Parallélisme des nids de boucles pour l'optimisation du temps d'exécution et de la taille du code

Yaroub Elloumi

UNIVERSITÉ PARIS-EST

THÈSE

En cotutelle

Pour obtenir le grade de

Docteur

De l'Université Paris-Est
École Doctorale Mathématiques et STIC
Spécialité : Informatique

&

De l'Université de Sfax
École Doctorale Sciences et Technologies
Spécialité : Ingénierie des Systèmes Informatiques

Présentée par

Yaroub Elloumi

Parallélisme des nids de boucles pour l'optimisation du temps d'exécution et de la taille du code

Soutenue publiquement le 16/12/2013 devant le jury composé de :

Président :	**Habib Mehrez**	**Professeur**	**Université Pierre et Marie Curie (France)**
Rapporteurs :	**Bachir Elayeb**	**Professeur**	**Université de Monastir (Tunisie)**
	Guy Gogniat	**Professeur**	**Université de Bretagne-Sud (France)**
Examinateur :	**Thierry Grandpierre**	**Maître de conférence**	**ESIEE Paris (France)**
Directeurs :	**Mohamed Akil**	**Professeur**	**ESIEE Paris (France)**
	Mohamed Hedi Bedoui	**Professeur**	**Université de Monastir (Tunisie)**

Remerciements

Je remercie tout d’abord Monsieur Guy Gogniat, professeur à l’Université de Bretagne-Sud, et Monsieur Bachir Elayeb, professeur à l’Université de Monastir, pour avoir accepté de rapporter ce travail.

Je remercie également Monsieur Habib Mehrez, professeur à l’Université Pierre et Marie Curie, et Monsieur Thierry Grandpierre, maître de conférence à l’ESIEE Paris, d’avoir bien voulu examiner ce travail et de faire partie de mon jury de thèse.

Je remercie tout particulièrement Monsieur Mohamed Hedi Bedoui, Professeur à l’Université de Monastir, et Monsieur Mohamed Akil, Professeur à l’ESIEE Paris, qui ont dirigé ma thèse et pour la confiance qu’ils m’ont accordée.

Je salue également tous les thésards (et anciens thésards) au Laboratoire Technologie et Imagerie Médicale (LTIM) et à l’équipe Algorithmes, Architectures, Analyse et Synthèse d'Images (A3SI) pour les discussions, les pauses cafés et tous les bons moments que j’ai eus l’occasion de partager avec eux.

Enfin, je remercie mes parents ainsi que tous les membres de ma famille pour leur soutien.

Parallélisme des nids de boucles pour l'optimisation du temps d'exécution et de la taille du code

Yaroub ELLOUMI

Résumé

Les algorithmes des systèmes temps réels incluent de plus en plus de nids de boucles, qui sont caractérisés par un temps d'exécution important. De ce fait, plusieurs démarches de parallélisme des boucles imbriquées ont été proposées dans l'objectif de réduire leurs temps d'exécution. Ces démarches peuvent être classifiées selon deux niveaux de granularité : le parallélisme au niveau des itérations et le parallélisme au niveau des instructions.

Dans le cas du deuxième niveau de granularité, les techniques visent à atteindre un parallélisme total des instructions appartenant à une même itération. Cependant, le parallélisme est contraint par les dépendances des données inter-itérations ce qui implique le décalage des instructions à travers les boucles imbriquées, provocant ainsi une augmentation du code proportionnelle au niveau du parallélisme. Par conséquent, le parallélisme total au niveau des instructions des nids de boucles engendre des implémentations avec des temps d'exécution non-optimaux et des tailles du code importantes.

Les travaux de cette thèse s'intéressent à l'amélioration des stratégies de parallélisme des nids de boucles. Une première contribution consiste à proposer une nouvelle technique de parallélisme au niveau des instructions baptisée « retiming multidimensionnel décalé ». Elle vise à ordonnancer les nids de boucles avec une période de cycle minimale, sans atteindre un parallélisme total. Une deuxième contribution consiste à mettre en pratique notre technique dans le contexte de l'implémentation temps réel embarquée des nids de boucles. L'objectif est de respecter la contrainte du temps d'exécution tout en utilisant un code de taille minimale. Dans ce contexte, nous avons proposé une première démarche d'optimisation qui consiste à utiliser notre technique pour déterminer le niveau parallélisme minimal. Par la suite, nous avons décrit une deuxième démarche permettant de combiner les parallélismes au niveau des instructions et au niveau des itérations, en utilisant notre technique et le « loop striping ».

Mots clés :
Nid de boucles, parallélisme, pipeline logiciel, performance, conception.

Nested loop parallelism for execution time and code size optimization

Yaroub ELLOUMI

Abstract

The real time implementation algorithms always include nested loops which require important execution times. Thus, several nested loop parallelism techniques have been proposed with the aim of decreasing their execution times. These techniques can be classified in terms of granularity, which are the iteration level parallelism and the instruction level parallelism. In the case of the instruction level parallelism, the techniques aim to achieve a full parallelism. However, the loop carried dependencies implies shifting instructions in both side of nested loops. Consequently, these techniques provide implementations with non-optimal execution times and important code sizes, which represent limiting factors when implemented on embedded real-time systems.

In this work, we are interested on enhancing the parallelism strategies of nested loops. The first contribution consists of purposing a novel instruction level parallelism technique, called “delayed multidimensional retiming”. It aims to scheduling the nested loops with the minimal cycle period, without achieving a full parallelism. The second contribution consists of employing the “delayed multidimensional retiming” when providing nested loop implementations on real time embedded systems. The aim is to respect an execution time constraint while using minimal code size. In this context, we proposed a first approach that selects the minimal instruction parallelism level allowing the execution time constraint respect. The second approach employs both instruction level parallelism and iteration level parallelism, by using the “delayed multidimensional retiming” and the “loop striping”.

Key words:
Nested loops, parallelism, software pipeline, performance, design.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

CHAPITRE 1 : INTRODUCTION GÉNÉRALE

1.1 Domaine d'intérêt

Les systèmes informatiques sont spécifiés par leurs domaines d'utilisation et les fonctionnalités qui les sont associées. Plusieurs domaines d'utilisation exigent des contraintes temporelles lors de la génération automatique des résultats finaux. Les systèmes informatiques sont ainsi dans l'obligation d'exécuter les applications en toutes circonstances tout en respectant un délai de réponse limité, comme exemples d'applications on peut citer : les applications audiovisuelles, le contrôle des systèmes industriels et le transport. Ces applications sont qualifiées de temps réel et les systèmes les implémentant de « systèmes temps réel ». Plusieurs de ces systèmes doivent aussi s'adapter à leurs environnements. Vu le phénomène exhaustif de portabilité, ils doivent aussi répondre à des contraintes d'embarquabilité telles que la taille, la consommation et le coût. On s'intéresse dans cette thèse à ce type de système : les systèmes temps réel embarqués.

En effet, les fonctionnalités assurées par les systèmes informatiques ont influé directement sur les valeurs des contraintes des systèmes temps réel embarqués. Les applications à implémenter se caractérisent par une complexité toujours croissance, impliquant une augmentation de la puissance de calculs. Les traitements de ces applications nécessitent des ressources matérielles importantes pour assurer leurs exécutions (processeurs, mémoires). Cette augmentation est proportionnelle aux coûts des systèmes. De plus, elle limite leurs facteurs d'embarquabilité, surtout pour le cas des systèmes portables (autonomie, poids, etc). De ce fait, les concepteurs sont ainsi dans l'impératif de faire recours à des techniques optimisations. Ces techniques consistent à modifier la spécification initiale à travers tous les niveaux de conception dans l'objectif de respecter les contraintes temporelles tout en réduisant des ressources matérielles. Les techniques d'optimisation à haut niveau offrent des espaces de solutions plus importants, ce qui conduit généralement au respect des contraintes temporelles de conception. Ces techniques d'optimisation assurent la modification de l'ordonnancement les traitements dans le but de minimiser les paramètres temporels de l'application, tout en garantissant une utilisation minimale des ressources matérielles. Les travaux de cette thèse s'intéressent aux méthodes et techniques d'optimisation appliquées aux niveaux algorithmiques, pour l'optimisation des contraintes du temps d'exécution et des ressources matérielles de l'implémentation.

1.2 Contexte

Les spécifications algorithmiques des systèmes temps réels intègrent de plus en plus de nids de boucles impliquant une augmentation des temps d'exécution des implémentations. Par ailleurs, les architectures actuelles ont subi une grande évolution en matière d'augmentation des unités de traitement. De ce fait, plusieurs travaux de recherche ont été proposés pour exploiter les potentialités offertes par les architectures dans le but d'augmenter le niveau du parallélisme des nids de boucles, afin de réduire leurs temps d'exécution. Ces travaux décrivent des techniques de ré-ordonnancement des traitements des structures itératives, permettant de sélectionner les traitements indépendants pour les exécuter dans le même espace temporel. Ces techniques de parallélisme procèdent à la modification de la structure algorithmique du nid de boucles tout en conservant le comportement global de l'application. Nous pouvons classifier les techniques de parallélisme des nids de boucles selon leur granularité : le parallélisme au niveau des itérations et le parallélisme au niveau des instructions. Le premier type consiste à sélectionner les itérations indépendantes en vue de les exécuter en parallèle [5, 9, 17, 15, 14]. Le deuxième type assure le parallélisme de l'exécution des instructions appartenant à une même itération [16, 22, 4]. D'autres travaux de recherche se sont intéressés à l'utilisation conjointe des deux types de parallélisme dans l'objectif

d'améliorer les performances des implémentations et l'utilisation des ressources matérielles [10, 8, 6, 3].

Pour le cas du parallélisme au niveau des instructions, plusieurs techniques ont été proposées pour le parallélisme d'une seule boucle. Certaines techniques sont appliquées à un niveau spécifique de boucle tel que la boucle interne [7, 23] ou la boucle externe [21]. D'autres techniques consistent à choisir le niveau de boucle dont le parallélisme assure la meilleure amélioration des performances [16]. Cependant, l'apport de ces techniques est limité au parallélisme d'un seul niveau de boucle. Un nombre restreint de techniques adressent le parallélisme des boucles imbriquées [20, 22, 1, 2, 4]. Elles visent généralement à atteindre un parallélisme total au niveau des instructions dans le but d'ordonnancer l'implémentation avec la période de cycle minimale. Dans le cadre de cette thèse, nous nous intéressons aux techniques de parallélisme des instructions des nids de boucles intitulées « retiming multidimensionnel » [1, 2, 4]. Elles modélisent les nids de boucles par l'intermédiaire des Graphes Flot de Données Multidimensionnels (GFDM), dont le formalisme permet une représentation explicite des instructions ainsi que leurs dépendances de données [4, 5, 12, 13, 17]. Par la suite, elles formalisent le parallélisme par l'intermédiaire de transformations basées sur la théorie des graphes.

1.3 Problématique

Le choix du parallélisme au niveau des instructions des boucles imbriquées est effectué en se basant sur les dépendances de données. Ce choix est contraint par les dépendances des données inter-itérations. De ce fait, le parallélisme implique le décalage des blocs de codes en amont et en aval des boucles imbriquées, intitulé respectivement prologue et épilogue [1, 2, 11].

Les techniques de retiming multidimensionnel visent à ordonnancer le nid de boucles avec une période de cycle minimale. Pour atteindre cette objectif, elles augmentent le niveau du parallélisme jusqu'à atteindre un parallélisme total. En effet, chaque parallélisme engendre l'ajout des blocs de code de prologue et d'épilogue, ce qui a pour effet d'accroitre la taille du code du nid des boucles proportionnellement avec le niveau du parallélisme. Les codes ajoutés impliquent une utilisation importante des mémoires locales et caches [18, 19]. De plus, ils nécessitent un ensemble de période de cycles pour leurs exécutions, ce qui présente une perte en temps [1, 4].

Par conséquent, pour atteindre une période de cycle minimale, le parallélisme total des nids de boucles engendre aussi bien des tailles importantes de codes, que des temps d'exécution non-optimaux, ce qui présente un facteur limitant pour les cas des implémentations embarquées. Respecter la contrainte du « temps d'exécution » tout en utilisant une « taille du code » minimale est un chalenge pour les concepteurs des systèmes embarqués temps réel [17].

1.4 Contribution

Dans le cadre des travaux de cette thèse, nous nous intéressons à améliorer les stratégies de parallélisme au niveau des instructions des nids de boucles. Par la suite, nous visons la mise en œuvre de ces stratégies dans le cadre de la conception des systèmes temps réel embarqués.

Notre première contribution consiste à proposer une nouvelle technique de parallélisme des instructions du nid de boucles, intitulée « retiming multidimensionnel décalé » [83, 84], permettant l'ordonnancement du nid de boucles avec une période de cycle minimale, sans atteindre un parallélisme total. L'idée principale de cette approche consiste, non pas à décaler

des instructions de calcul, mais de décaler des chemins de données entiers. Son formalisme permet d'extraire les caractéristiques temporelles et les dépendances de tous les chemins de données, puis d'en faire l'analyse afin de déterminer la liste des chemins à paralléliser. Par la suite, elle extrait le vecteur de parallélisme des chemins sélectionnés. Nous avons proposée une extension de cette technique dans [86]. Elle se base sur le choix d'un vecteur optimal de parallélisme et son utilisation itérative, jusqu'à atteindre la période de cycle minimale.

Notre deuxième contribution consiste à mettre en pratique notre technique dans le contexte d'implémentation temps réel embarquée des nids de boucle. L'objectif est de respecter la contrainte de temps d'exécution tout en utilisant un code de taille minimale. Dans ce contexte, nous avons proposé une première démarche d'optimisation utilisant notre technique pour déterminer le niveau parallélisme minimal permettant de respecter l'objectif. Elle permet de prédire les temps d'exécution et les tailles des codes en fonction du retiming multidimensionnel, puis de sélectionner le niveau minimal. Par la suite, nous avons proposé une deuxième méthode permettant de combiner les parallélismes au niveau des instructions et au niveau des itérations [85]. Cette deuxième méthode est basée sur l'utilisation conjointe du « retiming multidimensionnel décalé » et du « loop striping », pour respecter une contrainte de temps d'exécution tout en générant une taille de code minimale.

Ces contributions sont évaluées et validées par l'implémentation d'applications en traitement du signal et d'images, sur les architectures multi-GPUs. Cette étape a montré que la technique de « retiming multidimensionnel décalé » génère des implémentations avec des temps d'exécution et des tailles de codes inférieurs à celles générées par les techniques similaires existantes. Pour la deuxième contribution, la validation expérimentale a montré que l'utilisation conjointe de la technique (« retiming multidimensionnel décalé ») et le « loop striping » permet de respecter des contraintes de temps d'exécution, que chacune de ces deux techniques appliquée séparément ne le permettait. De plus, même si la contrainte est satisfaite, elle permet de générer une solution avec une taille de code inférieure à celles générées par chacune de ces deux techniques.

Les travaux de recherche menés et présentés dans ce manuscrit ont été effectués au Laboratoire Informatique Gaspard Monge (LIGM), Equipe ESIEE A3SI, Unité Mixte CNRS-UMLV-ESIEE (UMR 8049) de l'Université Paris-Est (en France), et au Laboratoire Technologie et Imagerie Médicale (LTIM-LR12ES06) de l'université de Monastir (en Tunisie), dans le cadre d'une thèse en cotutelle entre l'université de Paris-Est et l'université de Sfax.

1.5 Plan de la thèse

Le deuxième chapitre permet de détailler le contexte du parallélisme des nids de boucles. Nous débutons par décrire la structure des nids de boucles et les dépendances des données. Par la suite, nous traitons le problème du temps d'exécution des nids de boucles et nous montrons l'impératif de les optimiser pour le cas des implémentations temps réel embarquées. Après, nous décrivons la notion du parallélisme des nids de boucles en détaillant les étapes de la démarche et en mettant l'accent sur la contrainte des dépendances de données inter-itérations. Par la suite, nous listons les modèles de représentation des nids de boucles et les techniques de parallélisme qui y sont associées. Nous finissons par présenter une synthèse des modèles de formalisme des nids de boucles en se référant sur les critères algorithmiques qu'elles présentent.

Dans le troisième chapitre, nous décrivons la modélisation des nids de boucles par les Graphes Flot de Données Multidimensionnel (GFDMs). Par la suite, nous étudions les techniques du parallélisme des GFDMs au niveau des instructions et au niveau des itérations.

Cette étude nous permettra d'établir une synthèse des techniques d'optimisation des GFDMs et de dégager les limites de celles appliquant le parallélisme au niveau des instructions.

Dans le quatrième chapitre, nous proposons une nouvelle technique de retiming multidimensionnel intitulé « retiming multidimensionnel décalé » permettant d'ordonnancer le nid de boucles avec la période de cycle minimale, sans atteindre un parallélisme total. Nous démontrons la théorie permettant d'explorer les paramètres temporels et de dépendances des données du GFDMs, et d'en déduire le vecteur de parallélisme. Par la suite, nous étendons l'étude théorique dans l'objectif d'augmenter le niveau du parallélisme en utilisant le même vecteur déduit. Cette technique est validée par une étude expérimentale permettant de comparer les résultats générés par la technique que nous avons proposée, par rapport à ceux générés par les techniques existantes.

Dans le cinquième chapitre, nous mettons en pratique la technique du « retiming multidimensionnel décalé » pour la conception des implémentations respectant une contrainte du temps d'exécution en utilisant une taille de code minimale. Dans la première partie, nous décrivons une approche d'optimisation permettant de déterminer le niveau de parallélisme minimal permettant de respecter la contrainte du temps d'exécution. Dans une deuxième partie, nous combinons le « retiming multidimensionnel décalé » avec une technique de parallélisme au niveau des itérations. Nous décrivons une démarche d'estimation du temps d'exécution et de la taille du code, en fonction des deux techniques. Par la suite, nous montrons la théorie nécessaire pour le choix de l'ordre d'utilisation des techniques. Ce chapitre est clôturé par une étude expérimentale comparant les résultats générés par cette approche à ceux générés par chacune des techniques séparément.

Dans le sixième chapitre, nous résumons les contributions proposées et nous évaluons leurs apports pour la conception des implémentations temps réel embarquées. Par la suite, nous décrivons la continuité de ce travail en détaillant les perspectives envisagées. Ces perspectives s'intéressent à l'extension de la théorie algorithmique des stratégies du parallélisme, et à l'orientation de cette théorie vers l'optimisation d'autres contraintes de conception telle que la taille de la mémoire et les unités de calcul.

CHAPITRE 2 : PARALLÉLISME DES NIDS DE BOUCLES

2.1 Introduction

Les nids de boucles représentent des blocs de code nécessitant une fraction importante du temps d'exécution total d'une application. De plus, leurs implémentations exigent le recours à des ressources matérielles considérables, tel que la taille de la mémoire cache et le nombre des unités de calcul. Une grande partie de ces applications sont soumises à des contraintes temps réel et d'embarquabilité. Ces contraintes ne cessent d'être de plus en plus strictes, vu les exigences des consommateurs en matière de temps de réponse, de taille, de consommation, etc. Par conséquent, Il s'avère nécessaire de recourir à des procédures d'optimisation des nids de boucles, dans l'objectif de générer une implémentation respectant les contraintes.

L'optimisation des nids de boucles consiste à exploiter les potentialités des architectures en matière de multitude d'unités de calcul, afin de paralléliser le traitement. La démarche du parallélisme consiste à dégager les traitements itératifs pouvant être exécutés dans la même période du temps, puis d'affecter chacune à une unité de calcul distincte. Le choix du parallélisme dépend considérablement de l'ordre des dépendances de données des nids de boucles. L'absence des dépendances de données inter-itérations présentent une faciliter dans le choix du parallélisme ainsi que dans son niveau. Dans le cas contraire, elle contraint le choix des traitements à exécuter en parallèle. Dans ce contexte, les concepteurs procèdent à représenter les nids de boucles en modèle formel, reflétant les détails des dépendances de données. Puis, les démarches du parallélisme sont formulées autant que modification du modèle, dont nous verrons leurs principes dans ce chapitre.

Nous débutons par définir les nids de boucles ainsi que les dépendances de données. Ensuite, nous décrivons la notion du parallélisme des nids de boucles et son impact sur les performances de l'implémentation. La dernière partie est consacrée à la description des différents modèles de représentation des nids de boucles, ainsi que les démarches de parallélisme y associées.

2.2 Nid de boucles

2.2.1 Structure d'une boucle

Une « boucle » est considérée comme étant une structure itérative d'un programme fini, dont la structure algorithmique est schématisée dans la figure 2.1 [77, 108].

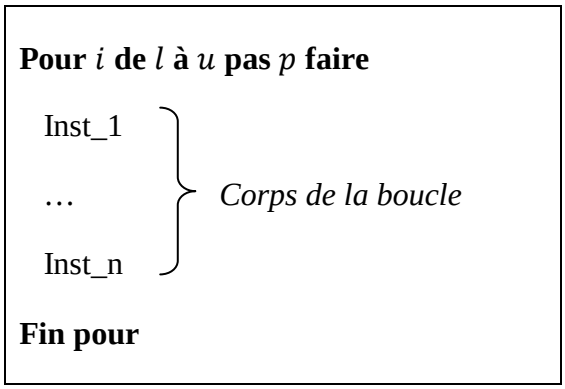


Figure 2.1 Structure d'une boucle

Où i est une variable entière appelée « indice » ou « itérateur », l et u sont des expressions entières représentant respectivement les bornes minimale et maximale de l'indice i, p est une variable entière indiquant les pas de l'indice. Dans le cas où $p = 1$, on parle de boucle régulière.

Chaque itération d'indice i exécute l'ensemble des instructions délimitées par « faire » et « fin pour », appelée corps de la boucle. Une instruction est le plus petit élément du code exécutable. Elle peut être simple telle qu'une affectation, un appel de fonction, ou bien composée telle qu'une structure conditionnelle ou une autre boucle.

2.2.2 Nid de boucles

Un nid de boucles est une imbrication d'un nombre fini n boucles (B_1, B_2, …, B_n) avec $n > 1$ [60, 61, 62], un exemple de cette structure est illustrée dans la figure 2.2.

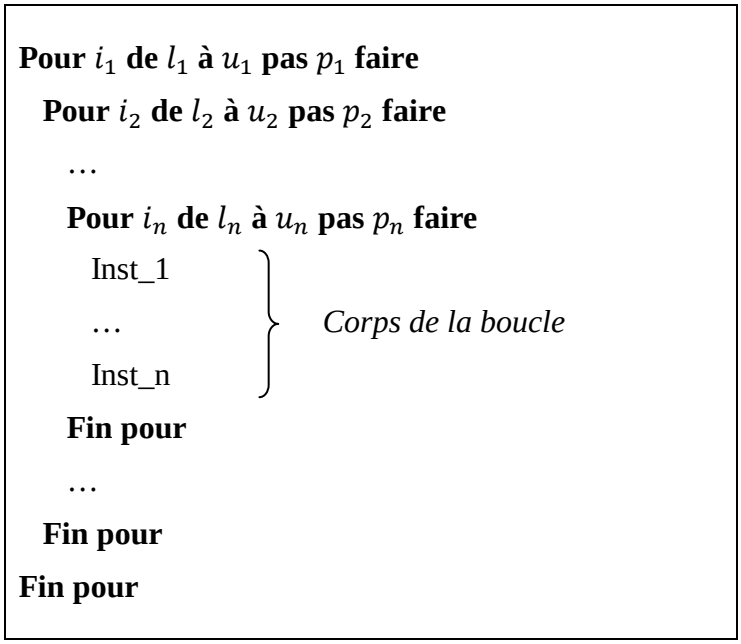


Figure 2.2 Structure d'un nid de boucles

n est appelé profondeur ou dimension du nid. On dit que B_1 est la boucle la plus externe et B_n la boucle la plus interne. La profondeur d'une boucle B_j est le nombre des boucles externes qui l'englobent. Un nid de boucles est dit « uniforme » si toutes les instructions appartiennent à la boucle la plus interne. De plus, il est dit « parfait » si le corps des instructions est composé d'une seule instruction. Chaque itération est représentée par un vecteur d'itération $I = (i_1, i_2, \dots, i_n)$ tel que $l_k \leq i_k \leq u_k$ et $1 \leq k \leq n$.

2.3 Dépendances de données

Une dépendance de donnée représente le transfert d'une donnée entre deux instructions. Elle peut être classifiée selon l'ordre des actions de lecture et d'écriture dans les variables utilisées, dont on distingue 3 types :

- Dépendance de flot : une instruction i_1 écrit une variable, suivie d'une autre instruction i_2 qui la lit.
- Anti-dépendance : une instruction i_1 lit la variable avant qu'une instruction i_2 l'écrive.
- Dépendance de sortie : deux instructions i_1 et i_2 écrivent dans la même variable.

Les anti-dépendances et les dépendances de sorties peuvent être éliminées en modifiant le programme initial par l'utilisation de nouvelles variables [40, 41].

Par ailleurs, les dépendances des données dans les nids de boucles peuvent être classifiées selon l'appartenance des instructions productrices et consommatrices aux itérations, dont nous distinguons deux types [63, 64, 65]:

- Dépendance de donnée inter-itération : est une dépendance de données entre deux instructions appartenant à deux itérations différentes.

- Dépendance de donnée intra-itération : est une dépendance de données entre deux instructions appartenant à une même itération. Dans le cas d'un nid de boucles uniforme, ces deux instructions appartiennent à la boucle la plus interne.

2.4 Caractéristiques temporelles des nids de boucles

2.4.1 Temps d'exécution des nids de boucles

Un nid de boucle uniforme assure l'exécution répétitive du corps de la boucle interne, tel que le nombre des répétitions est défini en fonction des itérateurs des boucles imbriquées. Le temps théorique nécessaire pour l'exécution d'un nid de boucles T_{NB} est calculé par la multiplication du temps d'exécution du corps de la boucle T_i et les valeurs des itérateurs des boucles, tel que indiqué dans l'équation 2.1.

$$T_{NB} = T_i \times \prod_{b=1}^{b=n}(u_n - l_n) \quad (2.1)$$

Avec n est la dimension du nid de boucles, l_n et u_n sont respectivement la valeur minimale et la valeur maximale de l'itérateur de la boucle n.

Les nids de boucles ont connu une augmentation excessive dans leurs domaines d'utilisation, telles que les applications de traitement de l'image et du signal [46], de télédétection, de reconstruction 3D/4D [94, 95], etc. Ce phénomène a été favorisé par leurs capacités de traiter un ensemble important de données. Les fonctionnalités assurées par ces applications sont caractérisées par des tailles de données de plus en plus importantes. En prenant l'exemple des images numériques, la dimension matricielle est exprimée de l'ordre de dizaines de Méga pixels. De plus, l'amélioration de la qualité des images a engendré l'augmentation des marges des couleurs, exprimées en termes de millions de couleurs. Ces deux critères impliquent l'augmentation des intervalles des itérateurs $[u_n - l_n]$ des nids de boucles des applications de traitement d'images. De ce fait, ils entraînent l'augmentation du temps d'exécution du nid de boucles T_{NB}. Par ailleurs, les applications actuelles ont recours à des nids de boucles avec des dimensions plus importantes. Pour le cas des applications de reconstruction 4D, le nid de boucles assurant la reconstruction spatiale est imbriqué dans les boucles assurant la répétition du traitement à l'échelle temporel. De ce fait, ces critères entraînent une augmentation dramatique du temps d'exécution du nid de boucles, au point qu'il représente la fraction la plus importante du temps d'exécution total de l'application.

2.4.2 Implémentation temps réel des nids de boucles

Les propriétés informant sur la rapidité du traitement sont devenues des caractéristiques primordiales de chaque système informatique. Plusieurs applications, dont la structure est basée sur les nids de boucles, sont généralement soumises à des contraintes temporelles pour l'exécution. De plus, les exigences de l'utilisation de ces applications ont imposé aux concepteurs des valeurs de contraintes de plus en plus strictes. Ces applications sont qualifiées de temps réel. Par définition, Un système *temps réel* est « *un système dont l'exactitude des résultats ne dépend pas seulement de l'exactitude logique des calculs mais aussi de la date à laquelle le résultat est produit. Si les contraintes temporelles ne sont pas satisfaites, on dit*

qu'une défaillance système s'est produite » [47]. La borne maximale sur le temps d'exécution est appelée *contrainte temps réel*. Ces applications se caractérisent essentiellement par les notions de réactivité et de temps. Le terme réactif qualifie ainsi le comportement de ces systèmes alors que le terme temps réel est relatif aux contraintes qui leurs sont imposées [48].

On peut répartir les contraintes temporelles en deux types :

- contraintes temporelles relatives : chaque tâche du système correspond à une échéance. Cependant, une exigence temporelle non respectée n'entraîne pas une violation des fonctionnalités du système. Elle est ainsi tolérable en dépit d'une dégradation dans la qualité du service. Le retard de quelques millisecondes ou la perte d'une trame lors d'une projection audio-visuelle ne nuira qu'au confort de l'utilisateur.

- contraintes temporelles strictes [71, 72, 73]: le système doit assurer des tâches à contraintes strictes. Une échéance est associée à chaque tâche ; le système doit impérativement respecter cette échéance. Une défaillance temporelle peut engendrer des conséquences catastrophiques, telles que les applications de supervision médicale ou de pilotage automatique.

Notre travail s'intéresse aux implémentations ayant des contraintes temporelles strictes. Les implémentations intégrant des nids de boucles risquent de ne pas respecter les contraintes temporelles, en vue de leurs temps d'exécution importants. Par conséquent, il s'avère indispensable de réduire les temps des implémentations des applications. Cet objectif ne peut être atteint sans l'*optimisation* des temps d'exécution des nids de boucles.

2.5 Optimisation des nids de boucles

2.5.1 Optimisation Vs Niveau de conception

Les concepteurs sont de plus en plus confrontés au difficile problème de déterminer les implémentations qui respectent les contraintes de conception. Cet objectif nécessite le recours à des approches d'optimisation assurant l'exploration des espaces de solutions, afin de trouver une adéquation entre l'architecture et l'application développée. L'exploration peut intervenir à tous les niveaux d'abstraction du flot de conception.

L'optimisation à bas niveau de conception nécessite la mise en œuvre d'un flot de synthèse ou de compilation complet de l'application sur chacune des architectures à comparer. Ce niveau de conception assure l'obtention des mesures de performance très précises. Cependant, la synthèse met en jeu des algorithmes complexes et requiert des outils matériels et logiciels de synthèse pour chaque architecture. De ce fait, l'exploration à ce niveau est caractérisée par un temps et un coût important. Par conséquent, le concepteur est dans l'obligation de se limiter à un espace de solutions restreint.

L'optimisation à haut niveau de conception s'appuie sur l'exploration de la spécification algorithmique des solutions. Dans ce cas, les étapes de synthèse ou de compilation sont remplacées par des algorithmes de faible complexité et plus rapides que ceux utilisés lors de la synthèse. Par conséquent, elle permet d'explorer un nombre plus important de solutions. Les performances estimées peuvent être peu précises, mais la précision n'est pas le critère essentiel pour le cas de comparaison des différentes solutions. Il est par contre indispensable d'assurer une erreur relative constante pour garantir un choix fiable des résultats.

D'après [106], plus l'approche d'optimisation est appliquée à un haut niveau, plus l'espace d'exploration est plus grand, ce qui permet généralement de sélectionner une solution plus optimale, tel que formulé dans la figure 2.3. Donc, il s'avère indispensable d'appliquer

des techniques d'optimisation à haut niveau pour respecter les contraintes temporelles. Dans ce travail, nous nous intéressons à l'optimisation des nids de boucles au niveau algorithmique de conception.

