui provoque l'admiration de ses étudiants . Je les suis reconnaissant de m'avoir donné ma chance et de m'avoir fait le point de départ dans le monde de la recherche.

Je tient également à remercier les membres de jury. Merci bien à Christian Ronse notre responsable de DEA d'avoir nous acceptés à suivre nos études en France, merci à ses orientations, ses encouragements et son soutient régulier durant toute l'année. Je remercier Cédric Wemmert d'avoir accepté de rapporter ma mémoire de stage et de m'avoir prodigé de nombreux conseils qui m'ont permis de clarifier cette étude. En fin je remercie JC Spehner et Thoma Noël d'être des membres de mon jury.

Après mon jury, je tient de remercier Michel Salomon de me supporté dans son bureau effleuré de l'ambiance. Je remercie mon collègue de bureau Frédéric Wagner d'être un exemple d'accompagnateur et être serviable. Ainsi a mon ami Rachid Seghir et Son encadreur Vincent Loechner qui m'aide toujours à effectuer mes tests sur leur programme de la librairie Polilyb, sans oublier mes deux amis Mohamed et Dominique, ainsi à tous les personnes de laboratoire ICPS et bien sur tous mes collègues de DEA et à tous les personnels du Pôle API de l'ULP Strasbourg.

Particulièrement, j'adresse toute ma gratitude à ma famille et spécialement à mes parents qui attend de loin ma réussite, loin mais proche à mon coeur. Je leur ai jamais assez dit, mais je peux au moins leur écrire.

Pour terminer, je revient de remercier Frédéric Vivien pour son coup de pouce et ses aides pour m'adapter au niveau des autres étudiants et m'aider à rédiger le rapport qui je lui fait fatigué avec mes fautes de rédaction, je lui dis : « Sorry my teacher... ».

Chapitre 1

Introduction

L'évolution des techniques de compilation et de conception automatique de matériels permet aujourd'hui de concevoir de manière conjointe *Co-Design* des systèmes électroniques intégrés sur une seule puce de silicium appelé ***"Système on Chip" SoC***. Ces systèmes dans leurs versions embarquées (ou enfouies) doivent répondre à des contraintes spécifiques de place, de vitesse et de consommation électrique. De plus, les capacités sans cesse croissantes de ces systèmes permettent aujourd'hui de développer des applications plus complexes telles que les applications multimédia.

Ces dernières travaillent entre autres sur des images et des signaux de grande taille, engendrant de gros besoins en espace mémoire et des transferts de données volumineux, traités par des boucles imbriquées. Il faut donc se concentrer sur la réduction mémoire lors de la conception de telles applications dans le monde de l'embarqué.

La mémoire constitue le facteur de coût et de performance dominant dans les systèmes embarqués , car elle est très consommatrice d'énergie, ce qui pousse le programmeur à concevoir des programmes utilisant pour leur exécution le moins de mémoire possible. Mais un tel travail d'optimisation de code s'avère très difficile et accumule d'autres contraintes aux contraintes des fonctionnalités à développer. De cela vient la nécessité de développer des méthodes d'optimisation automatique qui peuvent être intégrées dans les compilateurs. La compression mémoire est un objectif majeur.

De nombreuses méthodes d'optimisation mémoire dédiées aux nids de boucles accédant à des tableaux multidimensionnels ont été proposées, ces méthodes se classent en trois catégories :

1. Optimisation de l'utilisation des registres.
2. compression des variables des programmes en assignation unique avec examen des durées de vie.
3. Amélioration des localités temporelles et spatiales des données.

Cette dernière a engendré de nombreux travaux proposant l'amélioration de la localité des données à travers :

1. Des transformations des domaines d'itération :
   - Modification de l'ordre dans lequel les itérations d'un nid de boucles sont parcourues. Cette transformation nécessite la modification des instructions, et en particulier des références aux tableaux. Plusieurs techniques ont été proposées tel que :

     L'échange de boucles : inversion du niveau d'imbrication de deux boucles [9, 35].

     L'inversion d'indice : décrémentation d'indice [35].

     La torsion de boucle (Skewing) : décalage affine des bornes d'un indice de boucle [55].

     Le pavage (Tiling ou Blocking) : pavage de l'espace d'itérations en sous-domaines d'itérations [9, 39, 40, 14, 6].

     La localisation de séquence de données : C'est une combinaison entre la localité temporelle et spatiale [16, 40, 36, 37].

- Transformation entre plusieurs nids de boucles, consistant à associer à chaque instruction son propre domaine d'itération, puis à translater dans l'espace des indices l'un des domaines. On distingue deux opérations de ce type de transformation :

  fusion de boucles : C'est de combiner plusieurs nids de boucles en un seul [2, 34, 7, 35, 25, 10, 49, 26, 16].

  fission de boucles : C'est l'opération inverse de la précédente qui consiste à générer des nids de boucles a partir d'un ou plusieurs autres nids de boucles.

2. Réorganisation des données en mémoire, qui consiste à modifier l'adressage des éléments d'un tableau selon l'ordre dans lequel ils sont accédés par la boucle [38, 16, 4].

Bien que ces méthodes cessent permettant de réduire la taille mémoire de manière directe ou indirecte, le problème d'une efficacité déterminante reste ouvert. Notre travail se situe dans ce contexte de réduction mémoire, en se basant sur l'analyse de la durée de vie des variables dans les nids de boucles pour la minimiser. Notre méthode consiste à rapprocher les instances d'écriture (production) des instances de lecture (consommation) d'une même variable. Par cela on améliore le traitement des boucles manipulant la mémoire d'une manière répétitive en assurant une meilleure réutilisation mémoire. Ainsi on contractant les tableaux multidimensionnel de grandes dimensions en des tableaux monodimensionnels d'une dimension nécessaire pour contenir les variables vivantes. on exploitant les résultat des travaux sur le modèle polyédrique et l'ordonnancement. Plus précisément, notre travail se découpera ainsi :

Le chapitre 2 est consacré à quelques définitions et notions sur les nids de boucles et leurs modélisation polyédrique, les dépendances exactes et son contexte, l'ordonnancement et son utilisation dans les nids boucles, ainsi que les principales notions du modèle polyédrique tel que le polynôme d'Ehrhart.

Le chapitre 3 présente une étude sur les durées de vie des variables, tout on essayons de définir une borne inférieure sur la quantité mémoire nécessaire pour contenir les données du programme à un instant donnée. Autrement dit, on basant sur la durée de vie des variables, quelles est le nombre minimum de variables vivantes à un instant donnée ?

Nous présentons ensuite une méthode de minimisation de la durée de vie dans le cas d'un ordonnancement monodimensionnel. Cette méthode est baisée principalement sur l'analyse exacte des dépendances et ces contraintes d'ordonnancement.

Le chapitre 4 présente la génération de code afin d'assurer une réduction mémoire par contraction des tableaux, on utilisant l'analyse des durées de vie définie dans le chapitre précédent, ainsi on donne un exemple pour montrer comment mis à jour les ordonnancements et réindéxer les tableaux. Nous y présentons en fin de chapitre le problématique de la minimisation dans le cas d'un ordonnancement multidimensionnel, donnant un exemple qui montre la limite de la méthode.

Le chapitre 5 présente une étude sur les travaux similaires qui entre dans le même domaine, ainsi leurs complexités et leurs performances.

Le chapitre 6 conclut ce travail par des perspectives sur la réduction mémoire par analyse de durée de vie, en utilisant le modèle polyédrique et les travaux sur l'ordonnancement, tel que de définir des méthodes heuristiques d'optimisation de cette durée de vie dans le cas d'un ordonnancement variable multidimensionnel plutôt que monodimensionnel.

Chapitre 2

Définitions et notations

2.1 Nids de boucles :

Définition 1
Nous appelons **nid de boucles***, ou ensemble de* **boucles imbriquées***, une structure de programme finie de la forme :*

For $i = l(N_1, \ldots, N_p)$ to $u(N_1, \ldots, N_p)$ step r do
  B_1
  $\vdots$
  B_s
EndFor

où B_j est une instruction ou un nid de boucles, i une variable entière appelée **indice** *ou* **compteur** *de boucle, r une variable entière appelée* **incrément** *ou* **pas** *de la boucle, et l et u sont des fonctions définissant les* **bornes de boucle***. L'ensemble des blocs B_1, ..., B_s constitue le* **corps** *de la boucle.*

Dans notre travail, nous ne considérons pas des nids aussi généraux. Nous supposons que la fonction l (resp. u) est une fonction affine de paramètres de structure et de compteurs de boucles englobantes ou est le maximum (resp. minimum) d'un ensemble de telles fonctions. . Sauf indication contraire nous supposons que les pas sont toujours strictement positifs et nous nous servons de cette propriété tout au long de ce mémoire, implicitement. Cette restriction n'est pas contraignante, un simple changement de variable transformant une boucle à incrément négatif en une boucle à un incrément positif. De même, nous ne nous intéressons qu'à des nids dont toutes les boucles sont de pas unitaire ($r = 1$) et nous omettons généralement de spécifier la valeur de ce pas Pour des raisons toutes autres, nous excluons du corps de boucle toute fonction d'arrêt intempestif ou de transfert de contrôle (goto, appel de procédure). Nous appelons **nid de boucles parfait**, **nid parfait**, ou **boucles parfaitement imbriquées**, un nid dont le corps de toute boucle contient exactement, soit un nid, soit un bloc d'instructions. Avec nos hypothèses, un nid parfait est donc de la forme :

For $i_1 = l(N_1, \ldots, N_p)$ to $u(N_1, \ldots, N_p)$ do
  For $i_2 = l(N_1, \ldots, N_p, i_1)$ to $u(N_1, \ldots, N_p, i_1)$ do
    $\vdots$
    For $i_n = l(N_1, \ldots, N_p, i_1, \ldots, i_{n-1})$ to $u(N_1, \ldots, N_p, i_1, \ldots, i_{n-1})$ do
      S_1
      $\vdots$
      S_s
    EndFor
    $\vdots$

```
  EndFor
EndFor
```

où $S_1, \ldots, S_s$ sont des instructions. Que le nid étudié soit parfait ou non, nous notons toujours ses instructions de la sorte et la variable s représente le nombre total d'instructions du nid. n est appelé la **profondeur** du nid. Dans le cas de nids non parfaits, la profondeur d'une instruction est égale au nombre de boucles qui l'entourent. Afin de simplifier les notations, nous omettons les délimiteurs EndFor dans nos exemples, l'imbrication des boucles et instructions étant alors spécifiée par les niveaux d'indentation.

Nous identifions les itérations des n_S boucles englobants l'instruction S avec les vecteurs de $\mathbb{Z}^{n_S}$, appelés **vecteurs d'itération**, contenus dans un polyèdre convexe $\mathcal{D}_S$, appelé **domaine d'itération**, borné par les bornes de boucles. Dans le cas d'un nid parfait, toutes les instructions ont le même domaine d'itération $\mathcal{D}$. La $i^{ème}$ composante d'un vecteur d'itération est égale à la valeur du $i^{ème}$ compteur de boucle, en comptant à partir de la boucle la plus externe. Dans un nid, les différentes itérations (instances) d'une même instruction sont donc exécutées dans l'ordre lexicographique[1] de leurs vecteurs d'itération. Nous notons $\prec$ l'**ordre lexicographique** strict :

$$\forall \boldsymbol{i}, \boldsymbol{j} \in \mathbb{Z}^n, \quad \boldsymbol{i} \prec \boldsymbol{j} \quad \Leftrightarrow \quad \exists k \leq n, \forall l \leq k-1, \quad \boldsymbol{i}_l = \boldsymbol{j}_l \text{ et } \boldsymbol{i}_k < \boldsymbol{j}_k$$

et $\preccurlyeq$ l'ordre lexicographique large.

2.1.1 Programme à contrôle statique

Pour que l'analyse de dépendance des données soit exacte, il faut que le programme à analyser soit à contrôle statique, c'est-à-dire qu'il ait les caractéristiques suivantes :

1. Les structures de données sont des scalaires, réels ou des tableaux d'entiers ou réels.
2. Les instructions élémentaires sont des affectations sur des scalaires ou des éléments de tableaux.
3. Les structures de contrôle sont des boucles For (ou Do) et des compositions séquentielles.
4. Les indices des tableaux et les bornes des compteurs de boucles sont des fonctions affines entières des compteurs et des paramètres de structure.
5. Le programme ne comprend aucun appel de procédures, pas de goto, aucun pointeur, aucune instruction conditionnelle If.

2.1.2 Analyse des dépendances exactes

L'analyse des dépendances de données est une étape essentielle dans l'application de transformations de parallélisation et d'optimisation. Elle consiste à trouver les opérations qui entrent en conflit, c'est-à-dire qui doivent être exécutées dans un ordre prédéterminé afin de préserver la sémantique du programme original. On dit que deux opérations sont indépendantes si leur éventuel accès à un même emplacement mémoire (une variable ou un élément de tableau) ne se fait qu'en lecture. De nombreuses méthodes de détection des dépendances ont été proposées. Citons les travaux de U. Banerjee[3], P. Feautrier[21], F. Irigoin et al. sur PIPS [27], W. Pugh sur Omega [43].

Bernstien [5] a proposé une classification de trois types de dépendances :

• *Dépendance de flot :* générée par une écriture suivie d'une lecture dans un même emplacement mémoire.

• *Dépendance de sortie :* générée par une écriture suivie d'une autre écriture dans un même emplacement mémoire.

•*Anti-dépendance :* générée par une lecture suivie d'une écriture dans un même emplacement mémoire.

Ces dépendances sont représentées géométriquement par un ***"vecteur de dépendance"*** entre deux itérations $\boldsymbol{i}$ et $\boldsymbol{j}$ $\quad d_{S,T} = \boldsymbol{j} - \boldsymbol{i}$, pour le même niveau de boucle.

Remarque : Le vecteur de dépendance $d_{S,T}$ doit être toujours lexicographiquement positif : $d_{S,T} \succeq 0$.

L'ordre d'exécution d'un programme est représenté par ***un graphe de dépendance*** dont les noeuds désignent les points entiers associés aux instances des instructions et les arcs désignent les dépendances entres les instances des instructions.

[1] Cette propriété n'est bien évidemment vérifiée que par les nids dont toutes les boucles sont à incréments positifs, ce qui est notre cas.

L'analyse des dépendances exactes est possible dans les programmes à contrôle statique [21]. Ou chaque relation de dépendance exacte e de l'instruction S_e à l'instruction T_e est caractérisée par un polyèdre $\mathcal{D}_e$(le domaine d'existence de la relation de dépendance) et par une fonction quasi-affine h_e définie par :

pour toute valeur de $\boldsymbol{j} \in \mathcal{D}_e \Rightarrow S_e(h_e(\boldsymbol{j}, N), N) \rightarrow T_e(\boldsymbol{j})$

où N est le vecteur de paramètres. Evidemment, la description des dépendances exactes entre deux instructions peut impliquer l'union de plusieurs relations de dépendance. Une relation de dépendance e définit pour toute valeur $\boldsymbol{j} \in \mathcal{D}_e$ une dépendance entre les deux opérations $S_e(h_e(\boldsymbol{j}, N), N)$ et $T_e(\boldsymbol{j})$.

2.1.3 Mise en assignation unique :

Dans ce travail, nous considérons que les dépendances de flot directes. C'est-à-dire on considère que notre programme ne contient pas de dépendances artificielles tels que les dépendances de sortie et les anti-dépendances ou les dépendances de flots parasites qui n'ont aucune influence sur la sémantique du programme. Pour cela, une solution radicale a été proposée dans [20], basée sur l'expansion totale du programme qui détruit systématiquement toutes les dépendances artificielles.

L'idée de cette méthode d'expansion totale est d'assurer que chaque élément de tableau ou scalaire ne se voit affecter qu'une seule valeur (emplacement mémoire) durant l'exécution du programme, en gardant que les dépendances de flot directes.

L'algorithme d'expansion totale de la mémoire se base sur trois étapes :

1. Renommage systématique des variables, c'est à dire dans chaque instruction d'affectation, donner un nouveau nom de variable dans sa LHS (Left Hand Side).
2. Expansion totale des structures de données, c'est-à-dire pour chaque instruction, donner à chaque nouvelle variable la forme du domaine d'itération dans la LHS, on remplaçant la fonction d'indexation originale pour une liste des compteurs englobants.
3. Reconstruction du flot de données, c'est-à-dire remplacer chaque référence en lecture par la fonction source.

Vue la taille mémoire considérable utilisé par cette méthode, une autre technique d'expansion partielle a été proposée [31], mais cette méthode vise à extraire du parallélisme tout en essayant de limiter l'explosion mémoire qui leur est associée. Tant dis que notre objectif est de ce concevoir des méthodes exactes pour bien exploiter la taille mémoire.