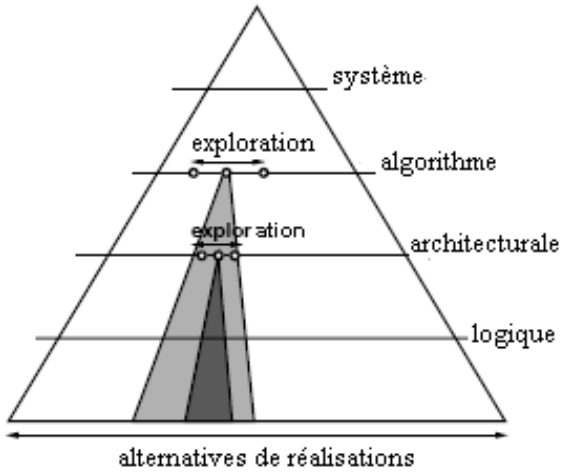


Figure 2.3. Pyramide d'abstraction : exploration successive [106]

2.5.2 Optimisation au niveau algorithmique : le parallélisme

Le principe de l'optimisation est de modifier la spécification algorithmique dans l'objectif de minimiser les caractéristiques temporelles. Dans le cas des nids de boucles, l'optimisation consiste à explorer, d'une part l'aspect répétitif au niveau de l'algorithme, et d'autre part les potentialités offertes par l'architecture, pour augmenter le niveau du parallélisme des nids de boucles [81]. La modification de la structure algorithmique est dans l'impératif de conserver les mêmes résultats que l'algorithme initial. Le parallélisme décrit une démarche de ré-ordonnancement des structures itératives, permettant de sélectionner les traitements indépendants pouvant être exécutés dans le même espace temporel. Le parallélisme est traduit par la duplication des blocs de code correspondant aux traitements à exécuter en parallèle, et ainsi par l'allocation des ressources matérielles nécessaires pour les exécuter. Par conséquent, le parallélisme améliore les temps d'exécution des nids de boucles en dépit d'une augmentation des ressources matérielles.

Le parallélisme engendre une augmentation proportionnelle en matière de coût, de consommation et de portabilité des implémentations finales. Cependant, plusieurs implémentations à base de nid de boucles sont soumises à des contraintes à la fois de temps réel et d'embarquabilité. Cette classe d'applications temps réel embarquées, principalement ciblées par nos activités de recherche, occupe une place de plus en plus importante dans le monde qui nous entoure. On les trouve maintenant aussi bien dans les produits grand public (téléphonie, automobile, appareil photo, équipements hifi et audio…) que dans les équipements industriels lourds (chaine de fabrication, ferroviaire, avionique, aérospatial …). D'où, la sélection des traitements à paralléliser est contrainte par les ressources matérielles nécessaires à leurs exécutions. Donc, il s'avère important de prendre en considération les contraintes de temps d'exécution et des ressources matérielles lors du choix du parallélisme.

2.6 Concepts de base du parallélisme

2.6.1 Démarche du parallélisme

Le parallélisme des nids de boucles se base principalement sur l'exploration des dépendances des données, dans l'objectif d'identifier les instructions à exécuter en parallèle.

Il s'avère difficile d'identifier les dépendances à partir des appels et des affectations des variables dans les instructions. La complexité de cette tâche augmente avec la taille des instructions et des chemins de données. D'une part, ce point est un facteur limitant pour définir un ordonnancement parallèle des instructions. D'autre part, le nombre des solutions de parallélisme est restreint.

Dans ce contexte, les concepteurs procèdent à représenter les nids de boucles par un modèle formel. Ce modèle doit garantir la représentation de toutes les caractéristiques algorithmiques du nid de boucles tel que les dépendances des données, les opérateurs de calcul, les intervalles des itérateurs, etc. De plus, la structure du modèle doit offrir les potentialités nécessaires pour appliquer le parallélisme, dont l'algorithme ne le permet. Un modèle de représentation est indépendant de la structure du nid de boucles à représenter, tel que soit son traitement de calcul. De plus, la structure du modèle permet un retour adéquat vers la structure algorithmique de l'application.

Par la suite, le parallélisme est formulé par une démarche d'exploration du modèle pour sélectionner les traitements indépendants, puis par une transformation de la structure du modèle. Cette transformation doit préserver les fonctionnalités initiales du nid de boucles. Chaque technique de parallélisme est caractérisée par la granularité des composantes à paralléliser, dont nous distinguons trois types :

- Parallélisme au niveau des instructions [1, 2, 4, 51, 52, 53, 54, 55] : les approches visent à réduire la taille des chemins de données appartenant à une même itération, dont l'exécution est effectuée dans un le même espace temporel. Elles procèdent à décaler des séquences des chemins dans le but d'atteindre une exécution parallèle des opérations.

- Parallélisme au niveau des itérations [5, 9, 56, 57, 58, 59, 79]: la réduction du nombre des cycles consiste à réduire le nombre d'itérations des boucles appartenant au chemin critique. Les approches procèdent ainsi à la modification des structures des boucles dans le but de paralléliser l'exécution d'un ensemble d'itérations.

- Utilisation conjointe des deux niveaux de parallélisme [10, 8, 6, 3, 31] : la démarche consiste à prédire l'apport de chaque parallélisme dans l'objectif de choisir le(s) parallélisme(s) offrant une meilleur amélioration des performances et des ressources matérielles des implémentations finales.

2.6.2 Contrainte des dépendances de données

L'analyse des dépendances des données est primordiale pour le choix du parallélisme. Pour le cas des nids de boucles sans dépendance inter-itération, les instructions du corps de la boucle interne peuvent être réordonnancer en utilisant les techniques de retiming ou de pipeline des graphes flot de données acycliques [24, 93]. De plus, les techniques de déroulage des boucles permettent de réordonnancer les N itérations du nid de boucles sur M unités de calcul tel que $1 \leq M \leq N$ [90, 91, 92]. Le parallélisme maximal consiste à allouer chaque itération à une unité de calcul. Ces deux niveaux de parallélismes peuvent être utilisés conjointement. De ce fait, les nids de boucles sans dépendances de données inter-itérations offrent un espace important de solution de parallélisme, ce qui permet au concepteur de choisir l'implémentation adéquate aux contraintes du temps d'exécution et de ressources matérielles.

Pour le cas des nids de boucles avec des dépendances inter-itérations, autres techniques ont été présentées pour assurer le parallélisme. Plusieurs d'entre elles ont été proposées pour

le parallélisme des instructions d'une seule boucle du nid. Certaines techniques sont appliquées à un niveau spécifique de boucle tel que la boucle interne [7, 23] ou la boucle externe [21]. D'autres techniques consistent à choisir le niveau de boucle dont le parallélisme assure une meilleure amélioration des performances [16]. Cependant, l'apport de ces techniques est limité au parallélisme d'un seul niveau de boucle.

Un nombre restreint de techniques adressent le parallélisme à travers des boucles imbriquées. Pour le cas du parallélisme au niveau des instructions, les techniques procèdent à décaler les instructions dans des itérations différentes à celles initiales, afin les exécuter en parallèle [20, 22, 1, 2]. Pour le parallélisme au niveau des itérations, les techniques regroupes les itérations indépendantes dans l'objectif de les exécuter en parallèle [5, 9, 14]. En présence des dépendances inter-itérations, les deux types d'ordonnancement parallèle engendrent un décalage des blocs de codes en amont et en aval des boucles imbriquées [1, 2, 9, 11], ce qui requiert plus de ressources matérielles pour l'implémentation. De ce fait, proposer une implémentation avec un rapport adéquat en « temps d'exécution » et en « ressources matérielles » est un chalenge pour les concepteurs des systèmes embarqués temps réel [17]. Dans ce manuscrit, on s'intéresse aux techniques du parallélisme à travers les niveaux du nid de boucles ayant des dépendances inter-itérations.

2.7 Modèle de représentation des nids de boucles

2.7.1 Graphe factorisé et conditionné de dépendances de données

Un nid de boucles peut être représenté par un Graphe Factorisé et Conditionné de Dépendances de Données (GFCDD), permettant la mise en évidence du parallélisme potentiel de l'algorithme. Ce graphe est modélisé par l'intermédiaire du couple (O, D) tel que [87]:

- O est l'ensemble fini des nœuds.

- D est l'ensemble fini des arcs représentant des dépendances de données.

La fonction $\gamma(d_i)$ retourne les nœuds récepteurs de l'arc d_i, et la fonction $\gamma^{-1}(d_i)$ retourne le nœud émetteur. L'ensemble des nœuds du GFCDD sont répartis en nœuds de calcul et nœuds de factorisation. Un nœud de calcul représente une instruction élémentaire telle qu'une addition, une soustraction, etc. Les nœuds de factorisation permettent de représenter l'aspect itératif des traitements dans le graphe [88, 89]. Ces nœuds sont classifiés selon le flux des données de la boucle. Les données en entrée à la boucle sont représentées par un nœud « *Fork* » (F) et les données en sortie sont représentées par un nœud « *Join* » (J). Une boucle est représentée par l'encapsulation des nœuds de calcul par les nœuds de factorisation, dont la structure est appelée frontière de factorisation. Une boucle est représentée dans le GFCDD par :

- l'insertion de nœuds de factorisation dans les arcs d'entrée et de sortie des données à la boucle.

- l'étiquetage des arcs entrants et sortants des nœuds de factorisations par les tailles des données acheminées.

- l'étiquetage de la frontière par le nombre des itérations.

Prenons l'exemple du nid de boucles de la figure 2.4(a) assurant la multiplication de deux vecteurs V et V' dans le vecteur résultat R, dont le GFCDD est schématisé dans la figure 2.4(b). La boucle est représentée par une frontière de factorisation, schématisée par un trait

discontinu. Les données en entrée des vecteurs V et V' sont modélisées par deux sommets *Fork* et les données en sortie du vecteur R sont modélisées par un sommet *Join*.

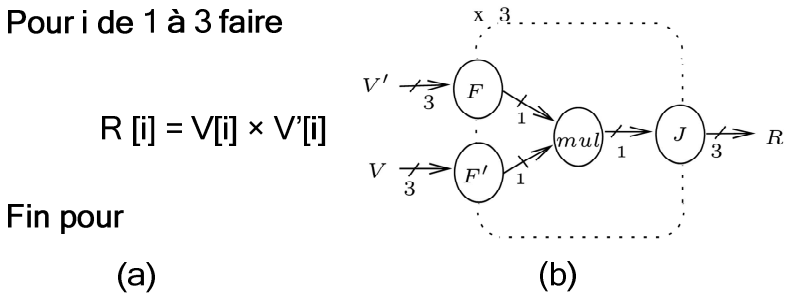


Figure 2.4 Multiplication de deux vecteurs : (a) le code de la boucle, (b) le GFCDD [87]

Le formalisme des GFCDDs permet de modéliser les dépendances de données entre deux itérations successives d'une même boucle, par l'intermédiaire d'un nœud de factorisation intitulé « *Iterate(I)* » [88]. Ce nœud récupère par un arc entrant une donnée de l'itération (i) et la fournit par un arc sortant à l'itération $(i + 1)$. Prenons le cas de la boucle de la figure 2.5(a) assurant le produit scalaire d'un vecteur M et d'un entier V dont les résultats sont rangés dans un vecteur S. Dans le GFCDD de la figure 2.5(b), le nœud *Iterate* récupère le résultat du nœud d'addition autant que contenu de $S[i]$ et fournit la même donnée au nœud d'addition pour le calcul de $S[i + 1]$.

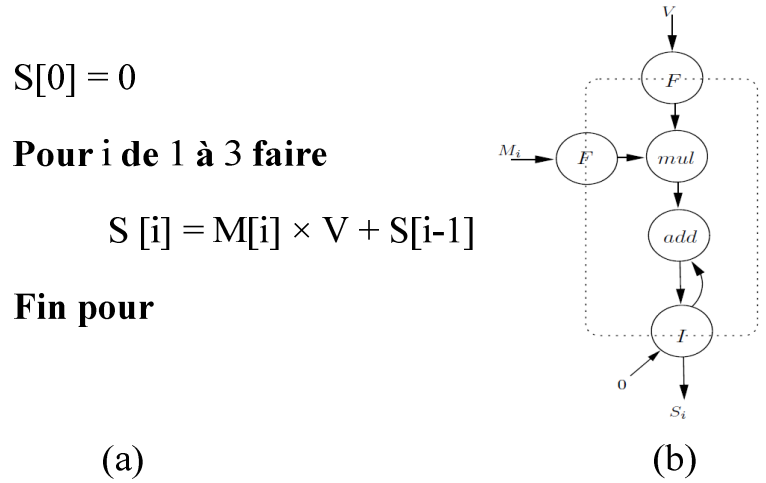


Figure 2.5 Produit scalaire d'un vecteur par un entier : (a) le code de la boucle, (b) le GFCDD [87]

2.7.2 Graphe flot de données multidimensionnel

Un Graphe Flot de Données Multidimensionnel GFDM [4, 5, 12, 13, 17, 109] permet de modéliser un nid de boucles par l'intermédiaire d'un graphe sous la forme de $G = (V, E, d, t)$ tels que V est l'ensemble des nœuds, E est l'ensemble des arcs, $d(e_i)$ est le délai multidimensionnel de l'arc e_i tel que $e_i \in E$, et $t(v_j)$ est le temps d'exécution du nœud v_j, tel que $v_j \in V$ [31, 32]. Chaque instruction est modélisée par un nœud et chaque dépendance de données est modélisée par un arc. Le sens d'une dépendance de donnée est formulé par un délai $d(e)$ sous la forme d'un vecteur de n entiers tel que $d(e) = (c_1, \dots, c_n)$. Chaque paramètre c_k indique la différence entre l'itération du nœud d'arrivée et l'itération d'exécution du nœud de départ. Une dépendance de donnée intra-itération est modélisée par un arc de délai nul $d(e) = (0,0)$. Une dépendance de donnée inter-itération est modélisée par un arc dont le délai contient au moins un indice non-nul.

Prenons l'exemple du filtre numérique d'ondelettes de la figure 2.6. L'algorithme est composé par deux boucles imbriquées, d'où chaque arc est étiqueté par un délai sous la forme d'un vecteur de deux indices $d(e) = (d.x, d.y)$, dont « $d.x$ » et « $d.y$ » correspondent respectivement à la différence des itérations par rapport à la boucle externe et à la boucle interne. L'instruction $D(i,j)$ représente la multiplication de deux variables calculées dans des itérations différentes. De ce fait, le GFDM contient deux arcs $e_4: B \rightarrow D$ et $e_5: C \rightarrow D$ ayant des délais non-nuls.

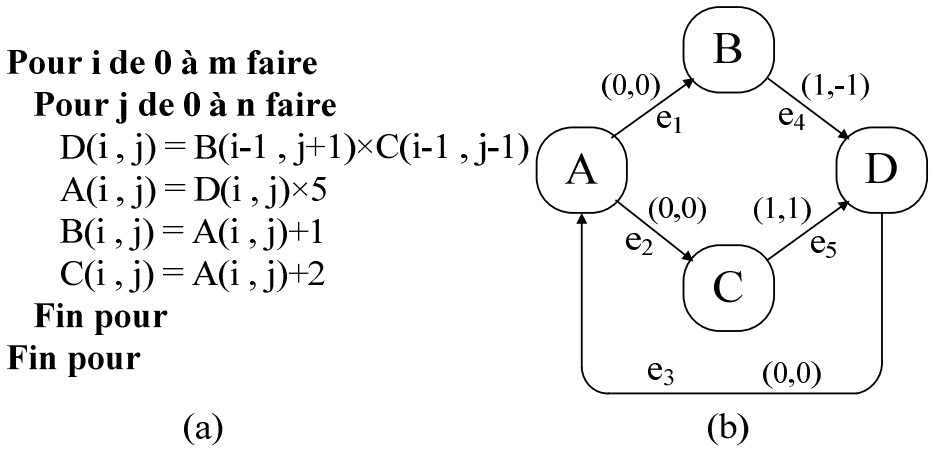


Figure 2.6 Filtre numérique d'ondelettes : (a) l'algorithme, (b) le GFDM

2.7.3 Modèle polyédrique

Le modèle polyèdrique permet de représenter le nid de boucles dans une structure matricielle [66, 67, 68]. Il assure la formulation de l'aspect itérative des boucles ainsi que les dépendances des données. Pour un nid de boucles S, le modèle polyédrique représente les itérateurs des boucles encapsulant S par le vecteur d'itération $\vec{x}$ [78]. Par la suite, il définit les bornes des itérateurs du corps S dans une structure matricielle, intitulée domaine D^S.

```
Pour i de 0 à M-1 pas 1 faire
    Pour i de 0 à N-1 pas 1 faire
        S1 : A[i,j] = A[i-1][j+1] × 3
        S2 : B[i,j] = A[i,j] + 2
    Fin pour
Fin pour
```

Figure 2.7 Nid de boucles

Prenons l'exemple du nid de boucles de la figure 2.7. Les bornes inférieures et supérieures des itérateurs des boucles peuvent être représentées sous la forme de l'ensemble des inéquations (2.2).

$$\begin{Bmatrix} i-1 \geq 0 \\ j-1 \geq 0 \\ i+N \geq 0 \\ j+N \geq 0 \end{Bmatrix} \quad (2.2)$$

Les inéquations sont représentées dans le domaine des itérations D^S sous la forme de multiplication de deux matrices, tel que définit dans l'équation (2. 3). La deuxième matrice

est composée par les itérateurs et les constantes utilisées, tandis que la première est composée de leurs coefficients dans les inéquations :

$$D^S = \left\{ \binom{i}{j} \in Z^2 \mid \begin{pmatrix} 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq 0 \right\} \qquad (2.3)$$

Chaque dépendance de données est modélisée en se basant sur les indices des variables appelées dans les instructions. Dans le cas de l'instruction $S1$, la lecture de la variable $A[i-1][j+1]$ correspond à une fonction $f_{R^A}(\vec{x})$. De même, l'écriture du résultat dans $A[i][j]$ est formulée par $f_{W^A}(\vec{x})$. Le modèle polyédrique représente les fonctions sous la forme de produit de matrices, tel que indiqué dans les équations (2.4) et (2.5).

$$f_{R^A}(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \quad (2.4)$$

$$f_{W^A}(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} (2.5)$$

2.8 Contraintes d'optimisation

Les systèmes temps réel embarqués requiert le respect d'un ensemble de contraintes lors de l'implémentation de l'algorithme sur l'architecture finale. Elles doivent être prédites à partir du modèle algorithmique du nid de boucles pour vérifier le respect des contraintes avant l'implémentation physique. De plus, les concepteurs sont dans l'obligation de quantifier l'évolution des paramètres en fonction des transformations du parallélisme du modèle. En fait, les architectures actuelles se caractérisent par une divergence des paramètres physiques et technologiques (tels que la fréquence, le type de la mémoire, etc) et des structures des architectures parallèles (telle que la répartition mémoire et communication inter-processeurs). Générer une technique d'optimisation pour chaque architecture cible est loin d'être une solution adéquate. Par ailleurs, le principe du parallélisme consiste à assurer un apport sur la performance de l'implémentation indépendamment de la cible.

Dans ce contexte, plusieurs travaux de recherche simulent d'une façon formelle les valeurs des contraintes à partir du modèle de représentation des nids de boucles tel que le temps d'exécution, le nombre des processeurs, etc. D'autres travaux se réfèrent à des paramètres algorithmiques tels que la taille du code et l'accélération. Ces paramètres permettent d'en déduire les contraintes de l'implémentation. De plus, elles garantissent les mêmes évolutions que les grandeurs physiques.

2.8.1 Contraintes temporelles

2.8.1.1 Temps d'exécution

Un algorithme peut être considéré comme une fonction de transfert de données, à laquelle est associé un retard équivalent au temps de calcul : c'est le temps d'exécution. Pour garantir que le temps d'exécution ne dépassera jamais une certaine valeur critique, à partir de laquelle les performances du système ne seraient plus satisfaisantes ou à partir de laquelle le système serait instable, il est nécessaire de borner ce temps d'exécution. Cette borne est appelée aussi *contrainte de latence [74, 75]*. De manière générale, elle correspond à l'intervalle de temps maximal pendant lequel le système réactif doit produire les actions engendrées par les données d'entrée. Elle est définie comme étant la durée totale d'une seule exécution de

l'algorithme. Au niveau algorithmique, le temps d'exécution est exprimé en fonction d'une unité élémentaire, intitulé généralement « période de cycle (P_{Cycle})» [69, 70]. Le temps d'exécution est le nombre des périodes de cycles N_{Cycles} nécessaire pour l'exécution de l'algorithme [96, 97].

- *Période du cycle [103, 2, 4]*

Le temps d'exécution de chaque instruction est généralement exprimé en fonction d'unité de temps, dont les valeurs sont proportionnelles à leurs temps d'exécution mesurés. La période de cycle P_{cycle} est définie par la durée nécessaire pour l'exécution d'une suite d'instructions aléatoires. Pour calculer ce temps du cycle, on cherche à évaluer le chemin le plus "long" (appelé chemin critique) que les données empruntent après l'implantation sur l'architecture. La longueur de ce chemin critique T_{cc}, une fois identifiée, détermine la valeur minimale de la période du cycle de l'application P_{cycle} : $P_{cycle} \geq T_{cc}$.

- *Nombre de cycles*

Cette contrainte représente le nombre de stabilité des données, depuis l'interception des données en entrées jusqu'à la génération des résultats. Cette valeur est liée à l'ordonnancement choisi pour l'implantation de l'algorithme. Le nombre de cycles dépend fortement des structures itératives appartenant à l'algorithme. Pour le cas d'un nid de boucles, la valeur représente la multiplication du nombre de cycles d'une itération par les nombres des itérations des boucles. De plus, il dépend aussi de la récursivité des dépendances de données : une opération ne peut être lancée à la période du cycle n que si toutes les données en entrées sont prêtes à la période du cycle $(n-1)$.

2.8.1.2 Accélération

Ce terme est proportionnel au temps d'exécution sur des architectures parallèles. Il représente le rapport entre le travail de l'algorithme $W(n)$, Avec n est la taille des données, et le temps d'exécution sur p processeurs $T(n,p)$ [98], tel que indiqué dans l'équation 2.6.

$$A(n) = W(n)/T(n,p) \quad (2.6)$$

Théoriquement, le temps d'exécution $T(n,p)$ diminue en fonction du nombre des processeurs. De ce fait, plus la valeur de l'accélération est grande, plus la performance de l'algorithme est importante.

2.8.1.3 Période d'itération [4]

Elle représente le temps de calcul moyen d'une itération. Pour le cas d'un déroulage d'une boucle en f instances, la période de l'itération est le temps d'exécution d'une itération divisé par f. La valeur minimale de la période d'itération T_{min} est décrite dans l'équation 2.7.

$$T_{min} = t_{max}(V)/f \quad (2.7)$$

Avec $t_{max}(V)$ est temps maximal d'exécution des instructions V, et f est le facteur de déroulage. La valeur minimale est susceptible d'être atteint dans le cas ou les instructions du corps de la boucle sont ordonnancées avec un parallélisme maximal.

2.8.2 Contraintes matérielles

La spécification logicielle ne peut être complètement indépendante de la spécification matérielle. Les opérations à exécuter ainsi que les entrées et les sorties de l'algorithme doivent correspondrent aux ressources matérielles. Il est donc nécessaire d'aboutir à une adéquation entre l'algorithme et l'architecture à partir du haut niveau de conception.

2.8.2.1 Nombre des unités de calcul

L'ordonnancement du nid de boucles consiste à explorer le modèle de représentation pour identifier les traitements à exécuter en parallèle. De ce fait, chaque traitement doit être exécuté sur une unité de calcul (CPU ou GPU). Chaque démarche de parallélisme consiste à augmenter les traitements parallèles. Cependant, ce parallélisme est contraint par le nombre des unités disponible dans l'architecture. D'où, plusieurs techniques de parallélisme procèdent à réordonnancer le nid de boucles en respectant une contrainte de nombre maximal d'unités de calcul.

2.8.2.2 Mémoire

Les mémoires assurent la collecte et l'échange des données entre les unités de calcul, suivant l'ordonnancement du traitement exigé par le compilateur. Vue les différents types de dépendances de données, l'utilisation de la mémoire diffère d'une application à une autre. Dans ce contexte, les concepteurs ont proposé des architectures à plusieurs unités de calcul, dont les structures diffèrent selon la stratégie de répartition de la mémoire. L'architecture à mémoire partagée est adéquate pour le cas d'échange intensif de données générées par les traitements parallèles des processeurs. En contraste, l'architecture à mémoire dédiée est utilisée dans le cas où les traitements affectés aux processeurs sont proportionnellement indépendants. Plusieurs autres architectures de réseaux de processeurs sont générées, combinant l'utilisation des mémoires dédiées et partagées, en se basant sur les caractéristiques d'exécution d'une famille d'application bien spécifiques. Par conséquent, les démarches d'optimisation sont dans l'impératif de choisir la hiérarchie de mémoire adéquate dans l'objectif d'améliorer les performances de l'implémentation.

2.8.2.3 Taille du code

Les techniques de parallélisme entraînent une augmentation de la taille du code du nid de boucles. Le parallélisme au niveau des itérations engendre la duplication du code itératif des corps de boucles, ce qui correspond à une duplication similaire en ressources matérielles. De plus, pour le cas des nids de boucles avec dépendances inter-itérations, les techniques de parallélisme impliquent l'ajout des blocs de code de prologue et d'épilogue à travers les structures des boucles. Ces instructions engendrent une utilisation importante des registres pour le stockage des variables de l'itération en aval. Par conséquent, la taille du code du nid de boucles est similaire à l'utilisation des ressources matérielles de l'architecture.

2.9 Techniques du parallélisme

2.9.1 Parallélisme au niveau des instructions

2.9.1.1 La technique « Outer Loop Pipelining »

Cette technique procède à réordonnancer le nid des boucles de la boucle interne à la boucle externe [21]. Ce travail affirme que l'inconvénient du pipeline logiciel est le temps d'exécution des instructions en prologue et épilogue, surtout dans le cas des latences réduites des itérations. La première étape consiste à appliquer le parallélisme au niveau des itérations de la boucle interne. La deuxième assure le parallélisme des prologues et des épilogues due au premier parallélisme.

Le temps du corps de la boucle interne est représenté sous la forme d'une latence exprimée en unité de périodes de cycle. Le parallélisme consiste à appliquer le pipeline aux périodes de cycle des itérations de la boucle interne. Prenons l'exemple du nid de boucles de la figure 2.8(a), dont la boucle interne est composée de 2 itérations. Pour le cas où la latence de l'itération est égale à 4, le pipeline décale l'exécution des itérations de la boucle interne par

un seul cycle. Cette ordonnancement est répartie en un corps de boucle incluant n cycles de chaque itération, limité par un prologue et un épilogue, tel que affiché dans la figure 2.8(b). Par la suite, elle vise à réduire leurs temps de calcul en exécutant en parallèle l'épilogue appartenant à l'itération (i) de la boucle externe avec le prologue appartenant à l'itération $(i+1)$. L'ordonnancement final du code de la figure 2.8(a) est schématisé dans la figure 2.9.

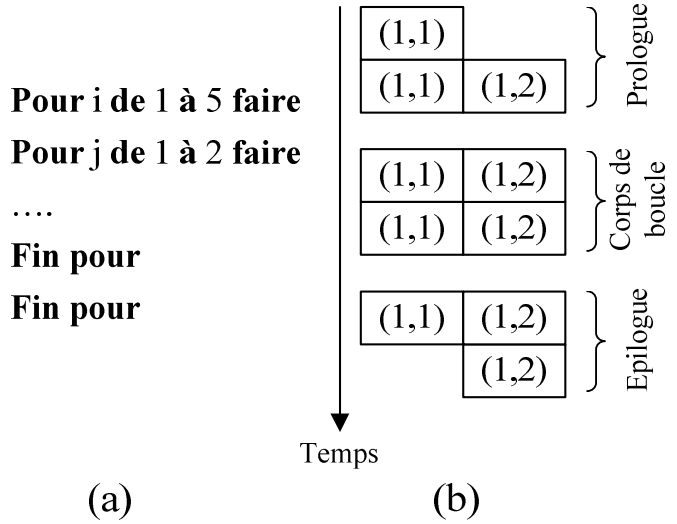


Figure 2.8 (a) Code du nid de boucles ; (b) Ordonnancement parallèle des itérations de la boucle interne

Ce parallélisme est contraint par les dépendances des données des séquences des instructions à exécuter pendant la même période de cycle. Ces dépendances de données sont celles liant l'épilogue de l'itération (i), les instructions de l'itération $(i+1)$ et le prologue le l'itération $(i+2)$. La technique est dans l'obligation de retarder le déclenchement de l'exécution de l'itération suivante de la boucle externe.

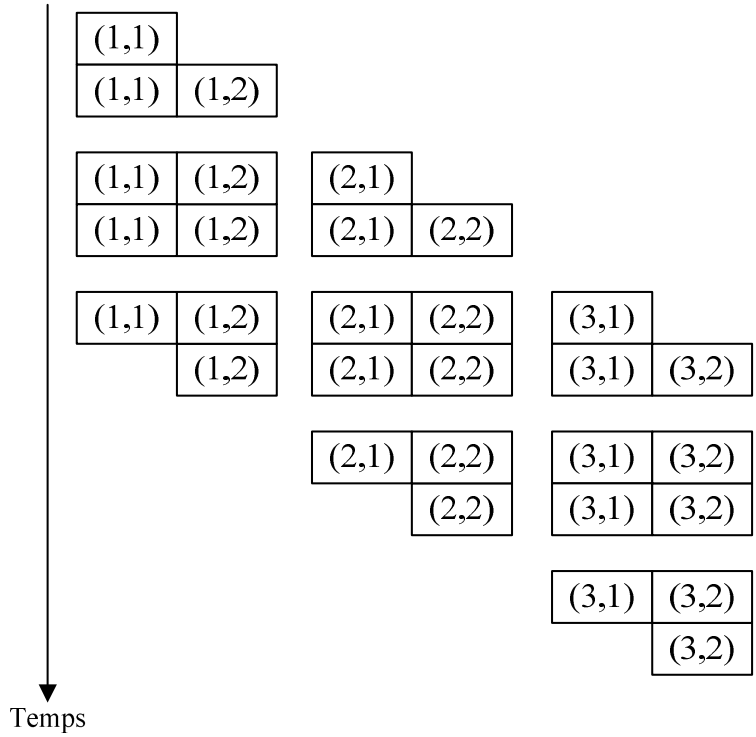


Figure 2.9 Ordonnancement du nid de boucles après la technique « Outer loop pipelining »

2.9.1.2 La technique « Polyhedral Bubble Insertion »

Cette technique représente le nid des boucles en utilisant le modèle polyédrique. Le problème des dépendances de données inter-itérations est résolu en deux étapes en fonction des dépendances de données inter-itérations [20].

Dans la première étape, la technique procède à paralléliser les instructions appartenant à la boucle interne. En fait, les instructions ne sont pas représentées explicitement. Mais, le temps d'exécution total d'une itération est représenté sous la forme d'un ensemble de périodes de cycle. Par la suite, l'ordonnancement assure le parallélisme des périodes de cycles dont le traitement est indépendant. Prenons l'exemple du nid de boucles de la figure 2.10 (a), dont le nombre des itérations de la boucle interne est égal à 5. Dans le cas où le temps d'exécution d'une itération est égal à 4 cycles, l'ordonnancement est répartit en 4 stages ; i.e., le stage numéro i assure l'exécution des $i^{ème}$ cycles des itérations. D'où, les itérations de la boucles internes sont ordonnancées telles que schématisées dans la figure 2.10 (b).