2.2 Ordonnancement

Une fonction d'ordonnancement d'un ensemble d'opérations attribue à chaque opération une valeur prise dans un ensemble totalement ordonné $\mathcal{E}$. Pour notre problème spécifique, les ordonnancements doivent satisfaire des contraintes de précédence : si l'opération q dépend de l'opération p, alors p doit être ordonnée avant q, c'est-à-dire $\Theta(p) < \Theta(q)$, où Θ est la fonction d'ordonnancement et « $<$ » l'ordre total sur $\mathcal{E}$. Les ordonnancements définissent des ordres partiels.
Nous ne nous intéressons pas à n'importe quel type d'ordonnancement.
Nous ne considérons que des fonctions relativement régulières à image dans un ensemble discret, et ce pour faciliter leur recherche, leur manipulation et la réécriture du code. Les ordonnancements les plus classiques sont à valeur dans $\mathbb{Z}^m$, où m est un entier. Suivant la valeur de m nous parlerons d'ordonnancement monodimensionnel ($m = 1$) ou multidimensionnel ($m > 1$).

Définition 2 (Ordonnancement monodimensionnel)
Un ordonnancement monodimensionnel est une fonction Θ qui associe à chaque itération de chaque instruction d'un nid de boucles un entier (date) choisi de manière à préserver les dépendances :

$$\Theta : V \times \mathbb{Z}^n \rightarrow \mathbb{Z} \text{ tel que } S(\boldsymbol{i}) \Rightarrow T(\boldsymbol{j}) \quad \Longrightarrow \quad \Theta(S, \boldsymbol{i}) < \Theta(T, \boldsymbol{j})$$

Par conséquent, une fonction Θ à valeur dans $\mathbb{Z}$ est un ordonnancement si et seulement si, pour tout couple de vecteurs d'itération $(\boldsymbol{i}, \boldsymbol{j})$ et tout couple d'instructions (S, T), elle vérifie :

$$\Theta(T, \boldsymbol{j}) \geq 1 + \Theta(S, \boldsymbol{i}) \tag{2.1}$$

si $T(\boldsymbol{j})$ dépend de $S(\boldsymbol{i})$.

La notion d'ordonnancement est étendue aux fonctions à valeurs dans $\mathbb{Q}$: est un ordonnancement (cf. [46]) la partie entière d'une fonction à valeur rationnelle qui satisfait l'équation 2.1. Par abus de notation, nous appellerons « ordonnancement » toute fonction rationnelle dont la partie entière est un ordonnancement. Sauf indication contraire, nous ne considérerons que des ordonnancements à valeurs rationnelles.

Nous ne nous préoccupons que d'ordonnancement réguliers : ordonnancements linéaires, linéaires décalés (ou affine de partie linéaire unique) et affines. Ce sont des extensions des hyperplans introduits par Lamport [30].

Définition 3 (Ordonnancement affine)
Un ordonnancement affine est une application Θ de la forme :

$$\begin{array}{rcl} V \times \mathbb{Z}^n & \rightarrow & \mathbb{Q} \\ \Theta:\ (v, \boldsymbol{i}) & \mapsto & X_v \boldsymbol{i} + \rho_v \end{array}$$

où les variables X_v représentent des vecteurs de $\mathbb{Q}^n$, les variables ρ_v des constantes rationnelles et où le produit $X_v\boldsymbol{i}$ est le produit scalaire.

Si tous les vecteurs X_v sont identiques, l'ordonnancement est dit **affine de partie linéaire unique** ou **linéaire décalé**. Si, de plus, toutes les constantes sont nulles ou toutes égales, l'ordonnancement est dit **linéaire**.

Nous définissons maintenant la notion d'ordonnancement multidimensionnel, c'est-à-dire d'ordonnancement à valeurs dans un espace $\mathbb{Q}^m$ de dimension éventuellement strictement supérieure à un. Ces ordonnancements ont été introduits et étudiés par Karp, Miller et Winograd [28], Lamport [30], Rao [47], Roychowdhury [48] et Feautrier [23].

Définition 4 (Ordonnancement multidimensionnel)
Un ordonnancement multidimensionnel de dimension d est une fonction qui définit un ordre d'exécution par l'association d'un vecteur entier de dimension d à chaque itération de chaque instruction. Les vecteurs image sont ordonnés entre eux grâce à l'ordre lexicographique (noté $\prec$) et sont choisis de manière à respecter les dépendances. Un tel ordonnancement est donc une application de $V \times \mathbb{Z}^n$ dans $\mathbb{Z}^d$.

Les différentes composantes de l'ordonnancement peuvent être vues comme des jours, des heures, des minutes, des secondes, etc, sachant que tous les jours n'ont pas forcément autant d'heures, les heures de minutes, etc. L'ordre séquentiel d'exécution d'un nid de boucles de profondeur supérieure ou égale à deux est un ordonnancement multidimensionnel.

Comme pour les ordonnancements monodimensionnels, nous étendons la notion d'ordonnancement aux fonctions à valeurs rationnelles et nous ne nous intéressons qu'à des fonctions un tant soit peu régulières.

Définition 5 (Ordonnancement affine multidimensionnel)
Un ordonnancement affine multidimensionnel de dimension d est une application Θ de la forme :

$$\begin{array}{rcl} V \times \mathbb{Z}^n & \rightarrow & \mathbb{Q}^d \\ \Theta:\ (v, \boldsymbol{i}) & \mapsto & \left(X_v^1 \boldsymbol{i} + \rho_v^1, \ldots, X_v^d \boldsymbol{i} + \rho_v^d\right) \\ S(\boldsymbol{i}) \Rightarrow T(\boldsymbol{j}) & \Longrightarrow & \left\lfloor \left(X_S^1 \boldsymbol{i} + \rho_S^1, \ldots, X_S^d \boldsymbol{i} + \rho_S^d\right) \right\rfloor \prec \left\lfloor \left(X_T^1 \boldsymbol{j} + \rho_T^1, \ldots, X_T^d \boldsymbol{j} + \rho_j^d\right) \right\rfloor \end{array}$$

où les variables X_v^i représentent des vecteurs de $\mathbb{Q}^n$ et les variables ρ_v^i des constantes rationnelles.

2.3 Polyèdres

2.3.1 Modèle polyédrique et polynômes d'Ehrhart

Le modèle d'analyse et de transformation de programmes dans lequel se place ce travail est *le modèle polyédrique* développé par P. Feautrier [24] . La contribution essentielle de ce modèle est qu'il permet de calculer les dépendances exactes. Cependant, nous allons utiliser un modèle polyédrique spécifique et ses extensions aux polyèdres paramétrés, ainsi que la librairie *Polylib* développés par P. Clauss et V. Loechner [12, 13] dans l'équipe ICPS/LSIIT de l'Université Louis Pasteur de Strasbourg.

Définition 6 (Polyèdre convexe) *Un polyèdre $\mathcal{P}$ est un sous-espace de $\mathbb{Q}$ borné par un nombre fini d'hyperplans. $\mathcal{P}$ s'écrit sous forme d'intersection d'un nombre fini de demi-espaces affines.*

On appel représentation implicite la définition du polyèdre par un système d'inégalités et d'égalités, ou contraintes :

$$\mathcal{P} = \{x \in \mathbb{Q}^n | Ax = a, Bx \geq b\}$$

ou : A et B sont des matrices entières, a et b sont des vecteurs entiers, $n \in \mathcal{N}$ est la dimension de l'espace dans lequel est défini le polyèdre. Cette forme est l'intersection des demi-espaces affines définis par les inégalités suivantes : $Ax \geq a, Ax \leq a, Bx \geq b$.

Définition 7 (Polytope) *: Un polytope est un polyèdre convexe borné d'un ensemble fini de points. C'est l'intersection bornée d'un nombre fini de demi-espaces.*

La modélisation des nids de boucles par des polyèdres permet la mise en évidence de la propriété suivante :

Propriété 1 *Dans le cas ou la boucle de niveau m contient une boucle de niveau m-1 au maximum (boucles parfaitement imbriquées), les bornes de boucles déterminent un polytope simple. Dans le cas contraire, les bornes déterminent une union de polytopes $\mathcal{D}$.*

Définition 8 (Polyèdre homothétique/bordé) *Un polytope homothétique/bordé est un polyèdre paramétré dont les sommets sont des fonctions affines(strictes) des paramètres.*

Propriété 2 *Tout polytope paramétré est homothétique/bordé par domaine de validité.*

Exemple : Soit le fragment de programme suivant :

```
do i=1,4
   do j=i+1,i+5
      ...
   enddo
enddo
```

Il est modélisé par le polyèdre défini par : $1 \leq i \leq 4 \quad et \quad i+1 \leq j \leq i+5$

2.3.2 Polyèdre paramétré :

Définition 9 (Polyèdre paramétré) *On appel le polyèdre paramétré un polyèdre défini par un système de contraintes dont la partie constante dépend linéairement de paramètres :*

$$\mathcal{N} = \begin{pmatrix} N_1 \\ N_2 \\ \cdots \\ N_p \end{pmatrix} \in \mathbb{Q}^p.$$

$$\mathcal{P}(\mathcal{N}) = \{x \in \mathbb{Q}^n | Ax = A'\mathcal{N} + a, Bx \geq B'N + b\}$$

ou : A, A', B et B' sont des matrices entières, a et b sont des vecteurs entiers .

Ce polyèdre peut être représenté dans un espace combiné des indices et des paramètres de la manière suivante :

$$D' = \{\begin{pmatrix} x \\ \mathcal{N} \end{pmatrix} \in \mathbb{Q}^{n+p} \mid [A| - A'] \begin{pmatrix} x \\ \mathcal{N} \end{pmatrix} = a, [B| - B'] \begin{pmatrix} x \\ \mathcal{N} \end{pmatrix} \geq b\}$$

D' est un polyèdre paramétré, définit dans un espace de dimension $n + p$

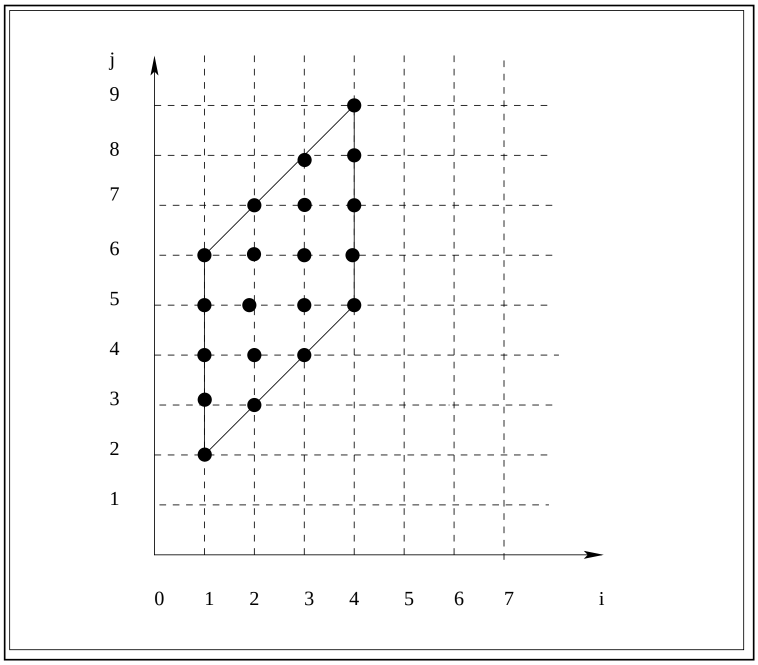

FIG. 2.1 – Polyèdre convexe

2.3.3 Nombre périodique et pseudo-polynôme

Définition 10 (Nombre périodique unidimensionnel) *Soit une suite (u_i), définie pour tout entier naturel i, de période q. On appelle nombre périodique unidimensionnel la fonction $u(i) = u_{i \ mod \ q}$, notée :*

$$u_i = [u_0, u_1, ..., u_{q-1}]_i$$

Définition 11 (Nombre périodique multidimensionnel) *Soit I un vecteur entier de dimension p, $I = (I_j)_{j \in [1,p]}$. Un nombre p-périodique u(I) est défini par un tableau u de p dimensions et de taille $q_1 \times q_2 \times ... \times q_p$ tel que :*

$$u(I) = u_{N_1,N_2,...,N_P}, avec \quad N_j = I_j \bmod q_j, \forall j \in [1,p]$$

Le vecteur $q = (q_1, q_2, ..., q_p)$ est appelé la multi-période de u(I). Le plus petit commun multiple des coefficients de ce vecteur est appelé la période de u(I).

Exemples :

1. Le nombre périodique $[3, \quad 7, \quad 7]_N$ est de dimension 1 et de période 3 en N. Il vaut :

$$\begin{cases} 3 & si & N \bmod 3 = 0 \\ 7 & si & N \bmod 3 = 1 \\ 7 & si & N \bmod 3 = 2 \end{cases}$$

2.Le nombre périodique :

$$\begin{pmatrix} 1 & 3 & 7 \\ 3 & 7 & 5 \\ 7 & 3 & 1 \end{pmatrix}_{N,M}$$

est de dimension 2 et de période 3×3. il vaut :

$$\begin{cases} 1 & si & N \bmod 3 = 0 & et & M \bmod 3 = 0 \\ 3 & si & N \bmod 3 = 0 & et & M \bmod 3 = 1 \\ 7 & si & N \bmod 3 = 2 & et & M \bmod 3 = 2 \\ 3 & si & N \bmod 3 = 1 & et & M \bmod 3 = 0 \\ 7 & si & N \bmod 3 = 1 & et & M \bmod 3 = 1 \\ & & ... & & \\ 3 & si & N \bmod 3 = 2 & et & M \bmod 3 = 2 \end{cases}$$

Définition 12 (Pseudo-polynôme) *Soit N un vecteur entier de dimension p, $N = (N_1, N_2, ..., N_p)$. Un pseudo-polynôme de degré d est défini par :*

$$f(N) = \sum_{i_1=0}^{d} \sum_{i_2=0}^{d-i_1} ... \sum_{i_p=0}^{d-i_1-i_2-\cdots-i_{p-1}} C_{i_1,i_2,...,i_p} N_1^{i_1} N_2^{i_2} ... N_p^{i_p}$$

où chaque coefficient $C_{i_1,i_2,...,i_p}$ est un nombre périodique. On appelle période du pseudo-polynôme le plus petit commun multiple des périodes des $C_{i_1,i_2,...,i_p}$.

Exemples

- pour tout entier $i \geq 2$:

$$\sum_{1}^{\frac{i}{2}} 1 = \frac{1}{2}i + \left[0, \frac{1}{2}\right]_i$$

- Pour tout couple d'entiers i et j positifs :

$$\sum_{1}^{\frac{i}{2}} \sum_{1}^{\frac{j}{2}} 1 = \left(\frac{1}{4}i + \left[\frac{1}{2}, \frac{1}{4}\right]_i\right) j + \left[\left(\frac{1}{2}i + \left[1, \frac{1}{2}\right]_i\right), \left(\frac{1}{4}i + \left[\frac{1}{2}, \frac{1}{4}\right]_i\right)\right]_j$$

$$= \frac{1}{4}ji + \left[\frac{1}{2}, \frac{1}{4}\right]_i j + \left[\frac{1}{2}, \frac{1}{4}\right]_j i + \begin{bmatrix} 1 & 1\backslash 2 \\ 1\backslash 2 & 1\backslash 4 \end{bmatrix}_{j,i}$$

2.3.4 Polynômes d'Ehrhart

La modélisation des nids de boucles par des polyèdres, nous a permet d'utiliser un résultat de travaux sur les polyèdres, ce résultat est le *Polynôme d'Ehrhart* [17, 18, 19], ce polynôme est utilisé pour dénombrer (énumérer) les points à coordonnées entières contenus dans un polyèdre paramétré.

Le théorème suivant donné par Ehrhart [17] pour un seul paramètre a été étendu a un nombre quelconque de paramètres par Clauss dans [12].

Théorème 1 (Théorème d'Ehrhart-Clauss) *Soit $\mathcal{P}(p)$ un polyèdre paramétré homothétique-bordé de dimension d, ayant m partamètres $p = (p_1, p_2, ..., p_m)$.*
Le nombre de points entiers contenus dans $\mathcal{P}(p)$, appelé "Polynôme d'Ehrhart", est un pseudo-polynôme en p de degré d et dont la période est le dénominateur de $\mathcal{P}(p)$. la forme générale du polynôme d'Ehrhart de $\mathcal{P}(p)$ est :

$$E(\mathcal{P}) = \sum_{i_1=0}^{d} \sum_{i_2=0}^{d-i_1} ... \sum_{i_m=0}^{d-(i_1+i_2+...+i_{m-1})} C_{i_1,i_2,...,i_m} P_1^{i_1} P_2^{i_2} ... P_m^{i_m}$$

$C_{i_1,i_2,...,i_m}$ sont des nombres m-périodiques ou plus, et dont la période est inférieur ou égale au dénominateur de $\mathcal{P}(p)$.

Le nombre de points contenus dans un polyèdre paramétré quelconque est donné par suos-domaine des paramètres, qui sont les domaines de validité des sommets du polyèdre paramétré. Un polynôme d'Ehrhart est calculé pour chacun des domaines de validité. Connaissant la forme générale du polynôme d'Ehrhart dans chaque domaine de validité, il suffit de résoudre un système d'égalités linéaires déduit d'un certain nombre de valeurs initiales du polynôme. Ces valeurs initiales sont obtenues en comptant le nombre de points entiers dans les polyèdres non-paramétrés correspondant à des instanciations des paramètres dans le domaine de validité concerné.

2.3.5 Lemme de Farkas

Ce lemme [22, 50] prévoit la forme de certaines formes affines.

Théorème 2 (Forme affine du lemme de Farkas) *Soit $\mathcal{D}$ un polyèdre non vide défini par p inégalité : $a_k x + b_k \geq 0, k \in \{1, ..., p\}$. Une forme affine Φ est positive en tout point de $\mathcal{D}$ si et seulement si elle est une combinaison affine positive des formes affines définissant $\mathcal{D}$:*

$$\Phi(x) = \lambda_0 + \sum_{k=1}^{p} \lambda_k(a_k x + b_k), \forall k \in [0, P]\lambda_k \geq 0$$

*Les valeurs positives λ_k sont appelées **multiplicateur de Farkas**.*

Ce théorème est très utilisé dans les programmes à contrôle statique, tous les ensembles importants sont des polyèdres : domaines d'itérations, domaines d'existence de dépendance [21],...etc. Il permet donc d'exprimer une fonction affine positive comme une combinaison affine des formes affines qui définissent son domaine de positivité. Feautrier l'utilise pour exprimer les fonctions d'ordonnancement et pour simplifier un ensemble de contraintes de dépendances. Il obtient ainsi un système d'équations linéaires dont les inconnues sont les multiplicateurs de Farkas paurontt être résolu par programmation linéaire.

2.3.6 Méthode des sommets

Cette méthode repose sur la représentation de Minkowski des polyèdres.

Définition 13 (Représentation de Minkowski des polyèdres) *Un polyèdre peut toujours être décomposé comme la somme d'un polytope (un polyèdre borné) et d'un cône polyédrique, appelé le cône caractéristique (pour de plus amples détails, voir [51]). Un polytope est défini par ses sommets, et tout point du polytope est une combinaison barycentrique (positive) des sommets du polytope. Un cône polyédrique est engendré par un nombre fini de rayons et de lignes. Tout point d'un cône polyédrique est la somme d'une combinaison positive de ses rayons et d'une combinaison quelconque de ses lignes. Par conséquent, un polyèdre $\mathcal{D}$ peut être défini par un ensemble de sommets, $\{v_1, \ldots, v_\omega\}$, un ensemble de rayons, $\{r_1, \ldots, r_\rho\}$ et un ensemble de lignes, $\{l_1, \ldots, l_\lambda\}$. Le polyèdre $\mathcal{D}$ est alors l'ensemble des points p tels que :*

$$p = \sum_{i=1}^{\omega} \mu_i v_i + \sum_{i=1}^{\rho} \nu_i r_i + \sum_{i=1}^{\lambda} \xi_i l_i \tag{2.2}$$

où $\mu_i \in \mathbb{Q}^+$, $\nu_i \in \mathbb{Q}^+$, $\xi_i \in \mathbb{Q}$ et $\sum_{i=1}^{\omega} \mu_i = 1$.

Les sommets, rayons et lignes d'un polyèdre peuvent être calculés même pour les polyèdres paramétrés [33]. La méthode des sommets, introduite par Quinton [45], utilise la représentation de Minkowski pour récrire un ensemble d'inéquations qui doit être satisfait sur tout un polyèdre.

Théorème 3 (Méthode des sommets) *Soit un polyèdre non vide $\mathcal{D}$ défini par un ensemble de sommets, $\{v_1, \ldots, v_\omega\}$, un ensemble de rayons, $\{r_1, \ldots, r_\rho\}$ et un ensemble de lignes, $\{l_1, \ldots, l_\lambda\}$. Soit une forme affine Φ de partie linéaire A et de partie constante b ($\Phi(x) = A.x + b$). La forme affine Φ est positive sur $\mathcal{D}$ si et seulement si 1) Φ est positive sur chacun des sommets de $\mathcal{D}$ et 2) la partie linéaire de Φ est positive (respectivement nulle) sur les rayons (resp. les lignes) de $\mathcal{D}$. Ce qui peut être écrit mathématiquement :*

$$\forall p \in \mathcal{D},\ A.p + b \geq 0 \quad \Leftrightarrow$$
$$\forall i \in [1, \omega],\ A.v_i + b \geq 0,\ \forall i \in [1, \rho],\ A.r_i \geq 0,\ et\ \ \forall i \in [1, \lambda],\ A.l_i = 0.$$

Chapitre 3

Durée de vie des valeurs dans les nids de boucles

Dans ce chapitre nous présentons le concept sur lequel se base notre technique de réutilisation mémoire à partir de l'analyse mémoire des nids de boucles à savoir *la durée de vie (ou intervalle d'utilité).*

Dans cette analyse on doit tout d'abord pouvoir identifier précisément à quels moments une valeur produite par une opération doit nécessairement être en mémoire en se basant sur l'ordre d'exécution défini par la fonction d'ordonnancement. L'ensemble des moments où une valeur doit être en mémoire définit son intervalle d'utilité, dont la durée est *la durée de vie de la valeur.*

3.1 Durée de vie d'une valeur

Définition 14 *La durée de vie d'une valeur est le temps d'exécution compris entre la date d'écriture (production) de cette valeur et sa date de dernière lecture (consommation).*

Illustration sur l'exemple 1

Nous illustrons le calcul de la durée de vie des valeurs dans les nids de boucles avec l'exemple 1 présenté figure 3.1.

```
    For i=1,N−1
      For j=1,N−1
S:        A[i,j]=(B[i,j−1]+B[i−1,j]) / 2
T:        B[i,j]= (A[i,j]+A[i−1,j]+A[i,j−1]+A[i−1,j−1])/ 4
      EndFor
   EndFor
```

FIG. 3.1 – Exemple 1.

Dépendances et ordonnancements

Cet exemple contient quatre dépendances de flot par rapport au tableau A et deux autres par rapport au tableau B :

- $e_{S,1} : S(i,j) \to T(i,j)$, $\mathcal{D}_{e_{S,1}} = \{(i,j) | 1 \leq i \leq N-1, 1 \leq j \leq N-1\}$;
- $e_{S,2} : S(i-1,j) \to T(i,j)$, $\mathcal{D}_{e_{S,2}} = \{(i,j) | 2 \leq i \leq N-1, 1 \leq j \leq N-1\}$;
- $e_{S,3} : S(i,j-1) \to T(i,j)$, $\mathcal{D}_{e_{S,3}} = \{(i,j) | 1 \leq i \leq N-1, 2 \leq j \leq N-1\}$;
- $e_{S,4} : S(i-1,j-1) \to T(i,j)$, $\mathcal{D}_{e_{S,4}} = \{(i,j) | 2 \leq i \leq N-1, 2 \leq j \leq N-1\}$;
- $e_{T,1} : T(i,j-1) \to S(i,j)$, $\mathcal{D}_{e_{T,1}} = \{(i,j) | 1 \leq i \leq N-1, 2 \leq j \leq N-1\}$;
- $e_{T,2} : T(i-1,j) \to S(i,j)$, $\mathcal{D}_{e_{T,2}} = \{(i,j) | 2 \leq i \leq N-1, 1 \leq j \leq N-1\}$;

Nous supposons que les fonctions d'ordonnancement des deux instructions sont définies comme suit :

$$\begin{aligned} \Theta_S(i,j) &= 2i + 2j \\ \Theta_T(i,j) &= 2i + 2j + 1 \end{aligned}$$

Ces ordonnancements sont légaux car ils satisfont les contraintes de dépendance (toutes les dépendances étant ici uniformes). Prenons par exemple la relation de dépendance $e_{S,1}$. Cette dépendance génère la contrainte d'ordonnancement : $\Theta_T(i,j) - \Theta_S(i,j) = 2i + 2j + 1 - (2i + 2j) = 1 \geq 1$ qui est vérifiée, donc ces ordonnancements satisfont la relation de dépendance $e_{S,1}$. De la même façon, on peut montrer qu'ils satisfont toutes les relations de dépendance de notre programme.

Durée de vie de l'élément de tableau $B[i,j]$

- Sa date d'écriture (production) est :

$$\Theta_T(i,j) = 2i + 2j + 1.$$

- Sa date de dernière lecture (consommation) est : $\Theta_S(i,j+1) = \Theta_S(i+1,j) = 2i+2j+2$, à cause de la relation de dépendance $e_{T,1}$ (si $1 \leq i \leq N-1, 2 \leq j \leq N-1$) ou à cause de la relation de dépendance $e_{T,2}$ (si $2 \leq i \leq N-1, 1 \leq j \leq N-1$).
- Sa durée de vie est donc :

$$DV(B[i,j]) = \Theta_S(i,j+1) - \Theta_T(i,j) = \Theta_S(i+1,j) - \Theta_T(i,j) = 1.$$

Durée de vie de l'élément de tableau $A[i,j]$

Considérons l'élément de tableau A[i,j] :

- À cause de la dépendance $e_{S,1}$, $A[i,j]$ a au moins une durée de vie de :

$$\Theta_T(i,j) - \Theta_S(i,j) = 2i + 2j + 1 - (2i + 2j) = 1$$

- À cause de la dépendance $e_{S,2}$, $A[i,j]$ (si $2 \leq i \leq N-1, 1 \leq j \leq N-1$) a au moins une durée de vie de :

$$\Theta_T(i,j) - \Theta_S(i-1,j) = 2i + 2j + 1 - (2i - 2 + 2j) = 3$$

- À cause de la dépendance $e_{S,3}$, $A[i,j]$ (si $1 \leq i \leq N-1, 2 \leq j \leq N-1$) a au moins une durée de vie de :

$$\Theta_T(i,j) - \Theta_S(i,j-1) = 2i + 2j + 1 - (2i + 2j - 2) = 3$$

- À cause de la dépendance $e_{S,4}$, $A[i,j]$ (si $2 \leq i \leq N-1, 2 \leq j \leq N-1$) a au moins une durée de vie de :

$$\Theta_T(i,j) - \Theta_S(i-1,j-1) = 2i + 2j + 1 - (2i - 2 + 2j - 2) = 5$$

Par conséquent, la durée de vie de la valeur $A[i,j]$ produite par l'opération $S(i,j)$ est de 5 (si $2 \leq i \leq N-1, 2 \leq j \leq N-1$).

3.2 Borne inférieure sur la quantité de mémoire nécessaire

La modélisation des nids de boucles par des polyèdres convexes permet l'utilisation des outils mathématiques existants portant sur la géométrie convexe et l'arithmétique en nombres entiers. Plusieurs méthodes utilisant les descriptions polyédriques [29, 32, 22, 23, 24] se sont montrées très puissantes pour résoudre les problèmes de transformation de boucles. Les problèmes sont posés sous forme de systèmes d'inéquations linéaires.

Le principe de notre méthode pour définir une borne inférieure de la quantité mémoire nécessaire à un instant donnée est le suivant :

- soit Θ un ordonnancement quelconque, soient t_d et t_f les dates de début et de fin de l'exécution de Θ (on peut prendre $t_d = 0$ et t_f la date de la dernière opération exécutée par le nid, dans le but d'encadrer notre travail sur un domaine borné) et soit $t_0 \in [t_d, t_f]$;
- dans ce contexte, nous cherchons a répondre aux questions : « Combien de valeurs sont vivantes à la date t_0 ? », « Quelle est la date t_0 à laquelle un maximum de valeurs sont vivantes ? », l'idée étant que nous aurons besoin d'au moins autant de mémoire qu'il y aura de valeurs simultanément en vie.

3.2.1 Résolution

Soit $e : \forall \boldsymbol{j} \in \mathcal{D}_e \quad S_e(h_e(\boldsymbol{j}, N)) \to T_e(\boldsymbol{j})$, une relation de dépendance entre les deux instructions S_e et T_e. On note $\mathcal{D}_{S_e}(t_0, e)$ l'ensemble des éléments de $\mathcal{D}_{S_e}$ qui correspondent à une valeur « en vie » à l'instant t_0 à cause de la dépendance e.

Un élément $\boldsymbol{j}_0$ de $\mathcal{D}_{S_e}$ appartient à $\mathcal{D}_{S_e}(t_0, e)$ si :

1. Il est source d'une dépendance incluse dans le domaine de dépendance $\mathcal{D}_e$:
$$\exists \boldsymbol{j} \in \mathcal{D}_e, \boldsymbol{j}_0 = h_e(\boldsymbol{j}, N);$$
2. La valeur est produite par l'instruction S_e avant l'instant t_0 :
$$\Theta(S_e, \boldsymbol{j}_0, N) \leq t_0;$$
3. La valeur produite doit être consommée après l'instant t_0 :
$$\Theta(T_e, \boldsymbol{j}, N) \geq t_0.$$

On a alors :

$$\mathcal{D}_{S_e}(t_0, e) = \{\boldsymbol{j}_0 \in \mathcal{D}_{S_e} | \exists \boldsymbol{j} \in \mathcal{D}_e, \boldsymbol{j}_0 = h_e(\boldsymbol{j}, N), \Theta(S_e, \boldsymbol{j}_0, N) \leq t_0, \Theta(T_e, \boldsymbol{j}, N) \geq t_0\}.$$

Pour une instruction S donnée on définit $\mathcal{D}_S(t_0)$ comme l'ensemble des éléments de $\mathcal{D}_S$ qui correspondent à une valeur « en vie » à l'instant t_0 :

$$\mathcal{D}_S(t_0) = \bigcup_{e | S_e = S} \mathcal{D}_{S_e}(t_0, e)$$

Dans ce cas, on ne peut pas effectuer une recherche de l'ordonnancement Θ tel que la valeur :

$$\max_{t_0 \in [t_d, t_f]} \sum_S | \mathcal{D}_S(t_0) |$$

soit minimale, car l'expression définissant $\mathcal{D}_S(t_0)$ est paramétrée par t_0, N et Θ, et contient un produit de paramètres ($\Theta(T_e, \boldsymbol{j}, N)$ contient le produit de Θ et N). Les méthodes polyédriques ne peuvent donc pas être mises en œuvre. Par contre, Θ étant fixé, on peut calculer le nombre d'éléments de $\mathcal{D}_{S_e}(t_0)$, en utilisant les fonctions offertes par la librairie Polylib [42], ces fonctions ont comme résultat un polynôme d'Ehrhart qui donne le nombre d'éléments du polyèdre paramétré $\mathcal{D}_S(t_0)$ en fonction de t_0.

Nous cherchons à calculer :

$$\max_{t_0 \in [t_d, t_f]} \sum_S | \mathcal{D}_S(t_0) |$$

ce qui correspond à écrire :

$$\max_{t_0 \in [t_d, t_f]} \sum_S \left| \bigcup_{e | S_e = S} \mathcal{D}_{S_e}(t_0, e) \right|$$

Cette somme par rapport aux différentes instructions constitue une union de polyèdres $\mathcal{D}_{S_e}(t_0, e)$, ces polyèdres pouvant avoir des éléments en commun. Il n'est donc pas possible de calculer le nombre d'éléments de $\mathcal{D}_S(t_0)$ en effectuant la somme des tailles des ensembles $\mathcal{D}_{S_e}(t_0, e)$. Il nous faut décomposer l'ensemble $\mathcal{D}_S(t_0)$ en une union de polyèdres *disjoints* (par exemple au moyen de la librairie Polylib), puis calculer le nombre des éléments contenus dans cette union disjointe. Le résultat sera exprimé sous la forme de polynômes d'Ehrhart, les différents polynômes correspondant à différents domaines de validités (des paramètres ou de la date t_0).

La valeur maximale des polynômes d'Ehrhart obtenus est le nombre maximal de valeurs simultanément en vie et est donc une borne inférieure de la quantité mémoire nécessaire pour stocker les valeurs calculées par le programme.

Les exemples suivants donnent une illustration de la méthode proposée pour calculer la borne inférieure de la quantité de mémoire nécessaire.

3.2.2 Application à l'exemple 1

Soit le code de la figure 3.1. Ce code contient le dépendances :

- $e_{S,1} : S(i,j) \to T(i,j)$, $\mathcal{D}_{e_{S,1}} = \{(i,j) | 1 \le i \le N-1, 1 \le j \le N-1\}$;
- $e_{S,2} : S(i-1,j) \to T(i,j)$, $\mathcal{D}_{e_{S,2}} = \{(i,j) | 2 \le i \le N-1, 1 \le j \le N-1\}$;
- $e_{S,3} : S(i,j-1) \to T(i,j)$, $\mathcal{D}_{e_{S,3}} = \{(i,j) | 1 \le i \le N-1, 2 \le j \le N-1\}$;
- $e_{S,4} : S(i-1,j-1) \to T(i,j)$, $\mathcal{D}_{e_{S,4}} = \{(i,j) | 2 \le i \le N-1, 2 \le j \le N-1\}$;
- $e_{T,1} : T(i,j-1) \to S(i,j)$, $\mathcal{D}_{e_{T,1}} = \{(i,j) | 1 \le i \le N-1, 2 \le j \le N-1\}$;
- $e_{T,2} : T(i-1,j) \to S(i,j)$, $\mathcal{D}_{e_{T,2}} = \{(i,j) | 2 \le i \le N-1, 1 \le j \le N-1\}$.