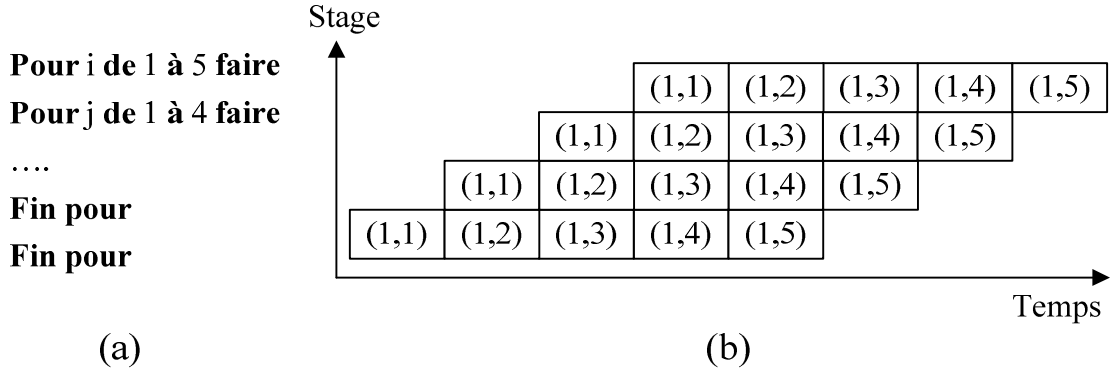


Figure 2.10 (a) Code du nid de boucles ; (b) Ordonnancement parallèle des itérations de la boucle interne

dans la deuxième étape, la technique réordonnance les traitements des itérations de la boucle externe, par l'exécution sérielle de leurs blocs de code. L'ordonnancement assure l'exécution successive des itérations (i, j) et $(i, j + 1)$, telle que schématisée dans la figure 2.11.

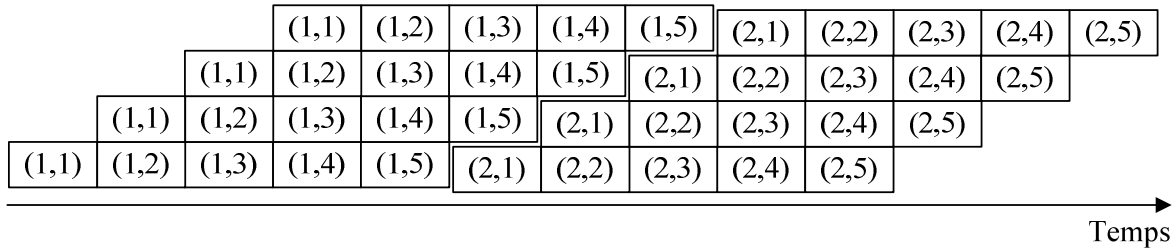


Figure 2.11 Ordonnancement du nid de boucles généré par la technique « Polyhedral Bubble Insertion »

Une étape de vérification est effectuée sur l'ordonnancement précédent : le regroupement des blocs peut altérer l'ordre de lecture et d'écriture des dépendances de données inter-itérations. Dans ce cas, des périodes de cycle de retard sont insérés entre les blocs pour assurer une exécution cohérente du nid de boucles.

2.9.1.3 L'approche du « Retiming Multidimensionnel »

Cette technique procède à redistribuer les opérations dans les itérations des boucles imbriquées. Tout algorithme est modélisé sous la forme d'un Graphe Flot de Données Multidimensionnel (GFDM) [1, 2, 3, 4]. L'algorithme de la figure 2.12(a) est formé par deux boucles imbriquées, dont le graphe bidimensionnel est schématisé dans la figure 2.12(b).

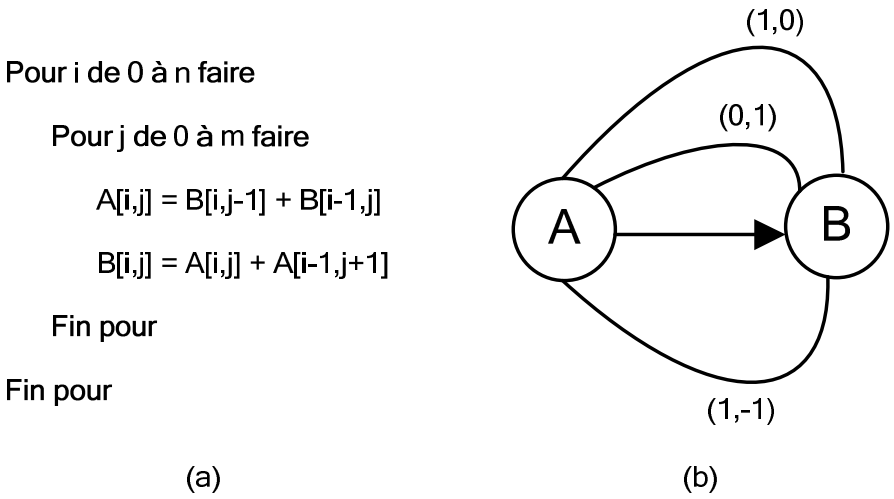


Figure 2.12 (a) Nid de boucles (b) GFDM du nid de boucles

Les arcs sont étiquetés par l'ordre des dépendance de données inter-itérations. Vue que tout chemin de données appartenant à une même itération est exécuté dans une même période de cycle, cette approche procède à réduire la taille de ces chemins en décalant l'exécution des nœuds à travers les itérations du nid de boucles. Les travaux proposés décrivent une méthode formelle pour le décalage des nœuds, en supposant que le temps d'exécution de tous les nœuds est égal à une seule unité de temps. Les techniques existantes basées sur l'approche de retiming multidimensionnel procèdent aux décalages des nœuds jusqu'à atteindre un parallélisme maximal.

2.9.2 Parallélisme au niveau des itérations

2.9.2.1 La technique « loop striping »

La technique de "loop striping" répartit les itérations du nid des boucles en se basant sur deux paramètres qui sont le facteur f et l'offset g. le premier paramètre représente le nombre des itérations collectées dans le même groupe. Le deuxième paramètre définit le sens de collection des itérations. Le choix de l'offset g est effectué de façon qu'il n'existe aucune dépendance de données entre les itérations collectées [5].

Les instructions des itérations collectées sont regroupées dans la boucle interne pour les exécuter en parallèle. Les dépendances de données inter-itérations implique l'exécution d'un nombre n d'itérations en amont de la boucle interne initiale, intitulées prologue. Simultanément, d'autres itérations sont à exécuter en aval, intitulées épilogue. L'augmentation du niveau du parallélisme consiste à l'augmentation du facteur f, dans l'objectif d'améliorer les performances des nids de boucles. Cependant, cette augmentation est proportionnelle à l'augmentation de la taille du code ce qui engendre l'augmentation des ressources matérielles de l'implémentation.

2.9.2.2 La technique « Loop tiling »

Cette technique répartit les itérations du nid de boucles en partitions formulées par n vecteurs, tel que n est le nombre des boucles imbriquées. Chaque vecteur correspond à une boucle et indique le sens d'appartenance des itérations de la boucle à la partition. La répartition est effectuée de façon qu'il n'existe aucune boucle de dépendances de données entre deux groupes, pour garantir un ordonnancement cohérent. Les techniques du « loop tiling » différent selon leurs démarches de répartition qui sont basées principalement sur l'exploitation des dépendances des données et sur les choix des vecteurs des répartitions. Les

travaux [9, 99, 100] décrivent des démarches dont les partitions ont des formes architecturales similaires point de vue vecteurs de groupement et nombre d'itérations. Dans le travail proposée dans [101], les répartitions ne sont pas définies avec les mêmes vecteurs, telles que indiquées dans l'exemple de l'espace d'itérations de la figure 2.13. La technique permet d'augmenter le niveau du parallélisme en incrémentant les itérations collectées dans le même groupe. Cependant, la taille du corps de la boucle et le nombre des itérations à exécuter en amont et en aval des boucles augmentent, entrainant ainsi une taille de code plus importante de l'implémentation. Notons que cette démarche a été décrite pour le cas des nids de boucles modélisés par le modèle polyédrique ou par d'autres modèles de représentation des nids de boucles à deux dimensions.

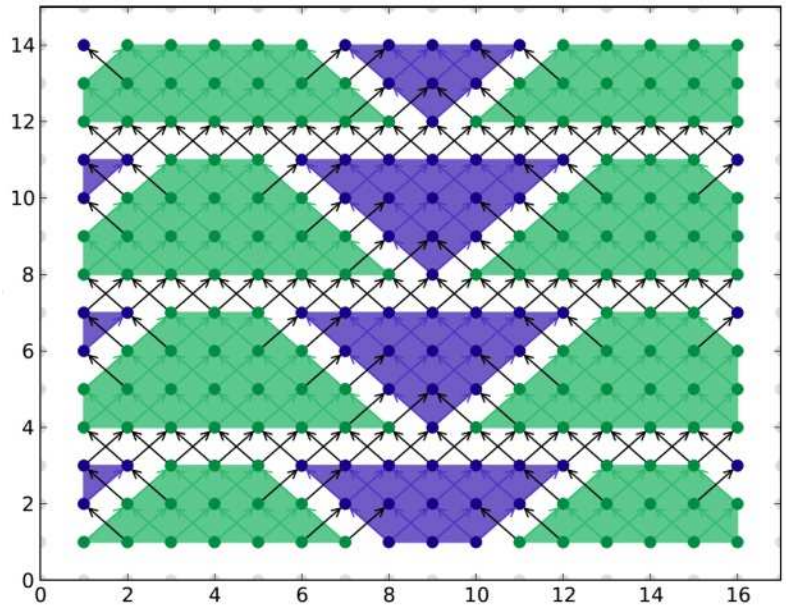


Figure 2.13 Espaces d'itérations du nid de boucles [9]

2.10 Synthèse des modèles de représentation des nids de boucles

2.10.1 Critères de classification des modèles de représentation

Après avoir passé en revue les plus importants modèles de représentation des nids de boucles et leurs techniques d'optimisation, nous avons jugé indispensable de rappeler les principales exigences et caractéristiques que doit disposer un modèle de représentation, avant de dresser leur bilan comparatif. Une telle liste pourrait servir à indiquer les défis actuels ainsi que les degrés d'efficacité de ces modèles. Ces caractéristiques portent sur plusieurs aspects de la spécification algorithmique :

- *Dimension du nid de boucles* : indique le nombre des boucles imbriquées du nid. Ce paramètre est de plus en plus primordial en vue de l'augmentation de la complexité des applications en matière de traitement itératif. Un modèle présentant explicitement la dimension du nid permet d'augmenter le nombre des scénarios du parallélisme et ainsi de choisir une solution plus optimale.

- *Occurrences des itérations des boucles :* représente le nombre des itérations de chaque boucle. Ce paramètre est indispensable lors du choix du parallélisme, vue la contrainte de décalage d'un nombre d'itérations en dehors des structures des boucles.

- *Granularité au niveau des itérations :* indique la représentation explicite de chaque itération des boucles. Ce paramètre renseigne principalement sur les caractéristiques de chaque itération tel que le vecteur d'itérations, le temps d'exécution, etc.

- *Granularité au niveau des instructions :* informe sur les caractéristiques des instructions appartenant au nid de boucles en matière de type de calcul et de temps d'exécution.

- *Dépendances des données inter-itérations :* indique la représentation des dépendances des données à travers deux itérations différentes.

- *Dépendances des données intra-itérations :* représente la formulation des dépendances de données entre deux instructions d'une même itération. Ce critère reflète l'ordre d'exécution des instructions ainsi que les différents chemins de données appartenant à une même itération. Ce paramètre est nécessaire pour définir le nombre des unités de traitement nécessaires pour l'exécution d'une seule itération.

2.10.2 Bilan des modèles de représentations des nids de boucles

Nous tentons à comparer les modèles de représentation des nids de boucles en se basant sur les critères détaillées dans le paragraphe 2.10.1. Cette évaluation est représentée dans le tableau 2-1, dont nous avons coché les critères assurés par chaque modèle. Cette étude nous a permis de relever les constatations suivantes :

Tableau 2-1 Critères des modèles de représentation des nids de boucles

Modèle / Critère	GFCDD	GFDM	Modèle polyédrique
Dimension du nid de boucles	×	×	×
Occurrences des itérations des boucles	×		×
Granularité au niveau des itérations		×	×
Granularité au niveau des instructions	×	×	
Dépendances des données inter-itérations		×	×
Dépendances des données intra-itérations	×	×	

- La formulation de l'aspect itératif diffère entre les modèles. Même si les trois modèles reflètent la dimension des nids de boucles, les occurrences des itérations sont explicitement formulées par le modèle polyédrique à travers les inéquations des valeurs des compteurs, et par le GFCDD à travers l'étiquetage des frontières de factorisation. Cependant, le GFDM fait recours à la représentation des espaces d'itérations, à partir du quel il est possible de déterminer les occurrences.

- Le GFDM et le modèle polyédrique assurent une représentation détaillée des itérations. Ce critère est validé par les potentialités offertes pour le parallélisme des itérations.

- Les modèles de représentation graphique, tel que le GFCDD et le GFDM, permettent de décrire explicitement la granularité des instructions en matière de nombre et de temps d'exécution. Ce critère reflète la quantité du traitement du corps de la boucle, ce qui assure une exploration efficace des instructions lors du choix du parallélisme.

- Le GFDM permet de modéliser les dépendances des données à l'intérieur d'une même itération. le balayage du graphe en suivant l'ordre des dépendances des données permet de calculer le temps d'exécution d'une itération et le nombre des instructions appartenant au chemin critique. Ces deux informations sont utilisées lors du choix du parallélisme au niveau des instructions. Par contre, le modèle polyédrique ne permet pas de représenter les instructions du corps de la boucle ainsi que leurs dépendances. De ce fait, les techniques du parallélisme du modèle polyédrique (« Outer Loop Pipelining » et « Polyhedral Bubble Insertion ») sont dans l'obligation d'exprimer le temps d'exécution d'une itération en fonction du nombre des périodes de cycle, sans décrire leur mode de calcul. De plus, l'absence des dépendances des données intra-itération ne permet pas de dégager les éventuels parallélismes pour l'exécution des instructions et ainsi le nombre des unités de calcul nécessaires. Par ailleurs, le formalisme du GFCDD ne permet de représenter que les dépendances inter-itérations d'une même boucle, ce qui limite l'utilisation de ce graphe pour un nombre restreint de nids de boucles.

2.11 Conclusion

Ce chapitre a présenté la structure du nid de boucles et a montré la nécessité de son optimisation en parallélisant les traitements indépendants. Le parallélisme consiste à modéliser le nid de boucles par un modèle formel, puis d'appliquer des transformations reflétant le parallélisme sur le modèle. Chaque modèle assure la représentation d'un ensemble de critères algorithmiques du nid de boucles. Une technique de parallélisme explore un ensemble de critères, représentés par le modèle, pour le choix du parallélisme. De ce fait, plus un modèle est capable de représenter des critères algorithmiques, plus il offre des potentialités de parallélisme aux techniques d'optimisation.

Nous avons comparé dans la section précédente les critères des trois modèles de représentation des nids de boucles. Nous avons constaté que le GFDM est spécifié par une représentation explicite de la granularité au niveau des itérations et au niveau des instructions. De plus, il assure la représentation de différents types des dépendances de données des instructions, que ce soient inter ou intra itérations. Ces critères offrent un espace de solution important pour le choix du parallélisme. Nous décrivons dans le chapitre suivant la représentation des nids de boucles par les GFDM, et nous présentons explicitement ses techniques d'optimisation en vue d'étudier l'exploitation des critères précédemment décris.

CHAPITRE 3 : GRAPHE DE FLOT DE DONNÉES MULTIDIMENSIONNEL ET TECHNIQUES DE PARALLÉLISME

3.1 Introduction

Nous avons montré dans le chapitre précédent que les applications intégrant des nids de boucles sont généralement soumises à des contraintes temporelles et de ressources matérielles. Dans ce contexte, le processus d'optimisation consiste à représenter le nid de boucles par un modèle, et à formuler le parallélisme autant qu'une transformation du modèle. A partir de l'étude comparative des modèles de représentation effectuée dans la section 2.10, nous avons constaté que le Graphe Flot de Données Multidimensionnel "GFDM" permet une représentation efficace des granularités au niveau des itérations et au niveau des instructions. Cette représentation graphique a été le sujet de plusieurs techniques d'optimisation qui visent à augmenter le niveau du parallélisme de l'ordonnancement des nids de boucles afin de réduire le temps d'exécution. Le principe du parallélisme consiste à explorer les dépendances de données du graphe dans l'objectif de sélectionner les traitements à exécuter en parallèle. Nous distinguons des techniques de parallélisme au niveau des itérations et autres au niveau des instructions. Ces dernières assurent le pipeline logiciel dont la démarche est intitulée « retiming multidimensionnel ».

Nous débutons ce chapitre par l'étude du formalise des GFDMs ainsi que les graphes d'ordonnancement permettant de décrire l'ordre d'exécution des itérations et des instructions. Par la suite, nous décrivons les techniques d'optimisation et leurs démarches de parallélisme. Cette étude nous permettra d'établir à la fin du chapitre un bilan de synthèse des techniques d'optimisation existantes.

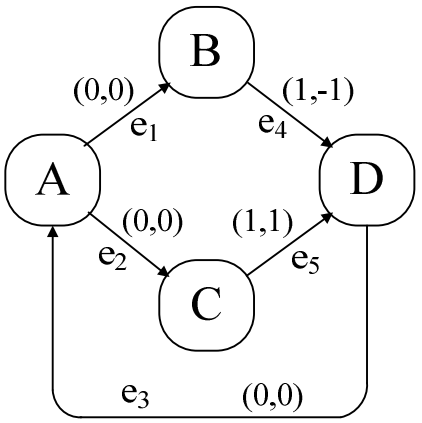
3.2 Formalisme graphique des nids de boucles

3.2.1 Graphe flot de données multidimensionnel

Un GFDM est une extension du graphe flot de données acyclique. Son formalisme est adéquat à la représentation des algorithmes contenant des structures répétitives et récursives imbriquées. Ce graphe est modélisé sous la forme de $G = (V, E, d, t)$ tels que V est l'ensemble des nœuds, E est l'ensemble des arcs, $d(e_i)$ est le délai multidimensionnel de l'arc e_i tel que $e_i \in E$, et $t(v_j)$ est le temps d'exécution du nœud v_j, tel que $v_j \in V$ [31, 32, 109]. Pour un algorithme donné, chaque instruction est modélisée par un nœud et chaque dépendance de donnée est modélisée par un arc. Chaque GDFM est caractérisé par une dimension n qui correspond au nombre de boucles imbriquées. Les dépendances de données sont schématisées par des délais qui représentent des vecteurs d'étiquetage des arcs. Pour un GFDM de n dimensions, un arc $e: u \rightarrow v$ est étiqueté par un délai sous la forme d'un vecteur de n entiers tel que $d(e) = (c_1, \dots, c_n)$. Chaque paramètre c_k indique l'ordre d'exécution du nœud d'arrivée par rapport au nœud de départ. La valeur de ce paramètre représente la différence entre le numéro de l'itération exécutant v et le numéro de l'itération exécutant u par rapport à la boucle d'indice k.

Prenons l'exemple du filtre numérique d'ondelettes dont l'algorithme illustré dans la figure 3.1(b) est composé par deux boucles imbriquées. Sa représentation graphique correspond à un graphe flot de données bidimensionnel tel que schématisé dans la figure 3.1(a). Chaque arc est étiqueté par un délai sous la forme d'un vecteur de deux indices $d(e) = (d.x, d.y)$, dont « $d.x$ » et « $d.y$ » correspondent respectivement à la différence des itérations par rapport à la boucle externe et à la boucle interne. Un arc $e: u \rightarrow v$ tel que $d(e) = (0,0)$, appelé délai nul, correspond à l'exécution des nœuds u et v dans la même itération de la boucle externe que la boucle interne, tel que les arcs $e_4: D \rightarrow A$, $e_1: A \rightarrow B$ et $e_2: A \rightarrow C$ du GFDM de la figure 3.1(a). Pour un arc $e: u \rightarrow v$, un délai $d(e) = (0, x)$ indique que les nœuds u et v sont exécutés dans une même itération par rapport à la boucle

externe. Pour la boucle interne, si le nœud u est exécuté dans l'itération k, le nœud v est exécuté dans l'itération $(k + x)$. Par exemple, le délai $d(e4) = (1, -1)$ allant du nœud B à D signifie que le nœud B est exécuté une itération avant le nœud D, par rapport à la boucle externe, et que le nœud B est exécuté une itération après le nœud D, par rapport à la boucle interne.



(a)

Pour i de 0 à m faire
  Pour j de 0 à n faire
    D(i , j) = B(i-1 , j+1)×C(i-1 , j-1)
    A(i , j) = D(i , j)×5
    B(i , j) = A(i , j)+1
    C(i , j) = A(i , j)+2
  Fin pour
Fin pour

(b)

Figure 3.1 Filtre numérique d'ondelettes : (a) le GFDM, (b) l'algorithme

Un chemin de données p est défini comme étant une succession de nœuds liés par des dépendances de données, noté par $p: v_i \xrightarrow{e_m} v_{i+1} \xrightarrow{e_{m+1}} \dots \xrightarrow{e_n} v_j$ ou $v_i \overset{p}{\Rightarrow} v_j$. Les extrémités d'un chemin doivent être impérativement des nœuds de calcul, ce qui implique que le nombre des nœuds excède le nombre des dépendances de données par une seule unité. Le temps d'exécution $t(p)$ d'un chemin $p: v_i \xrightarrow{e_m} \dots \xrightarrow{e_n} v_j$ est la somme des temps d'exécution des nœuds de ce chemin, tel que $t(p) = \sum_{k=i}^{k=j} t(v_k)$. De même, le vecteur de délais $d(p)$ d'un chemin p est la somme des vecteur des délais des arcs appartenant au chemin tel que $d(p) = \sum_{k=m}^{k=n} d(e_k)$. Deux nœuds interconnectés par un arc de délais nuls sont exécutés dans la même itération, et ainsi exécutés dans la même période de cycle. La période d'un cycle $C(G)$ d'un graphe GFDM est égale à la valeur maximale des temps d'exécution des chemins p ayant $d(p) = (0, \dots, 0)$, tel que indiquée dans l'équation 3.1 [7].

$$C(G) = max\{t(p), d(p) = (0, \dots, 0)\} \quad (3.1)$$

Dans le cas du GFDM de la figure 3.1(a), la période de cycle est calculée en se référant au chemin $p: D \xrightarrow{e_3} A \xrightarrow{e_1} B$ ou $p: D \xrightarrow{e_3} A \xrightarrow{e_2} C$, dont les arcs e_1, e_2 et e_3 ont des délais nuls. Dans le cas où les temps d'exécution des nœuds sont $t(A) = t(B) = t(C) = t(D) = 1$, la valeur de la période de cycle est $C(G) = 3$, tel que illustré dans l'ordonnancement statique de la figure 3.2.

3.2.2 Les graphes d'ordonnancement des GFDMs

L'ordonnancement d'un algorithme dépend principalement des dépendances de données inter-itérations et intra-itérations. De ce fait, l'espace des itérations permet de représenter graphiquement les différentes itérations du nid de boucles ainsi que toutes les occurrences des instructions et les dépendances de données. Nous illustrons dans la figure 3.3(a) l'espace d'itérations du GFDM de la figure 3.1(a). C'est une représentation cartésienne sous la forme d'une grille à deux axes, dont l'axe vertical correspond à la boucle interne et l'axe horizontal à la boucle externe. Chaque partition de la grille représente une itération englobant tous les

nœuds qui sont exécutés dans la même itération de l'algorithme. Chaque itération est étiquetée par un vecteur indiquant les numéros d'itérations par rapport aux boucles. On note que l'itération (0,0) est la première à être exécutée. Dans l'exemple rapporté par la figure 3.3, on se limite à schématiser (3×3) itérations. L'espace d'itérations total peut être directement déduit de la séquence illustrée dans la figure 3.3(a). Les dépendances intra-itérations sont représentées avec des arcs discontinus, et les dépendances inter-itérations par des arcs continus. Les instances des arcs e_4 et e_5 du GFDM sont représentées respectivement par des arcs des extrémités vides et des arcs des extrémités remplies. Le sens des dépendances $C \rightarrow D$ indique l'incrémentation des indices de l'itération par rapport à la boucle externe et la boucle interne.

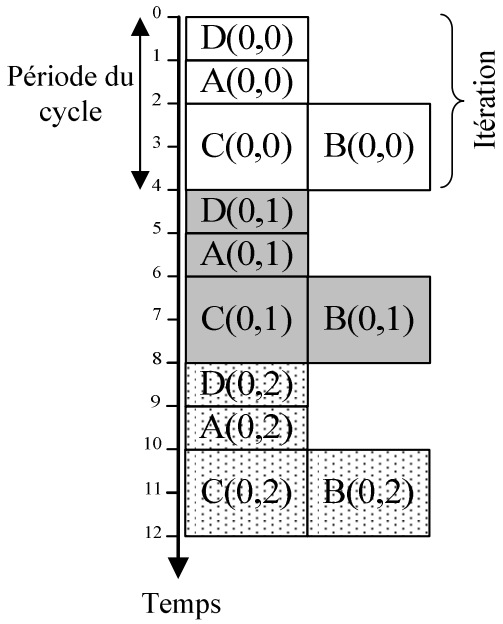


Figure 3.2 Ordonnancement des itérations du filtre numérique d'ondelettes

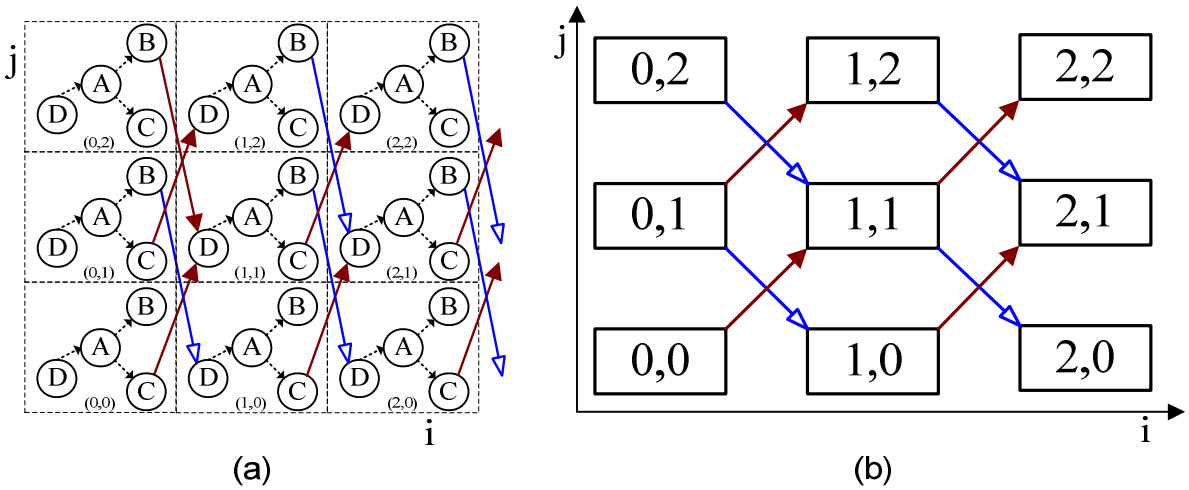


Figure 3.3 Filtre numérique d'ondelettes : (a) espace d'itérations, (b) graphe de dépendance de cellules

Le Graphe de Dépendance de Cellules (GDC) est une représentation déduite de l'espace des itérations, permettant de schématiser uniquement les dépendances de données inter-itérations du GFDM dont les délais sont non-nuls. pour le cas du filtre numérique d'ondelettes, le GDC n'assure la représentation que des arcs e_4 et e_5, tel que affiché dans la figure 3.3(b).

De ce fait, il est formé par un ensemble de cellules représentant chacune une itération, et des arcs représentant les dépendances des données entre les itérations. Un modèle mathématique pour le graphe de dépendance des données peut être présenté sous la forme d'un n-uplets (I^n, D) avec I^n est l'ensemble des indices des cellules, et D est une matrice contenant tous les vecteurs des délais inter-itérations. Pour le nid de boucles schématisé dans la figure 3.3 avec m=30 et n=30, le graphe de dépendances de cellule est décrit par le couple (I^2, D) dont les structures sont décrites respectivement dans l'ensemble (3.2) et la matrice (3.3).

$$I^2 = \{(i,j): 0 \leq i \leq 30, 0 \leq j \leq 30\} \quad (3.2)$$

$$D = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \quad (3.3)$$

3.2.3 Vecteur d'ordonnancement

Le graphe de dépendances de données est utilisé pour déterminer un ordre d'exécution des itérations, permettant un déroulement adéquat du nid de boucles et ainsi vérifiant une exécution cohérente de l'application. Cet ordre d'exécution doit garantir qu'une itération n'est exécutée que si les données en provenance des autres itérations (acheminer par des arcs de délais non nuls) ont été bien exécutées. Si cet ordre d'exécution existe alors le GFDM $G = (V, E, d, t)$ est qualifié *réalisable.* Un formalise mathématique a été proposé par [2] permettant de vérifier cette caractéristique. Il consiste à prouver l'existence d'un vecteur d'ordonnancement s du graphe de dépendance de cellules tel que $d \times s \geq 0$ pout tout $d \in G$. Un vecteur d'ordonnancement s représente un vecteur normal à tout les hyper-plans représentant les arcs de délais non-nuls. L'ordre d'exécution correspondant au vecteur s ne doit entraîner aucun cycle entre les cellules du GDC. Un vecteur assurant un ordre d'exécution des cellules du graphe de dépendances de données, est un vecteur qui respecte l'exécution de G. Le GFDM du filtre numérique d'ondelettes peut être exécuté suivant le vecteur d'ordonnancement $s = (0,1)$. Cet ordre signifie que les itérations peuvent être exécutées en incrémentant le compteur de la boucle interne. Par contre, on peut facilement constater à partir du graphe de dépendances des cellules que le vecteur d'ordonnancement $s = (1,0)$ ne permet pas d'obtenir un GDFM réalisable, vue la dépendance de données $B \rightarrow D$. On note que pour deux vecteurs bidimensionnels $P = (P.x, P.y)$ et $Q = (Q.x, Q.y)$, l'addition des deux vecteurs est égale à $P + Q = (P.x + Q.x, P.y + Q.y)$, et le produit scalaire est égal à $P.Q = P.x \times Q.x + P.y \times Q.y$.

Ce vecteur est déterminé en fonction des délais non nul du GFDM. L'ensemble de tous les vecteurs d'ordonnancement possibles à un GFDM est intitulé sous espace d'ordonnancement tel que décrit dans la définition 3.1.

> ***Définition 3.1.*** *[1, 2] Un sous espace d'ordonnancement S d'un GFDM réalisable $G = (V, E, d, t)$ est la région de l'espace où il existe des vecteurs d'ordonnancement qui maintiennent un ordonnancement réalisable de G, i.e., si le vecteur $s \in S$ donc $d(e) \times s \geq 0$ pout tout $e \in E$.*

L'espace d'ordonnancement peut être schématisé par l'intersection d'un ensemble d'hyperplans dont chacun correspond à un délai non-nul du graphe. Prenons le cas du filtre numérique d'ondelettes initial dont l'espace d'ordonnancement est coloré en gris dans figure 3.4. Les dépendances de données e_4 et e_5 définissent un demi-plan, respectivement limité par les droites pointillées. Les équations de ces droites sont déterminées à partir des délais des arcs. Chaque hyperplan contient l'ensemble des vecteurs s dont les résultats de leurs multiplications

avec les délais sont supérieurs ou égaux à zéro. A partir du sous espace d'ordonnancement, le GFDM de la figure 3.1(a) peut être ordonnancé en suivant le vecteur $s = (1,0)$.