Les fonctions d'ordonnancements sont (par hypothèse) :

$$\begin{aligned} \Theta_S(i,j) &= 2i + 2j \\ \Theta_T(i,j) &= 2i + 2j + 1 \end{aligned}$$

L'ensemble des valeurs vivantes à un instant t_0 à cause de la dépendance e est défini par :

$$\mathcal{D}_{S_e}(t_0, e) = \{\boldsymbol{j}_0 \in \mathcal{D}_{S_e} | \exists j \in \mathcal{D}_e, \boldsymbol{j}_0 = h_e(\boldsymbol{j}, N), \Theta(S_e, \boldsymbol{j}_0, N) \le t_0, \Theta(T_e, \boldsymbol{j}, N) \ge t_0\},$$

$$\text{avec ici} \quad \mathcal{D}_{S_e} = \{(i,j) | 1 \le i \le N-1, 1 \le j \le N-1\}.$$

Valeurs produites par l'instruction S

Dans la relation de dépendance $e_{S,1}$, chaque élément de tableau $A[i,j]$ doit être produit par l'instruction S avant t_0 et être consommé par l'instruction T après l'instant t_0. Autrement dit, on détermine les vecteurs d'itération $\boldsymbol{j}_o(i_0, j_0) \in \mathcal{D}_S$ tels que :

1. $\exists \boldsymbol{j} = (i,j) \in \mathcal{D}_e, h_{e_{S,1}}(i,j) = (i_0, j_0) \Leftrightarrow i_0 = i$ et $j_0 = j$;
2. L'élément $A[i,j]$ est produit avant l'instant t_0 : $\Theta_S(i_0, j_0) \le t_0$;
3. L'élément $A[i,j]$ est consommé après l'instant t_0 : $\Theta_T(i,j) \ge t_0$.

Les contraintes ci-dessus définissent le polyèdre :

$$\mathcal{D}_S(t_0, e_{S,1}) = \left\{ (i_0, j_0) \left| \begin{array}{l} 1 \le i_0 \le N-1, 1 \le j_0 \le N-1, \\ 1 \le i \le N-1, 1 \le j \le N-1, \\ 2i_0 + 2j_0 \le t_0, 2i + 2j + 1 \ge t_0, \\ i_0 = i, j_0 = j \end{array} \right. \right\}$$

De la même manière, on trouve les polyèdres définis pour les autres dépendances :

– La dépendance e_{S_2} est caractérisée par le polyèdre :

$$\mathcal{D}_S(t_0, e_{S,2}) = \left\{ (i_0, j_0) \;\middle|\; \begin{array}{l} 1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, 2 \leq i \leq N-1, 1 \leq j \leq N-1, \\ 2i_0 + 2j_0 \leq t_0, 2i + 2j + 1 \geq t_0, i_0 = i - 1, j_0 = j \end{array} \right\}$$

– La dépendance $e_{S,3}$ est caractérisée par le polyèdre :

$$\mathcal{D}_S(t_0, e_{S,3}) = \left\{ (i_0, j_0) \;\middle|\; \begin{array}{l} 1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, 1 \leq i \leq N-1, 2 \leq j \leq N-1, \\ 2i_0 + 2j_0 \leq t_0, 2i + 2j + 1 \geq t_0, i_0 = i, j_0 = j - 1 \end{array} \right\}$$

– La dépendance $e_{S,4}$ est caractérisée par le polyèdre :

$$\mathcal{D}_S(t_0, e_{S,4}) = \left\{ (i_0, j_0) \;\middle|\; \begin{array}{l} 1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, 2 \leq i \leq N-1, 2 \leq j \leq N-1, \\ 2i_0 + 2j_0 \leq t_0, 2i + 2j + 1 \geq t_0, i_0 = i - 1, j_0 = j - 1 \end{array} \right\}$$

Donc le nombre de valeurs produites par l'instruction S_e qui sont « en vie » à l'instant t_0 est :

$$\mathcal{D}_{S_e}(t_0) = \bigcup_{e|S_e=S} \mathcal{D}_{S_e}(t_0, e) = \mathcal{D}_S(t_0, e_{S,1}) \cup \mathcal{D}_S(t_0, e_{S,2}) \cup \mathcal{D}_S(t_0, e_{S,3}) \cup \mathcal{D}_S(t_0, e_{S,4}).$$

Pour déterminer cette union, on utilise les fonctions d'énumération d'image d'une union de polyèdres paramétrés de la librairie polyédrique Polilyb comme suit :

1. Au départ on trouve les images (projection sur j_0) de l'union des différents polyèdres, on fait appel à la fonction :
   `Polyhedron-Image (Polyhedron*Pol,Matrix*Func,unsigned NbMaxConstrs)`
   Cette fonction donne l'image d'un ensemble de polyèdres `Pol` par une transformation affine représentée par une matrice `Func`. Ce qui donne les images :

$$Imag1 = \{1 \leq i_0 \leq N-2, 1 \leq j_0 \leq N-2, t_0 - 5 \leq 2i_0 + 2j_0 \leq t_0\}$$
$$Imag2 = \{1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-2, t_0 - 3 \leq 2i_0 + 2j_0 \leq t_0\}$$
$$Imag3 = \{1 \leq i_0 \leq N-2, 1 \leq j_0 \leq N-1, t_0 - 3 \leq 2i_0 + 2j_0 \leq t_0\}$$
$$Imag4 = \{1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, t_0 - 1 \leq 2i_0 + 2j_0 \leq t_0\}$$

2. Ensuite, on fait l'union disjointe de ces images, par un simple appel à la fonction :
   `Disjoint-Domain (Polyhedron*Pol,Int*Flag,unsigned NbMaxRays)`
   Cette fonction donne une union disjointe de polyèdres égale à un ensemble de polyèdres (images) `Pol`. Une telle union disjointe permet de dénombrer le nombre exact de points entiers dans l'union (et d'éviter les points redondants). Ce qui nous donne les polyèdres suivants :

$$\begin{array}{rl} UD1 = & \{i_0 = N-1, 1 \leq j_0 \leq N-1, t_0 + 1 \leq 2j_0 + 2N \leq t_0 + 2\} \\ UD2 = & \{j_0 = N-1, 1 \leq i_0 \leq N-2, t_0 + 1 \leq 2i_0 + 2N \leq t_0 + 2\} \\ UD3 = & \{j_0 = N-1, 1 \leq i_0 \leq N-2, t_0 - 1 \leq 2i_0 + 2j_0 \leq t_0\} \\ UD4 = & \{1 \leq i_0 \leq N-2, 1 \leq j_0 \leq N-2, t_0 - 3 \leq 2i_0 + 2j_0 \leq t_0\} \\ UD5 = & \{1 \leq i_0 \leq N-2, 1 \leq j_0 \leq N-2, t_0 - 5 \leq 2i_0 + 2j_0 \leq t_0 - 4\} \end{array}$$

3. Enfin, on calcule le nombre de points de chacun des polyèdres de l'union des polyèdres. On fait appel à la fonction :
   `Domain_Enumerate (Polyhedron*D, Polyhedron*C, NbMaxRays, Chr**pn)`
   Cette fonction donne l'énumération de l'union disjointe des polyèdres. Ce qui donne les polynômes d'Ehrhart selon leurs domaines de validité :

$$\begin{aligned}
EP1 = &\quad t_0 + [-3, -4]_{t_0} \text{ si } 7 \leq t_0 \leq 8 \text{ et } N \geq 6 \\
EP2 = &\quad 2N - \frac{1}{2}t_0 + [1, \frac{3}{2}]_{t_0} \text{ si } 4N - 5 \leq t_0 \leq 4N - 4 \text{ et } N \geq 6 \\
EP3 = &\quad 2N - \frac{1}{2}t_0 + [3, \frac{7}{2}]_{t_0} \text{ si } 4N - 7 \leq t_0 \leq 4N - 6 \text{ et } N \geq 6 \\
EP4 = &\quad 6N - \frac{3}{2}t_0 + [-2, \frac{-1}{2}]_{t_0} \text{ si } 2N + 3 \leq t_0 \leq 4N - 8 \text{ et } N \geq 6 \\
EP5 = &\quad 4N - \frac{1}{2}t_0 + [-5, \frac{-9}{2}]_{t_0} \text{ si } 2N + 1 \leq t_0 \leq 2N + 2 \text{ et } N \geq 6 \\
EP6 = &\quad \frac{1}{2}t_0 + [-4, \frac{-9}{2}]_{t_0} \text{ si } 2N - 2 \leq t_0 \leq 2N \text{ et } N \geq 6 \\
EP7 = &\quad \frac{3}{2}t_0 + [-6, \frac{-15}{2}]_{t_0} \text{ si } 9 \leq t_0 \leq 2N - 2 \text{ et } N \geq 6
\end{aligned}$$

Valeurs produites par l'instruction T

- La dépendance $e_{T,1}$ est caractérisée par le polyèdre :

$$\mathcal{D}_{e_{T,1}}(t_0, e_{T,1}) = \left\{ (i_0, j_0) \;\middle|\; \begin{array}{l} 1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, 1 \leq i \leq N-1, 2 \leq j \leq N-1, \\ 2i_0 + 2j_0 \leq t_0, 2i + 2j + 1 \geq t_0, i_0 = i, j_0 = j - 1 \end{array} \right\}$$

- La dépendance $e_{T,2}$ par le polyèdre :

$$\mathcal{D}_{e_{T,2}}(t_0, e_{T,2}) = \left\{ (i_0, j_0) \;\middle|\; \begin{array}{l} 1 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-1, 2 \leq i \leq N-1, 1 \leq j \leq N-1, \\ i_0 - 1 + j_0 \leq t_0, i + j \geq t_0, i_0 = i - 1, j_0 = j \end{array} \right\}$$

Donc le nombre de valeurs produites par l'instruction T_e qui sont « en vie » à l'instant t_0 est :

$$D_T(t_0) = \bigcup_{e | T_e = T} \mathcal{D}_{T_e}(t_0, e) = \mathcal{D}_T(t_0, e_{T,1}) \cup \mathcal{D}_T(t_0, e_{T,2})$$

De la même façon que pour l'instruction S, on trouve le nombre de valeurs « en vie » comme suit :

1. L'image de ces deux polyèdres est :

$$Imag1 = \{1 \leq i_0 \leq N-2, 2 \leq j_0 \leq N-1, t_0 - 2 \leq 2i_0 + 2j_0 \leq t_0 - 1\}$$

$$Imag2 = \{2 \leq i_0 \leq N-1, 1 \leq j_0 \leq N-2, t_0 - 2 \leq 2i_0 + 2j_0 \leq t_0 - 1\}$$

2. L'union disjointe de ces images est :

$$\begin{aligned}
UD1 = &\quad \{i_0 = N-1, 1 \leq j_0 \leq N-2, t_0 \leq 2j_0 + 2N \leq t_0 + 1\} \\
UD2 = &\quad \{j_0 = 1, \frac{t_0 - 4}{2} \leq i_0 \leq \frac{t_0 - 3}{2}, 2 \leq i_0 \leq N - 2\} \\
UD3 = &\quad \{2 \leq i_0 \leq N-2, 2 \leq j_0 \leq N-2, t_0 - 2 \leq 2i_0 + 2j_0 \leq t_0 - 1\} \\
UD4 = &\quad \{j_0 = N-1, 1 \leq i_0 \leq N-2, t_0 \leq 2i_0 + 2j_0 \leq t_0 + 1\} \\
UD5 = &\quad \{i_0 = 1, 2 \leq j_0 \leq N-2, \frac{t_0 - 4}{2} \leq j_0 \leq \frac{t_0 - 3}{2}\}
\end{aligned}$$

3. On dénombre les points contenus dans cette union disjointe. Le résultat est exprimé par des polynômes d'Ehrhart, un par domaine de validité :

$$
\begin{aligned}
EP1 = &\quad \frac{1}{2}t_0 + [-4, -\frac{7}{2}]_{t_0} \text{ si } t_0 = 2N+1, t_0 \geq 13 \\
EP2 = &\quad 2 \text{ si } 8 \leq t_0 \leq 9, N \geq 6 \\
EP3 = &\quad \frac{1}{2}t_0 + [-2, -\frac{3}{2}]_{t_0} \text{ si } 10 \leq t_0 \leq 2N-1, N \geq 6 \\
EP4 = &\quad \frac{1}{2}t_0 + [-5, \frac{-9}{2}]_{t_0} \text{ si } t_0 = 2N \text{ et } t_0 \geq 12 \\
EP5 = &\quad 2 \text{ si } t_0 = 4N-5 \text{ et } t_0 \geq 19 \\
EP6 = &\quad 1 \text{ si } 4N-7 \leq t_0 \leq 4N-6 \\
EP7 = &\quad 2N - \frac{1}{2}t_0 + [0, \frac{-1}{2}]_{t_0} \text{ si } 2N+2 \leq t_0 \leq 4N-7
\end{aligned}
$$

Donc le nombre de valeurs « en vie » du programme à un instant donné est constitué de la somme des polynômes obtenus pour chacune des deux instructions. Un polynôme d'Ehrhart dénombrant le nombre de points d'un polyèdre atteignant ses valeurs extrémales aux extrémités de son domaine de validité, on calcule le nombre maximal de valeurs en vie à un instant donné en ne considérant que les dates extrémités de domaines de validité. Les valeurs à considérer sont regoupées dans le tableau 3.1. Le nombre maximal de valeurs vivantes est égal à $6N-13$ pour $t_0 = 2N-3$.

t_0	7	8	9	10	12	13	19	$2N-3$	$2N-2$
S	3	5	6	9	12	12	22	$3N-4$	$3N-9$
T	0	2	2	3	1	3	2	$3N-9$	$3N-9$
$S+T$	3	7	8	12	13	15	24	**6N-13**	$6N-18$
t_0	$2N-1$	$2N$	$2N+1$	$2N+2$	$4N-8$	$4N-7$	$4N-6$	$4N-5$	$4N-4$
S	$N-4$	$N-4$	$3N-5$	$3N-6$	10	7	6	4	3
T	$N-3$	$N-5$	$N-3$	$N-1$	4	3	1	2	0
$S+T$	$2N-7$	$2N-9$	$4N-8$	$4N-7$	14	10	7	6	3

TAB. 3.1 – Nombre de valeurs en vie pour l'exemple 1 aux dates extrémités de domaine de validité.

3.2.3 Application à l'exemple 2

Le nid de boucles étudié est présenté figure 3.2.

```
    For i=2,2N−1
S1:     A[2i]   =C[i−1]+ C[i−2]
S2:     B[i+N] = A[i]
S3:     C[i]     = B[i]

    EndFor
```

FIG. 3.2 – Exemple 2.

Dépendances

Les dépendances dans ce code sont :

$$
\begin{aligned}
e_{S_1} : &\quad S_1(i) \to S_2(2i) &\quad \text{sur le domaine } \mathcal{D}_{e_{S_1}} = \{i | 2 \leq i \leq N-1\} &\quad (3.1)\\
e_{S_2} : &\quad S_2(i-N) \to S_3(i) &\quad \text{sur le domaine } \mathcal{D}_{e_{S_2}} = \{i | 2+N \leq i \leq 2N-1\} &\quad (3.2)\\
e_{S_{3,1}} : &\quad S_3(i-1) \to S_1(i) &\quad \text{sur le domaine } \mathcal{D}_{e_{S_{3,1}}} = \{i | 3 \leq i \leq 2N-1\} &\quad (3.3)\\
e_{S_{3,2}} : &\quad S_3(i-2) \to S_1(i) &\quad \text{sur le domaine } \mathcal{D}_{e_{S_{3,2}}} = \{i | 4 \leq i \leq 2N-1\} &\quad (3.4)
\end{aligned}
$$

Ordonnancements

Nous considérons les ordonnancements :

$$
\begin{aligned}
\Theta_{S_1}(i) &= 2i\\
\Theta_{S_2}(i) &= i+1\\
\Theta_{S_3}(i) &= i
\end{aligned}
$$

Ces ordonnancements sont légaux car ils satisfont toutes les contraintes de dépendance.