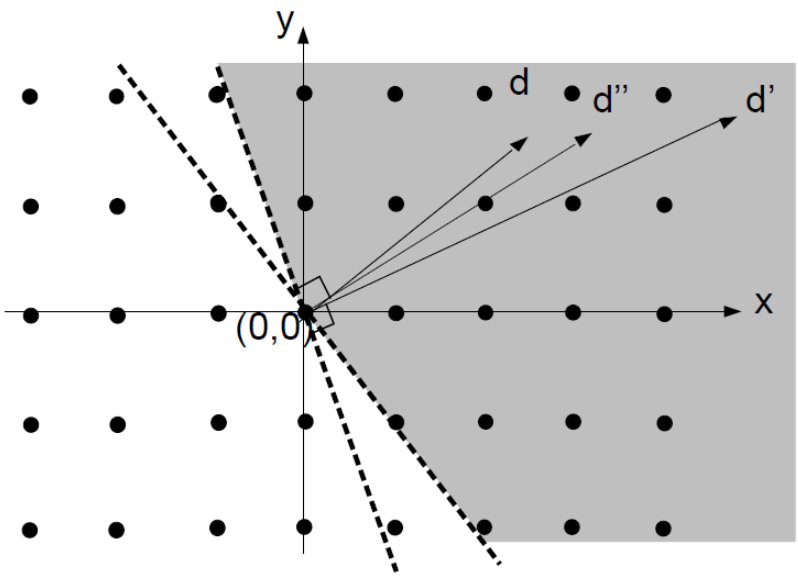


Figure 3.4 Espace d'ordonnancement du filtre numérique d'ondelettes

3.3 Parallélisme au niveau des itérations

3.3.1 Principe

Une famille des techniques d'optimisation vise à augmenter le niveau de parallélisme des itérations dans les applications multidimensionnelles, dans l'objectif de réduire le nombre des cycles nécessaires pour l'exécution du nid de boucles. Cette transformation exige que les itérations regroupées n'aient pas de dépendances de données entre elles. La sélection des itérations à exécuter en parallèle doit respecter les fonctionnalités initiales de l'application. Le choix de cette répartition est basé sur les dépendances de données inter-itérations ainsi que de la contrainte du temps d'exécution à atteindre.

3.3.2 Les techniques de parallélisme

3.3.2.1 Loop striping

La technique du "loop striping" modifie le GFDM en se référant à deux paramètres qui sont le facteur f et l'offset g [5]. Le premier paramètre représente le nombre des itérations qui sont collectées dans le même groupe. Le deuxième paramètre indique le sens de collection des itérations ; i.e, l'itération $(1,0)$ et l'itération $(0,g)$ sont placées dans le même groupe, si le facteur f est égal à 2. Le choix de l'offset g est effectué de façon qu'il n'existe pas de dépendances de données entre les itérations collectées [6]. Ce critère est la différence majeure entre la technique de loop striping et le déroulage de boucles. Prenons l'exemple du GFDM de la figure 3.1(a), le loop striping permet de regrouper des itérations en suivant les paramètres $f = 2$ et $g = 2$, dont le groupe est représenté par une ellipse dans le GDC de la figure 3.5(a).

Les instructions des itérations collectées sont regroupées dans la boucle interne. l'itérateur de la boucle externe est incrémenté par un pas égal au facteur f. De plus, dans le cas d'une valeur d'offset g non-nulle, les itérations $(0,x)$, dont $0 < x < f$, n'appartiennent à aucun groupe. Par conséquence, elles doivent être exécutées avant la boucle interne. De ce fait, une boucle est ajoutée en amont de la boucle interne initiale, intitulée prologue. Simultanément,

une autre boucle est ajoutée en aval, intitulée épilogue. Par exemple, le loop striping avec $f = 2$ et $g = 2$ appliquée au filtre numérique d'ondelettes génère l'algorithme de la figure 3.6(b). Les instructions de la boucle interne sont dupliquées par rapport à l'algorithme initial. Les itérations $(0,0)$ et $(0,1)$ de l'algorithme initial sont exécutées dans le prologue, tandis que les itérations $(n, i+1)$ et $(n-1, i+1)$ sont exécutées dans l'épilogue.

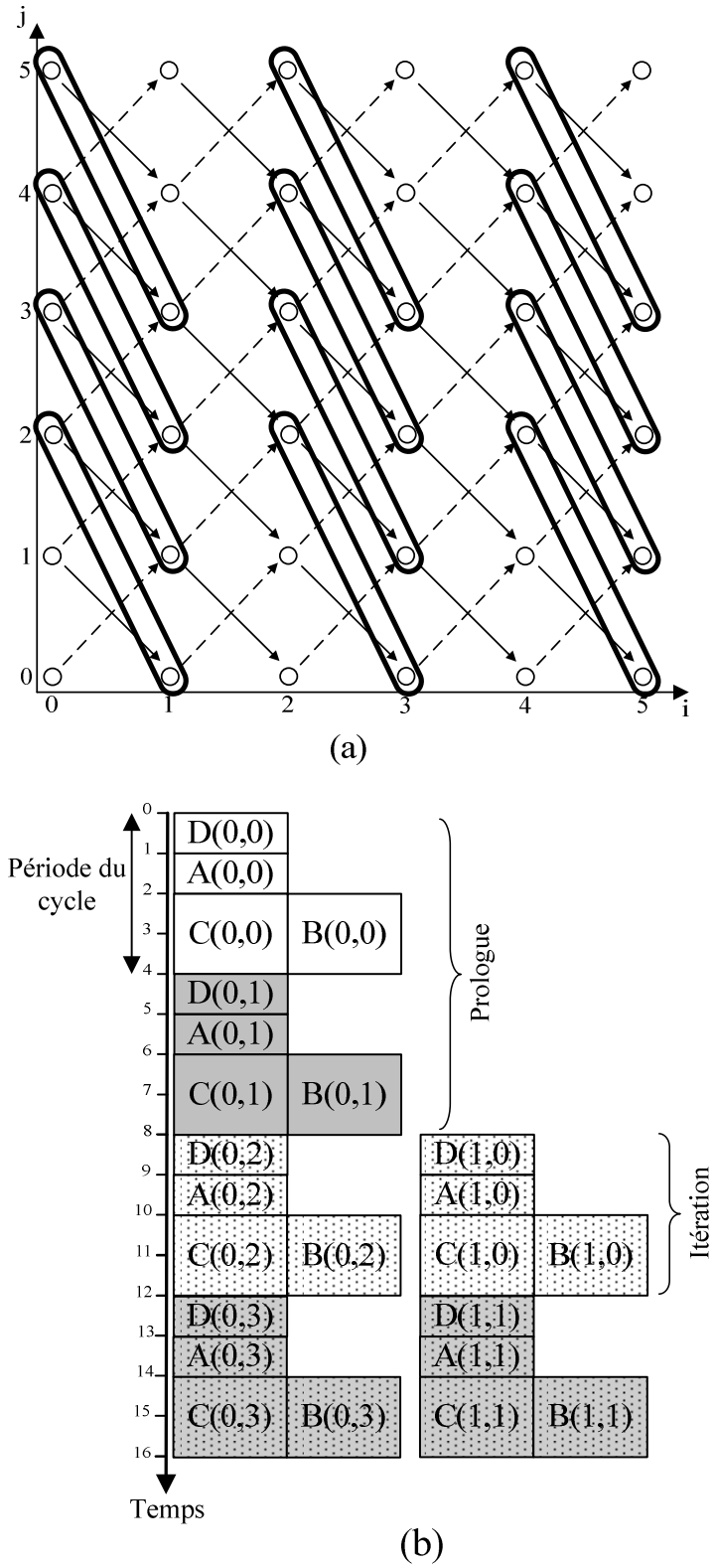
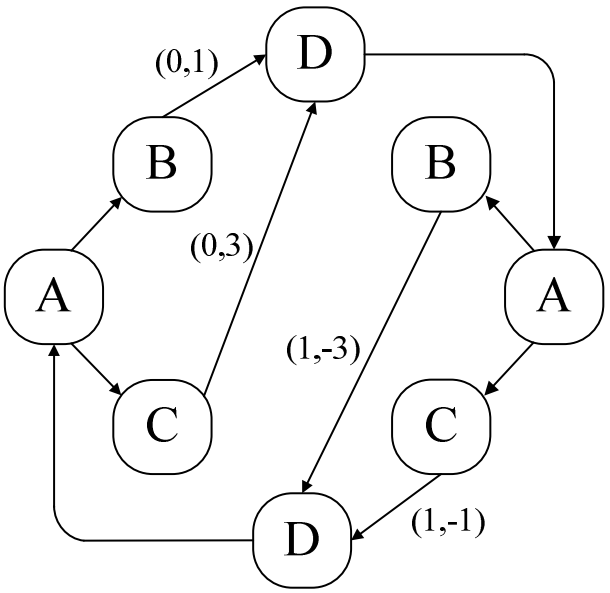


Figure 3.5 Filtre numérique d'ondelettes après loop striping: (a) le graphe de dépendance de cellules, (b) l'ordonnancement statique des itérations

Due à la duplication des instructions dans chaque itération, le GFDM généré par la technique de loop srtiping intègre deux occurrences de chaque nœud appartenant au GFDM initial. Le GFDM du filtre numérique d'ondelettes après loop striping est schématisé dans la figure 3.6(a). Cette technique n'entraîne pas la modification des dépendances de données intra-itérations. De ce fait, elle préserve un délai nul dans les arcs $D \rightarrow A$, $A \rightarrow B$ et $A \rightarrow C$, tel que le GFDM initial. Les délais non nuls sont modifiés en se référant au groupement des itérations illustré dans le GDC de la figure 3.5(a). L'ordonnancement statique de l'algorithme après loop striping est représenté dans la figure 3.5(b), dans laquelle les instructions appartenant à la même itération dans le code initial sont représentées par le même motif.

Dans le cas ou $m = 20$ et $n = 20$, la technique de loop striping permet de réduire le nombre des cycles de 441 à 291, et ainsi réduire le temps d'exécution de 1764 à 1164 unité de temps, représentant une amélioration de 34.01%. En revanche, ce parallélisme engendre l'ajout de 17 instructions de prologue et d'épilogue, représentant d'une augmentation de 50% de la taille du code final.



(a)

Pour i de 0 à m pas 2 faire
Pour j de 0 à g-1 faire
D(i , j) = B(i-1 , j+1)×C(i-1 , j-1)
A(i , j) = D(i , j)×5
B(i , j) = A(i , j)+1
C(i , j) = A(i , j)+2
Fin pour
Prologue
Pour j de g à n faire
D(i , j) = B(i-1 , j+1)×C(i-1 , j-1)
A(i , j) = D(i , j)×5
B(i , j) = A(i , j)+1
C(i , j) = A(i , j)+2
D(i+1 , j-2) = B(i , j-1)×C(i , j-3)
A(i+1 , j-2) = D(i+1 , j-2)×5
B(i+1 , j-2) = A(i+1 , j-2)+1
C(i+1 , j-2) = A(i+1 , j-2)+2
Fin pour
Pour j de n-(g-1) à n faire
D(i+1, j) = B(i , j+1)×C(i , j-1)
A(i+1, j) = D(i+1 , j)×5
B(i+1 , j) = A(i+1 , j)+1
C(i+1 , j) = A(i+1 , j)+2
Fin pour
Epilogue
Fin pour

(b)

Figure 3.6 Filtre numérique d'ondelettes après loop striping: (a) le GFDM, (b) l'algorithme

3.3.2.2 Technique du « retiming itérationnel »

Cette technique procède au parallélisme des itérations en deux étapes : la première consiste à répartir les itérations en groupes similaires; la deuxième étape consiste à paralléliser l'exécution des itérations du même groupe [13, 76]. Cette technique vise à réduire le temps moyen d'exécution d'une itération du nid de boucles.

Dans ce contexte, elle explore les dépendances de données de l'espace d'itérations. Les partitions sont définies en se basant sur n vecteurs, tel que n est la dimension du nid de boucles. Chaque vecteur d'une boucle indique le sens de regroupement des itérations de la boucle. Pour le cas de l'espace d'itération de la figure 3.7, les partitions sont encadrées par des traits pointillés. L'espace d'itération est répartie en groupe dont les vecteurs de la boucle externe et interne sont respectivement $(0,1)$ et $(2,-2)$. La répartition est faite de façon qu'il n'existe pas de boucle de dépendances entre deux partitions, afin de permettre une exécution cohérente. Les dépendances des données inter-itérations peuvent impliquer l'exécution d'un ensemble d'itérations en dehors des groupes, sous la forme de prologue et d'épilogue, tel que les itérations (0,0), (0,1) et (1,0) de la figure 3.7.

Par la suite, les itérations appartenant à la même partition sont représentées dans un graphe intitulé « Graphe de Flot d'Itérations (GFI) » dont les nœuds représentent des itérations et les arcs représentent les dépendances des données inter-itérations. Ces dépendances de données sont étiquetées par des délais indiquant l'ordre d'exécution des itérations. Chaque partition de l'espace des itérations de la figure 3.7 est modélisée par le GFI de la figure 3.8. La démarche de retiming itérationnel fait recours au retiming des graphes synchrones [24] pour exécuter en parallèle tout les itérations d'une même partition.

L'augmentation du niveau de parallélisme consiste à augmenter le nombre des itérations dans chaque groupe. Cependant, un niveau de parallélisme élevé engendre l'augmentation du nombre des itérations à exécuter en prologue et en épilogue. Cette technique a été étendue dans [76] en décrivant la façon de répartir les itérations dans l'objectif de réduire le temps d'accès à la mémoire cache et ainsi d'améliorer la performance de l'implémentation.

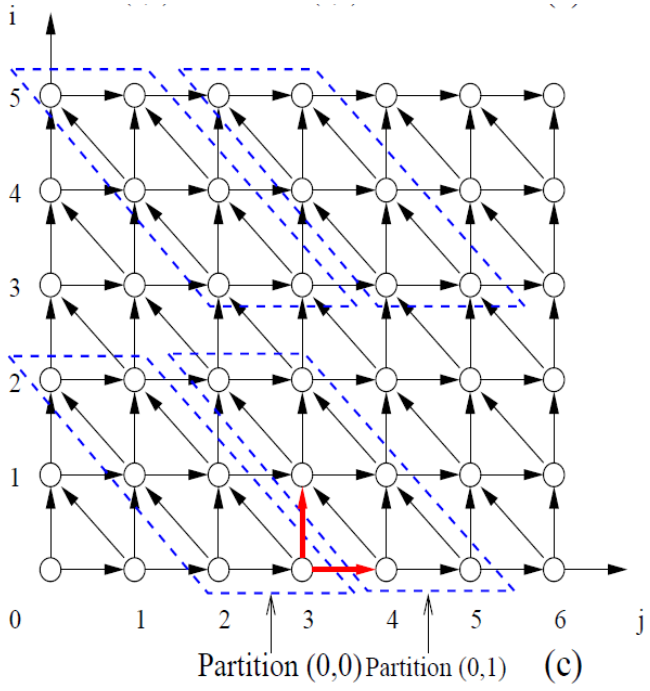


Figure 3.7 Espaces d'itérations de l'algorithme multidimensionnel [76]

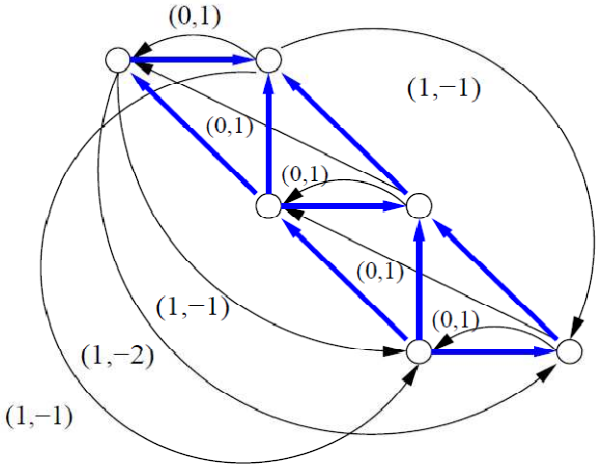


Figure 3.8 Graphe flot d'itérations [76]

3.4 Parallélisme au niveau des instructions

3.4.1 Principe

Cette famille de techniques procède à modifier l'ordre d'exécution des instructions du corps de la boucle pour paralléliser leurs traitements. Ce parallélisme correspond à un pipeline logiciel intitulé « retiming multidimensionnel ». Dans le cas des GFDMs, le retiming multidimensionnel consiste à déplacer des valeurs de délais à travers les arcs du graphe. Un retiming multidimensionnel r appliqué sur le nœud u consiste à soustraire le vecteur r des arcs entrants à u et de l'ajouter aux arcs sortant de u, tel que $u \in V$ De ce fait, cette transformation modifie l'ordre d'exécution des nœuds à travers les itérations. Un retiming multidimensionnel r est une fonction dans V qui modifie l'ordonnancement des nœuds du graphe de façon qu'elle entraîne l'exécution du nœud dans une itération, autre que celle d'origine. Cette modification est appliquée sur toutes les occurrences d'un même nœud pour toutes les itérations.

Pour un GFDM de n dimensions, une fonction de retiming multidimensionnel r appliquée sur le nœud u est modélisée sous la forme d'un vecteur de taille n ($r(u) = (r_1, \dots, r_n)$). Chaque indice r_i correspond au décalage de l'exécution de nœud u par rapport à la boucle. l'occurrence du nœud originalement exécuté dans l'itération x est exécutée dans l'itération $(x + r_1)$.

Prenons le cas du GFDM de la figure 3.1(b) dont les nœuds $D(i,j)$, $A(i,j)$, $B(i,j)$ et $C(i,j)$ sont exécutés dans l'itération (i,j). Dans le cas de l'application de la fonction de retiming multidimensionnel $r = (0,1)$ sur le nœud D, un vecteur $(0,1)$ est retiré des arcs e_4 et e_5 et ajouté à l'arc e_3, tel que illustré dans le GFDM de la figure 3.9(a). La transformation du délai nul de l'arc $e_3: D \rightarrow A$ en délai de valeur $(0,1)$ implique que les nœuds D et A ne sont plus exécutés dans la même période de cycle. Leurs exécutions sont décalées d'une seule itération par rapport à la boucle interne : si D est exécuté dans l'itération i alors A est exécuté dans l'itération $(i + 1)$. Cependant, les nœuds sont toujours exécutés dans la même itération par rapport à la boucle externe. Ce décalage est fait pour toutes les occurrences des nœuds D et A dans toutes les itérations de l'application.

Pour le cas de la première occurrence du nœud A appartenant à la première itération de la boucle interne, le retiming multidimensionnel entraîne l'exécution de la première instance de l'instruction $D(i,0)$ en amont de la structure de la boucle interne, telle que décrit dans l'algorithme du Figure 3.9 (b). L'ensemble des instructions avant chaque structure itérative dû

à l'application du retiming multidimensionnel est intitulé prologue. De même, les dernières instances des instructions $A(i,n)$, $B(i,n)$ et $C(i,n)$ appartenant à la dernière itération sont exécutées en aval de la boucle interne, appelées épilogue. la nouvelle valeur du délais $d(e_3) = (0,1)$ ne permet plus l'exécution du GFDM en suivant le vecteur $s = (0,1)$.

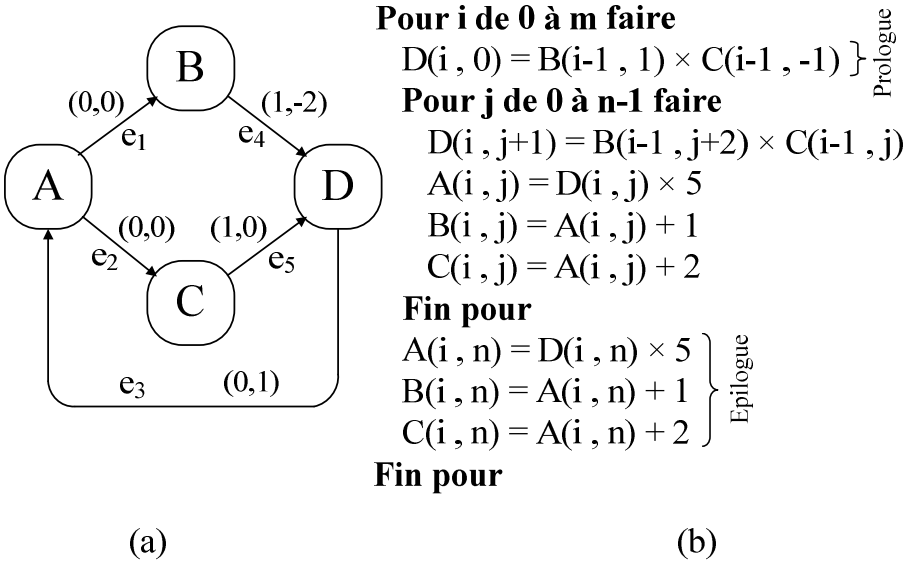


Figure 3.9 Filtre numérique d'ondelettes après retiming multidimensionnel : (a) le GFDM, (b) l'algorithme

On illustre dans la figure 3.10(a) l'espace d'itérations correspondant au GFDM de figure 3.9(a). Cette représentation graphique permet de représenter les instructions D appartenant au prologue, dont les nœuds sont ordonnancés en dehors de l'espace cartésien des itérations. L'espace d'itérations reflète clairement que les dépendances de donnée $D \rightarrow A$ sont représentées par des arcs continus avec des extrémités doubles.

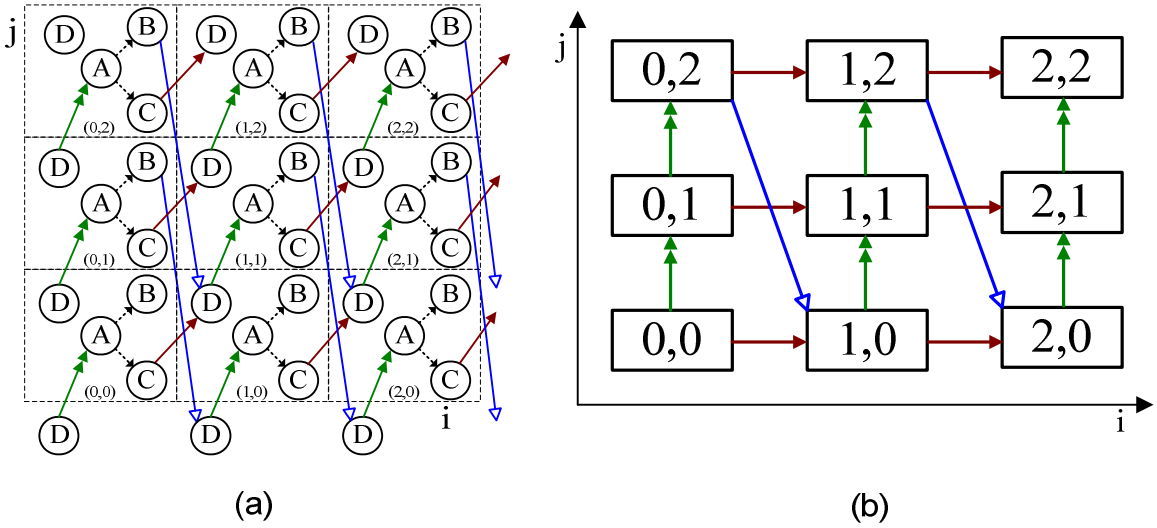


Figure 3.10 Filtre numérique d'ondelettes après retiming multidimensionnel : (a) espace d'itérations, (b) graphe de dépendances de cellules

L'indice des instructions au niveau de l'algorithme nous montre que toute instruction $D(i,j+1)$ appartenant à la boucle interne n'a aucune dépendance de données directe avec les nœuds $A(i,j)$, $B(i,j)$ et $C(i,j)$, appartenant à la même itération. Ce phénomène est visualisable explicitement à travers les arcs d'une même cellule de l'espace des itérations.

Cela permet une exécution parallèle du nœud D avec les autres nœuds. La figure 3.11 illustre l'ordonnancement statique des itérations après le retiming $r(D) = (0,1)$, où on choisit d'exécuter en parallèle les nœuds D et A. Cette transformation entraîne une diminution des tailles des chemins critiques dont les nouvelles structures sont respectivement $A \rightarrow B$ ou $A \rightarrow C$. De ce fait, elle entraîne une réduction similaire pour la valeur de la période de cycle. Prenons le même cas où $m = 10$ et $n = 10$ et $t(A) = t(B) = t(C) = t(D) = 1$, la période minimale de cycle est réduite de 3 à 2 unités de temps. Le temps d'exécution du filtre numérique d'ondelettes est réduit de 300 à 210 unités de temps ce qui reflète un gain de 30 % du temps d'exécution de l'application, par rapport à l'algorithme original, en dépit d'une augmentation de 50% de la taille du code.

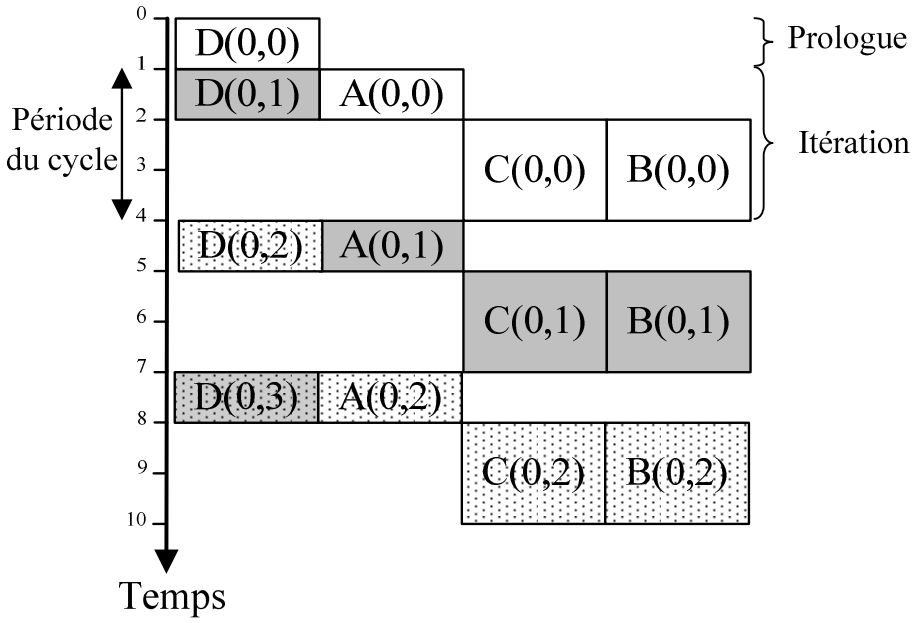


Figure 3.11 Ordonnancement statique du filtre numérique d'ondelettes après retiming multidimensionnel

3.4.1 Sélection de la fonction du retiming multidimensionnel

Une fonction de retiming multidimensionnel est dite légale si le GFDM est qualifié de réalisable après le retiming. Cette transformation doit éviter des valeurs de délais qui génèrent des conflits et doit assurer un ordre d'exécution cohérent du GFDM. La propriété 3.1 démontrée dans [1, 2] liste les conditions qu'un GFDM doit vérifier après l'application du retiming multidimensionnel légal. Les deux premières conditions de la propriété s'intéressent à la notion du déplacement des délais de part et d'autre du nœud décalé et de l'invariance des délais des cycles de données dans le GFDM. Les deux dernières conditions consistent à l'existence d'un ordre d'exécution pour l'application à partir du vecteur d'ordonnancement du graphe de dépendances des cellules.

Propriété 3.1. *[1] Soit $G = (V, E, d, t)$ un GFDM réalisable, r un retiming multidimensionnel, et s un vecteur d'ordonnancement du graphe $G_r = (V, E, d_r, t)$, donc :*

1) *Pour tout chemin $v_i \overset{p}{\Rightarrow} v_j$, on a $d_r(e) = d(e) + r(u) - r(v)$*
2) *Pour tout cycle $l \in G$, on a $d_r(l) = d(l)$*
3) *Pour tout arc $u \overset{e}{\rightarrow} v$, $d_r(e) \times s \geq 0$*
4) *Pas de cycle dans le GDC du MDFG G.*

Les techniques existantes adoptent une démarche formelle permettant de sélectionner une fonction de retiming multidimensionnel, en se basant sur la propriété 3.1. La sélection de la fonction doit se baser sur la valeur des délais initiaux des arcs du GFDM. Elle consiste à identifier le sous-espace $S+$ des vecteurs s vérifiant l'inéquation $d(e) \times s > 0$. Par la suite, elle procède à sélectionner un vecteur parmi ceux qui appartiennent à l'hyperplan. La fonction du retiming multidimensionnel r est donc le vecteur orthogonale à s. Cette démarche permet d'assurer une GFDM légal après l'application du retiming multidimensionnel.

L'approche du retiming multidimensionnel vise à la réduction des chemins critiques du GFDM. Dans ce contexte, les techniques existantes procèdent à appliquer le retiming sur les premiers nœuds des chemins critiques. Le travail décrit dans [2] propose une méthode pour prédire une fonction du retiming multidimensionnel pour les nœuds ayant des arcs entrants de délais non-nul, tel que décrit le dans théorème 3.1.

Théorème 3.1. *[1, 2] soit $G = (V, E, d, t)$ un GDFM réalisable, S+ un sous espace d'ordonnancement strictement positive de G, $u \in V$ un nœud dont tous les arcs entrants avec un délai non-nul. Un retiming multidimensionnel légal r du nœud u est n'importe quel vecteur orthogonal à s.*

Prenons l'exemple du GFDM dans la figure 3.1(b) à partir duquel on déduit l'exactitude des inéquations $(1,0) \times (1,1) \geq (0,0)$ et $(1,0) \times (1,-1) \geq (0,0)$. De ce fait, le vecteur $(1,0)$ est un vecteur d'ordonnancement légal, permettant un ordre d'exécution cohérent du GFDM. D'après le théorème 3.1, les fonctions du retiming multidimensionnel orthogonales à $(1,0)$ sont respectivement $(0,1)$ ou $(0,-1)$. D'où, l'application de la fonction $r(D) = (0,1)$ génère un GFDM légal tel que affiché dans la figure 3.9(a), dont toutes les conditions de la propriété 3.1 sont vérifiées.

3.4.2 Les techniques du retiming multidimensionnel

Les techniques de retiming multidimensionnel procèdent à appliquer les fonctions de retiming multidimensionnel jusqu'à atteindre un parallélisme total, dans le but d'exécuter l'application avec la valeur minimale de la période de cycle. Elles peuvent être étendues pour s'appliquer sur les boucles non-uniformes, contenant des instructions inter-itérations. Leur démarche procède à appliquer des fonctions de retiming multidimensionnel d'une façon répétitive, jusqu'à obtenir un GFDM sans aucune dépendance de données de délais nul. Les travaux [2,3] simulent l'exécution parallèle des nœuds à l'obtention d'une période de cycle égale à un, en supposant que les instructions ont la même valeur du temps d'exécution égale à une seule unité de temps. La théorie 3.2 présentée dans [2] liste les contraintes pour l'atteinte du parallélisme total pour les GFDMs.

Théorème 3.2. *[1, 2] Soit $G = (V, E, d, t)$ est un GFDM tel que $t(v_i) = 1$ pour tout $v \in V$. r est un retiming multidimensionnel légal de G tel que $C(G_r) = 1$ si et seulement si :*

1) *Le graphe de dépendances de cellules du nouveau GFDM $G_r = (V, E, d_r, t)$ ne contient aucun cycle.*

2) *$d_r(e) \neq (0, \dots, 0)$ pour tout $e \in E$.*

3) *Si le temps d'exécution d'un chemin p dans G_r est plus grand que 1, donc $d_r(p) \neq (0, \dots, 0)$.*

Les démarches des techniques existantes procèdent à choisir une fonction du retiming multidimensionnel telle que indiquée dans le théorème 3.2 : elles supposent que les temps d'exécution des nœuds sont égaux à une période de cycle. De ce fait, l'atteinte de la période d'horloge minimale $C(G_r) = 1$ nécessite l'application des fonctions de retiming jusqu'à atteindre un GFDM sans aucune dépendance de données de délai nul.

3.4.2.1 Retiming multidimensionnel progressif

Cette technique vise à atteindre une exécution totalement parallèle des instructions du corps de la boucle interne [2]. Elle procède à sélectionner et appliquer itérativement le retiming multidimensionnel telle que décrit dans les théorèmes 3.1 et 3.2. Elle débute par identifier les premières instructions exécutées dans le corps de la boucle pour décaler leurs exécutions. Dans cet objectif, elle identifie tous les nœuds X du GFDM ayant des arcs entrant de délais non-nuls. Puis, elle sélectionne un vecteur d'ordonnancement s_1 tel que $|S_x| + |S_y|$ est minimale et en déduire une fonction de retiming multidimensionnel r_1 pour l'appliquer aux nœuds X [2, 3]. Cette transformation réduit les tailles des chemins critiques en retirant les nœuds X. Cette modification entraîne que les nœuds Y successeurs X ont des arcs entrants de délais non-nuls et des arcs sortants de délais nuls. De ce fait, ils représentent les nœuds à exécuter en premier dans le corps de la boucle. D'où, la technique progressive procède à sélectionner un deuxième vecteur d'ordonnancement s_2 et en déduire une fonction de retiming multidimensionnel r_2 pour l'appliquer aux nœuds Y. La technique progressive applique itérativement ces étapes, en balayant progressivement le GFDM dans l'ordre des dépendances des données jusqu'à atteindre un parallélisme maximal. Nous illustrons dans les graphes de la figure 3.12 le GFDM du filtre numérique d'ondelettes généré par la technique du retiming multidimensionnel progressive. La première étape consiste à appliquer la fonction $r = (0,1)$ sur le nœud D. Le parallélisme total est atteint en appliquant la fonction $r = (1, -3)$ sur le nœud A.

En fait, les nouvelles valeurs des délais impliquent l'augmentation des vecteurs d'ordonnancement. Pour le cas du filtre numérique d'ondelettes, le vecteur d'ordonnancement initial est égal à $(1,0)$ tandis que le deuxième est égal à $(3,1)$. En se basant sur le théorème 3.1, plus l'ordre d'application une fonction de retiming multidimensionnel $r = (d.x, d.y)$ est plus grand, plus les valeurs des indices "$d.x$" et "$d.y$" sont importants.

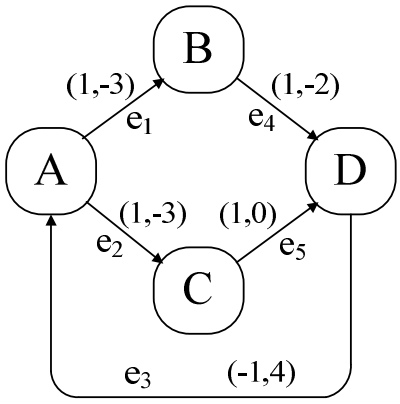


Figure 3.12 GFDM totalement parallèle du filtre numérique d'ondelette par la technique du retiming multidimensionnel progressif

3.4.2.2 Retiming multidimensionnel enchaîné

Cette technique génère un GFDM totalement parallèle en utilisant une seule fonction de retiming multidimensionnel [2, 3]. Pour deux nœuds successifs u et v tel que $u \to v$, une

fonction de retiming multidimensionnel est utilisée comme suit : décaler le nœud u par $(r \times (k+1))$ et v par $(r \times k)$ tel que $k > 0$. Dans ce contexte, la technique enchaînée débute par sélectionner une fonction de retiming optimale multidimensionnel r. Ensuite, elle procède à balayer le GFDM dans l'objectif de déterminer les facteurs de multiplication k de chaque nœud. Le retiming est par la suite appliqué à tous les nœuds du GFDM ayant des arcs sortants de délais nuls. Pour le cas du filtre numérique d'ondelettes, la technique applique la fonction $r = (0,2)$ sur le nœud D puis la fonction $r = (0,1)$ sur le nœud A, dont le GFDM final est illustré dans la figure 3.13.

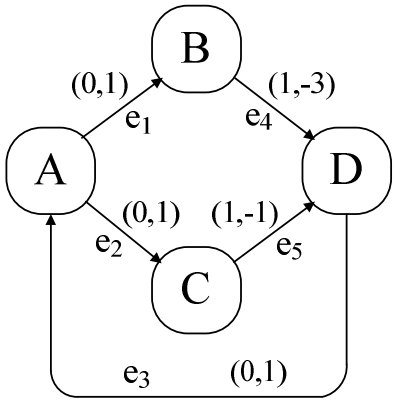


Figure 3.13 GFDM du filtre numérique d'ondelettes généré par la technique du retiming multidimensionnel enchaîné

La technique de retiming multidimensionnel enchaîné génère des implémentations avec des temps d'exécution et des tailles de codes inférieurs à ceux générés par la technique progressive. Cette amélioration est due à l'utilisation itérative d'une même fonction optimale de retiming multidimensionnel.

3.4.2.3 La technique SPINE

Cette technique vise à atteindre un parallélisme total tout en réduisant la taille du code du prologue et de l'épilogue. Dans cette objectif, elle procède à regrouper tous les délais non-nuls dans l'objectif de les disperser par la suite sur tous les arcs du GFDM. le théorème 3.3 présenté dans [4] décrit les conditions nécessaires qu'un GFDM doit vérifier pour atteindre une période de cycle minimale, dans le cas où $t(v) = 1$. Ce théorème est spécifique pour le cas d'un GFDM avec un vecteur d'ordonnancement $s = (1,0)$ et $r = (0,1)$.

Théorème 3.3. *[4] Soit un GFDM $G = (V, E, d, t)$, $r = (0,1)$ est une fonction de retiming multidimensionnel et l un cycle dans G :*

- *La période de cycle minimale $C_{min}(l) = 1$ est atteinte si $\sum d(e) = (i,j)$ avec $i > 0$*
- *si $\sum d(e) = (0,k)$ avec $K > 0$, alors la période minimale de cycle $c(l) = arrondie\ (t(l)/k)$ peut être atteinte, avec $t(l) = \sum t(v)$*

L'idée de base de cette technique consiste à transformer les GFDMs pour qu'ils vérifient la première condition. Par la suite, elle procède à déplacer les délais $(0,k)$, pour les regrouper dans un arc de délais (i,j) tel que $i > 0$. De ce fait, les arcs du GFDM seront étiquetés par des délais nuls ou de type (i,j). Cette structure du GFDM permet d'atteindre la période minimale du cycle $C(G) = 1$. Dans le cas contraire, le théorème exige que l'indice j du délais (i,j) (correspondant à la boucle interne) doit être supérieur au nombre des nœuds du chemin critique pour atteindre un parallélisme total. Si aucun des vecteurs ne peut être appliqué, elle

sélectionne un vecteur d'ordonnancement s en déployant la même démarche de la technique de retiming multidimensionnel enchaînée.

La répartition du délai total se base sur le principe du retiming des graphes synchrones [24]. Elle convertit le GFDM en un graphe de données unidimensionnel. Ensuite, elle transforme les délais existants en entiers. Ces nouvelles valeurs d'étiquetage permettent d'appliquer la démarche du retiming [24]. Si le graphe est susceptible d'être ordonnancé avec la période de cycle minimale, la technique procède à décaler l'exécution des noeuds et génère le graphe correspondant. Par la suite, elle transforme les délais du graphe, de la structure entier à la structure multidimensionnel. Dans le cas contraire, la période de cycle minimale ne pourra pas jamais être atteinte.

3.4.3 Contraintes du retiming multidimensionnel

Les études effectuées dans [26] ont dégagé les limites des techniques du retiming multidimensionnel pour atteindre un parallélisme total. L'atteinte de ce parallélisme nécessite le respect d'un ensemble de contraintes concernant la relation entre les nombres d'itérations des boucles et les valeurs de retiming à appliquer. Prenons le cas du filtre numérique de l'algorithme de la figure 2.6, dont le chemin critique est composé de 3 nœuds. Si le nombre d'itérations de la boucle externe est $i = 0$ et le nombre d'itérations de la boucle interne est $j < 3$, alors il n'existe aucun vecteur du retiming multidimensionnel $(0, p)$ permettant d'atteindre un parallélisme total pour l'exécution des nœuds de ce graphe.

Pour cela, ces travaux ont définis la notion du vecteur du nombres d'itérations $I = \{i_0, i_1, \dots, i_k\}$, avec k est la dimension du GFDM, et dont les valeurs sont comprises entre $L = \{l_0, l_1, \dots, l_k\}$ et $U = \{u_0, u_1, \dots, u_k\}$ tel que $l_j \leq i_j \leq u_j$. La contrainte Sc du GFDM est définie dans l'équation (3.4).

$$Sc = [(u_0 - l_0 + 1), (u_1 - l_1 + 1), \dots, (u_k - l_k + 1)] \quad (3.4)$$

En se basant sur la démarche du retiming multidimensionnel, le parallélisme maximal est atteint si la fonction de retiming $r(u) = (r_0, r_1, \dots, r_k)$ appliquée à n'importe quel nœud u, satisfait la condition de l'inéquation (3.5).

$$r_j < s_j \; tel \; que \; 0 \leq j \leq k \quad (3.5)$$

3.5 Application conjointe des niveaux du parallélisme

Chaque niveau de granularité explore des critères spécifiques du GFDM pour le choix du parallélisme. En se basant sur l'étude effectuée, aucun niveau de parallélisme n'est qualifiée d'optimale par rapport l'autre. De plus, le parallélisme au niveau des itérations des GFDMs préserve la structure des instructions du corps de la boucle, ce qui préserve les potentialités du parallélisme au niveau des instructions, et vice versa. De ce fait, les deux niveaux de parallélisme peuvent être appliqués conjointement dans l'objectif de générer des implémentations avec des contraintes plus optimales.

Dans de contexte, nous distinguons un seul travail dont la démarche applique conjointement la technique de retiming multidimensionnel SPINE et le retiming itérationnel, dans l'objectif d'optimiser la période d'itération tout en utilisant une taille de code minimale [33]. Le choix du/des parallélisme(s) est basé sur le respect de la contrainte de la période d'itération. La démarche vérifie la possibilité d'appliquer le parallélisme au niveau des instructions en testant les deux vecteurs d'ordonnancement $s = (1,0)$ et $s = (0,1)$. Si la

contrainte n'est pas encore respectée, la démarche fait recours au retiming itérationnel pour le regroupement des itérations en partitions. Le choix du nombre des itérations regroupées est effectué en fonction de la contrainte de période d'itérations à atteindre.

3.6 Synthèse des techniques de parallélisme des GFDMs

L'étude explicite effectuée sur les techniques d'optimisation des GFDMs nous a permis de dégager l'ensemble des constatations suivantes :

- *Principe du parallélisme :* les techniques d'optimisation des GFDMs visent à atteindre un parallélisme maximal, que ce soit au niveau des instructions qu'au niveau des itérations. Ces techniques se basent sur la notion d'égalité des temps d'exécution des traitements à paralléliser. Cette notion est triviale pour le cas des itérations, due à l'aspect itératif du nid de boucles. Cependant, pour le cas du parallélisme au niveau des instructions, les techniques prétendent que tous les instructions requièrent le même temps d'exécution.

- *Contrainte temporelle :* les techniques d'optimisation des GFDMs visent à atteindre un parallélisme maximal, que ce soit au niveau des instructions qu'au niveau des itérations. L'objectif principal est de réduire la période d'itération du nid de boucles. Cet objectif est similaire à ordonnancer le GFDM avec la période de cycle minimale. Cependant, toutes les techniques du parallélisme engendrent l'ajout de blocs de code en amont et en aval des structures des boucles dues aux dépendances des données inter-itérations. En se basant sur les paramètres du parallélisme, ces blocs de codes nécessitent un temps d'exécution considérable par rapport au temps global du nid de boucles. De ce fait, atteindre la période d'itération minimale ne correspond pas à l'atteinte du temps d'exécution minimal. A notre connaissance, il n'existe aucune technique de parallélisme des GFDMs visant à respecter une contrainte de temps d'exécution.

- *Contrainte de ressources matérielles :* le parallélisme au niveau des itérations entraîne l'augmentation de la taille du code dans le corps de la boucle. Cet augmentation de la taille est similaire à l'augmentation des unités de traitements pour l'exécution parallèle. Pour le cas du parallélisme au niveau des instructions, les blocs de code du prologue et d'épilogue sont proportionnel à la taille de la mémoire cache utilisée ainsi qu'à la fréquence d'accès [13]. Par conséquent, la taille du code correspondante au GFDM est proportionnelle aux ressources matérielles de l'implémentation.

- *Utilisation conjointe des niveaux de parallélisme :* malgré la multitude des techniques de parallélisme du GFDM, nous distinguons un seul travail assurant l'utilisation conjointe des deux niveaux du parallélisme. De plus, ce travail a été proposé pour l'optimisation des mêmes contraintes ciblées par les techniques du parallélisme appliquées séparément qui est la période d'itération et la taille du code.

3.7 Conclusion

Nous avons présenté dans ce chapitre une étude explicite du formalisme des GFDMs. Cette étude a été suivie par la présentation des techniques d'optimisation des GFDMs qui nous a permis d'établir une synthèse des techniques existantes du parallélisme.

En premier lieu, les techniques d'optimisation au niveau des instructions visent à ordonnancer le nid de boucles avec la période de cycle minimale dans le but atteindre le

parallélisme total. Cet objectif implique l'insertion du code de prologue et d'épilogue à travers les structures des boucles. Par conséquent, les techniques existantes génèrent des implémentations avec des temps d'exécution non-optimaux et des tailles importantes des codes. D'où, les techniques de retiming multidimensionnel existantes ne sont pas adéquates pour l'optimisation des implémentations temps réel embarquées. Pour cela, nous proposons dans le chapitre 4 une nouvelle stratégie de parallélisme au niveau des instructions permettant de réduire le temps d'exécution et la taille du code lors de l'ordonnancement du nid de boucles avec une période de cycle minimale.

En deuxième lieu, nous avons constaté que toutes les techniques d'optimisation visent à optimiser la période d'itération du nid de boucles. En se basant sur les tailles des prologues et des épilogues, l'optimisation de cette contrainte n'est pas similaire à l'optimisation du temps d'exécution de l'implémentation. De plus, nous avons constaté qu'aucune technique ne propose une démarche de parallélisme pour le respect d'une contrainte de temps d'exécution qui s'avère nécessaire pour le cas des implémentations des systèmes temps réel. Dans ce contexte, nous nous intéressons dans le chapitre 5 à la mise en pratique les techniques de parallélisme des GFDMs pour le respect des contraintes de temps d'exécution.

CHAPITRE 4 : PROPOSITION D'UNE NOUVELLE TECHNIQUE « RETIMING MULTIDIMENSIONNEL DÉCALÉ »

4.1 Introduction

Nous avons détaillé dans le chapitre précédent les principes du retiming multidimensionnel ainsi que ses techniques existantes visant à atteindre un parallélisme total. Nous avons déduit que le parallélisme maximal engendre des tailles de code importantes et des temps d'exécution non optimaux des nids de boucles. L'évolution de ces paramètres nous amène à poser la question suivante : est-il nécessaire d'atteindre un parallélisme total pour ordonnancer une application multidimensionnel avec la période de cycle minimale ?

Les théories du retiming multidimensionnel présentées dans le chapitre précédent supposent que toutes opérations nécessitent le même temps d'exécution qui est égal à une seule unité de temps. Dans ce cas, l'atteinte de la période de cycle minimale correspond à l'ordonnancement avec un parallélisme total. Cependant, les techniques duparallélisme au niveau des instructions décrites dans [8, 24, 104] explorent la notion des temps d'exécution élémentaires des opérations : le choix des traitements à paralléliser est basé sur leurs temps d'exécution.

Dans ce chapitre, nous proposons une nouvelle approche de retiming multidimensionnel permettant d'explorer les temps d'exécution élémentaires des opérations lors du choix du parallélisme dans les boucles imbriquées. Cette technique permet le décalage des chemins de données tout en assurant leurs exécutions dans la limite de la période de cycle minimale. Nous démontrons l'ensemble des théorèmes, permettant d'extraire les chemins à décaler, puis de sélectionner les fonctions légales du retiming multidimensionnel. Ce chapitre est clôturé par une étude expérimentale permettant s'évaluer les apports de notre technique en matière de temps d'exécution et de taille du code, par rapport aux techniques existantes.

4.2 Limites des techniques du retiming multidimensionnel existantes

Les techniques de retiming multidimensionnel consistent à transformer les structures des GFDMs dans le but de minimiser la période du cycle. Cette transformation entraîne une modification de l'algorithme correspondant à l'application multidimensionnel. Donc, elle affecte la taille du code et le temps d'exécution de l'implémentation finale. Nous décrivons dans le premier paragraphe l'ordre d'évolution de ces caractéristiques en fonction du retiming multidimensionnel. Puis, nous dégageons les paramètres influant sur la modification de ces grandeurs.

4.2.1 Evolution de la taille du code et du temps d'exécution

La démarche du retiming multidimensionnel consiste à décaler l'ordre d'exécution des instructions. Elle procède à distribuer les nœuds appartenant à un même chemin de délai nul dans différentes itérations. Cette modification entraîne l'exécution d'un ensemble d'occurrences des nœuds en dehors des boucles affectés par le retiming. Une fonction de retiming multidimensionnel $r = (d_1, \dots, d_n)$ consiste à décaler les nœuds appartenant à la boucle j tel que $d_j \neq 0$. Si r est appliqué sur un ensemble de nœuds $X \in V$, elle implique l'exécution d'un ensemble d'occurrences des nœuds X en amont de la boucle j, intitulé prologue. De même, un ensemble d'occurrences des nœuds Y tel que $Y = V - X$ sont exécuté en aval de la boucle j, intitulé épilogue. Ces nœuds correspondent à l'ajout de blocs de codes en dehors des structures itératives. De ce fait, ils requièrent un ensemble de périodes de cycles supplémentaires pour leurs exécutions. En outre, la démarche de retiming multidimensionnel vise à augmenter le niveau de parallélisme des traitements itératifs, et non pas au niveau des traitements en amont et en aval des boucles. Par conséquent, le nombre des cycles nécessaires pour l'exécution du prologue et d'épilogue est souvent plus important que le nombre des cycles nécessaires pour l'exécution d'une seule itération. Cette augmentation est

proportionnelle à celle du temps d'exécution total. On déduit que le retiming multidimensionnel implique une augmentation considérable du nombre de périodes de cycles nécessaires pour l'exécution de l'application multidimensionnel. De plus, la transformation due au retiming multidimensionnel affecte l'intervalle des itérateurs des structures itératives. Les valeurs minimales et maximales de ces intervalles dépendent du vecteur d'ordonnancement utilisé lors de l'exécution de l'application. Il s'avère donc indispensable d'ajouter des instructions pour le re-calcul des nouvelles valeurs des itérateurs. Ces instructions correspondent à un temps d'exécution et une taille du code supplémentaires de l'implémentation finale.

En fait, l'évolution des caractéristiques temporelles et du code varient en fonction de deux critères qui sont : la valeur de la fonction du retiming multidimensionnel, et le nombre des fonctions appliquées.

4.2.2 Impact de la fonction du retiming multidimensionnel

Le nombre des instructions décalées en prologue et épilogue dépend des valeurs des indices de la fonction du retiming multidimensionnel $r = (c_1, c_2, \dots, c_n)$. Chaque indice renseigne sur la taille de l'ensemble des instructions à décaler en dehors de la structure itérative ; i.e., Si r est appliqué sur un ensemble de nœuds $X \in V$ et $c_i \neq 0$, alors $|c_i|$ occurrences des nœuds X sont décalées en amont de la boucle i (prologue). De même, un ensemble de $|c_i|$ occurrences des nœuds Y, tel que $Y = V - X$, sont exécuté en aval de la même boucle (épilogue). Nous constatons que plus l'indice $|c_i|$ est petit, plus la taille du code décalé est moins importante. Un indice $|c_i| = 0$ permet de maintenir la structure de la boucle i et ainsi une taille minimale du code correspondant à cette boucle. Pour le cas du GFDM du filtre numérique d'ondelette, un décalage des instructions dans l'ordre des lignes avec la fonction $r = (0,1)$ ou dans l'ordre des colonnes avec la fonction $r = (1,0)$ génère une taille d'algorithme inférieure à celle de l'algorithme généré après le décalage des mêmes nœuds en utilisant l'ordre des deux dimensions avec la fonction $r = (1,1)$ [4].

Pour remédier à cet inconvénient, les techniques du retiming multidimensionnel visent à choisir des fonctions permettant de générer le minimum de décalage d'instructions. Les techniques incrémentale et enchaînée procèdent à choisir des vecteurs d'ordonnancement $s = (s.x, s.y)$ tel que la somme $|s.x| + |s.y|$ est minimale, puis à sélectionner une fonction de retiming sous la forme de $r = (s.y, -s.x)$. Le formalisme des GFDMs implique que les vecteurs d'ordonnancement augmentent de valeurs en fonction des retimings multidimensionnels appliqués. Dans ce contexte, la technique enchaînée procède à utiliser itérativement la premier fonction déduite du premier vecteur, afin de réduire les effets du prologue et d'épilogue. Dans ce même objectif, la technique SPINE teste successivement les vecteurs d'ordonnancement *(0,1), (1,0)* et *(1,1)* pour en déduire un retiming légal. Dans le cas contraire, elle sélectionne une fonction de retiming multidimensionnel tel que décrit dans les techniques progressive et enchaînée. Nous tenons à signaler qu'il n'existe aucun travail de recherche décrivant une démarche de sélection de vecteur d'ordonnancement et de fonction de retiming multidimensionnel aboutissant à une implémentation optimale.

4.2.3 Impact du nombre des fonctions du retiming multidimensionnel

La taille du code et le nombre des périodes de cycles augmentent simultanément avec le nombre des fonctions du retiming multidimensionnel. En fait, toutes les techniques procèdent à appliquer les transformations à base du retiming multidimensionnel tant qu'il existe encore des dépendances de données de délais nuls : si le chemin de données critique est composé de X nœuds, alors toute technique de retiming multidimensionnel utilise $(X - 1)$ fonctions. Elles sont appliquées d'une façon incrémentale à partir du premier nœud ayant un arc entrant de

délais nul, jusqu'au dernier nœud du chemin. Les blocs de codes correspondant au prologue, épilogue et instructions de re-calcul sont ajoutés avec chaque fonction de retiming. De plus, la technique de retiming multidimensionnel n'assure pas le décalage des structures itératives uniquement, mais aussi des blocs de codes déjà existant en amont et en aval de ces structures. Donc, la taille du code et le nombre des cycles augmentent d'une façon exponentielle en fonction du nombre des fonctions de retiming appliquées. Prenons l'exemple du filtre à réponse impulsionnelle infinie [2] : Chaque technique existante applique quatre fonctions de retiming pour obtenir un GFDM totalement parallèle. Nous choisissons d'appliquer la technique enchaînée avec une fonction de retiming optimale $r = (1, -1)$. Ensuite, nous générons le GFDM après chaque fonction de retiming dans le but de déterminer sa taille de code et son nombre de périodes de cycle, dont les valeurs sont affichées dans les courbes de la figure 4.1.

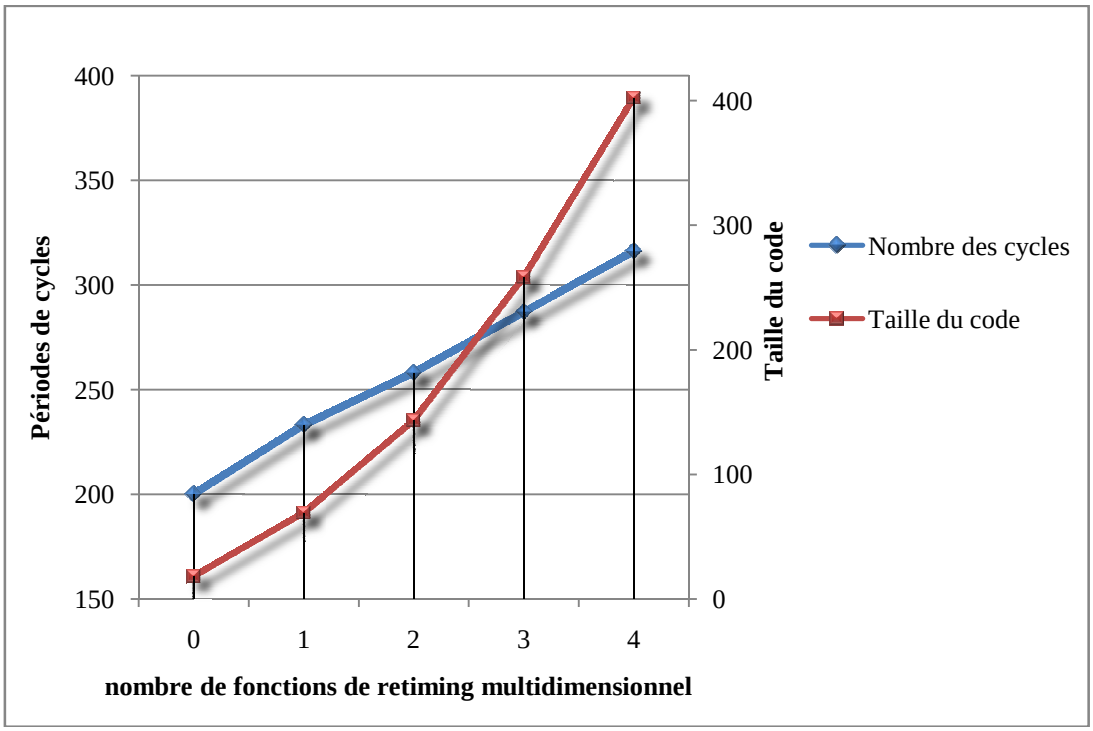


Figure 4.1 Evolution du nombre des cycles et de la taille des codes en fonction du retiming multidimensionnel

En se basant sur ces valeurs expérimentales, nous constatons que plus on applique une fonction de retiming, plus le nombre des périodes de cycle augmente et plus la taille du code ajouté est importante. Le code de l'algorithme totalement parallélisé est 22 fois plus grand que celui de l'algorithme initial. D'autre part, le nombre des cycles nécessaires pour l'exécution du GFDM totalement parallélisé a augmenté de 58% par rapport à l'ordonnancement du graphe initial. On peut conclure donc que les solutions générées par les techniques existantes ne permettent pas de respecter des contraintes strictes du temps d'exécution. De plus, les tailles des codes importantes engendrent une augmentation des ressources matérielles des implémentations. Par conséquent, le parallélisme offert n'est pas adéquat pour l'implémentation dans des systèmes temps réel embarqués.

4.3 Motivation et principes

Nous présentons dans cette section l'idée de base de notre contribution. Nous expliquons dans le premier paragraphe l'apport de notre idée par l'intermédiaire d'une application

multidimensionnel. Par la suite, nous décrivons dans le deuxième paragraphe les principes et la démarche de notre contribution.

4.3.1 Exemple de motivation

La période de cycle minimale d'un graphe flot de données est définie comme étant le temps d'exécution maximal parmi les chemins de délai nul $max\{t(p), d(p) = 0\}$. Pour un chemin de données $p: u_1 \rightarrow \cdots \rightarrow u_n$, les techniques du retiming multidimensionnel impliquent l'exécution de chaque nœud u_k dans une période de cycle séparément, tel que $1 \leq k \leq n$. De ce fait, la période de cycle minimale d'un graphe totalement parallélisé est égale à la valeur maximale des temps d'exécution des nœuds appartenant au GFDM ($max\{t(v), v\epsilon V\}$). Prenons le cas du filtre numérique d'ondelettes, composé de deux types de nœuds (addition et multiplication). Assumant que le premier type nécessite deux unités de temps et le deuxième nécessite une seule, le GFDM totalement parallélisé est ordonnancé avec la période de cycle minimale $c_{min} = 2$. L'ordonnancement statique de GFDM totalement parallélisé est affiché dans la figure 4.2, dont les nœuds appartenant à la même itération dans le MDFG initial sont illustrés avec le même motif.

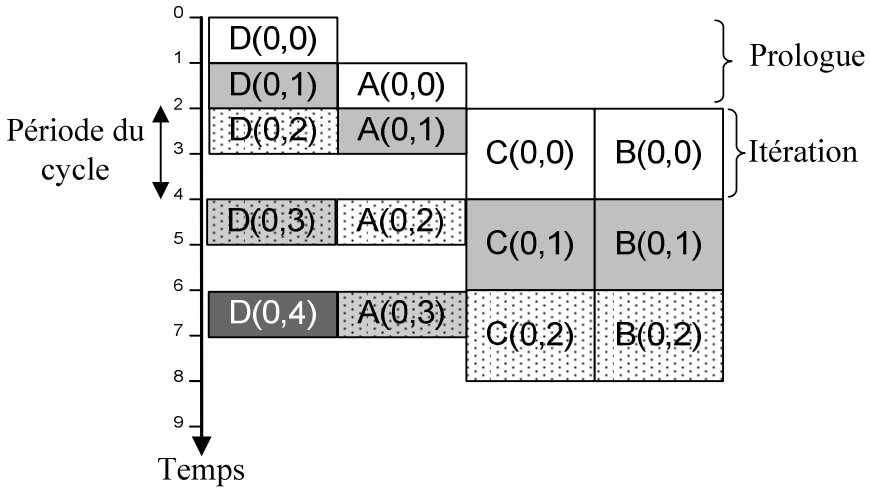


Figure 4.2 Ordonnancement statique du filtre numérique d'ondelettes totalement parallélisé

Ce choix du parallélisme se base sur la notion d'égalité des temps d'exécution des nœuds de calcul. Cependant, d'une façon général, les instructions de calcul des graphes de flots de données n'ont pas le même temps d'exécution ; Ils dépendent des natures des tâches à effectuer (addition, multiplication, division, …).

En fait, le GFDM totalement parallélisé peut être exécuté suivant le vecteur d'ordonnancement $s = (1,0)$. Vue que ce dernier a deux vecteurs orthogonaux qui sont $(0,1)$ et $(0,-1)$, et que les délais des arcs e_4 et e_5 sont non nuls, nous procédons à appliquer le fonction de retiming $r(D) = (0,-1)$, dont le GFDM résultat est affiché dans la figure 4.3(b). Essayons de focaliser sur la structure du GFDM de la figure 4.3(a). L'arc e_3 est de délai nul, ce qui implique que les nœuds D et A sont exécutés dans la même période de cycle et ainsi dans la même itération, que le GFDM initial. Les délais des arcs e_1 et e_2 sont égaux à $(0,1)$ ce qui signifie que le chemin $p: D \rightarrow A$ est exécuté une période de cycle avant celle des nœuds B et C. Par conséquent, le chemin $p: D \rightarrow A$ appartenant à la première itération du code initial est exécuté dans le prologue de la boucle interne, et chaque chemin $p: D \rightarrow A$ appartenant à l'itération i, tel que $i > 1$, est exécuté dans l'itération $(i-1)$. D'une façon similaire, les instructions B et C appartenant à la dernière itération de la boucle interne dans le

code initial sont exécutées en aval de la boucle. De ce fait, l'algorithme correspondant au GFDM de la figure 4.3(b) est décrit dans la figure 4.3(a).

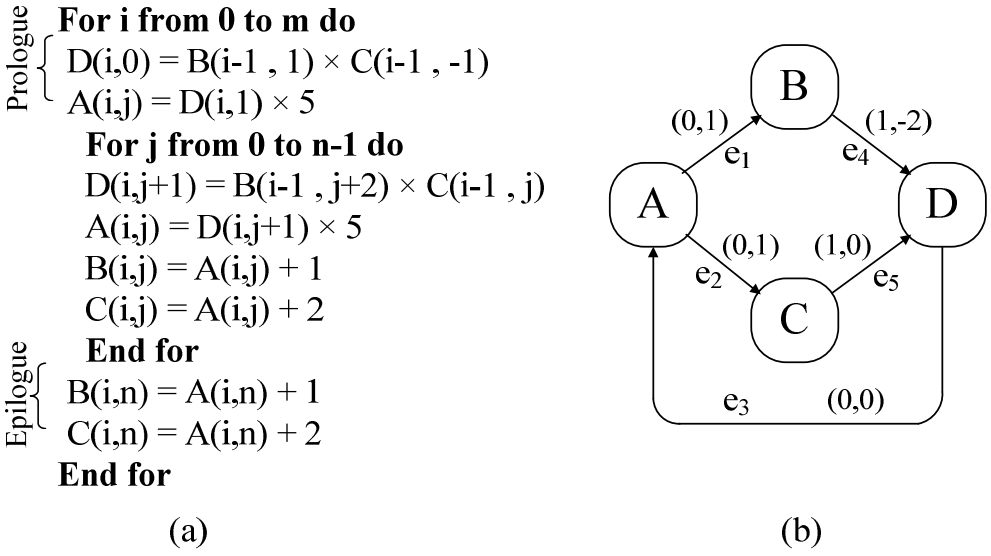


Figure 4.3 Filtre numérique d'ondelettes généré par la technique de retiming multidimensionnel décalé : (a) l'algorithme, (b) le MDFG

Le GFDM de la figure 4.3(b) contient 3 chemins de délais nuls, qui sont B, C et $p : D \rightarrow A$. Admettant que leurs temps d'exécution est égal à 2 unités de temps, l'algorithme est ordonnancé avec la période de cycle $C(G) = 2$. Nous déduisons que même si le graphe n'est plus totalement parallélisé, il est toujours exécuté avec la période minimale de cycle $c_{min} = 2$. A partir de la figure 4.4, nous déduisons que les périodes de cycles sont totalement exploitées, et que les données générées sont directement consommées dans la période de cycle suivante. D'autre part, le code correspondant ne contient que deux instructions en amont et en aval de la boucle interne, telles que affichées dans la figure 4.3(a). La taille du code est réduite de 25% par rapport au code correspondant au graphe totalement parallélisé. De même, le nombre de périodes de cycle est réduit de 7.69% et ainsi une même amélioration en temps d'exécution. Par conséquent, même si la solution générée n'est pas totalement parallèle, elle nécessite un temps d'exécution et une taille de code inférieurs à ceux du parallélisme total.