- La dépendance e_{S1} génère la contrainte d'ordonnancement suivante :

$$\Theta_{S_2}(2i) - \Theta_{S_1}(i) = (2i+1) - 2i = 1 \geq 1$$

- La dépendance e_{S_2} génère la contrainte d'ordonnancement :

$$\Theta_{S_3}(i) - \Theta_{S_2}(i-N) = i - (i-N) = N \geq 1$$

- La dépendance $e_{S_{3,1}}$ génère la contrainte d'ordonnancement :

$$\Theta_{S_1}(i) - \Theta_{S_3}(i-1) = 2i - (i-1) = i+1 \geq 1$$

- La dépendance $e_{S_{3,2}}$ génère la contrainte d'ordonnancement :

$$\Theta_{S_1}(i) - \Theta_{S_3}(i-2) = 2i - (i-2) = i+2 \geq 1$$

Valeurs produites par l'instruction S_1

Dans la relation de dépendance e_{S_1}, chaque élément de tableau $A[2i]$ doit être produit par l'instruction S avant d'être consommé par l'instruction T. Autrement dit, il existe un vecteur d'itération $i_0 \in \mathcal{D}_S$ tel que :

1. $\exists i \in \mathcal{D}_e, 2i_0 = i$;
2. L'élément $A[2i]$ est produit avant l'instant t_0 : $\Theta_{S_1}(i_0) \leq t_0$;
3. L'élément $A[2i]$ est consommé après l'instant t_0 : $\Theta_{S_2}(i) \geq t_0$;

ce qui définit le polyèdre :

$$\mathcal{D}_{S_1}(t_0, e_{S_1}) = \{2 \leq i_0 \leq N-1, 2 \leq i \leq 2N-1, 2i_0 \leq t_0, i+1 \geq t_0, 2i_0 = i\}.$$

Donc le nombre de valeurs produites par l'instruction S_1 qui sont « en vie » à l'instant t_0 est :

$$D_{S_1}(t_0) = \mathcal{D}_{S_1}(t_0, e_{S_1})$$

Pour déterminer ce dernier, on utilise les fonctions d'énumération d'image d'une union de polyèdres paramétrés (ou bien l'énumération de l'image d'un seul polyèdre parce qu'on a un seul polyèdre) :

1. L'image (projection sur i_0) du polyèdre $\mathcal{D}_{S_1}(t_0, e_{S_1})$ est :

$$Img(\mathcal{D}_S(t_0, e_{S_1})) = \{2 \leq i_0 \leq N-1, 2i_0 \leq t_0 \leq 2i_0 + 1\}.$$

2. L'union disjointe de cette image donne l'image elle-même :

$$Img(\mathcal{D}_S(t_0, e_{S_1})) = \{2 \leq i_0 \leq N-1, 2i_0 \leq t_0 \leq 2i_0+1\}.$$

3. L'énumération d'image donne le polynôme d'Ehrhart (suivant le domaine de validité) suivant :

$$EP_1 = 1 \text{ si : } 5 \leq t_0 \leq 2N-2$$

$$EP_2 = 1 \text{ si : } 4 \leq t_0 \leq 5, N = 3$$

Valeurs produites par l'instruction S_2

La dépendance e_{S_2} est caractérisée par le polyèdre :

$$\mathcal{D}_{S_2}(t_0, e_{S_2}) = \{2 \leq i_0 \leq N-1, 2 \leq i \leq 2N-1, i_0+1 \leq t_0, i \geq t_0, i_0 + N = i\}.$$

Donc le nombre de valeurs produites par l'instruction S_2 qui sont « en vie » à l'instant t_0 est :

$$\mathcal{D}_{S_2}(t_0) = \mathcal{D}_{S_2}(t_0, e_{S_2}).$$

1. L'image (projection sur i_0) du polyèdre $D_{S_2}(t_0, e_{S_2})$ est :

$$Img(\mathcal{D}_{S_2}(t_0, e_{S_2})) = \{2 \leq i_0 \leq N-1, i_0 \leq t_0 \leq i_0 + N\}.$$

2. L'union disjointe de cette image donne l'image elle même :

$$UDImag(\mathcal{D}_{S_2}(t_0, e_{S_1})) = \{2 \leq i_0 \leq N-1, i_0 \leq t_0 \leq i_0 + N\}.$$

3. L'énumération d'image donne deux polynômes d'Ehrhart suivant leurs domaines de validité :

$$EP_1 = t_0 - 1 \text{ si : } 2 \leq t_0 \leq N-1$$

$$EP_2 = 2N - t_0 \text{ si : } N+2 \leq t_0 \leq 2N-1$$

Valeurs produites par l'instruction S_3

La dépendance $e_{S_{3,1}}$ est caractérisée par le polyèdre :

$$\mathcal{D}_S(t_0, e_{S_{3,1}}) = \{3 \leq i_0 \leq 2N-1, 2 \leq i \leq 2N-1, i_0 \leq t_0, 2i \geq t_0, i_0 = i-1\}.$$

La dépendance $e_{S_{3,2}}$ est caractérisée par le polyèdre :

$$\mathcal{D}_S(t_0, e_{S_{3,2}}) = \{4 \leq i_0 \leq 2N-1, 2 \leq i \leq 2N-1, i_0 \leq t_0, 2i \geq t_0, i_0 = i-2\}.$$

1. Donc le nombre de valeurs produites par l'instruction S_3 qui sont « en vie » à l'instant t_0 est :

$$D_{S_3}(t_0) = \mathcal{D}_{S_3}(t_0, e_{S_{3,1}}) \bigcup \mathcal{D}_{S_{3,2}}(t_0, e_{S_{3,2}}).$$

2. L'image (projection sur i_0) des polyèdres $\mathcal{D}_{S_{3,1}}(t_0, e_{S_{3,1}})$ et $D_{S_{3,2}}(t_0, e_{S_{3,2}})$ est :

$$Img(\mathcal{D}_{S_3}(t_0, e_{S_{3,1}})) = \{2 \leq i_0 \leq 2N-3, i_0 \leq t_0 \leq 2i_0 + 4\}$$

$$Img(\mathcal{D}_{S_3}(t_0, e_{S_{3,2}})) = \{2 \leq i_0 \leq 2N-2, i_0 \leq t_0 \leq 2i_0 + 2\}$$

3. L'union disjointe de ces images donne les mêmes images (les images d'origine sont disjointes) :

$$\begin{aligned} UDImg1 &= \{i_0 = 2N-2, 2N-2 \leq t_0 \leq 4N-2, N \geq 2\} \\ UDImg2 &= \{2 \leq i_0 \leq 2N-3, i_0 \leq t_0 \leq i_0 + 2\} \\ UDImg3 &= \{2 \leq i_0 \leq 2N-3, 2i_0 + 3 \leq t_0 \leq 2i_0 + 4\} \end{aligned}$$

4. L'énumération d'image donne les polynômes d'Ehrhart selon leurs domaines de validité :

$$
\begin{aligned}
&EP_1 = 1 \text{ si : } 2 \leq t_0 \leq 5 \text{ et } N = 2 \\
&EP_2 = \tfrac{1}{2}t_0 + [3, \tfrac{5}{2}]_{t_0} \text{ si : } 8 \leq t_0 \leq 2N-3 \\
&EP_3 = 2N - \tfrac{1}{2}t_0 + [1, \tfrac{1}{2}]_{t_0} \text{ si : } 2N-2 \leq t_0 \leq 4N-4 \\
&EP_4 = 2 \text{ si : } t_0 = 4N-3 \\
&EP_5 = 2N-3 \text{ si : } t_0 \leq 8 \text{ et } t_0 \geq 4N-3 \\
&EP_6 = 2N-3 \text{ si : } t_0 \leq 6 \text{ et } t_0 \geq 2N-2 \\
&EP_7 = t_0 - 1 \text{ si : } 2 \leq t_0 \leq 2N-3 \\
&EP_8 = \tfrac{1}{2}t_0 + [2, \tfrac{3}{2}]_{t_0} \text{ si : } 6 \leq t_0 \leq 2N-3 \\
&EP_9 = 2N - \tfrac{1}{2}t_0 + [0, \tfrac{1}{2}]_{t_0} \text{ si : } 2N-2 \leq t_0 \leq 4N-4
\end{aligned}
$$

Donc, le nombre maximal de valeurs « en vie » du programme est constitué de la somme des polynômes résultats des dépendances entre e_{S_1}, e_{S_2} et e_{S_3}, et est donné par le tableau 3.2. Ce qui nous donne le nombre maximal de valeurs simultanément en vie, et donc la taille mémoire minimale pour exécuter notre programme : $3N-1$.

t_0	2	5	6	7	8	$N-1$
S_1	$N-2$	$N-2$	1	1	1	1
S_2	1	4	5	6	7	$N-2$
S_3	$2N-3$	$2N-3$	$2N-3$	$2N-3$	$2N-3$	$1/2.N-1$
$S_1+S_2+S_3$	$3N-4$	**3N-1**	$2N+3$	$2N+4$	$2N+5$	$3/2.N-2$
t_0	$N+2$	$2N-3$	$2N-2$	$2N-1$	$4n-4$	$4N-3$
S_1	1	1	$N-2$	$N-2$	$N-2$	$N-2$
S_2	$N-2$	3	2	1	0	0
S_3	$1/2N$	$N+2$	$N+1$	$2N-3$	$2N-3$	$2N-3$
$S_1+S_2+S_3$	$3/2.N-1$	$N+6$	$2N+1$	$3N-4$	$3N-5$	$3N-5$

TAB. 3.2 – Nombre de valeurs en vie pour l'exemple 2 aux dates extrémités de domaine de validité.

3.3 Minimisation de la durée de vie des valeurs : cas monodimensionnel

Dans cette partie, nous allons étudier la durée de vie des variables en vue de la minimiser, en se basant sur les contraintes de dépendances et la fonction d'ordonnancement. Nous allons faire l'étude dans le cas d'un ordonnancement monodimensionnel. Soit la relation de dépendance suivante :

$$e : \forall \boldsymbol{j} \in \mathcal{D}_e, S_e(h_e(\boldsymbol{j}, N)) \to T_e(\boldsymbol{j})$$

cette relation de dépendance engendre la contrainte d'ordonnancement suivante entre les instructions du programme :

$$\forall \boldsymbol{j} \in \mathcal{D}_e : \quad \Theta(T_e, \boldsymbol{j}, N) - \Theta(S_e, h_e(\boldsymbol{j}, N), N) - 1 \geq 0.$$

tandis que la durée de vie de la valeur correspondant à cette dépendance (pour un vecteur d'itération $j \in \mathcal{D}_e$ donné) est définie par la différence entre la date de consommation de la valeur et la date de sa production :

$$\Theta(T_e, \boldsymbol{j}, N) - \Theta(S_e, h_e(\boldsymbol{j}, N), N).$$

On cherche à minimiser la durée de vie des valeurs qui génèrent la relation de dépendance e. Pour ce faire, on majore ces durées de vie par une borne, puis on minimisera cette borne (procédé classique en programmation linéaire). Soit d_{S_e} cette borne. L'expression de la durée de vie étant une fonction affine des paramètres et du vecteur d'itération, lequel est défini sur un domaine potentiellement paramétré, le maximum des durées de vie est une fonction affine des paramètres. On note donc $d_{S_e} = d_{S_e,1}N + d_{S_e,0}$ où $d_{S_e,0}$ est une constante et $d_{S_e,1}$ un vecteur constant. Dans ce cas on peut écrire :

$$\forall \boldsymbol{j} \in \mathcal{D}_e, d_{S_e,1}N + d_{S_e,0} \geq \Theta(T_e, \boldsymbol{j}, N) - \Theta(S_e, h_e(\boldsymbol{j}, N), N)$$
$$\Leftrightarrow \forall \boldsymbol{j} \in \mathcal{D}_e, d_{S_e,1}N + d_{S_e,0} - \Theta(T_e, \boldsymbol{j}, N) + \Theta(S_e, h_e(\boldsymbol{j}, N), N) \geq 0.$$

Cette inégalité devant être vérifiée sur tout le domaine $\mathcal{D}_e$ qui est potentiellement paramétré, nous nous trouvons devant le même problème que celui rencontré lors de recherche d'ordonnancements : la résolution d'un système potentiellement infini d'(in)égalités linéaires. Pour simplifier ce système nous utilisons les mêmes procédés que lors de la recherche classique d'ordonnancements (cf. [22, 53]). L'inégalité ci-dessus est affine positive sur le polyèdre $\mathcal{D}_e$. Donc on peut appliquer le lemme de Farkas (cf. section 2.3.5). Nous notons :

$$\mathcal{D}_e = \{x | \forall i \in [1, p_e], A_{e,i}.x + B_{e,i}.N + C_{e,i} \geq 0\}$$

($\mathcal{D}_e$ est un domaine défini par p_e inégalités). D'après le lemme de Fakas, il existe des valeurs appelées *multiplicateurs de Farkas*, $\lambda_{e,0}, \lambda_{e,1}, \cdots, \lambda_{e,p_e} \geq 0$, telles que :

$$d_{S_e,1}N + d_{S_e,0} - \Theta(T_e, \boldsymbol{j}, N) + \Theta(S_e, h_e(\boldsymbol{j}, N), N) \equiv \lambda_{e,0} + \sum_{i=1}^{p_e} \lambda_{e,i}(A_{e,i}.\boldsymbol{j} + B_{e,i}.N + C_{e,i}).$$

Par identification des coefficients de $\boldsymbol{j}$, de N et des constantes, nous obtenons finalement un système composé d'un nombre fini d'équations et d'inéquations qui peut être résolu au moyen de n'importe quel système de recherche de solutions pour les programmes linéaires. Il nous reste juste à spécifier notre fonction objective. Nous avons une durée de vie par instruction, et nous cherchons à minimiser la mémoire totale utilisée par le programme, soit la somme des durées de vie (nous justifions cette somme par l'utilisation des durées de vie, au chapitre suivant, pour réduire la mémoire utilisée) :

$$\sum_S \mid d_{S,1}.N + d_{S,0} \mid$$

Nous voulons bien évidemment que notre fonction objective soit linéaire, or le produit de la variable $d_{S,1}$ par le vecteur de paramètres N est quadratique. C'est pourquoi nous omettons les paramètres dans la fonction objective. Afin de conserver l'importance des paramètres, nous dissocions la minimisation des parties linéaires et des parties constantes des durées d_s. Nous choisissons alors comme fonction objective :

$$\min_{\preceq}(\sum_S \|d_{S,1}\|, \sum_S \mid d_{S,0} \mid).$$

Cette fonction est constituée d'un minimum lexicographique et, pour résoudre ce problème, on a le choix entre deux normes linéaires :

$$\|d_{S,1}\| = \|d_{S,1}\|_\infty = \max_i \mid d_{S,1}^{(i)} \mid$$

$$\|d_{S,1}\| = \|d_{S,1}\|_1 = \sum_i \mid d_{S,1}^{(i)} \mid$$

où $d_{S,1}^{(i)}$ désigne la i^{e} composante du vecteur $d_{S,1}$.

Remarque : Évidemment, dans ce cas, on espère d'avoir $d_{S,1} = 0$ pour obtenir un minimum constant en éliminant le paramètre N.