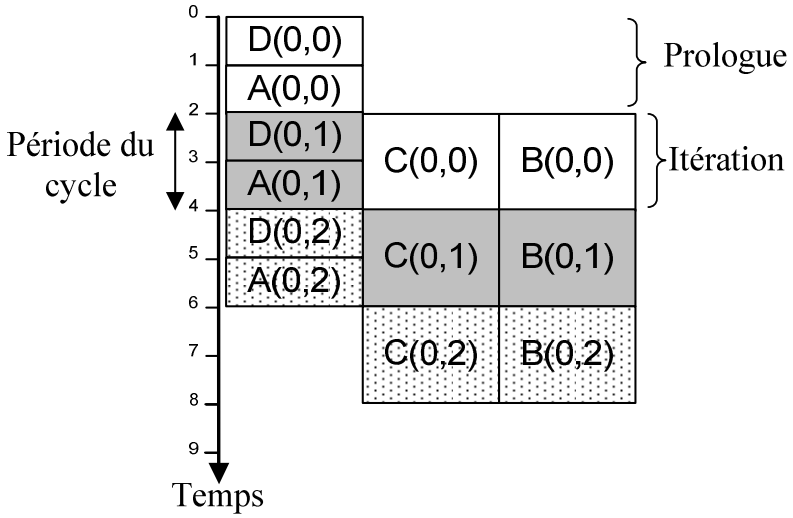


Figure 4.4 Ordonnancement statique du filtre numérique d'ondelettes généré par la technique du retiming multidimensionnel décalé

4.3.2 Principes du retiming multidimensionnel décalé

En se basant sur les valeurs des délais du GFDM de la figure 4.3(b), les cycles des nœuds conservent les valeurs des délais du graphe initial : l'équation $d_r(p) = d(p) + r(v) - r(u)$ est vérifiée pour tous cycles de nœuds appartenant au GFDM. De plus, l'espace d'itérations affiché dans la figure 4.5(a) ne contient aucun cycle entre les occurrences des itérations. On peut conclure qu'à partir du graphe de dépendances des cellules qu'il existe une infinité de vecteurs d'ordonnancement qui peuvent assurés un ordre d'exécution légal du filtre. On cite à titre d'exemple le vecteur d'ordonnancement $s = (3,1)$. Par conséquent, cette modification génère une GFDM réalisable, et préserve les fonctionnalités initiales du filtre.

Nous focalisons maintenant sur les détails de la transformation appliquée pour générer le GFDM de la figure 4.3(b). Nous remarquons à partir de l'ordonnancement statique de la figure 4.4 que les nœuds appartenant initialement à la même itération dans le graphe initial, sont partagés en deux périodes de cycle. En fait, le délai de l'arc e_3 est nul, ce qui signifie que tout chemin $D \rightarrow A$ est exécuté dans une seule période. L'espace d'itérations dans la figure 4.5(a) montre que chaque chemin $D \rightarrow A$ initialement exécuté dans l'itération interne (i), est exécuté dans l'itération $(i-1)$, tel que $i > 1$. De plus, chaque occurrence de ce chemin appartenant à la première itération de la boucle interne est exécutée en amont de cette boucle, et ainsi en dehors de l'espace d'itérations, telle que affichée dans la figure 4.5(a).

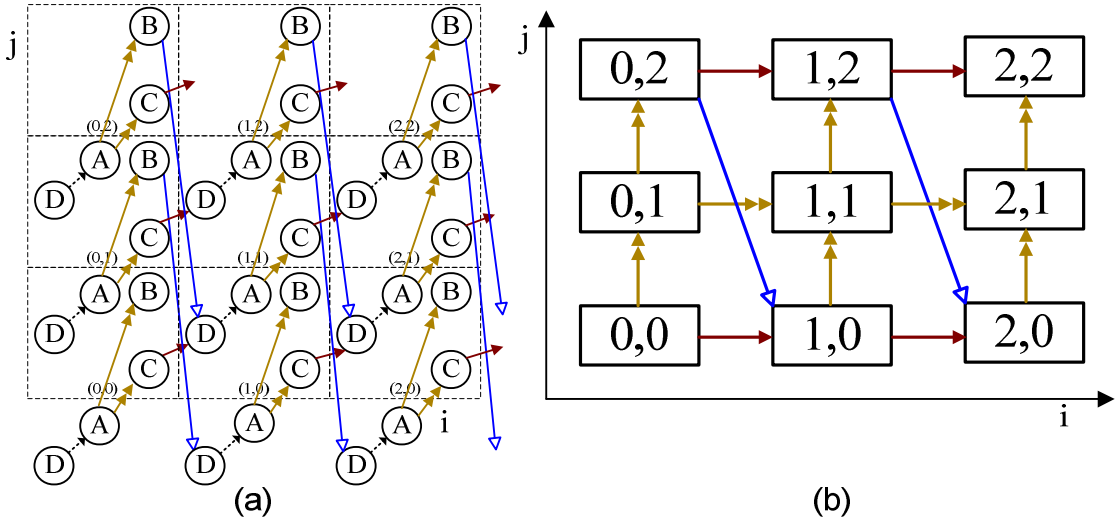


Figure 4.5 (a) Espace d'itérations du GFDM généré par le retiming multidimensionnel décalé ; (b) Graphe de dépendances de cellules

En se basant sur ces caractéristiques, la transformation peut être considérée comme étant l'application de la fonction de retiming multidimensionnel $r = (0,1)$ au chemin $p: D \rightarrow A$ du GFDM initial du filtre. Cette transformation consiste à soustraire un délai de $(0,1)$ des arcs e_4 et e_5 entrants à D et l'ajouter aux arcs e_1 et e_2 sortants de A. En fait, le chemin est exécuté en deux unités de temps. Malgré que le délai du chemin est nul et ainsi exécuté dans la même période de cycle, le GFDM peut être toujours ordonnancé avec la valeur minimale de période de cycle $c_{min} = 2$. En outre, ce résultat est atteint après l'application d'une seule fonction de retiming multidimensionnel. Le décalage des instructions est appliqué une seule fois, ce qui explique la réduction de la taille du prologue et d'épilogue par rapport à la solution totalement parallèle. Elle engendre une diminution de la taille du code et une diminution similaire dans le nombre des périodes de cycles nécessaires pour l'exécution de toute l'application. Nous constatons que cette transformation génère une solution plus optimale en matière de temps

d'exécution et de taille de code par rapport aux solutions générées par les techniques du retiming multidimensionnel existantes.

Nous proposons ainsi une nouvelle technique d'optimisation basée sur la démarche de retiming multidimensionnel. Cette technique ne procède pas à décaler tout les nœuds du GFDM, mais à décaler les chemins de données dont l'exécution est effectué dans la même itération. L'objectif de notre technique consiste à l'atteinte de la période de cycle minimale, en employant le moins de fonctions de retiming multidimensionnel possibles.

Cette technique nécessite l'exploration des temps d'exécution élémentaires des opérations, afin de sélectionner les chemins à exécuter en parallèle. De plus, la théorie du parallélisme démontrée dans les travaux précédents [2, 3] se limite à sélectionner une fonction de retiming pour le décalage d'un seul nœud. Cette théorie doit être étendue pour permettre le décalage de tout un chemin de données. Ces deux tâches sont décrites respectivement dans les sections 4.4 et 4.5.

4.4 Sélection des chemins de données

La sélection des chemins à paralléliser doit se baser sur leurs temps d'exécution, dans l'objectif de les ordonnancer dans la période de cycle minimale. De plus, ce choix doit prendre en considération le sens des dépendances de données des nœuds. Dans ce contexte, nous décrivons dans le premier paragraphe notre méthode pour extraire les temps d'exécution et les délais de tous les chemins du GFDM. Dans le deuxième paragraphe, nous détaillons notre démarche d'exploration de ces valeurs pour la sélection des chemins à exécuter en parallèle.

4.4.1 Propriétés du temps d'exécution et des dépendances de données du GFDM

Cette étape consiste à balayer le GFDM pour identifier tous les chemins de données dont les délais sont nuls, ainsi que leurs temps d'exécution. En effet, deux nœuds u et v peuvent être liés par plusieurs chemins de données, dont les délais sont généralement différents. Vue que notre démarche s'intéresse principalement aux chemins de délais nuls, un chemin $p : u \rightarrow v$ n'est décalé que si au moins un parmi les chemins liant u et v a un délai nul. Dans ce contexte, nous définissons une nouvelle grandeur $D(u, v)$ représentant le délai minimal des chemins liant u et v, telle que décrite dans l'équation (4.1). Si un seul chemin, parmi tous ceux liant u et v, est de délai nul alors $D(u, v) = (0, \dots, 0)$; Sinon, $D(u, v) \neq (0, \dots, 0)$.

$$D(u, v) = min\{d(p), p : u \rightarrow v\} \quad (4.1)$$

Par ailleurs, deux nœuds u et v peuvent être liés par différents chemins de délai nul. Ces chemins n'ont pas forcement le même temps d'exécution. Un chemin $p : u \rightarrow v$ ne sera décalé que si tous les chemins liant u et v sont exécutés dans la limite de la période de cycle, tel que soit leurs valeurs de temps d'exécution. Pou cela, nous définissons une deuxième grandeur $T(u, v)$ qui représente la valeur maximale parmi les temps d'exécution des chemins de délais nuls entre les nœuds u et v, telle que indiquée dans l'équation (4.2).

$$T(u, v) = max\{t(p), p : u \rightarrow v \; et \; d(p) = D(u, v)\} \quad (4.2)$$

Le principe de notre démarche implique le test de toutes les combinaisons des chemins de données du GFDM. Dans cet objectif, nous utilisons la notion de matrice permettant de représenter les chemins de données entre tous couples de nœuds du GFDM. Nous procédons à déclarer deux matrices D et T contenant respectivement les valeurs de $D(u, v)$ et $T(u, v)$. Chaque matrice est de dimension $V \times V$ dont les lignes et les colonnes sont indexées par les

noms des nœuds. Un chemin $p : u \to v$ est indexé par la cellule de la ligne u et de la colonne v. Les étapes de remplissage des matrices sont décrites dans l'algorithme 4.1. La démarche débute par le calcul des valeurs $D(u,u)$ et $T(u,u)$ correspondant aux chemins de données composés par un seul nœud, et représentant les cellules des diagonales des deux matrices. Par la suite, nous associons chaque nœuds u à ces successeurs v, dans le but de déterminer les valeurs $D(u,v)$ et $T(u,v)$. Cette étape est répétée d'une façon incrémentale dans le sens de dépendances de données jusqu'à remplir toutes les cellules des matrices.

Algorithme 4-1 Calcul des matrices D et T

Entrée : $G = (V, E, d, t)$ un GFDM réalisable
Sortie : les matrices D et T

1: Début
/ Calculer D et T pour chaque nœud et pour chaque arc */*
2: Pour chaque noeud $u \in V$ faire
3: $\mathrm{D}(u,u) = (0,0)$
4: $\mathrm{T}(u,u) = \mathrm{t}(u)$
5: Fin pour
6: Pour chaque arc $e \in E$ avec $e: u \to v$ faire
7: $\mathrm{D}(u,v) = \mathrm{d}(e)$
8: $\mathrm{T}(u,v) = \mathrm{t}(u) + \mathrm{t}(v)$
9: Ajouter l'élément $(u \xrightarrow{e} v, \mathrm{D}(u,v), \mathrm{T}(u,v))$ à la liste EDGE
10: Fin pour
/ Calculer D et T pour chaque chemin de données */*
11: PATH ← EDGE
12: Répéter
13: **Pour** chaque élément $(u_i \overset{p}{\Rightarrow} v_i, x_i, y_i)$ de la liste PATH **faire**
14: **Pour** chaque élément $(u_j \xrightarrow{e} v_j, x_j, y_j)$ de EDGE **faire**
15: **Si** $v_i = u_j$ & $(u_i \overset{p}{\Rightarrow} v_j, d, t) \notin$ R **alors**
16: Ajouter $\left(u_i \overset{p}{\Rightarrow} v_j, x_i + x_j, y_i + y_j\right)$ à la liste R liste
17: **Sinon si** $v_i = u_j$ & $(u_i \overset{p}{\Rightarrow} v_j, d, t) \in$ R **alors**
18: Ajouter $\left(u_i \to v_j, \min\left(d, x_i + x_j\right), \max\left(t, y_i + y_j\right)\right)$ à la liste R list
19: Fin si
20: Fin pour
21: **Fin pour**
22: **Pour** chaque élément $(u \overset{p}{\Rightarrow} v, x, y)$ de R **faire**
23: $\mathrm{D}(u,v) \leftarrow x$
24: $\mathrm{T}(u,v) \leftarrow y$
25: Fin pour
26: TEST ← R
27: Jusqu'à D et T sont totalement remplis
28: Fin

Prenons l'exemple du graphe flot de donnée bidimensionnelle du filtre numérique d'ondelette, composé de quatre nœuds. L'algorithme 4.1 génère les matrices D et T illustrées respectivement dans le tableau 4-1 et le tableau 4-2. Cette démarche permet de déterminer directement les chemins de données de délais nuls, dont les cellules correspondantes sont colorées en gris.

Tableau 4-1 Matrice D du filtre numérique d'ondelettes

u\v	A	B	C	D
A	(0,0)	(0,0)	(0,0)	(1,1)
B	(1,1)	(0,0)	(1,1)	(1,1)
C	(1,1)	(1,1)	(0,0)	(1,1)
D	(0,0)	(0,0)	(0,0)	(0,0)

Tableau 4-2 Matrice T du filtre numérique d'ondelettes

u\v	A	B	C	D
A	1	3	3	3
B	4	2	6	3
C	4	6	2	3
D	2	4	4	1

L'application de la fonction du retiming multidimensionnel légal $r(D) = (0,1)$ génère un nouveau GFDM dont la valeur du délai de l'arc e_3 est $(0,1)$. Cette modification affecte les délais des trois chemins $D \to A$, $D \to B$ et $D \to C$ tels que affichés dans la matrice D du tableau 4-3.

Tableau 4-3 Matrice D du filtre numérique d'ondelettes après r(D)=(0,1)

u\v	A	B	C	D
A	(0,0)	(0,0)	(0,0)	(1,1)
B	(1,1)	(0,0)	(1,1)	(1,1)
C	(1,1)	(1,1)	(0,0)	(1,1)
D	(0,1)	(0,1)	(0,1)	(0,0)

Tableau 4-4 Matrice T du filtre numérique d'ondelettes après r(D)=(0,1)

u\v	A	B	C	D
A	1	3	3	3
B	4	2	6	3
C	4	6	2	3
D	2	4	4	1

4.4.2 Sélection des chemins de données

Nous procédons à identifier les chemins de données à paralléliser, à partir des matrices D et T. Ces chemins doivent respecter des contraintes en matière du temps d'exécution et du délai. En fait, ordonnancer le GFDM avec la période de cycle c implique que les temps d'exécution de tous les chemins de données de délais nuls sont inférieurs à c. Nous formulons cette notion d'ordonnancement des GFDMs dans le théorème 4.1.

Théorème 4.1. *soit G =(V,E,t,d) un GFDM réalisable, et c une période de cycle quelconque. Les deux conditions suivantes sont équivalentes :*

- $\varphi(G) \leq c$
- *Pour tous nœuds u et v de V, si* $T(u,v) > c$ *alors* $D(u,v) \neq 0$.

Preuve. *On suppose que* $\varphi(G) \leq c$, *et u et v deux nœuds dans V tel que* $T(u,v) > c$. *Si* $D(u,v) = 0$, *alors il existe un chemin p de u à v dont le temps d'exécution est* $t(p) =$

$T(u,v)$ qui est supérieur à c et le délai $d(p) = D(u,v) = 0$, ce qui est contradictoire avec la supposition initiale.
On suppose maintenant que la deuxième condition est vérifiée, et soit $u \to v$ n'importe quel chemin de délai nul dans G, alors on a $D(u,v) = d(p) = 0$, ce qui implique $t(p) \leq T(u,v) \leq c$.

Donc, pour ordonnancer le GFDM avec la période de cycle minimale, notre technique doit transformer les valeurs des matrices en respectant la deuxième condition du théorème 4.1 : aucun chemin $p: u \to v$ ayant un temps d'exécution supérieur à c_{min} ne doit avoir un délai nul. Ce principe signifie que tout chemin ayant $T(u,v) > c$ et $D(u,v) = 0$ doit être affecté par le retiming multidimensionnel dans l'objectif d'avoir $D(u,v) \neq 0$.

En premier lieu, les valeurs des matrices D et T peuvent nous indiquées directement si la condition du théorème est respectée. Prenant l'exemple du filtre numérique d'ondelette dont les matrices D et T sont respectivement affichés dans le tableau 4-1 et le tableau 4-2, nous distinguons que les chemins critiques $D \to B$, $D \to C$, $A \to B$ et $A \to C$ ont des délais nuls dont leurs temps d'exécution excèdent c_{min}. En deuxième lieu, les valeurs des matrices nous renseignent sur les chemins de données à paralléliser. En fait, le retiming multidimensionnel est appliqué aux nœuds dont toutes les arcs entrants sont de délais non nuls. Pour le cas du GFDM de la figure 3.1, les premiers chemins à paralléliser sont ceux commençant par le nœud D. D'où, nous explorons les lignes des matrices correspondant à ce nœud. De plus, atteindre la période de cycle minimale par retiming multidimensionnel implique le parallélisme des chemins de données de délais nuls dont le temps d'exécution est inférieur à c_{min}. Pour le cas du filtre numérique d'ondelette, les cellules appartenant à la ligne D et ayant $D(n,x) = (0,\dots,0)$ et $t(x,n) \leq c_{min}$ sont les chemins D et $D \to A$.

4.5 Retiming multidimensionnel pour un chemin de données

En fait, les travaux existants ont décrit une procédure formelle pour la sélection d'une fonction de retiming multidimensionnel. Elle consiste à déterminer l'ensemble des vecteurs d'ordonnancement du graphe G, i.e., les vecteur s qui vérifient $d(e) \times s > 0$ pout tout $e \in E$, et par la suite choisir une fonction r orthogonale à s. Ce choix dépend uniquement des délais non nuls, et ne prend pas en considération les nœuds à paralléliser. Cependant, la théorie des techniques existantes se limitent à appliquer la fonction à des nœuds ayant des arcs sortants de délais nuls. En fait, ces derniers peuvent être considérés autant que des premiers nœuds appartenant à des chemins de délais nuls. Dans ce contexte, nous décrivons dans le théorème 4.2 la façon de déduire une fonction de retiming multidimensionnel pour un nœud ayant un arc entrant de délais nul.

Théorème 4.2. *$G = (V,E,t,d)$ un GFDM réalisable, u et $v \in V$ tel que v a un seul arc entrant $e: u \to v$ et $d(e) = (0,\dots,0)$. Si r une fonction de retiming légale de u, alors r est une fonction de retiming légale de v.*

Preuve. *Soit $p: \dots \xrightarrow{(a,b)} u \xrightarrow{(0,0)} v \xrightarrow{(x,y)} \dots$ un chemin dans G. $r(u) = (i,j)$ une fonction de retiming multidimensionnel légale de G signifie qu'il existe un vecteur d'ordonnancement s tel que $(i,j) \times s \geq 0$ et $(x,y) \times s \geq 0$. La somme de ces deux inéquations vérifie que $(x+i, y+j) \times s \geq 0$. Sachant que $(a-i, b-j) \times s \geq 0$ et $(0,0) \times s \geq 0$, le GFDM contenant $p: \dots \xrightarrow{(a-i,b-j)} u \xrightarrow{(0,0)} v \xrightarrow{(x+i,y+j)} \dots$ est réalisable, pouvant être exécuté dans l'ordre du vecteur d'ordonnancement s. Par conséquent, r(v) est une fonction de retiming multidimensionnel légal.*

Ce théorème prouve qu'une fonction de retiming légal d'un nœud peut être appliquée à son successeur, si ce dernier a un seul arc entrant. De plus, cette théorie peut être testée successivement sur une séquence de nœuds respectant la même condition. D'où, le théorème 4.2 permet d'identifier une fonction de retiming multidimensionnel pour un chemin p de données de délai nul, tel que soit le nombre des nœuds qu'il intègre et tant que $t(p) \leq c_{min}$.

Un GFDM peut intégrer plusieurs chemins de délais nuls ordonnancés dans le même espace temporel. Cependant, déterminer un vecteur de retiming pour chaque chemin à décaler est loin d'être une solution adéquate. Il s'avère donc indispensable de réduire les fonctions de retiming. Dans ce contexte, nous proposons de déterminer l'ensemble des chemins qui peuvent être décalés avec la même fonction de retiming. En fait, les travaux existants décrivent une procédure formelle pour décaler un ensemble de nœuds avec la même fonction. Cette démarche nécessite que tous les arcs entrants aux chemins soient de délais non nuls. Nous décrivons dans le théorème suivant une procédure de prédiction d'une fonction de retiming pour un ensemble de chemins de délais nuls.

Théorème 4.3. *Soient $X, Y \subseteq V$, $X \cap Y = \emptyset$, tel que tout arc entrant à y est sortant de x, avec $y \in Y$ et $x \in X$. Si $r(X)$ est un retiming multidimensionnel légal alors $r(Y)$ est un retiming multidimensionnel légal.*

Preuve. *Soient s un vecteur d'ordonnancement vérifiant $d(e) \times s > 0$ et r est un vecteur orthogonal à s. d'après [2], si X est un ensemble de nœuds dont tous les arcs entrants de délais non nul, alors r(X) est une fonction de retiming légal. Prenant en considération le théorème 4.2, on peut conclure que r(Y) est une fonction de retiming légal.*

4.6 Technique du retiming multidimensionnel décalé

4.6.1 Démarche

L'objectif de notre technique est d'ordonnancer le GFDM avec la période de cycle minimale. En premier lieu, le principe de notre technique consiste à sélectionner les chemins de données ayant des arcs entrant de délais non nuls. Ces chemins doivent avoir un délai nul et des temps d'exécution inférieurs à c_{min}. Cette sélection est effectuée en déterminant les nœuds ayant des dépendances de données non nul, et en explorant leurs lignes dans les matrices D et T. De ce fait, nous sélectionnons les chemins dont les lignes vérifient la deuxième condition du théorème 4.1. Cette tâche est décrite explicitement dans le paragraphe 4.5.2. En deuxième lieu, les chemins sélectionnés sont à paralléliser pour réduire le chemin critique du GFDM. En se basant sur le théorème 4.3, ils peuvent être décalés par la même fonction de retiming. Nous procédons par la suite à sélectionner une fonction de retiming tel que décrit dans le théorème 4.2. Pour atteindre la période de cycle minimale, ces étapes sont à répéter à partir des nœuds successeurs des chemins décalés. Vue la modification des valeurs de délais du GFDM, les matrices D et T sont ainsi à rectifier après chaque retiming multidimensionnel.

4.6.2 Identification des chemins

Cette étape consiste à explorer les matrices D et T dans le but de sélectionner les chemins à décaler, tel que décrit dans l'algorithme 4.2. Elle débute par déterminer les nœuds ayant des arcs entrants de délai non nul. Par la suite, elle explore les lignes des matrices D et T correspondante à chaque nœud dans l'objectif de sélectionner les chemins ayant $D(u, v) = (0, \dots, 0)$ et $t(u, v) \leq c_{min}$. Ces chemins sont ainsi rangés dans la liste L.

En fait, notre démarche doit employer le minimum de fonctions de retiming pour atteindre la période de cycle minimale. De ce fait, elle doit maximiser le nombre des nœuds dans les chemins sélectionnés, tout en gardant une exécution dans la limite de période de cycle. De ce fait, si deux chemins de liste L sont totalement superposés, la démarche doit choisir ceux ayant la taille maximale, dont la tâche est assurée par l'instruction 9 de l'algorithme.

Algorithme 4-2 Sélection les chemins

Entrée : un GFDM réalisable $G = (V, E, t, d)$
Sortie : liste de nœuds L
1: Début
2: Déterminer les nœuds n ayant tout les arcs entrants de délais non nuls et au moins un arc sortant de délai nul
3: **Pour** chaque nœud n
4: **Pour** chaque nœud x du GFDM
5: **Si** $D(n, x) = (0, \dots, 0)$ et $T(n, x) \leq c_{min}$ **alors**
6: Ajouter le chemin $p: n \rightarrow x$ à L
7: Fin si
8: Fin pour
9: Mettre à jour la liste L
10: Fin pour
11: Fin

4.6.3 Algorithme du retiming multidimensionnel décalé

Notre technique débute par le calcul de la période de cycle minimale. Puis, elle fait appel à l'algorithme 4.1 pour calculer les valeurs des cellules des matrices D et T correspondant au GFDM. Si les matrices contiennent des chemins qui ne satisfassent pas le théorème 4.1, la technique choisit une fonction de retiming r. Par la suite, elle fait appel à l'algorithme 4.2 pour identifier les chemins dans le but de les décaler par la fonction r. Les valeurs des matrices doivent être actualisées en fonction du décalage effectué. Ces étapes sont répétées jusqu'il n'existe aucun chemin critique p dans le GFDM dont $t(p) > c_{min}$, tel que décrit dans l'algorithme 4.3.

Prenons l'exemple du filtre numérique d'ondelette. L'algorithme 4.3 détermine une période de cycle $c_{min} = 2$. Les matrices D et T correspondant au GFDM initial sont affichées respectivement dans le tableau 4-1 et le tableau 4-2. La première itération de l'algorithme 4.3 détermine une fonction de retiming légal $r = (0,1)$. Puis, elle fait appel à l'algorithme 4.2 pour identifier les chemins à décaler. Elle se positionne au niveau de la ligne du nœud D, vue qu'il est le seul ayant tout les ars entrants de délais non nul. Les chemins susceptibles d'être décalé sont le nœud D et le chemin $p: D \rightarrow A$. D'où, l'algorithme 4.2 retourne le chemin $p: D \rightarrow A$ qui sera décalé $r(D \rightarrow A) = (0,1)$. Ce décalage entraîne la modification des valeurs des matrices D et T, dont le nouveau contenu est affiché dans le tableau 4-5 et le tableau 4-6, correspondant au GFDM de la figure 4.3(b). Ce graphe vérifie les conditions du théorème 4.2 et représente une solution plus optimale que celles générés par les techniques existantes.

4.7 Extension de la technique de retiming multidimensionnel décalé

4.7.1 Utilisation multiple d'une fonction de retiming

Dans le cas général, l'atteinte de la période de cycle minimale nécessite d'application de plusieurs fonctions de retiming multidimensionnel. Pour l'exemple du filtre à RII dont

$t(M) = t(A) = 1$, l'atteinte de la période de cycle minimale nécessite l'utilisation de 4 fonctions de retiming multidimensionnel. Cependant, nous avons montré dans la section 4.2 que les valeurs des indexes de la fonction de retiming $d.x$ et $d.y$ augmente en fonction du rang ; e.i, plus l'ordre de la fonction de retiming est grand, plus les valeurs des indexes sont importantes. Pour le cas du filtre à RII, la première fonction de retiming est égale à $(1, -1)$ tandis que la quatrième est égale à (8,-15) [2,3].

Algorithme 4-3 Retiming multidimensionnel décalé

Entrée : un GFDM réalisable $G = (V, E, t, d)$
Sortie : un GFDM réalisable $G_r = (V, E, t, d_r)$ avec la période de cycle $\varphi(G_r) = c_{min}$
1: Début
2: Déterminer la période de cycle minimale $\boldsymbol{c_{min}}$
3: Calculer les matrices $\boldsymbol{D}$ et $\boldsymbol{T}$
4: **Tant que** $\exists p \; tel \; que \; d(p) = (0, \dots, 0) \; et \; t(p) > c_{min}$ **faire**
5: Trouver un vecteur d'ordonnancement $s = (s.x, s.y)$ tel que $s.x + s.y$ est minimale
6: Choisir une fonction de retiming MD $\boldsymbol{r}$ orthogonal à $\boldsymbol{s}$
7: Identifier la liste des nœuds $\boldsymbol{L}$ à décaler (tel que indiqué dans l'algorithme 4.2)
8: **Pour** chaque chemin $\boldsymbol{ch}$ de la liste $\boldsymbol{L}$
9: Appliquer la fonction $\boldsymbol{r}(\boldsymbol{ch})$
10: Fin pour
11: Actualiser les matrices $\boldsymbol{D}$ et $\boldsymbol{T}$
12: Fin tant que
13: **Fin**

Tableau 4-5 Matrice D du filtre numérique d'ondelettes après retiming r(D→A)=(0,1)

u\v	A	B	C	D
A	(0,0)	(0,1)	(0,1)	(1,1)
B	(1,1)	(0,0)	(1,1)	(1,1)
C	(1,1)	(1,1)	(0,0)	(1,1)
D	(0,0)	(0,1)	(0,1)	(0,0)

Tableau 4-6 Matrice T du filtre numérique d'ondelettes après retiming r(D→A)=(0,1)

u\v	A	B	C	D
A	1	3	3	3
B	4	2	6	3
C	4	6	2	3
D	2	4	4	1

En premier lieu, les valeurs des indexes impliquent des tailles importantes de prologue et d'épilogue. La taille de ces blocs de code est similaire au nombre des périodes de cycle nécessaires pour leurs exécutions. En outre, plus la taille de prologue et d'épilogue est importante, plus l'implémentation requiert des ressources matérielles. Par conséquent, les fonctions de retiming multidimensionnel ayant des indices de valeurs importantes sont inadéquates pour des implémentations temps réel embarquées. En deuxième lieu, l'application du retiming multidimensionnel dépend du nombre d'itérations des boucles [5]. Le nombre des itérations d'une boucle x doit être impérativement supérieur à l'index $|d.x|$ de la fonction du retiming multidimensionnel. Pour le cas du filtre à RII, la dernière fonction de retiming multidimensionnel (8,-15) signifie que 15 itérations de la boucle interne sont à décaler. De ce fait, les fonctions de retiming multidimensionnel ayant des valeurs importantes d'indexes

admettent des limites lors de l'application à des nids de boucles dont les nombres d'itérations sont réduits. En troisième lieu, la démarche de sélection d'une fonction retiming multidimensionnel requiert un temps d'exécution d'une complexité de $O(|E|)$. Par conséquent, l'extraction multiple des fonctions de retiming engendre une augmentation du temps de conception pour la généralisation du GFDM final.

Cependant, la fonction du retiming ayant les valeurs minimales d'indexes (et engendrant le minimum de taille de prologue et d'épilogue) est la première fonction, intitulé f_1, dont les indexes sont extraits à partir du GFDM initial. De ce fait, dans l'objectif d'optimiser les GFDMs finaux, nous proposons une démarche permettant d'utiliser la même fonction de retiming multidimensionnel pour tous les chemins à décaler. Dans ce contexte, nous décrivons dans le théorème 4.4 la théorie permettant d'identifier différents fonctions de retiming multidimensionnel à partir d'une seule.