3.3.1 Application à l'exemple 1

Supposons que les ordonnancements sont de la forme suivante :
$\Theta_S(i,j) = ai + bj + cN + g$
$\Theta_T(i,j) = di + ej + fN + h$

- Les contraintes d'ordonnancements sont :
  1. $\Theta_T(i,j) - \Theta_S(i,j) = (d-a)i + (e-b)j + (f-c)N + (h-g) \geq 1$,
     $\mathcal{D}_{e_{S,1}} = \{(i,j) | 1 \leq i \leq N-1, 1 \leq j \leq N-1\}$
  2. $\Theta_T(i,j) - \Theta_S(i-1,j) = (d-a)i + (e-b)j + (f-c)N + (h+a-g) \geq 1$,
     $\mathcal{D}_{e_{S,2}} = \{(i,j) | 2 \leq i \leq N-1, 1 \leq j \leq N-1\}$
  3. $\Theta_T(i,j) - \Theta_S(i,j-1) = (d-a)i + (e-b)j + (f-c)N + (h+b-g) \geq 1$,
     $\mathcal{D}_{e_{S,3}} = \{(i,j) | 1 \leq i \leq N-1, 2 \leq j \leq N-1\}$
  4. $\Theta_T(i,j) - \Theta_S(i-1,j-1) = (d-a)i + (e-b)j + (f-c)N + (h+a+b-g) \geq 1$,
     $\mathcal{D}_{e_{S,4}} = \{(i,j) | 2 \leq i \leq N-1, 2 \leq j \leq N-1\}$
  5. $\Theta_S(i,j) - \Theta_T(i,j-1) = (a-d)i + (b-e)j + (c-f)N + (g+e-h) \geq 1$,
     $\mathcal{D}_{e_{T,1}} = \{(i,j) | 1 \leq i \leq N-1, 2 \leq j \leq N-1\}$
  6. $\Theta_S(i,j) - \Theta_T(i-1,j) = (a-d)i + (b-e)j + (c-f)N + (g+d-h) \geq 1$,
     $\mathcal{D}_{e_{T,2}} = \{(i,j) | 2 \leq i \leq N-1, 1 \leq j \leq N-1\}$

  L'équation (1) (resp. 5) devant être vraie pour toute valeur de $N \geq 1$. On déduit que $f - c \geq 0$ (resp. $c - f \geq 0$). Donc $f = c$, le paramètre n'a aucune incidence sur l'ordonnancement des opérations et *a fortiori* sur les durées de vie et on peut, sans perte de généralité, prendre $f = c = 0$.
  L'équation (1) doit être vraie pour $(i,j) = (N,1)$ et ce pour tout N. D'où $d - a \geq 0$. Un raisonnement similaire sur l'équation (5) nous donne $d - a \geq 0$ d'où $a = d$. De même on peut montrer que $b = e$.
  Les ordonnancements ont donc la forme suivante :

  $$\Theta_S(i,j) = ai + bj + g$$

  $$\Theta_T(i,j) = ai + bj + h$$

  Les contraintes de dépendances se récrivent alors :
  1. $(h - g) \geq 1$
  2. $(h + a - g) \geq 1$
  3. $(h + b - g) \geq 1$
  4. $(h + a + b - g) \geq 1$
  5. $(g + b - h) \geq 1$
  6. $(g + a - h) \geq 1$

- Les équations sur les durées de vie s'écrivent alors :

  $$\begin{cases} d_{S,1}.N + d_{S,0} \geq (h-g) \\ d_{S,1}.N + d_{S,0} \geq (h+a-g) \\ d_{S,1}.N + d_{S,0} \geq (h+b-g) \\ d_{S,1}.N + d_{S,0} \geq (h+a+b-g) \\ d_{T,1}.N + d_{T,0} \geq (g+b-h) \\ d_{T,1}.N + d_{T,0} \geq (g+a-h) \end{cases}$$

  De $(h + a - g) \geq 1$ et $(g + a - h) \geq 1$ on tire $a \geq 1$. De $(h + b - g) \geq 1$ et $(g + b - h) \geq 1$ on tire $b \geq 1$.
  Les contraintes sur les durées de vie deviennent :

  $$\begin{cases} d_{S,1}.N + d_{S,0} \geq (h-g) \\ d_{T,1}.N + d_{T,0} \geq (g+b-h) \\ d_{T,1}.N + d_{T,0} \geq (g+a-h) \end{cases}$$

  Donc dans une solution optimale :

  $$d_{S,1} = d_{T,1} = 0, d_{S,0} = h - g \text{ et } d_{T,0} = g - h + max(a,b).$$

On cherche donc une solution au problème :

$$\min\left(\mid h-g \mid + \mid g-h+\max(a,b) \mid\right).$$

Tel que :

$$\begin{cases} a \geq 1 \\ b \geq 1 \\ h-g \geq 1 \\ g+b-h \geq 1 \\ g+a-h \geq 1 \end{cases}$$

La solution optimale est :

$$\begin{cases} g-h=1 \\ g-h+\max(a,b)=1 \end{cases} \Leftrightarrow \begin{cases} h=1+g \\ a=2 \\ b=2 \end{cases}$$

d'où :

$$\Theta_S(i,j) = 2(i+j) \text{ et } \Theta_T(i,j) = 2(i+j)+1 \text{ (avec g=0).}$$

Les durées de vie maximales sont alors :

1. $\Theta_T(i,j) - \Theta_S(i,j) = 2(i+j)+1-2(i+j) = 1$
2. $\Theta_T(i,j) - \Theta_S(i-1,j) = 2(i+j)+1-2(i-1+j) = 3$
3. $\Theta_T(i,j) - \Theta_S(i,j-1) = 2(i+j)+1-2(i+j-1) = 3$
4. $\Theta_T(i,j) - \Theta_S(i-1,j-1) = 2(i+j)+1-2(i-1+j-1) = 5$
5. $\Theta_S(i,j) - \Theta_T(i,j-1) = 2(i+j)-(2(i+j-1)+1) = 1$
6. $\Theta_S(i,j) - \Theta_T(i-1,j) = 2(i+j)-(2(i-1+j)+1) = 1$

3.3.2 Application à l'exemple 2

Relations de dépendance

$$e_{S_1}: \quad S_1(i) \to S_2(2i) \quad \text{sur le domaine } \mathcal{D}_{e_{S_1}} = \{i | 2 \leq i \leq N-1\} \tag{3.5}$$

$$e_{S_2}: \quad S_2(i-N) \to S_3(i) \quad \text{sur le domaine } \mathcal{D}_{e_{S_2}} = \{i | 2+N \leq i \leq 2N-1\} \tag{3.6}$$

$$e_{S_{3,1}}: \quad S_3(i-1) \to S_1(i) \quad \text{sur le domaine } \mathcal{D}_{e_{S_{3,1}}} = \{i | 3 \leq i \leq 2N-1\} \tag{3.7}$$

$$e_{S_{3,2}}: \quad S_3(i-2) \to S_1(i) \quad \text{sur le domaine } \mathcal{D}_{e_{S_{3,2}}} = \{i | 4 \leq i \leq 2N-1\} \tag{3.8}$$

Expressions des ordonnancements

Nous supposons que les ordonnancements sont de la forme :

$$\begin{aligned} \Theta_{S_1}(i) &= a_1 + b_1 N + c_1 \\ \Theta_{S_2}(i) &= a_2 + b_2 N + c_2 \\ \Theta_{S_3}(i) &= a_3 + b_3 N + c_3 \end{aligned}$$

Délais induits par les ordonnancements

Le délai induit par un ordonancement sur une dépendance est égal à la différence des temps d'exécution des deux opérations en dépendance.

e_1 **:** $\Theta_{S_2}(2i) - \Theta_{S_1}(i) = (2a_2 - a_1)i + (b_2 - b_1)N + c_2 - c_1$, $i \in \mathcal{D}_{e_1} = \{i | 2 \leq i \leq N-1\}$.

e_2 **:** $\Theta_{S_3}(i) - \Theta_{S_2}(i-N) = (a_3 - a_2)i + (b_3 - b_2 + a_2)N + c_3 - c_2$, $i \in \mathcal{D}_{e_2} = \{i | 2+N \leq i \leq 2N-1\}$.

e_3 : $\Theta_{S_1}(i) - \Theta_{S_3}(i-1) = (a_1 - a_3)i + (b_1 - b_3)N + c_1 - c_3 + a_3$, $i \in \mathcal{D}_{e_3} = \{i | 3 \leq i \leq 2N-1\}$.
e_4 : $\Theta_{S_1}(i) - \Theta_{S_3}(i-2) = (a_1 - a_3)i + (b_1 - b_3)N + c_1 - c_3 + 2a_3$, $i \in \mathcal{D}_{e_4} = \{i | 4 \leq i \leq 2N-1\}$.

Les contraintes de dépendances s'expriment facilement en fonction du délai : « délai supérieur ou égal à un ». La majoration des délais de vie des valeurs, s'exprime tout aussi facilement : « majorant supérieur ou égal au délai ». Pour les deux types de contraintes, les inéquations obtenues doivent être valables sur tout un domaine d'existence, qui se trouve être paramétré. Une fois de plus nous nous devons de « linéariser » cet ensemble de contraintes potentiellement infini. Pour ce faire nous allons utiliser ici la méthode des sommets (cf. section 2.3.6) plutôt que la forme affine du lemme de Farkas (afin d'obtenir un résultat qui soit encore lisible par l'être humain).

Les contraintes que nous avons à satisfaire sont de la forme :

$$d_{S,1}N + d_{S,0} \geq (\alpha i + \beta N + \gamma) - (\lambda i + \mu N + \nu) \geq 1, \quad x + yN \leq i \leq u + vN$$

Chaque domaine d'existence d'une dépendance étant défini par deux équations sur l'indice d'itération i, il nous suffit (d'après la méthode des sommets, cf. section 2.3.6) de remplacer la quantification sur i par une instantiation de l'équation pour chacune des deux valeurs extrémales de i. Pour éliminer le paramètre N de la même manière, on commence par restreindre notre étude au cas $N \geq 3$. Il ne nous reste plus alors qu'à instancier les équations obtenus précédemment avec le cas $N = 3$ et à considérer la valeur du coefficient de N (les coefficients doivent vérifier le sens de l'inéquation : pour que l'inéquation $a + bN \geq c + dN$ soit vérifiée, il faut que l'on ait $b \geq d$). Par ce procédé on obtient les inéquations présentées figure 3.3.

Programme linéaire

Pour résoudre notre problème il nous suffit maintenant de collecter toutes les inéquations contraignant notre problème et de leur ajouter la fonction objective minimisant les durées de vie des variables. On obtient ainsi le programme linéaire présenté figure 3.4.

Résolution

Pour résoudre le système de la figure 3.4 on peut utiliser un logiciel tel que SPPoC [8] qui nous donne comme ordonnancement :

$$\begin{aligned} \Theta_{S_1}(i) &= i+1 \\ \Theta_{S_2}(i) &= N+1 \\ \Theta_{S_3}(i) &= i \end{aligned}$$

et comme durées de vie maximale :

$$\begin{aligned} d_{S_1,1}N + d_{S_1,0} &= N-2 \\ d_{S_2,1}N + d_{S_2,0} &= N-2 \\ d_{S_3,1}N + d_{S_3,0} &= 3 \end{aligned}$$

1. $d_{S_1,1}N+d_{S_1,0} \geq \Theta_{S_2}(2i)-\Theta_{S_1}(i) = (2a_2-a_1)i+(b_2-b_1)N+c_2-c_1 \geq 1$ avec $i \in \mathcal{D}_{e_1} = \{i|2 \leq i \leq N-1\}$.
   (a) $i = 2 : d_{S_1,1}N + d_{S_1,0} \geq (b_2 - b_1)N + 4a_2 - 2a_1 + c_2 - c_1 \geq 1$.
      i. $N = 3 : 3d_{S_1,1} + d_{S_1,0} \geq 4a_2 - 2a_1 + 3b_2 - 3b_1 + c_2 - c_1 \geq 1$.
      ii. Coefficients de $N : d_{S_1,1} \geq (b_2 - b_1) \geq 0$.
   (b) $i = N - 1 : d_{S_1,1}N + d_{S_1,0} \geq (2a_2 - a_1 + b_2 - b_1)N + c_2 - c_1 - 2a_2 + a_1 \geq 1$.
      i. $N = 3 : 3d_{S_1,1} + d_{S_1,0} \geq 4a_2 - 2a_1 + 3b_2 - 3b_1 + c_2 - c_1 \geq 1$.
      ii. Coefficients de $N : d_{S_1,1} \geq (2a_2 - a_1 + b_2 - b_1) \geq 0$.
2. $d_{S_2,1}N + d_{S_2,0} \geq \Theta_{S_3}(i) - \Theta_{S_2}(i - N) = (a_3 - a_2)i + (b_3 - b_2 + a_2)N + c_3 - c_2 \geq 1$ avec $i \in \mathcal{D}_{e_2} = \{i|2 + N \leq i \leq 2N - 1\}$.
   (a) $i = N + 2 : d_{S_2,1}N + d_{S_2,0} \geq (a_3 + b_3 - b_2)N + 2a_3 - 2a_2 + c_3 - c_2 \geq 1$.
      i. $N = 3 : 3d_{S_2,1} + d_{S_2,0} \geq 5a_3 - 2a_2 + 3b_3 - 3b_2 + c_3 - c_2 \geq 1$.
      ii. Coefficients de $N : d_{S_2,1} \geq (a_3 + b_3 - b_2) \geq 0$.
   (b) $i = 2N - 1 : d_{S_2,1}N + d_{S_2,0} \geq (2a_3 - a_2 + b_3 - b_2)N + c_3 - c_2 + a_2 - a_3 \geq 1$.
      i. $N = 3 : 3d_{S_2,1} + d_{S_2,0} \geq 5a_3 - 2a_2 + 3b_3 - 3b_2 + c_3 - c_2 \geq 1$.
      ii. Coefficients de $N : d_{S_2,1} \geq (2a_3 - a_2 + b_3 - b_2) \geq 0$.
3. $d_{S_3,1}N + d_{S_3,0} \geq \Theta_{S_1}(i) - \Theta_{S_3}(i - 1) = (a_1 - a_3)i + (b_1 - b_3)N + c_1 - c_3 + a_3 \geq 1$ avec $i \in \mathcal{D}_{e_3} = \{i|3 \leq i \leq 2N - 1\}$.
   (a) $i = 3 : d_{S_3,1}N + d_{S_3,0} \geq (b_1 - b_3)N + c_1 - c_3 + 3a_1 - 2a_3 \geq 1$.
      i. $N = 3 : 3d_{S_3,1} + d_{S_3,0} \geq 3a_1 - 2a_3 + 3b_1 - 3b_3 + c_1 - c_3 \geq 1$.
      ii. Coefficients de $N : d_{S_3,1} \geq (b_1 - b_3) \geq 0$.
   (b) $i = 2N - 1 : d_{S_3,1}N + d_{S_3,0} \geq (2a_1 - 2a_3 + b_1 - b_3)N + 2a_3 - a_1 + c_1 - c_3 \geq 1$.
      i. $N = 3 : 3d_{S_3,1} + d_{S_3,0} \geq 5a_1 - 4a_3 + 3b_1 - 3b_3 + c_1 - c_3 \geq 1$.
      ii. Coefficients de $N : d_{S_3,1} \geq (2a_1 - 2a_3 + b_1 - b_3) \geq 0$.
4. $d_{S_3,1}N + d_{S_3,0} \geq \Theta_{S_1}(i) - \Theta_{S_3}(i - 2) = (a_1 - a_3)i + (b_1 - b_3)N + c_1 - c_3 + 2a_3$ avec $i \in \mathcal{D}_{e_4} = \{i|4 \leq i \leq 2N - 1\}$.
   (a) $i = 4 : d_{S_3,1}N + d_{S_3,0} \geq (b_1 - b_3)N + c_1 - c_3 + 4a_1 - 2a_3 \geq 1$.
      i. $N = 3 : 3d_{S_3,1} + d_{S_3,0} \geq 4a_1 - 2a_3 + 3b_1 - 3b_3 + c_1 - c_3 \geq 1$.
      ii. Coefficients de $N : d_{S_3,1} \geq (b_1 - b_3) \geq 0$.
   (b) $i = 2N - 1 : d_{S_3,1}N + d_{S_3,0} \geq (2a_1 - 2a_3 + b_1 - b_3)N + 3a_3 - a_1 + c_1 - c_3 \geq 1$.
      i. $N = 3 : 3d_{S_3,1} + d_{S_3,0} \geq 5a_1 - 3a_3 + 3b_1 - 3b_3 + c_1 - c_3 \geq 1$.
      ii. Coefficients de $N : d_{S_3,1} \geq (2a_1 - 2a_3 + b_1 - b_3) \geq 0$.

FIG. 3.3 – Inéquations obtenues par application de la méthode des sommets aux contraintes de l'exemple 2.

$$
\begin{array}{lclcl}
3d_{S_1,1} + d_{S_1,0} & \geq & 4a_2 - 2a_1 + 3b_2 - 3b_1 + c_2 - c_1 & \geq & 1 \\
d_{S_1,1} & \geq & b_2 - b_1 & \geq & 0 \\
3d_{S_1,1} + d_{S_1,0} & \geq & 4a_2 - 2a_1 + 3b_2 - 3b_1 + c_2 - c_1 & \geq & 1 \\
d_{S_1,1} & \geq & 2a_2 - a_1 + b_2 - b_1 & \geq & 0 \\
3d_{S_2,1} + d_{S_2,0} & \geq & 5a_3 - 2a_2 + 3b_3 - 3b_2 + c_3 - c_2 & \geq & 1 \\
d_{S_2,1} & \geq & a_3 + b_3 - b_2 & \geq & 0 \\
3d_{S_2,1} + d_{S_2,0} & \geq & 5a_3 - 2a_2 + 3b_3 - 3b_2 + c_3 - c_2 & \geq & 1 \\
d_{S_2,1} & \geq & 2a_3 - a_2 + b_3 - b_2 & \geq & 0 \\
3d_{S_3,1} + d_{S_3,0} & \geq & 3a_1 - 2a_3 + 3b_1 - 3b_3 + c_1 - c_3 & \geq & 1 \\
d_{S_3,1} & \geq & b_1 - b_3 & \geq & 0 \\
3d_{S_3,1} + d_{S_3,0} & \geq & 5a_1 - 4a_3 + 3b_1 - 3b_3 + c_1 - c_3 & \geq & 1 \\
d_{S_3,1} & \geq & 2a_1 - 2a_3 + b_1 - b_3 & \geq & 0 \\
3d_{S_3,1} + d_{S_3,0} & \geq & 4a_1 - 2a_3 + 3b_1 - 3b_3 + c_1 - c_3 & \geq & 1 \\
d_{S_3,1} & \geq & b_1 - b_3 & \geq & 0 \\
3d_{S_3,1} + d_{S_3,0} & \geq & 5a_1 - 3a_3 + 3b_1 - 3b_3 + c_1 - c_3 & \geq & 1 \\
d_{S_3,1} & \geq & 2a_1 - 2a_3 + b_1 - b_3 & \geq & 0
\end{array}
$$

$$\min_{\prec}(d_{S_1,1} + d_{S_2,1} + d_{S_3,1}, d_{S_1,0} + d_{S_2,0} + d_{S_3,0})$$

FIG. 3.4 – Programme linéaire définissant les ordonnancements minimisant les durées de vie pour l'exemple 2.

3.3.3 Application à l'exemple 3

Soient les deux nids de boucles suivants :

```
    For i=1,2N−1
      For j=1,2N−1
S:       A[i,j]=(A[i−1,j]+A[i,j−1]) / 2
      EndFor
    EndFor
    For i=1, N−1
      For j=1,N−1
T:       B[i,j]= A[i+j, i+j] +3
      EndFor
    EndFor
```

FIG. 3.5 – Exemple 3.

Il y a trois relations de dépendance, deux dans la première boucle et une autre entre la première et la deuxième boucle :

$$
\begin{aligned}
e_{S,1} : S(i-1,j) \to S(i,j), \mathcal{D}_{e_{S,1}} &= \{(i,j) | 2 \leq i \leq 2N-1, 1 \leq j \leq 2N-1\} \\
e_{S,2} : S(i,j-1) \to S(i,j), \mathcal{D}_{e_{S,2}} &= \{(i,j) | 1 \leq i \leq 2N-1, 2 \leq j \leq 2N-1\} \\
e_{S,3} : S(i+j,i+j) \to T(i,j), \mathcal{D}_{e_{S,3}} &= \{(i,j) | 1 \leq i \leq N-1, 1 \leq j \leq N-1\}
\end{aligned}
$$

Supposons que les ordonnancements sont de la forme suivante :

$$\Theta_S(i,j) = a_1 i + b_1 j + c_1 N + d_1$$
$$\Theta_T(i,j) = a_2 i + b_2 j + c_2 N + d_2$$

– Les contraintes d'ordonnancements sont :

$$\Theta_S(i,j) - \Theta_S(i-1,j) = a_1 \geq 1, \text{ sur le domaine } \mathcal{D}_{e_{S,1}} = \{(i,j) | 2 \leq i \leq 2N-1, 1 \leq j \leq 2N-1\}.$$

$$\Theta_S(i,j) - \Theta_S(i,j-1) = b_1 \geq 1, \text{ sur le domaine } \mathcal{D}_{e_{S,2}} = \{(i,j) | 1 \leq i \leq 2N-1, 2 \leq j \leq 2N-1\}.$$

$$\Theta_T(i,j) - \Theta_S(i+j,i+j) = (a_2 - a_1 - b_1)i + (b_2 - b_1 - a_1)j + (c_2 - c_1)N + (d_2 - d_1) \geq 1,$$
$$\text{sur le domaine } \mathcal{D}_{e_{S,3}} = \{(i,j) | 1 \leq i \leq N-1, 1 \leq j \leq N-1\}.$$

– Les bornes de durée de vie à minimiser sont définies par :

$$d_{S,1}.N + d_{S,0} \geq a_1$$
$$d_{S,1}.N + d_{S,0} \geq b_1$$
$$d_{S,1}.N + d_{S,0} \geq (a_2 - a_1 - b_1)i + (b_2 - b_1 - a_1)j + (c_2 - c_1)N + (d_2 - d_1)$$

Recherchons une solution avec durée de vie non paramétrée, soit avec $d_{S,1} = 0$. Dans une telle solution on doit obligatoirement avoir :

$$\begin{cases} a_2 - a_1 - b_1 = 0 \\ b_2 - b_1 - a_1 = 0 \\ c_2 - c_1 = 0 \end{cases}$$

Nous obtenons alors le système (avec les contraintes de dépendances) :

$$\begin{cases} a_1 \geq 1 \\ b_1 \geq 1 \\ d_2 - d_1 \geq 1 \\ d_{S,0} \geq a_1 \\ d_{S,0} \geq b_1 \\ d_{S,0} \geq d_2 - d_1 \\ a_2 = a_1 + b_1 \\ b_2 = a_1 + b_1 \\ c_1 = c_2 \end{cases}$$

D'où $d_{S,0} \geq 1$. Supposons que l'on puise avoir égalité : $d_{S,0} = 1$ alors :

$$\begin{cases} a_1 = b_1 = 1 \\ d_2 = 1 + d_1 \\ a_2 = 2 \\ b_2 = 2 \\ c_1 = c_2 \end{cases}$$

Ce système est faisable avec $d_1 = 0$ et $c_1 = 0$. Ce qui donne les ordonnancements suivants :

$$\begin{aligned} \Theta_S(i,j) &= i + j \\ \Theta_T(i,j) &= 2(i+j) + 1 \end{aligned}$$

Donc les durées de vie maximales sont :

1. $\Theta_S(i,j) - \Theta_S(i-1,j) = (i+j) - (i-1+j) = 1$
2. $\Theta_S(i,j) - \Theta_S(i,j-1) = (i+j) - (i+j-1) = 1$
3. $\Theta_T(i,j) - \Theta_S(i+j,i+j) = 2(i+j) + 1 - (i+j+i+j) = 1$

Chapitre 4

Génération de code

Après la détermination de l'ordonnancement optimal minimisant la durée de vie des valeurs, on montre comment générer le nouveau code afin d'introduire une technique de réutilisation mémoire, en s'appuyant sur la notion de durée de vie d'une variable. Rajopadhye et Wilde [54] définissent des contraintes que doit respecter l'emplacement mémoire. Celles-ci sont basées sur l'analyse des dépendances du programme et la fonction d'ordonnancement, Dans le contexte du modèle polyédrique. Les contraintes prises en compte pour fixer la taille mémoire des données nécessaires du programme rejoignent celles de Cathoor et al. [11]. Cependant ces contraintes tentent de réduire l'occupation mémoire des données à un minimum en associant le calcul de l'ordonnancement avec l'optimisation de la mémoire. Ceci vient du fait qu'ils attachent autant d'importance à l'optimisation de la taille des données. Notre technique de génération de code consiste en une réindexation des tableaux avec une réécriture de boucles :

4.1 Réécriture de boucles

La réécriture du nid de boucles suivant un ordonnancement se fait en utilisant un des nombreux algorithmes existant ([1, 15] parmi beaucoup d'autres), et elle comporte deux parties :

- La génération des nouvelles boucles qui énumèrent exactement l'ensemble des couples (t, p) où t est une date et p un processeur : nous nous sommes restreints au cas des ordonnancements monodimensionnels.
- La mise à jour des fonctions d'accès dans le code qui doivent être exprimées en fonction des nouveaux indices de boucle et non plus des anciens.

4.2 Réindexation des tableaux

Après la réécriture des boucles, il faut réindexer les tableaux, en utilisant les durées de vie des valeurs calculées.

Le problème de réindexation est le suivant :

Soit la relation de dépendance : $S_e(h_e(\boldsymbol{j}, N), N) \rightarrow T_e(\boldsymbol{j}, N)$, entre les deux instructions S_e et T_e. Il s'agit de définir de nouvelles fonctions d'accès (index) des tableaux tenant compte des durées de vie. Pour résoudre ce problème, on procède comme suit :

Soit $\Theta^L_{S_e}$ et $\Theta^L_{T_e}$ les parties linéaires des ordonnancements : Θ_{S_e} et Θ_{T_e}.

Soit :

- $\boldsymbol{j}$ l'indice du vecteur d'itération de notre programme original ;
- $\boldsymbol{j}'$ l'indice du vecteur d'itération du nouveau programme à générer, tel que $\boldsymbol{j}' = A\boldsymbol{j} + b$.

On définit une matrice de transformation $\mathcal{M}_{S_e}$ (resp. $\mathcal{M}_{T_e}$) telle que la matrice $\begin{bmatrix} \Theta^L_{S_e} \\ \mathcal{M}_{S_e} \end{bmatrix}$ (resp . $\begin{bmatrix} \Theta^L_{T_e} \\ \mathcal{M}_{T_e} \end{bmatrix}$) soit régulière carrée de dimension égale à la dimension du domaine de validité de Θ_{S_e}, c'est-à-dire à la dimension de $\mathcal{D}_{S_e}$, dans le but d'être injective.

La durée de vie d'un élément de tableau $A[\boldsymbol{j}]$ écrit par l'instruction S_e est égale au maximum à :

$$d_{S_e} = d_{S_e,1}.N + d_{S_e,0}$$

Cette valeur détermine le nombre de cases mémoire nécessaires : $d_{S_e} + 1$: au maximum une valeur devra être stockée pendant $(d_{S_e} + 1)$ itérations.

Notre réindexation revient à référencer le tableau en écriture dans l'instruction S_e comme suit :

$$S_e : A[t_0 \bmod (d_{S_e} + 1), \mathcal{M}_{S_e}.\boldsymbol{j}] = ...$$

De la même manière, on référence le tableau dans l'instruction T_e :

Le vecteur d'itération $\boldsymbol{j} = A^{-1}(\boldsymbol{j}' - b)$ est source de dépendance si $\boldsymbol{j} \in \mathcal{D}_e \Leftrightarrow A^{-1}(\boldsymbol{j}' - b) \in \mathcal{D}_e$.

Donc la source de dépendance est :

$$h_e(\boldsymbol{j}, N) = h_e(A^{-1}(\boldsymbol{j}' - b), N).$$

Cette source est écrite à la date :

$$\Theta_{S_e}(h_e(\boldsymbol{j}, N)).$$

Donc elle a été écrite dans :

$$T_e : \left(\Theta_{S_e}(h_e(A^{-1}(\boldsymbol{j}' - b), N)) \bmod (d_{S_e} + 1), \mathcal{M}_{S_e}(A.h_e(A^{-1}(\boldsymbol{j}' - b), N) + b))\right)$$

Donc la génération de code est comme suit :

Initialement on a :

$S_e : A[f(\boldsymbol{j})] = ...$

$T_e :$ $\quad ... = A[g(\boldsymbol{j})]$

Après transformation on a :

$S_e : A[t_0 \bmod (d_{S_e} + 1), \mathcal{M}_{S_e}.\boldsymbol{j}] = ...$

$T_e :$ si $A^{-1}(\boldsymbol{j}' - b) \in \mathcal{D}_e$

$\quad ... = A[\Theta_{S_e}(h_e(A^{-1}(\boldsymbol{j}' - b), N)) \bmod (d_{S_e} + 1), \mathcal{M}_{S_e}(A.h_e(A^{-1}(\boldsymbol{j}' - b), N))] +$b))

sinon $\quad ... = A[g(A.h_e(A^{-1}(\boldsymbol{j}' - b))]$.

4.3 Génération de code de l'exemple 2

Soit le code de programme de la figure 3.2 :

On a déjà trouvé les fonctions d'ordonnancements optimales suivantes :

$$\Theta_{S_1}(i) = i + 1, \Theta_{S_1}(i) \in [3, 2N]$$
$$\Theta_{S_2}(i) = N + 1$$
$$\Theta_{S_3}(i) = i, \Theta_{S_3}(i) \in [2, 2N - 2]$$

D'où $t_0 \in [2, 2N]$.

Le processus de génération de code est le suivant :

– Pour l'instruction S_1 on a :

$$\theta_{S_1}(i) = t_0 = i + 1 \Leftrightarrow i = t_0 - 1 \text{ si } t_0 \in [3, 2N].$$

La durée de vie de S_1 est $d_{S_1} = N - 2$, ce qui définit un nombre de cases mémoire de $N - 1$. Alors l'instruction S_1 doit écrire dans le tableau $A[t_0 \bmod (N - 1)]$ (si $t_0 \leq 3$). Les sources de dépendances par rapport à l'instruction $S_1(i)$ sont $S_3(i-1)$ et $S_3(i-2)$ (c'est-à-dire $S_3(t_0-2)$ et $S_3(t_0-3)$). Comme la durée de vie du tableau C accédé par A est $d_{S_3}=3$, le nombre de cases mémoire réutilisables est de 4. Ce qui implique que les éléments du tableau A[2i] de l'instruction $S_1(t_0)$ doit référencer les éléments de tableau $C[(t_0-3) \bmod 4]$ et $C[(t_0-4) \bmod 4]$.

- La date d'exécutiondes instances de l'instruction S_2 est constante (indépendante de i). On crée alors une boucle sans changer les indices d'écriture et de lecture.
  L'instruction fait référence au tableau écrit par l'instruction S_1, ce qui signifie que les éléments de l'instruction S_2 lisent les élément de tableau $A[(j/2+1) \bmod (N-1)]$
- Pour l'instruction S_3, elle est référencée par l'instruction S_1, ce qui définit son indice d'écriture : $C[t_0 \bmod 4]$.
  Donc notre programme devient après la génération de code :

```
For t0= 2,2N
If t0>= 3   then
   A[ t0%(N−1) ] = C [ (t0−2)% 4] + C[ (t0−3)% 4]
If t0= N+1  then
   For j= 2, 2N−1
     B[j + N] = A[ ( J/2 +1)% (N−1)]
   EndFor
If t0 <= 2N−1  then
   C[t0%4] = B[t0]
EndFor
```

FIG. 4.1 – L'exemple .2. après la génération de code

4.4 Limite de la méthode

Considérons l'exemple 4 (présenté figure 4.2). Ce code contient cinq relations de dépendances :

e_1 : $S(i,j-1) \rightarrow S(i,j)$, $1 \leq i \leq N, 2 \leq j \leq N$ (dépendance de sortie).

e_2 : $T(i,j-1) \rightarrow T(i,j)$, $1 \leq i \leq N, 2 \leq j \leq N$.

e_3 : $T(i,j-2) \rightarrow T(i,j)$, $1 \leq i \leq N, 3 \leq j \leq N$.

e_4 : $S(i,j) \rightarrow T(i,j)$, $1 \leq i \leq N, 1 \leq j \leq N$.

e_5 : $T(i,j-1) \rightarrow S(i,j)$, $1 \leq i \leq N, 2 \leq j \leq N$ (anti-dépendance).

Comme d'habitude nous recherchons des ordonnancements affines *monodimensionnels* :

$$\Theta_S(i,j,N) = ai + bj + cN + d \quad \text{et} \quad \Theta_T(i,j,N) = \alpha i + \beta j + \gamma N + \delta.$$

Une étude fine de l'exemple montre que dans une solution minimisant les durées de vie on aura : $d_{S,1}=0$, $d_{S,0}=1$ (pour la durée de vie), $a=\alpha$, $b=\beta=2$, $c=\gamma$ et $\delta=d+1$. On choisira par exemple : $a=\alpha=0$, $c=\gamma=0$, $d=0$ et $\delta=1$. Autrement dit :

$$\Theta_S(i,j,N) = 2j \quad \text{et} \quad \Theta_T(i,j,N) = 2j+1.$$

On obtient ainsi le code de la figure 4.3. La boucle sur le temps, dans ce cas, étant la boucle sur j, et l'accès au tableau dans l'instruction S étant dès le départ monodimensionnel, l'utilisation des durées de vie ne nous fait rien gagner ici.

Plutôt que de se cantonner aux ordonnancements monodimensionnels comme nous l'avons fait jusqu'à présent, considérons l'ordonnancement multidimensionnel :

$$\Theta_S(i,j,N) = (i, 2j) \quad \text{et} \quad \Theta_T(i,j,N) = (i, 2j+1).$$

```
For i=1,N
   For j=1,N
      S : A[i] = 27*(i+j)+i*j
      T : B[i,j] = (B[i, j-1] - A[i]) * (B[i, j-2] - A[i])
   EndFor
EndFor
```

FIG. 4.2 – Exemple 4.

```
For j=1,N
   For i=1,N
      S : A[i] = 27*(i+j)+i*j
   EndFor
   For i=1,N
      T : B[i,j] = (B[i, j-1] - A[i]) * (B[i, j-2] - A[i])
   EndFor
EndFor
```

FIG. 4.3 – Exemple 4 récrit avec les ordonnancements : $\Theta_S(i,j,N) = 2j$ et $\Theta_T(i,j,N) = 2j+1$.

Cet ordonnancement est tout simplement l'ordonnancement de l'exemple de départ. La durée de vie des valeurs produites par l'instruction S est alors :

$$\Theta_T(i,j,N) - \Theta_S(i,j,N) = (0,1)$$

sachant qu'en fait l'écriture et la lecture de $A[i]$ ont lieu au sein de la même itération. La durée de vie effective est donc de $(0,0)$ en terme d'itérations (ce qui importe pour la quantité de mémoire à réserver). En utilisant cette durée de vie pour récrire le code de l'exemple 4 on obtient alors le code de la figure 4.4 ou plus simplement celui de la figure 4.5, où le tableau A est remplacé par un simple scalaire suivant une optimisation de notre exemple qui était facilement prédictible. En utilisant un ordonnancement multidimensionnel nous avons donc été capables de réduire la quantité de mémoire nécessaire ce que nous n'avons pas pu faire en utilisant un ordonnancement monodimensionnel (dans un exemple admettant des ordonnancements monodimensionnels).