Théorème 4.4. *soit $G = < V, E, d, t >$ un GFDM réalisable, $X, Y \subseteq V$, $X \cap Y = \emptyset$ tel que $y \in Y$ and $x \in X$ et $e: x \xrightarrow{(0,\dots,0)} y$ pour tout arc allant de x à y, et un entier $k > 1$. Si $r(X)$ est une fonction de retiming legal alors $(k \times r)(Y)$ est une fonction de retiming légal.*

Preuve. *r(X) est une fonction de retiming légal signifie qu'il existe un vecteur d'ordonnancement s du GFDM tel que $r \perp s$ et ainsi $r \times s = 0$. Sachant que k est strictement positif, $(k \times r) \times s = 0$. D'où, $(k \times r)(X)$ est une fonction de retiming multidimensionel légal. En se basant directement sue le théorème 4.3, $(k \times r)(Y)$ est un retiming multidimensionnel légal.*

Ce théorème offre une flexibilité dans le choix des fonctions de retiming multidimensionnel de façon qu'il permette de décaler deux chemins avec la même fonction. Prenons l'exemple du GFDM de la figure 4.6(a) contenant un chemin critique $p_{ch}: A \to B \to C \to D$. Dans le cas où $t(A) = t(B) = 1$ et $t(C) = t(D) = 2$, nous distinguons deux chemins $p_1: A \to B$ et $p_2: C$ à décaler. La premier fonction engendrant des tailles de prologue et d'épilogue optimales est $r = (0,1)$. Si le chemin p_1 est décalé avec la fonction $r(p_1) = (0,1)$, le chemin p_2 est dans l'obligation d'être décalé par la fonction $r(p_2) = (1,-2)$, engendrant une implémentation finale composée de 35 instructions et exécutée dans 150 unités de temps. Cependant, le théorème 4.4 permet d'appliquer la fonction $(2 \times (0,1))$ sur le chemin p_1, dont le GFDM résultat est schématisé dans la figure 4.6(b). Le délai de arc $B \to C$ permet de décaler le chemin p_2 par la fonction de retiming $(1 \times (0,1))$. L'implémentation du GFDM final dans la figure 4.6(c) est composée de 19 instructions et exécutée dans uniquement 95 unités de temps. Par conséquent, l'utilisation de la fonction de retiming minimale à plusieurs reprises résulte une implémentation avec un temps d'exécution et une taille de code réduits. Les pourcentages de gain en matière de temps d'exécution et de la taille du code, par rapport à la technique enchaînée, sont respectivement de 36.66% et 45.71%.

4.7.2 Démarche

Le théorème 4.4 montre qu'une succession de chemins de données de délais nuls peuvent être décalés successivement en utilisant une seul fonction de retiming : pour un GFDM composé de x chemins de données de délais nuls, chaque séquence du chemin peut être décalée avec la fonction $(L \times r)$ telle que L est un entier strictement positif et plus petit que celui du chemin prédécesseur. Nous proposons donc une démarche permettant d'identifier les chemins à décaler, de les étiqueter par des entiers positifs, dont les valeurs sont décroissantes dans le sens des dépendances de données.

Dans ce contexte, les chemins doivent être étiquetés par des valeurs minimales pour réduire la taille des prologues et des épilogues. D'où, le paramètre L est dans l'impératif de commencer par la valeur 1 et d'être incrémenté par une seule unité. Cependant, le retiming multidimensionnel est appliqué à partir des chemins ayant tout les arcs entrants de délais non nuls, et ayant les valeurs maximales d'étiquetage. Or, ces valeurs d'étiquetage ne sont déterminées qu'après l'étiquetage des chemins successeurs. Pour cela, nous procédons à répartir la démarche de cette technique en deux étapes : la première assure l'identification et l'étiquetage des chemins à décaler. Elle génère ainsi un « Graphe Flot de Données Multidimensionnel Etiqueté (GFDME) », dont la démarche est décrite dans le paragraphe 4.7.3. La deuxième étape consiste à sélectionner et appliquer une fonction de retiming multidimensionnel, dans le sens décroissant des valeurs d'étiquetage.

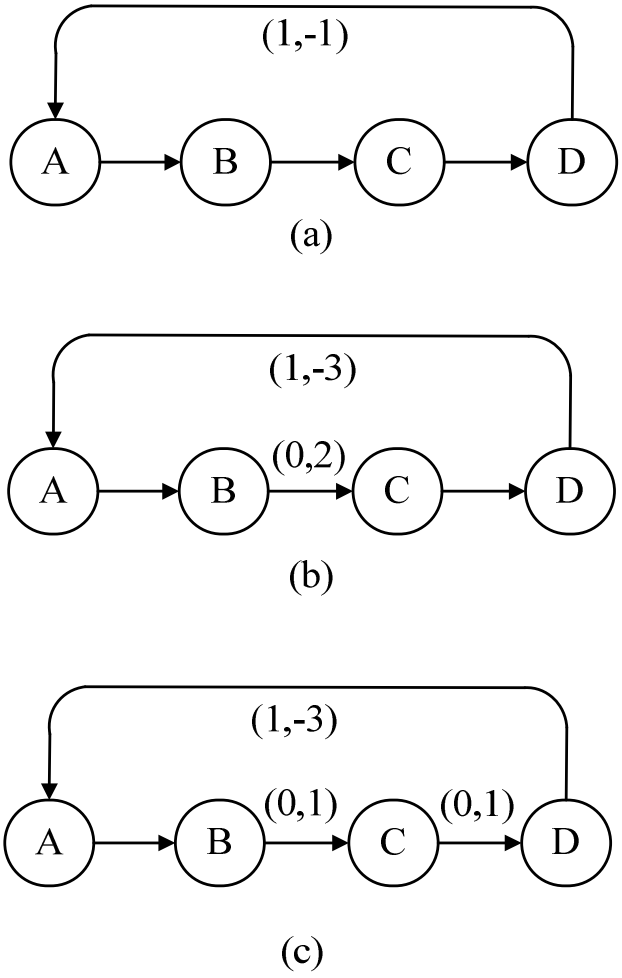


Figure 4.6 (a) GFDM initial, (b) GFDM après r(p_1)=2×(0,1) ;(c) GFDM après r(p_2)=(0,1)

4.7.3 Génération du graphe flot de données multidimensionnel étiqueté

Cette étape consiste à balayer d'une façon incrémentale le GFDM dans le sens inverse des dépendances de données, en commençant par les nœuds ayant tous les arcs entrants de délais non nuls. Pour chaque nœud n, on vérifie les colonnes d'indice n dans les matrices D et T : si un chemin $p : x \rightarrow n$ est de délais nuls et $t(x \rightarrow n) \leq c_{min}$ alors il est ajouté à liste des chemins à décaler. En se basant sur le principe de maximisation des nœuds dans les chemins de données, nous procédons à filter la liste L dans le but d'éliminer la redondance des chemins. Par la suite, ils sont étiquetés par $L = 1$. Ces étapes sont répétées en incrémentant la valeur de L, jusqu'à tester tous les nœuds du GFDM, telles que décrites dans l'algorithme 4.4.

Algorithme 4-4 Génération du GFDME

Entrée : $G = (V, E, d, t)$ un GFDM réalisable
Sortie : GFDME, valeur maximale du niveau L

1: Début
2: initialiser $L \leftarrow 0$
3: Déterminer la période de cycle minimale $\boldsymbol{c_{min}}$
4: Calculer les matrices $\boldsymbol{D}$ et $\boldsymbol{T}$
5: Extraire tous les arcs de délais non nuls
6: Remplir NC par tous les nœuds sans aucun arc sortant
7: Tant qu'existe des chemins de D non testé**s faire**
8: **Pour** chaque nœud n de NC
9: **Pour** chaque nœud x du GFDM
10: **Si** $\boldsymbol{D(x,n)} = (\boldsymbol{0}, \dots, \boldsymbol{0})$ et $\boldsymbol{T(n,x)} \leq \boldsymbol{c_{min}}$ **alors**
11: Ajouter le chemin $\boldsymbol{p}: \boldsymbol{x} \rightarrow \boldsymbol{n}$ à $\boldsymbol{R}$
12: Fin si
13: Fin pour
14: Fin pour
15: Raffiner la liste $\boldsymbol{R}$
16: Etiqueter les chemins de $\boldsymbol{R}$ par $\boldsymbol{L}$
17: Incrémenter $\boldsymbol{L}$
18: Remplir $\boldsymbol{NC}$ par les nœuds prédécesseurs des chemins de $\boldsymbol{R}$
19: Fin tant que
20: Fin

4.7.1 Algorithme de l'extension du retiming multidimensionnel décalé

Notre technique débute par choisir la fonction de retiming multidimensionnel optimale dans le but de l'appliquer sur les chemins sélectionnés. Dans ce contexte, le choix de la fonction de retiming est à effectuer tel que décrit dans [2]. Par la suite, elle fait recours à l'algorithme 4.4 pour générer le GFDME et la valeur maximale d'étiquetage. Notre démarche procède à appliquer la fonction de retiming sélectionnée sur les chemins selon l'ordre décroissant des valeurs d'étiquetage. Elle débute par l'exploration du GFDM par les nœuds S ayant tous les arcs entrants de délais non nuls jusqu'à trouver les nœuds A ayant la plus grande valeur d'étiquetage k. Par la suite, elle calcule le délai $r \times k$, le soustraire des arcs entrants aux nœuds S et l'ajouter aux arcs sortants des nœuds A. Ces étapes sont répétées jusqu'à atteindre les nœuds ayant la valeur d'étiquetage égal à 1, tel que décrit dans l'algorithme 4.5.

En utilisant la technique de retiming multidimensionnel décalé, un GFDM ordonnancé avec la période de cycle minimale est toujours achevé, dont l'efficacité de la démarche est démontrée dans le théorème 4.5.

Théorème 4.5. *Soit $G = (V, E, d, t)$ un GFDM réalisable. La technique de retiming multidimensionnel décalé transforme G en G_r tel que G_r est ordonnancé avec la période de cycle minimale, dont la complexité est $O(V^2 + E)$.*

Preuve. *L'algorithme 1 permet le calcul des matrices D et T pour chaque pair de nœuds du MDFG, avec une complexité de $O(E + V^2)$. Par la suite, l'algorithme 2 teste au maximum toutes les cellules des matrices D et T, dans $(n \times V)$ instructions tel que $n > 0$. Finalement, l'algorithme 3 sélectionne une fonction de retiming*

multidimensionnel nécessitant au maximum E instructions, et l'appliquer à chaque chemin sélectionné. D'où, la complexité de la technique de retiming multidimensionnel décalé est de $O(V^2 + E)$.

Algorithme 4-5 Extension de la technique de retiming multidimensionnel décalé

Entrée : un GFDM réalisable $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{t}, \boldsymbol{d})$
Sortie : un GFDM réalisable $\boldsymbol{G_r} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{t}, \boldsymbol{d_r})$ avec la période de cycle $\boldsymbol{\varphi(G_r) = c_{min}}$

1: Début
2: Trouver un vecteur d'ordonnancement $\mathbf{s} = (\mathbf{s.x}, \mathbf{s.y})$ tel que $\mathbf{s.x} + \mathbf{s.y}$ est minimal
3: Choisir une fonction de retiming multidimensionnel $\boldsymbol{r}$ orthogonal à $\boldsymbol{s}$
4: Générer le GFDME (tel que indiqué dans l'algorithme 4.4)
5: Définir l'ensemble S des nœuds ayant tous les arcs entrants de délais non-nul
6: **Pour** j de k à 1 **faire**
7: Définir l'ensemble des nœuds A étiquette par j
8: **Pour** tout n ∈ S **faire**
9: Soustraire (j × r) de tous arcs entrants à n
10: Fin pour
11: **Pour** tout n ∈ A **faire**
12: Ajouter (j × r) à tous arcs sortants de n
13: Fin pour
14: S ← (S-A) + successeurs (S ∩ A)
15: Fin pour
16: **Fin**

Prenons le cas du filtre numérique d'ondelettes schématisé dans la figure 3.1. L'algorithme principale choisit la fonction $r = (0,1)$. Par la suite, il fait appel à l'algorithme 4.4 pour générer le GFDME correspondant. Après la génération des matrices D et T affichées respectivement dans le tableau 4-1 et le tableau 4-2, l'algorithme extrait les arcs e_4 et e_5 et commence par tester les chemins ayant comme extrémité finale le nœud B ou C. Pour le nœud C, la démarche teste les cellules de la colonne C et identifie les chemins p dont $D(p) = 0$ et $T(p) \leq c_{min}$. Cette étape identifie deux chemins vérifiant la condition qui sont respectivement les nœuds B ou C. De ce fait, l'algorithme identifie leurs nœuds prédécesseurs A pour l'étiqueter par 1. La deuxième itération de l'algorithme 4-4 teste le chemin dont l'extrémité est le nœud A, et achève le nœud D sans que le temps d'exécution dépasse C_{min}. D'où, le GFDME final est illustré dans la figure 4.7. L'algorithme n'applique la fonction de retiming qu'une seule fois, dont le chemin décalé est $D \rightarrow A$.

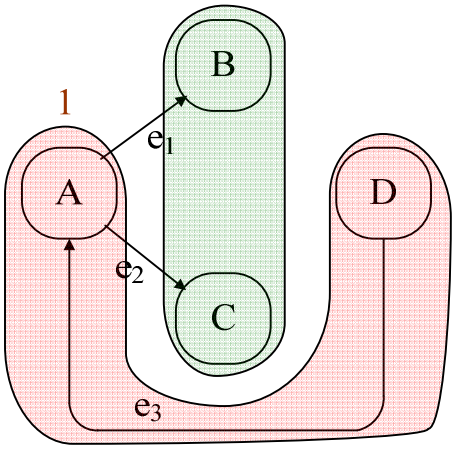


Figure 4.7 GFDME du filtre numérique d'ondelettes

4.8 Résultats expérimentaux

4.8.1 Principe

Nous validons dans cette section l'extension de la technique du retiming multidimensionnel décalé. Notre expérimentation consiste à comparer les résultats générés par notre technique à ceux générés par les techniques progressive et enchaînée. Cette validation évalue l'évolution de deux paramètres : le temps d'exécution et la taille du code. Nous utilisons dans notre démarche de validation deux applications multidimensionnelles types, qui sont respectivement : l'Algorithme de Jacobi (AJ) [102] et le Transformé de Walsh-Fourrier (TWF) [26]. Chaque application est implémentée pour le cas de différentes tailles de matrices en entrée. Nous déterminons ainsi le GFDM de chaque application, tel que décrit dans l'Annexe B. Les applications sont implémentées dans deux cibles d'architecture NVIDIA qui sont la carte Quadro-600 et la carte G-105-M dont les informations des nombres des GPUs, les fréquences des GPUs et les fréquences des mémoires sont indiquées dans le tableau 4-7.

Le choix du parallélisme de la technique du retiming multidimensionnel décalé est basé sur le temps d'exécution des instructions. Dans ce contexte, nous identifions les instructions de calcul des applications multidimensionnelles afin de mesurer leurs temps d'exécution, dont les valeurs sont affichées dans le tableau 4-7. Pour des raisons de simplification, ces temps d'exécution sont convertis en périodes de cycles, tel que indiquées dans la dernière ligne du tableau 4.7.

Tableau 4-7 Caractéristiques techniques des cartes NVIDIA

Architecture			QUADRO-600	G-105-M
GPUs			96	16
Fréquence GPU (GHZ)			1.28	1.23
Fréquence mémoire (MHZ)			800	790
Temps d'exécution des instructions	Addition	10^{-5} secondes	0.3	0.59
		Cycle	1	1
	Multiplication	10^{-5} secondes	0.31	0.61
		Cycle	1	1
	division	10^{-5} secondes	0.61	1.19
		Cycle	2	2

4.8.2 Validation de la technique du retiming multidimensionnel décalé

Les temps d'exécution des nœuds sont utilisés lors de l'application des trois fonctions de retiming multidimensionnel. Dans le cas du TWF, la technique de retiming multidimensionnel décalé utilise une seule fonction $(1,-1)$. La technique progressive et enchaînée utilisent chacune deux fonctions de retiming qui sont respectivement $\{(2,-2),(1,-1\}$ et $\{(1,-1),(2,-3)\}$. Dans le cas de l'application AJ, la technique de retiming multidimensionnel décalé nécessite une seule fonction de retiming $(0,1)$ tandis que les techniques progressives et enchaîné nécessite deux dont les valeurs sont $\{(0,2),(0,1\}$ et $\{(0,1),(1,-3)\}$. Les tailles des codes des implémentations générées par les trois techniques

sont affichées dans le tableau 4-8. En fait, les transformations dues au retiming multidimensionnel sont indépendantes des nombres des itérations des boucles. D'où, les tailles des codes des GFDMs générés par les trois techniques sont indépendantes de la taille de la matrice en entrée. En se basant sur ces valeurs, nous constatons que la technique de retiming multidimensionnel décalé génère des implémentations avec des tailles de codes nettement inférieures aux celles générées par les deux autres techniques, dont elle présente un gain de 70.31% par rapport à la technique progressive et 35.35% par rapport à la technique enchaînée.

Tableau 4-8 Tailles des codes des implémentations en fonction des techniques du retiming multidimensionnel

Applications	Retiming multidimensionnel décalé	Retiming multidimensionnel enchaîné	Retiming multidimensionnel progressif
TWF	22	42	68
AJ	10	13	37
Gain		35.35%	70.31%

À partir des GFDMs générés, nous déduisons six codes tel que chacun correspond à une taille de matrices d'entrée parmi les suivantes (8*8, 10*10, 12*12, 14*14, 16*16, 18*18). Chaque code est édité par la plateforme de programmation parallèle CUDA, et ordonnancé avec les directives offertes, telles que indiquées dans l'Annexe A. Chaque code est exécuté et compilé sur chacune des cibles citées dans le paragraphe 4.8.1. Les temps d'exécution de l'implémentation de la TWF sur QUADRO 600 et G-105-M sont illustrés respectivement dans les histogrammes des figures 4.8 et 4.9. De même pour l'AJ, les temps d'exécution sont illustrés dans les figures 4.10 et 4.11.

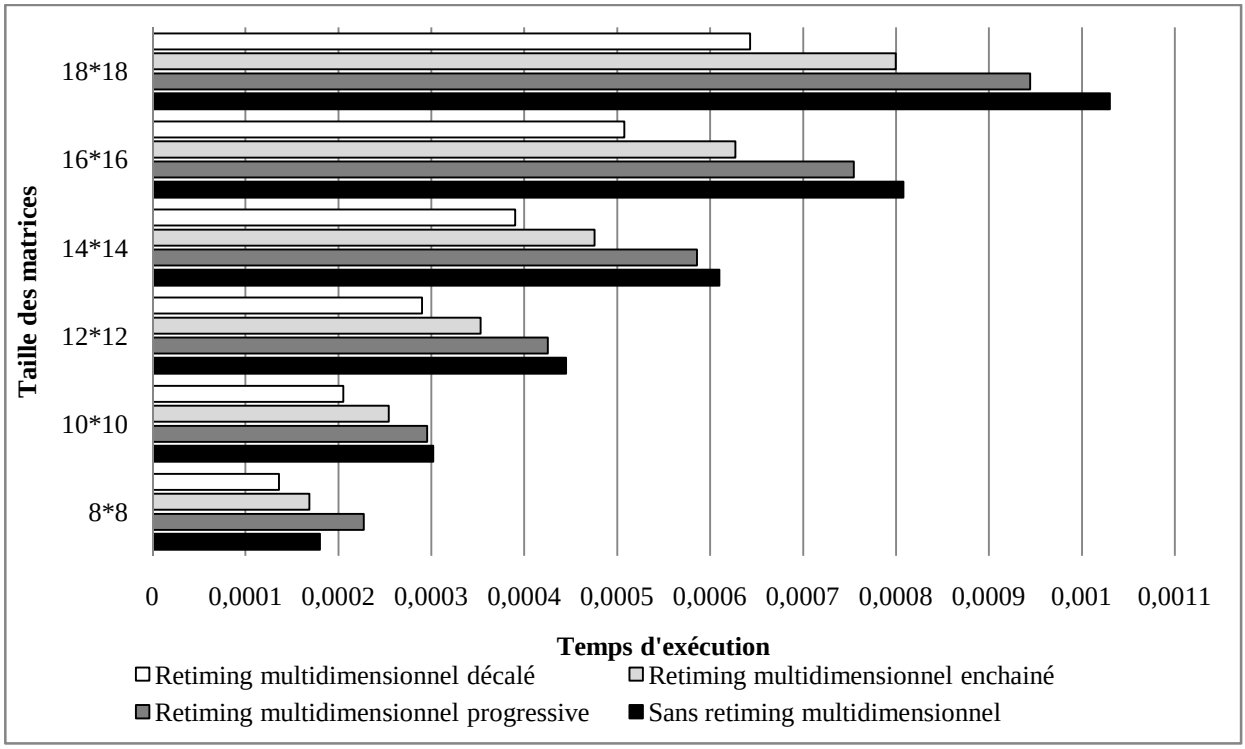


Figure 4.8 Temps d'exécution du transformée de Walsh Fourrier sur l'architecture QUADRO 600

Les valeurs expérimentales ont prouvé que notre technique présente un gain en matière de temps d'exécution par rapport aux résultats générés par les techniques existantes, permettant une moyenne de gain de 34.56% par rapport à la technique progressive et 20.67% par rapport à la technique enchaînée.

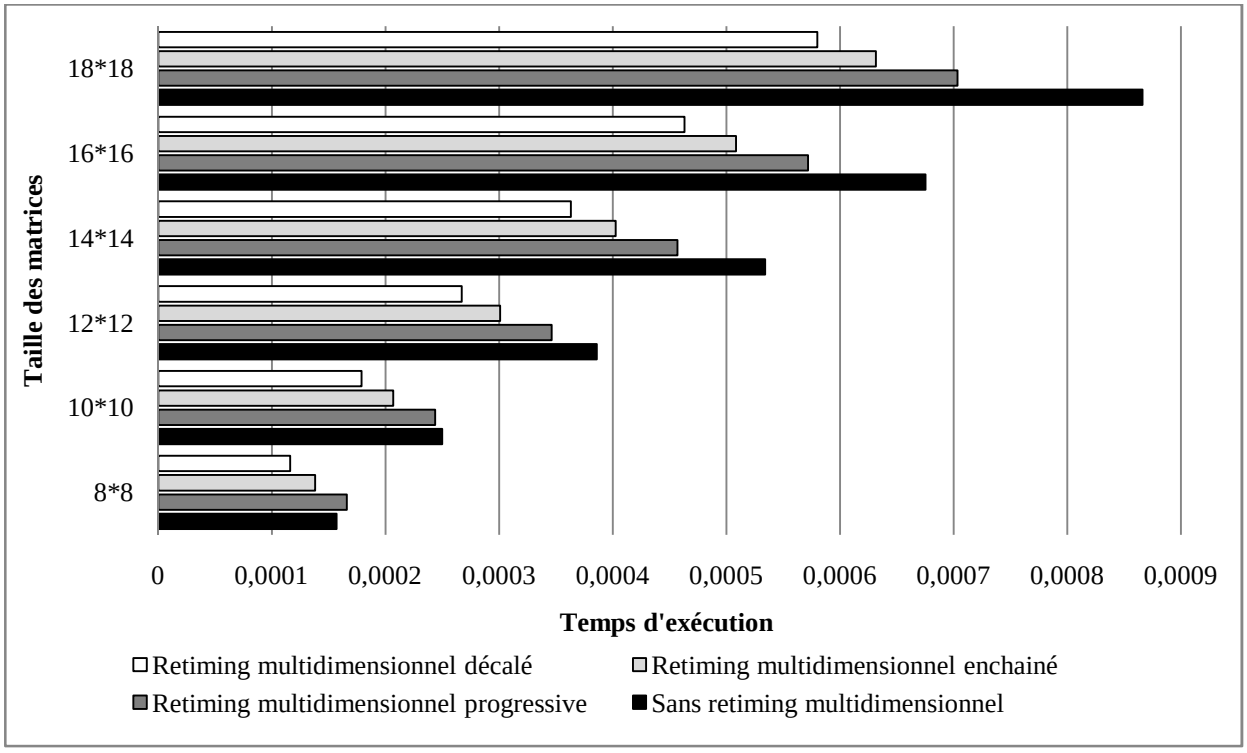


Figure 4.9 Temps d'exécution du transformée de Walsh Fourrier sur l'architecture G 105 M

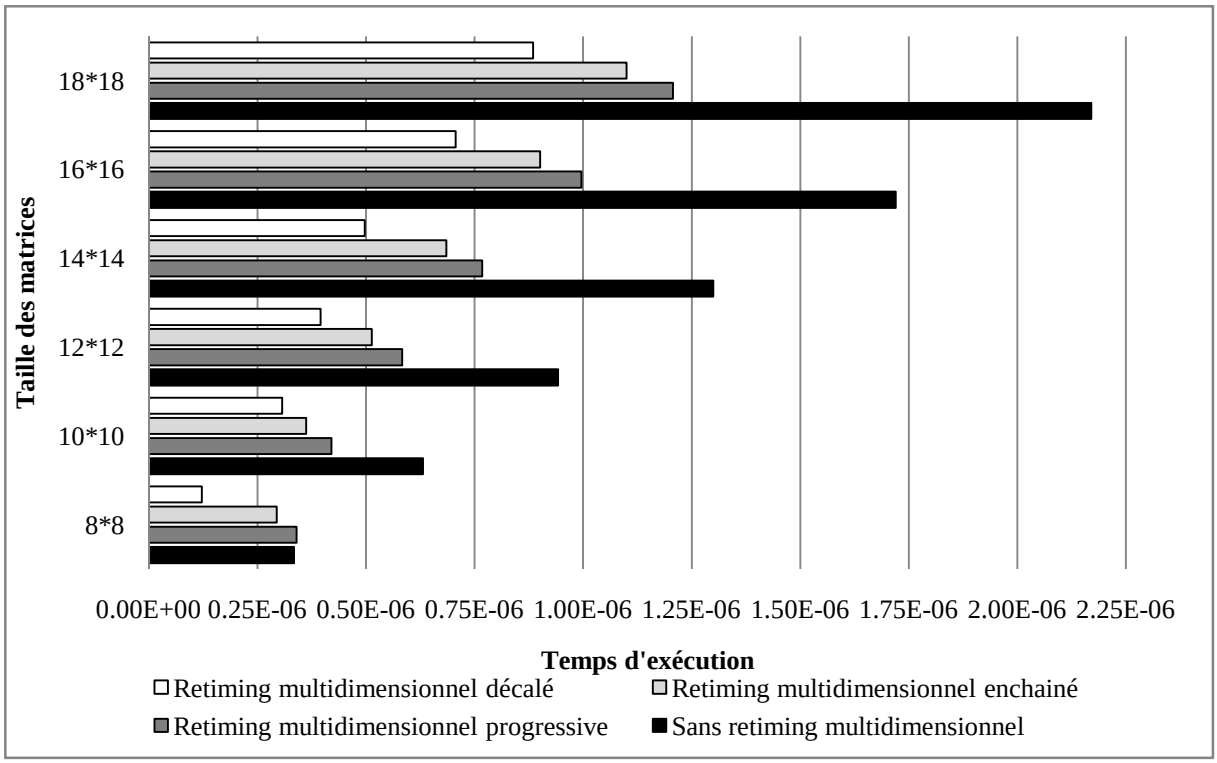


Figure 4.10 Temps d'exécution de l'algorithme du JACOBI sur l'architecture QUADRO 600

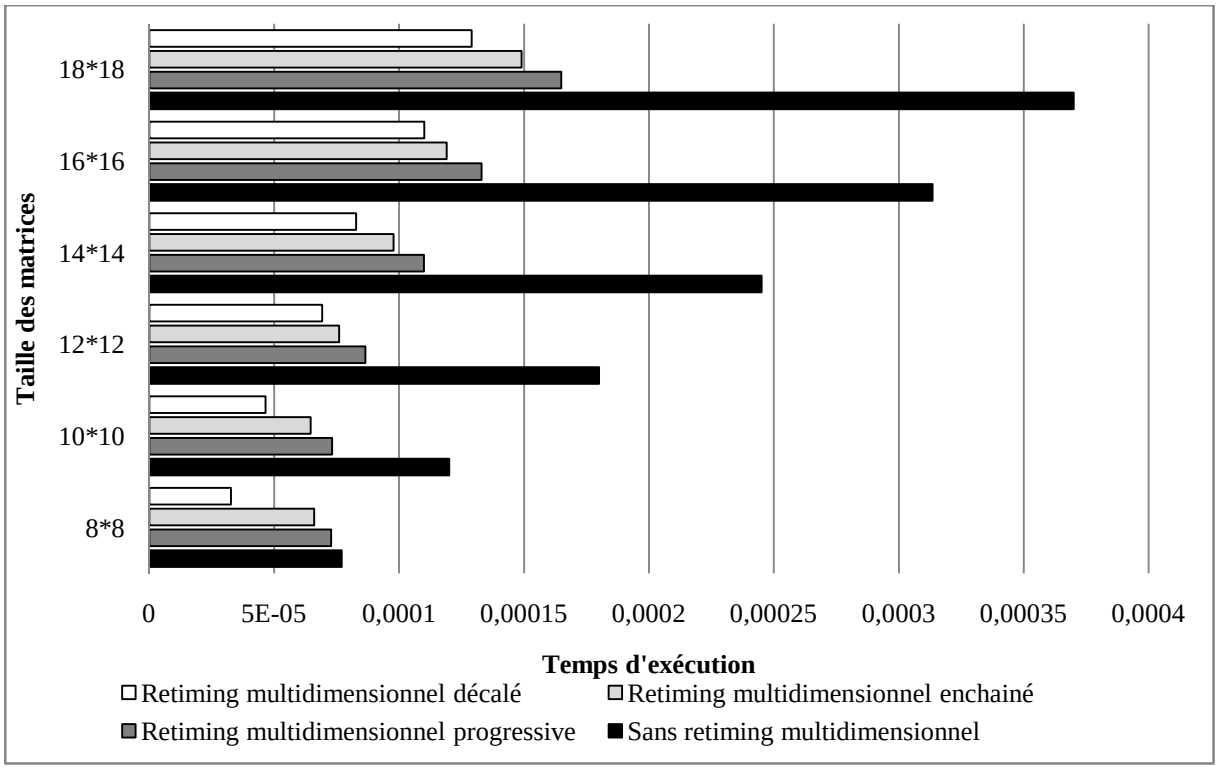


Figure 4.11 Temps d'exécution de l'algorithme du JACOBI sur l'architecture G 105

La technique de retiming multidimensionnel décalé offre une amélioration considérable par rapport au code initial, tel que soit la taille de la matrice d'entrée. Pour le cas du TWF, le code avec une matrice d'entrée de taille 8*8 présente le gain minimal par rapport aux autres tailles de matrice, dont la valeur est supérieure à 25%. Cependant, les techniques existantes n'assurent pas une amélioration pour tous les cas des matrices d'entrée. Par exemple, la technique progressive entraîne une augmentation du temps d'exécution par rapport au code initial du TWF pour le cas de la matrice d'entrée de taille 8*8. De même, la technique enchaînée n'assure qu'une amélioration de 9.20% uniquement. Ces résultats sont dus aux valeurs des fonctions de retiming multidimensionnel, qui engendre un décalage d'un nombre important d'itérations dont le code est exécuté en prologue et épilogue.

Par ailleurs, pour chaque taille de matrice d'entrée, les pourcentages d'évolution des temps d'exécution pour les deux cibles sont similaires. Ce critère est vérifié pour les trois techniques de retiming multidimensionnel. Par conséquent, la contribution de notre technique est garantie indépendamment de l'architecture parallèle utilisée.