En fait, en utilisant un ordonnancement monodimensionnel nous cherchons par définition à satisfaire par la première boucle du nouveau nid, celle sur le temps, toutes les dépendances. Un tel ordonnancement induit donc un délai positif et donc une durée de vie positive via cette boucle la plus externe. La durée de vie effective d'une valeur est alors proportionnelle au nombre d'itérations des boucles les plus internes : dans notre exemple N fois la durée de vie selon l'ordonnancement. Au contraire, nous aurions intérêt, afin de minimiser les durées de vie, à ce que les dépendances ne soient satisfaites que par les boucles les plus internes possibles, les boucles externes n'induisant qu'un délai nul sur les dépendances. La première dimension de l'ordonnancement multidimensionnel proposé pour l'exemple 4 induit un délai nul sur les dépendances : les dépendances sont donc satisfaites par la boucle la plus interne et la durée de vie des valeurs produites est alors au pire du nombre d'itération de cette boucle (N) et au mieux — ce qui est notre cas — constante... Même quand les codes considérés admettent des ordonnancements monodimensionnels nous avons intérêt, dans une optique d'optimisation mémoire, à utiliser des ordonnancements multidimensionnels minimisant les durées de vie. Encore faudrait-il savoir en construire...

```
For i=1,N
   For j=1,N
      S : A[i mod 1, j mod 1] = 27*(i+j)+i*j
      T : B[i,j] = (B[i, j-1] - A[i mod 1, j mod 1])
                   * (B[i, j-2] - A[i mod 1, j mod 1])
   EndFor
EndFor
```

FIG. 4.4 – Exemple 4 récrit avec les ordonnancements : $\Theta_S(i,j,N) = (i,2j)$ et $\Theta_T(i,j,N) = (i,2j+1)$.

```
For i=1,N
   For j=1,N
      S : A = 27*(i+j)+i*j
      T : B[i,j] = (B[i, j-1] - A)
                   * (B[i, j-2] - A)
   EndFor
EndFor
```

FIG. 4.5 – Exemple 4 récrit avec les ordonnancements : $\Theta_S(i,j,N) = (i,2j)$ et $\Theta_T(i,j,N) = (i,2j+1)$.

Chapitre 5

Travaux préexistants

Nous passons ici en revue quelques travaux sur la minimisation de la mémoire, travaux menés dans des cadres comparables.

Les travaux de Lefebvre et Feautrier

Ces travaux se placent dans le cadre de la parallélisation automatique. Cette étude est faite pour palier l'inconvénient de la méthode d'expansion totale (programme en assignation unique), en effectuant une expansion partielle afin d'assurer un maximum de parallélisme, mais avec un coût mémoire beaucoup plus réduit que l'expansion totale. Cette méthode se base sur l'analyse de dépendance, en n'éliminant que les dépendances artificielles qui n'ont pas d'influence sur la sémantique du programme, afin d'assurer une meilleure réutilisation mémoire.

Lefebvre et Feautrier ont proposé une méthode qui permet de réaliser une expansion partielle de la mémoire sans perte de parallélisme. Elle est basée sur le calcul effectif des intervalles d'utilité des valeurs du programme. Cette méthode reconstruit complètement le programme à partir uniquement du flot de données et de l'ordonnancement. La définition des nouvelles structures de données s'effectue de manière à n'établir que les dépendances de sortie neutres. Donc elle permet la redéfinition d'un programme indépendamment des structures de données originales, ce qui est beaucoup moins restrictif et offre un champ de manoeuvre beaucoup plus large. On se retrouve alors capable de corriger les programmes mal écrits, c'est-à-dire utilisant beaucoup d'espace mémoire, ou de prendre en compte du code directement fourni en assignation unique, ce qui permet d'optimiser des programmes générés à partir de programmes systoliques.

L'optimisation mémoire dans cette technique repose sur un ordonnancement choisi pour maximiser le parallélisme exposé. Cet ordonnancement peut-être très différent de celui minimisant la quantité nécessaire, ordonnancement que nous essayons de construire. Contrairement à nous, Lefebvre et Feautrier ne se limitent pas au cas monodimensionnel.

Les travaux de Quilleré et Rajopadhye

Quilleré et Rajopadhye [44] s'intéressent aux programmes à contrôle statique mis en assignation unique. Étant donné un ordonnancement multidimensionnel ils projettent les tableaux suivant un ensemble maximal de directions.

Quilleré et Rajopadhye n'ont donc pas le choix de l'ordonnancement et ils travaillent dans un cadre plus restrictif. Par contre ils manipulent des ordonnancements multidimensionnels et peuvent réduire plusieurs dimensions d'un tableau.

La contraction de tableaux

De nombreux travaux se sont focalisés sur la contraction de tableaux, c'est-à-dire, au sens strict, le remplacement d'un tableau par un scalaire. Par exemple Pike [41] propose un algorithme de contraction qui permet une contraction partielle (suppression de certaines des dimensions d'un tableau). Ses techniques ne s'appliquent qu'à une forme faible de programmes en assignation unique : des programmes où les fonctions d'accès aux tableaux sont unimodulaires (deux instances d'une même instruction ne peuvent pas écrire la même cellule mémoire, mais deux instances de deux instructions différentes le peuvent). En plus, un accès en lecture a un tableau ne peut pas dépendre de deux instructions différentes (suivant la valeur de l'indice d'itération, par exemple). Son cadre de travail est donc beaucoup plus restrictif que le nôtre.

Ce travail se rapproche du nôtre dans le sens où la contraction est différente pour chaque instruction. Son cadre est beaucoup plus restrictif mais Pike est capable de supprimer plusieurs dimensions d'un tableau, ce qui nous demanderait de manipuler des ordonnancements multidimensionnels.

Les travaux de Thies, Vivien, Sheldon et Amarasinghe

Thies, Vivien, Sheldon et Amarasinghe [52] se sont restreints aux programmes en assignation unique admettant des ordonnancements monodimensionnels. Dans ce cadre ils recherchent des *vecteurs d'occupation affines* qui sont des directions suivant lesquelles un tableau peut être projeté, la direction étant compatible soit avec un ordonnancement donné, soit avec tous les ordonnancements. Leur technique permet de replier les tableaux suivant une unique dimension, comme nous ici. Cette technique est combinable avec la notre, puisqu'ils ont la liberté du choix de l'ordonnancement. Il faudrait étudier finement la relation entre les durées de vie des valeurs et les vecteurs d'occupation, ces deux notions étant bien évidemment reliées comme le montre clairement les contraintes dites de stockage que doivent satisfaire les vecteurs d'occupations.

Chapitre 6

Conclusion et perspectives

L'étude présentée dans ce mémoire permet d'avoir une influence importante sur l'utilisation de la mémoire surtout pour la génération des systèmes embarqués. La technique proposée pour la réduction mémoire nous a permis d'exploiter les résultats des études effectuées sur l'ordonnancement et la modélisation polyédrique. Nous avons utilisé les polynômes d'Ehrhart pour le dénombrementdes points d'un ensemble de polyèdres afin de déterminer la taille mémoire nécessaire à un instant donné. Nous avons aussi défini un ordonnancement minimisant la durée de vie des variables lors de l'exécution d'un programme, afin de pouvoir réduire la quantité mémoire utilisée par le programme.

De nombreuses questions sont encore ouvertes concernant notament la détermination d'un ordonnancement optimal dans le cas multidimensionnel.

Cette question nous permettre de percevoir une évolution de nos travaux, si on veut pouvoir concevoir des systèmes embarques très performants répondant aux besoins actuels.

Bibliographie

[1] Corinne Ancourt et François Irigoin. Scanning polyhedra with DO loops. In *Proc. of the 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 39–50, Avril 1991.

[2] David F. Bacon, Susan L. Graham et Oliver J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Surveys*, 26(4), 1994.

[3] U. Banerjee. An introduction to a formal theory of dependence analysis. *Journal of Supercomputing*, 2:133–149, 1988.

[4] Luca Benini et Giovanni De Micheli. System-level power optimization : techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):115–192, Avril 2000.

[5] A. J. Bernstein. Analysis of programs for parallel processing. *IEEE Trans. on Elec. Computers*, EC-15, 1966.

[6] Youcef Bouchebaba et Fabien Coelho. Pavage pour une séquence de nids de boucles. *Technique et science informatiques*, 21, 2002. À paraître.

[7] Pierre Boulet, Alain Darte, Georges-André Silber et Frédéric Vivien. Loop parallelization algorithms : From parallelism extraction to code generation. *Parallel Computing*, 24(3-4):421–444, Mai 1998. Numéro spécial « Languages and Compilers for Parallel Computers ».

[8] Pierre Boulet et Xavier Redon. SPPoC : manipulation automatique de polyèdres pour la compilation. *Technique et Science Informatiques*, 20(8):1019–1048, 2001.

[9] Steve Carr. *Memory-Hierarchy Management.* PhD thesis, Rice University, Février 1993.

[10] Steve Carr et Ken Kennedy. Scalar replacement in the presence of conditional control flow. *Software - Practice and Experience*, 24(1):51–77, Janvier 1994.

[11] Francky Catthoor, Sven Wuytack, Eddy De Greef, Florin Balasa, Lode Nachtergaele et Arnout Vandecappelle. *Custom Memory Management Methodology : Exploration of Memory Organisation for Embedded Multimedia System Design.* Kluwer Academic Publishers, Boston, Septembre 1998. ISBN 0-7923-8288-9.

[12] Philippe Clauss et Vincent Loechner. Parametric analysis of polyhedral iteration spaces. In Fortes, Mongenet, Parhi et Taylor, éditeurs, *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, pages 415–424. IEEE Computer Society Press, August 1996.

[13] Philippe Clauss et Vincent Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, 1998. Kluwer Academic Pub.

[14] Stephanie Coleman et Kathryn S. McKinley. Tile size selection using cache organization. *ACM SIGPLAN Notices*, 30(6):279–290, Juin 1995.

[15] Jean-François Collard, Paul Feautrier et Tanguy Risset. Construction of DO loops from systems of affine constraints. *Parallel Processing Letters*, 5(3):421–436, Septembre 1995.

[16] Eddy De Greef. *Storage Size Reduction for Multimedia Application.* PhD thesis, Katholieke Universiteit Leuven and IMEC, Janvier 1998.

[17] Eugène Ehrhart. Sur les polyèdres rationnels homothétiques à n dimensions. *C.R. Acad. Sci. Paris*, 254:616–618, 1962.

[18] Eugène Ehrhart. Sur un problème de géométrie diophantienne linéaire i. *Reine Angew. Math.*, 226:1–29, 1967.

[19] Eugène Ehrhart. Sur un problème de géométrie diophantienne linéaire ii. *Reine Angew. Math.*, 227:25–49, 1967.

[20] Paul Feautrier. Array expansion. In *ACM International Conference on Supercomputing*, pages 429–441, 1988.

[21] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–51, 1991.

[22] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part I : One-dimensional time. *Int. J. Parallel Programming*, 21(5):313–348, Octobre 1992.

[23] Paul Feautrier. Some efficient solutions to the affine scheduling problem, part II : Multi-dimensional time. *Int. J. Parallel Programming*, 21(6):389–420, Décembre 1992.

[24] Paul Feautrier. Distribution automatique des données et des calculs. *Technique et science informatique*, 15(5):529–557, 1996.

[25] Antoine Fraboulet, Karen Godary et Anne Mignotte. Loop fusion for memory space optimization. In *IEEE International Symposium on System Synthesis*, pages 95–100, Montréal, Canada, Octobre 2001. IEEE Press.

[26] G. Gao, R. Olsen, V. Sarkar et R. Thekkath. Collective loop fusion for array contraction. In U. Banerjee, D. Gelernter, A. Nicolau et D. A. Padua, éditeurs, *The 5th Workshop on Languages and Compiler for Parallelism*, volume 757 des *Lecture Notes in Computer Science*, pages 281–295. Springer-Verlag, 1992.

[27] François Irigoin, Pierre Jouvelot et Rémy Triolet. Semantical interprocedural parallelization : An overview of the PIPS project. In *Proceedings of the 1991 ACM International Conference on Supercomputing*, Cologne, Germany, Juin 1991.

[28] R. M. Karp, R. E. Miller et S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, Juillet 1967.

[29] W. Kelly et W. Pugh. A framework for unifying reordering transformations. Rapport de recherche 3193, Dpt. of Computer Science, University of Maryland, April 1993.

[30] Leslie Lamport. The parallel execution of DO loops. *Communications of the ACM*, 17(2):83–93, Février 1974.

[31] Vincent Lefebvre. *Restructuration automatique des variables d'un programme en vue de sa parallélisation*. PhD thesis, Université de Versailles - Saint-Quentin, 1998.

[32] C. Lengauer. Loop parallelization in the polytope model. In *CONCUR 93*, 1993.

[33] Vincent Loechner et Doran K. Wilde. Parameterized polyhedra and their vertices. *International Journal of Parallel Programming*, 25(6), Décembre 1997.

[34] Naraig Manjikian et Tarek S. Abdelrahman. Fusion of loops for parallelism and locality. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):193–209, Février 1997.

[35] Kathryn S. McKinley, Steve Carr et Chau-Wen Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424–453, Juillet 1996.

[36] Benoît Meister. Localité de données dans les opérations stencil. In *RenPar'2001*, Avril 2001.

[37] Benoît Meister. Méthodes mathématiques d'optimisation des accès mémoire pour logiciels enfouis. Rapport de DEA, Université Louis Pasteur, Strasbourg, France, 2001.

[38] Preeti Ranjan Panda, Nikil D. Dutt et Alexandru Nicolau. *Memory Issues in Embedded Systems-On-Chip*. Kluwer Academic Publishers, 1999. ISBN 0-7923-8362-1.

[39] Preeti Ranjan Panda, Hiroshi Nakamura, Nikil D. Dutt et Alexandru Nicolau. Improving cache performance through tiling and data alignment. In *Irregular 1997*, pages 167–185, 1997.

[40] Preeti Ranjan Panda, Hiroshi Nakamura, Nikil D. Dutt et Alexandru Nicolau. Augmenting loop tiling with data alignment for improved cache performance. *IEEE Transactions on Computers*, 48(2):142–149, 1999.

[41] Geoffrey Roeder Pike. *Reordering and Storage Optimizations for Scientific Programs.* PhD thesis, University of California, Berkeley, 2002.

[42] Polylib - a library of polyhedral functions. `http://icps.u-strasbg.fr/polylib/`.

[43] William Pugh. The Omega test : A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–114, Août 1992.

[44] Fabien Quilleré et Sanjay Rajopadhye. Optimizing memory usage in the polyhedral model. *ACM Transactions on Programming Languages and Systems*, 22(5):773–815, Septembre 2000.

[45] Patrice Quinton. *Automata Networks in Computer Science*, chapitre The systematic design of systolic arrays. Manchester University Press, 1987.

[46] Patrice Quinton. The systematic design of systolic arrays. In Françoise Fogelman Soulie, Yves Robert et Maurice Tchuente, éditeurs, *Automata Networks in Computer Science*, chapitre 9, pages 229–260. Manchester University Press, 1987.

[47] Sailesh K. Rao. *Regular Iterative Algorithms and Their Implementations on Processor Arrays.* PhD thesis, Stanford University, Octobre 1985.

[48] Vwani P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms.* PhD thesis, Stanford University, Décembre 1988.

[49] V. Sarkar et G. Gao. Optimization of array accesses by collective loop transformations. In *ACM Int. Conf. Supercomputing*, pages 194–205, 1991.

[50] A. Schrijver. *Theroy of Linear and Integer Programming.* John Wiley and Sons, New York, 1986. ISBN 0-471-90854-1.

[51] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, New York, 1986.

[52] William Thies, Frédéric Vivien, Jeffrey Sheldon et Saman Amarasinghe. A Unified Framework for Schedule and Storage Optimization. In *Proceedings of the ACM SIGPLAN'01 conference on Programming Language Design and Implementation (PLDI)*, pages 232–242, Snowbird, UT, USA, Juin 2001.

[53] Frédéric Vivien. On the Optimality of Feautrier's Scheduling Algorithm. In *Euro-Par 2002*, Août 2002.

[54] D. Wilde et S. Rajopadhye. Memory reuse analysis in the polyhedral model. In Bougé, Fraigniaud, Mignotte et Robert, éditeurs, *Euro-Par'96 Parallel Processing*. Springer Verlag, 1996.

[55] Michael J. Wolfe. Loop skewing : The wavefront method revisited. *International Journal of Parallel Programming*, 15(4):279–293, Août 1986.