4.9 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle technique du retiming multidimensionnel, permettant d'atteindre l'ordonnancement avec une période de cycle minimale sans atteindre un parallélisme total. La technique proposée génère des solutions en employant moins de fonctions de retiming multidimensionnel, tout en assurant l'utilisation des mêmes vecteurs optimaux. L'étude expérimentale nous a montré que notre technique génère des solutions avec des temps d'exécution et des tailles du code nettement inférieures à ceux générés par les techniques progressive et enchaînée. D'où, la stratégie du parallélisme proposée par la technique du retiming multidimensionnel décalé permet de respecter des contraintes de temps d'exécution plus inférieurs et de minimiser les ressources matérielles de l'implémentation.

Cependant, le processus l'optimisation des systèmes temps réel embarqués exige la prédiction de l'évolution des performances en fonction de la technique du parallélisme. Cette étape est indispensable, d'une part pour choisir les paramètres du parallélisme engendrant un résultat optimal, et d'autre part pour vérifier le respect des contraintes. De ce fait, il s'avère obligatoire d'estimer le temps d'exécution et la taille du code des implémentations générées par la technique du retiming multidimensionnel décalé, pour la faire recours lors de la conception des implémentations temps réel embarquées.

CHAPITRE 5 : IMPLÉMENTATION TEMPS RÉEL EMBARQUÉES DES GFDMs

5.1 Introduction

Les techniques de parallélisme sont généralement utilisées pour la conception des systèmes temps réel embarquées. Ces systèmes nécessitent une implémentation qui respecte les contraintes de temps d'exécution, tout en utilisant le minimum de ressources matérielles. Cet objectif consiste à explorer les différentes solutions offertes par les techniques d'optimisation dont le choix du parallélisme se base principalement sur les améliorations qu'elles proposent en matière de performance et ressources matérielles. De ce fait, une étape d'estimation s'avère indisponsable pour vérifier le respect des contraintes. Cependant, les techniques de parallélisme des GFDMs procèdent à augmenter progressivement le niveau de parallélisme, sans pour autant vérifier si les performances sont déjà atteintes. De plus, la taille du code augmente en fonction du parallélisme. Donc, le niveau de paralléisme achevé par ces techniques d'optimisation est inadéquat pour le cas des implémentations temps réel embarquées.

Dans ce chapitre, nous utilisons des démarches de parallélisme des GFDMs pour générer des implémentations qui respectent une contrainte de temps d'exécution tout en utilisant une taille minimale de code. En premier lieu, nous utilisons la technique du retiming multidimensionnel décalé pour la réalisation de cet objectif. Nous détaillons la théorie d'estimation du temps en fonction du parallélisme choisi par le retiming multidimensionnel décalé. Puis, nous décrivons une démarche d'optimisation qui applique itérativement le retiming multidimensionnel dans l'objectif d'atteindre la contrainte de conception. Une deuxième partie est consacrée à l'utilisation conjointe du parallélisme au niveau des instructions et du parallélisme au niveau des itérations, dont le choix des techniques de parallélisme est basé sur leurs apports en matière de temps d'exécution et de taille de code.

5.2 Respect de contrainte de temps d'exécution par retiming multidimensionnel décalé

5.2.1 Exemple de motivation

Le retiming multidimensionnel décalé applique un parallélisme au niveau des instructions du nid de boucles dans le but d'atteindre la période de cycle minimale [1, 2, 4, 10]. Ce parallélisme entraîne une diminution dans le temps d'exécution de l'implémentation, en dépit d'une augmentation de la taille du code. Vu que la technique augmente le niveau du parallélisme itérativement, le temps d'exécution de l'application diminue à plusieurs reprises. Cette augmentation du niveau de parallélisme se poursuit indépendamment de la valeur du temps d'exécution réalisée. Cependant, le code des boucles imbriquées augmente considérablement en fonction du niveau du parallélisme. Nous procédons à étudier l'évolution du temps d'exécution et de la taille du code en fonctions du retiming multidimensionnel pour le cas du filtre à Réponse Impulsionnelle Infinie (RII). Dans le cas où $t(M_x) = t(A_y) = 1$, la technique du retiming multidimensionnel décalée applique 4 transformations de retiming multidimensionnel pour générer l'implémentation finale du filtre. Après chaque transformation, nous générons le GFDM et nous déduisons le code dans le but de calculer son temps d'exécution et sa taille, dont les valeurs sont illustrées dans la courbe de la figure 5.1. L'implémentation ordonnancée avec la période d'horloge minimale est exécutée en 632 unités de temps, avec une taille de code de 402 instructions. Par conséquent, même si le GFDM ordonnancée avec la période d'horloge minimale respecte la contrainte de temps d'exécution, il nécessite des ressources matérielles importantes pour l'implémenter.

Nous proposons d'implémenter le filtre à RII en imposant une contrainte de temps d'exécution égale à 900 unités de temps. En se basant sur les valeurs présentées dans la figure

5.1, l'implémentation générée par la troisième transformation de retiming multidimensionnel requiert un temps d'exécution égal à 861. De plus, la taille du code est de 258 instructions représentant ainsi une amélioration de 35.82% par rapport à l'implémentation totalement parallèle. D'où, la solution générée après la troisième fonction de retiming respecte la contrainte du temps d'exécution et utilise une taille de code inférieure par rapport à l'implémentation ordonnancé avec la période de cycle minimale.

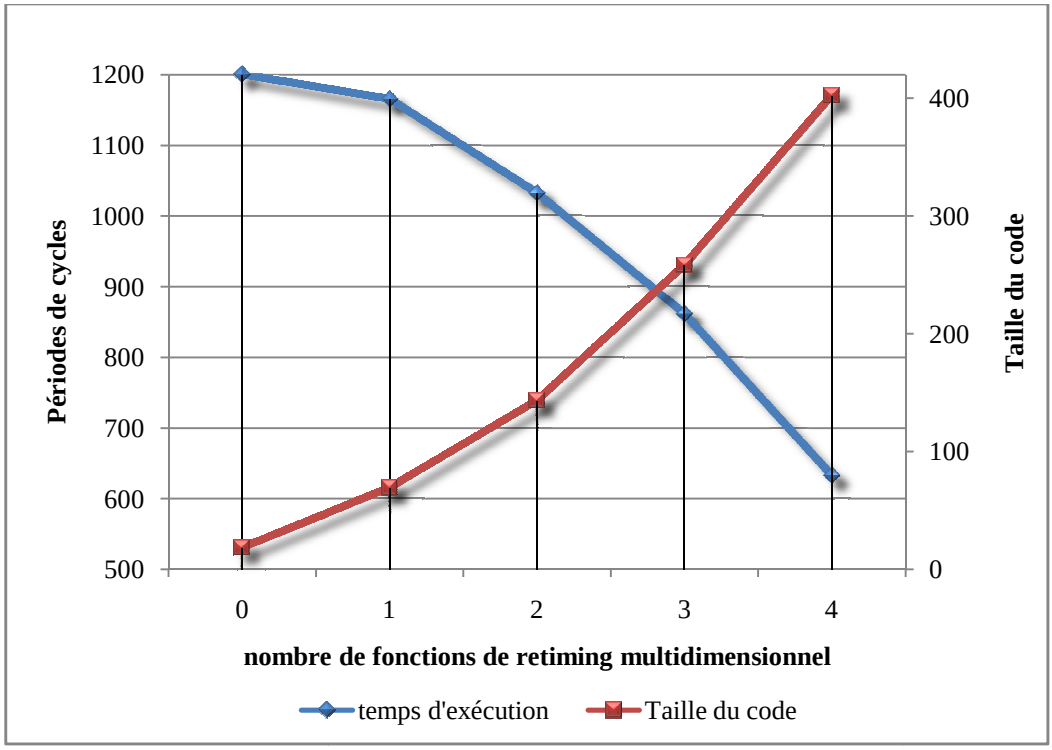


Figure 5.1 Evolution du temps d'exécution de la taille du code du fitre à RII en fonction du retiming multidimensionnel

5.2.2 Principes

Nous avons conclu de l'exemple précédent que, même si l'implémentation ordonnancée avec la période d'horloge minimale respecte la contrainte de temps d'exécution, elle nécessite des ressources matérielles importantes. Il s'avère donc bénéfique de se limiter au niveau de parallélisme permettant de respecter la contrainte du temps. Dans ce contexte, nous présentons une nouvelle approche d'optimisation permettant d'utiliser la technique de retiming multidimensionnel décalé pour déterminer le niveau du parallélisme minimale assurant le respect de la contrainte de temps d'exécution. De ce fait, la taille du code de l'implémentation résultat est inférieure ou égale à celle de l'implémentation ayant la période de cycle minimale. La démarche de cette approche procède à vérifier le temps d'exécution du GFDM après chaque fonction du retiming : si la contrainte n'est pas encore respectée, la fonction de retiming suivante est appliquée, jusqu'à atteindre la contrainte de temps d'exécution ou bien atteindre la période de cycle minimale, telle que modélisée dans le flot de la figure 5.2.

Ce travail consiste à prédire l'évolution du temps d'exécution en fonction des transformations du retiming multidimensionnel. De ce fait, notre approche procède à estimer le temps d'exécution à haut niveau de conception, à partir de la spécification algorithmique. Elle explore le GFDM et les paramètres de la fonction du retiming multidimensionnel dans l'objectif de prétendre la valeur du temps d'exécution. Nous décrivons dans le paragraphe 5.2.3 la théorie de l'estimation du temps d'exécution d'un GFDM après l'application du

retiming. Par la suite, nous présentons dans la partie 5.2.4 la démarche de notre approche d'optimisation en décrivant les algorithmes et leurs complexités. Même si le formalisme est décrit pour le cas des graphes bidimensionnels, il est directement extensible au cas général des boucles imbriquées. Nous tenons à signaler que le temps d'exécution est composé en deux parties : le temps de calcul et le temps de communication [38, 39]. Cette dernière correspond au temps nécessaire à l'ordonnancement des données et à l'accès mémoire, dont les valeurs dépendent de la technologie des processeurs et de l'hiérarchie de la mémoire. Dans ce travail, nous nous contentons de prendre en considération le temps de calcul.

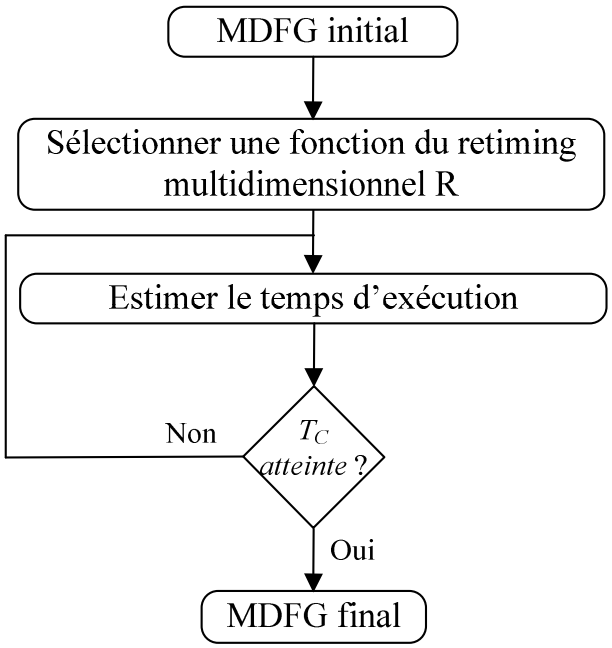


Figure 5.2 Flot de l'approche d'optimisation

5.2.3 Estimation du temps d'exécution

5.2.3.1 Temps d'exécution du MDFG uniforme

D'une façon générale, le temps d'exécution représente la multiplication de la période du cycle $C(G)$ et du nombre des cycles N_{CLK} nécessaire pour l'exécution de toute l'application. La période de cycle d'un GFDM est égale au temps d'exécution du chemin critique. De plus, chaque itération appartenant à une structure uniforme de boucles imbriquées est exécutée dans une seule période. De ce fait, pour le cas des nids de boucles uniformes, le nombre des périodes de cycles est égal à la multiplication des intervalles des itérateurs de toutes les boucles imbriquées, tel que formulé dans l'équation (5.1).

$$T = max\{t(p), d(p) = 0\} \times \prod_{b=1}^{b=n} (u_b - l_b) \quad (5.1)$$

Dont p est un chemin de données du GFDM, n est le nombre des boucles imbriquées, l_n et u_n sont respectivement la valeur minimale et maximale de l'itérateur de la boucle b.

Prenons l'exemple du GFDM schématisé dans la figure 5.3(b) dont le chemin critique est le suivant $p: A \rightarrow B \rightarrow C$. Dans le cas où $t(A) = t(B) = t(C) = 1$, la période du cycle est égale à 3 unités de temps. De plus, si les nombres des itérations de la boucle externe et interne sont respectivement 4 et 6, l'application nécessite 24 périodes de cycle, telle que schématisée dans l'espace d'itérations dans la figure 5.4 dont chaque cellule correspond à une itération.

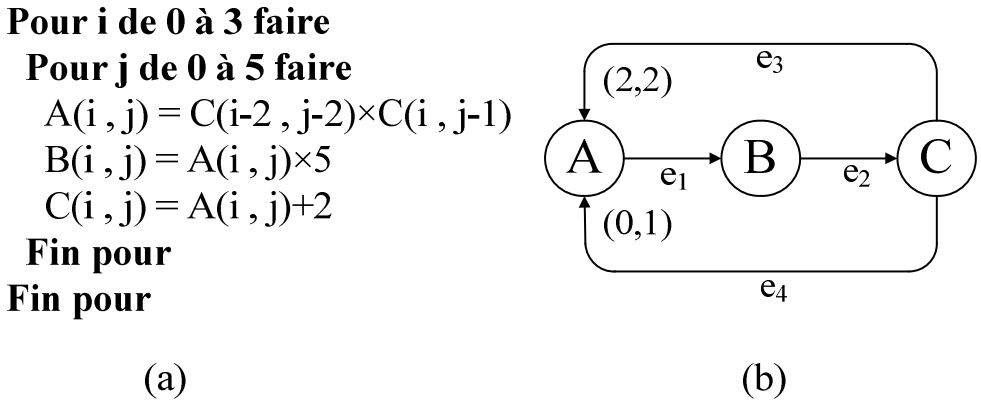


Figure 5.3 Nid de boucles : (a) l'algorithme, (b) le GFDM

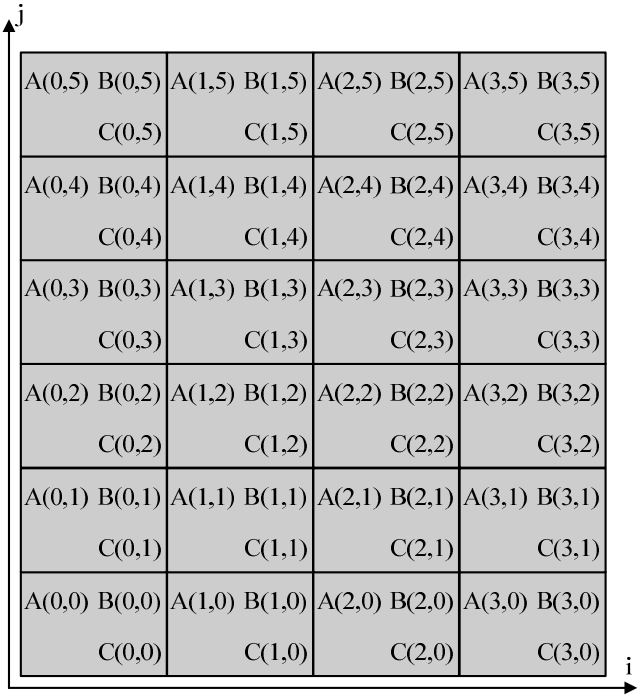


Figure 5.4 Espace d'itérations du GFD bi-dimensionnel

5.2.3.2 Temps d'exécution après une fonction du retiming multidimensionnel

Chaque transformation à base du retiming multidimensionnel affecte la valeur et le nombre des périodes de cycle. D'une part, le chemin critique est réduit après le retiming multidimensionnel, dont nous sommes dont l'obligation de déterminer la nouvelle valeur. Cette étape nécessite le parcours du graphe à partir des nœuds décalés jusqu'aux nœuds ayant tous arcs entrants de délais non nuls. Le GFDM après le retiming multidimensionnel $r(A) = (1, -1)$ est affiché dans la figure 5.5(a). Le chemin critique après retiming est $B \rightarrow C$ ayant un temps d'exécution égal à 2 unités de temps. D'autre part, le retiming multidimensionnel affecte le nombre des périodes de cycles correspondant aux itérations N_{loop} et ajoute des périodes de cycles $N_{pro-epi}$ pour l'exécution des prologues et des épilogues. De ce fait, pour calculer la valeur total du nombre de cycles N_{clk}, nous procédons à calculer indépendamment ces deux composantes, telles que indiquées dans l'équation (5.2).

$$N_{clk} = N_{loop} + N_{pro-epi} \quad (5.2)$$

Pour le N_{loop}, une fonction de retiming $r = (d.x, d.y)$ entraîne le décalage de $|d.x|$ itérations de la boucle externe en amont et en aval de sa structure. De ce fait, le nombre d'itérations de la boucle externe est réduit de $|d.x|$. De même, le terme $d.y$ entraîne la diminution des itérations de la boucle interne par $|d.y|$. Par conséquent, le nombre des cycles nécessaires pour l'exécution de la nouvelle structure des boucles est décrit dans l'équation (5.3).

$$N_{loop} = (n_1 - |d.x|) \times (n_2 - |d.y|) \quad (5.3)$$

dont $d.x$ et $d.y$ sont les termes de la fonction de retiming multidimensionnel tel que $r = (d.x, d.y)$, et dont n_1 et n_2 sont les nombres d'itérations respectivement de la boucle externe et de la boucle interne.

L'application du retiming multidimensionnel $r = (1, -1)$ sur le GFDM de la figure 5.3(b) résulte le GFDM de la figure 5.5(b). L'espace d'itérations correspondant est schématisé dans la figure 5.6, dont les instructions de la boucle interne sont modélisées par des cellules en gris. L'espace d'itérations nous montre que les nombres d'itérations de la boucle externe et interne décroissent respectivement de 6 à 5 et de 4 à 3.

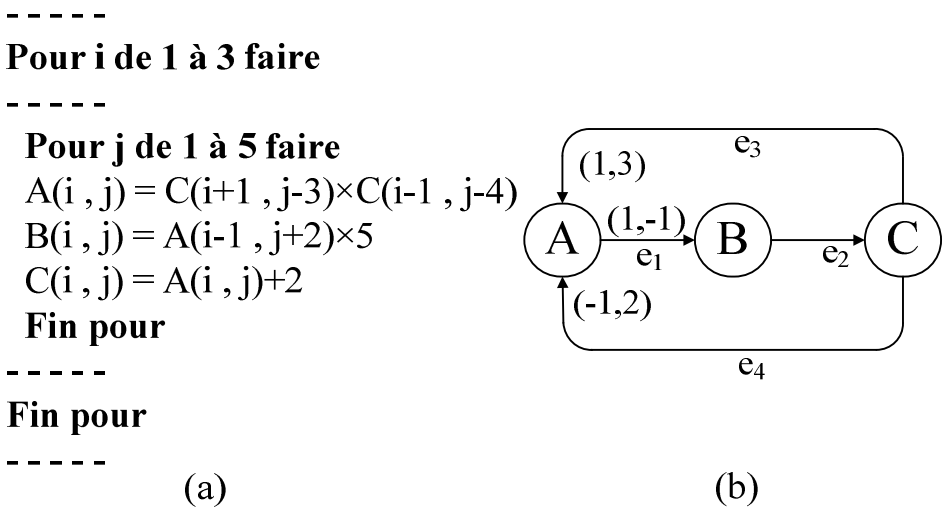


Figure 5.5 Nid de boucles après retiming multidimensionnel : (a) l'algorithme, (b) le GFDM

Pour le $N_{pro-epi}$, l'utilisation du retiming multidimensionnel $r(A) = (1, -1)$ implique l'exécution des nœuds A appartenant à toutes les itérations de la boucles internes $(0, j)$ et (i, n_2) en dehors de la structure des boucles, dont $0 \leq j \leq n_2$ et $0 \leq i \leq n_1$. Les nœuds A appartenant au prologue sont illustrés dans des carrés blancs dans l'espace d'itérations de la figure 5.6. Chaque cellule de l'espace d'itérations est exécutée en une période de cycle, ce qui implique que les instructions du prologue nécessitant $(n_2 + n_1 - 1)$ périodes de cycle. De même, les nœuds B et C appartenant initialement aux itérations $(i, 0)$ et (n_1, j) sont exécutés en dehors des itérations de la boucle interne, représentant ainsi l'épilogue. De ce fait, le prologue et l'épilogue nécessitent le même nombre de cycles pour leurs exécutions. Nous indiquons que les cellules d'indice $(n_1 + 1, 0)$ et $(0, n_2 + 1)$ appartiennent à la fois au prologue et au épilogue. Dans le cas général, le nombre de cycles nécessaires pour l'exécution du prologue et de l'épilogue est défini dans l'équation (5.4).

$$N_{pro-epi} = 2 \times \big((|d.x| \times n_2) + (|d.y| \times n_1) - (|d.x| \times |d.y|) \big) \quad (5.4)$$

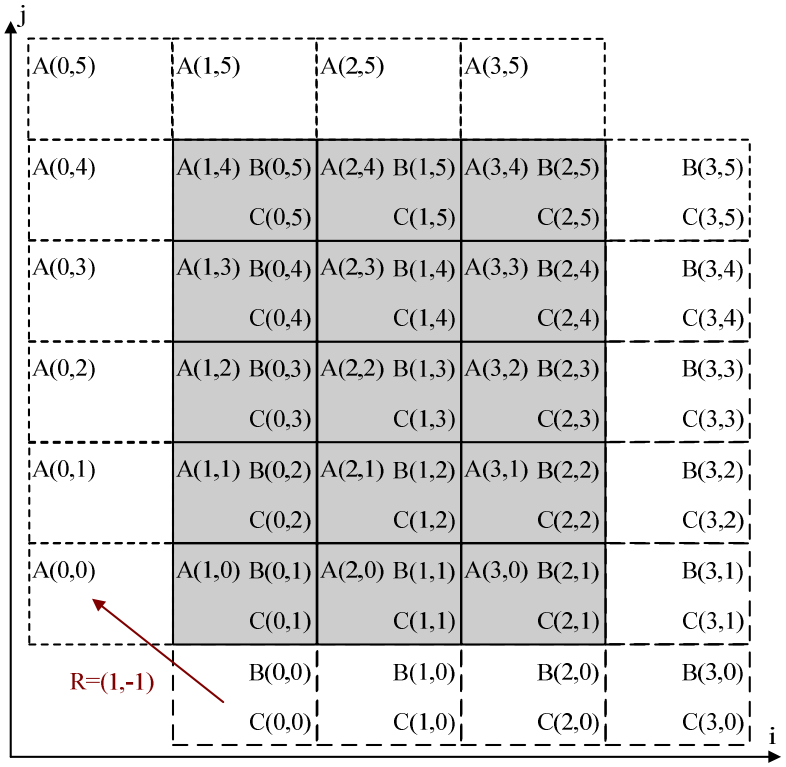


Figure 5.6 Espace d'itérations du GFDM après r(A)=(1,-1)

5.2.3.3 Temps d'exécution après plusieurs retiming multidimensionnel

Nous décrivons dans cette section une démarche pour l'estimation du temps d'exécution du GFDM final, dont la structure est atteinte après l'application itérative d'un ensemble de fonctions de retiming multidimensionnel. Prenons l'exemple du nid de boucles de la figure 5.3(a). Le GFDM final généré par la technique de retiming multidimensionnel décalé nécessite deux fonctions de retiming qui sont respectivement $r(A) = (2, -2)$ et $r(B) = (1 - 1)$. L'espace d'itérations correspondant à cette solution est représenté dans la figure 5.7. Chaque retiming multidimensionnel modifie le nombre des cycles du nid de boucles N_{loop} et le nombre des cycles du prologue et d'épilogue $N_{pro-epi}$. Les fonctions du retiming sont des multiples du vecteur r. De ce fait, le décalage des instructions est effectué dans le même sens, ce qui résulte des instructions de prologue et d'épilogue partiellement superposées. A partir de l'espace d'itérations de la figure 5.7, nous déduisons que les instructions $A(1,4)$ et $B(0,5)$ appartiennent à la même cellule, et ainsi exécutées dans la même période de cycle. Par conséquent, le nombre de cycles total ne peut pas être le cumul des nombres de cycle des prologues et des épilogues de chaque retiming.

En fait, le nombre des cycles est égal au nombre N_{CLK} des cellules utilisées de l'espace d'itérations. Ce nombre peut être calculé en déterminant la dimension totale de l'espace d'itérations N_{dim} et en soustrayant le nombre des cellules vacantes N_v, tel que définie dans l'équation 5.5. Nous procédons donc à calculer le N_{dim} et le N_v itérativement après chaque fonction de retiming.

$$N_{CLK} = N_{dim} - N_v \quad (5.5)$$

La dimension de l'espace d'itérations représente la taille initiale en ajoutant l'extension due aux fonctions du retiming multidimensionnel, telle que indiquée dans l'équation 5.6.

$$N_{CS} = (m, n) + n_{MDR} \times (|d.x|, |d.y|) \quad (5.6)$$

Dont m et n sont les nombres d'itérations respectivement de la boucle externe et la boucle interne, n_{MDR} est le nombre des fonctions de retiming et (r_x, r_y) est la fonction de retiming.

Le nombre des cellules vacantes N_v dépend de la fonction du retiming multidimensionnel r et de son nombre applications. Prenons l'exemple de l'espace d'itérations de la figure 5.7. Les cellules vacantes sont dues à la première et la deuxième fonction du retiming. Le cadre bleu englobe les cellules correspondantes au prologue et épilogue de la première fonction. Le deuxième cadre rouge contient les codes du prologue et d'épilogue due à la deuxième fonction. Le nombre des cellules correspondante au prologue et à l'épilogue est à déterminer directement par la multiplication des nombres des itérations et les valeurs des indexes de la fonction de retiming. La formule générale calculant le nombre N_v indépendamment du rang de la fonction de retiming est indiqué dans l'équation (5.7).

$$N_v = N_{dimX} \times N_{dimY} - (N_{dimX} - X_{PredIS}) \times (N_{dimY} - n)$$
$$+(N_{dimY} - Y_{PredIS}) \times (N_{dimX} - m) \quad (5.7)$$

Dont N_{dimX} et N_{dimY} sont la largeur et la hauteur de l'espace d'itérations, X_{PredIS} et Y_{PredIS} sont la largeur et la hauteur de l'espace d'itérations précédent, et m et n sont les nombres des itérations respectivement de la boucle externe et la boucle interne.

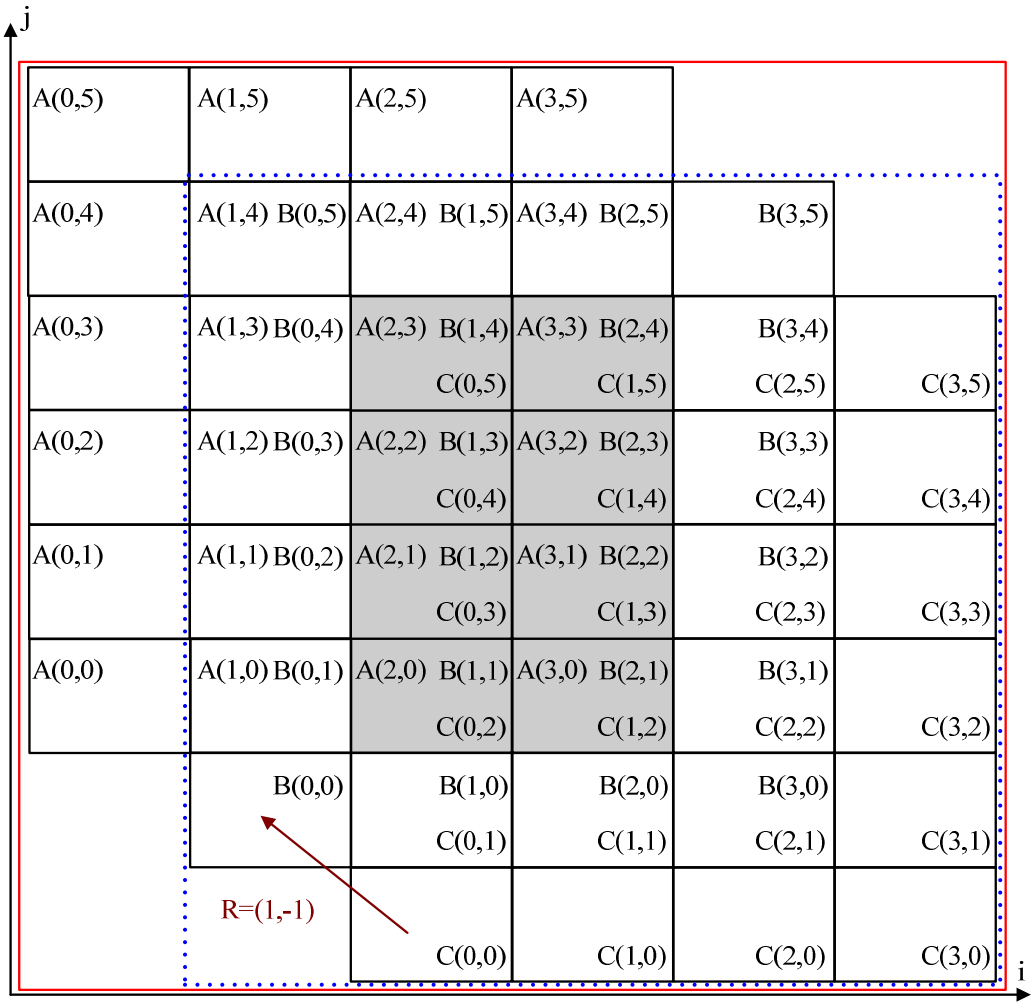


Figure 5.7 Espace d'itérations après r(A)=(2,-2) et r(B)=(1,-1)

5.2.4 L'algorithme de l'approche d'optimisation

Ce travail vise à générer une implémentation qui respecte une contrainte de temps d'exécution. De ce fait, notre approche définit le temps d'exécution initial dans le but de le comparer à la contrainte. Si cette valeur est supérieur à T_C, l'approche procède à sélectionner une fonction de retiming multidimensionnel R et l'appliquer au GFDM. Ensuite, elle estime le temps d'exécution correspondant à la fonction R. Cette étape est répété jusqu'à atteindre la contrainte de temps d'exécution T_C ou que le GFDM est totalement parallèle. Le flot de notre approche d'optimisation est décrit dans l'algorithme 5.1.

Algorithme 5-1 Respect de la contrainte de temps d'exécution par retiming multidimensionnel décalé

Entrée : GFDM $G = (V, E, d, t)$, contrainte de temps d'exécution T_C, les compteurs de la boucle externe m et de la boucle interne n
Sortie : GFDM respectant la contrainte

0: **Début**
/ calcul du temps d'exécution T et la période de cycle minimale C_{min} */*
1: Extraire les chemins critiques P_{ch}
2: Calculer la période de cycle courante $C = max\{t(p), p \in P_{ch}\}$
3: $N_{Clk} \leftarrow \prod_{i=1}^{i=n} f_i$
4: $T \leftarrow C \times N_C$
5: Sélectionner une fonction de retiming multidimensionnel $R = (x, y)$
6: Déterminer le nombre maximal n_{max} des fonctions de retiming multidimensionnel
7: $i \leftarrow 0, N_{VT} \leftarrow 0$
8: $OB = (m, n)$
9: $PredIS = (XPredIS, YPredIS)$
10: $Ndim = (NdimX, NdimY)$
/ appliquer itérativement le retiming multidimensionnel */*
11: **Tant que** $(T > T_C)$ et $(C > C_{min})$ **faire**
12: $i \leftarrow i + 1$
13: Calculer la valeur de N_v tel que l'équation 3
14: $N_{VT} = N_{VT} + N_v$
15: $N \leftarrow nx * ny - N_{VT}$
16: $\mathrm{N_{Clk_R}} \leftarrow (\mathrm{nx}, \mathrm{ny}) - \mathrm{N}$
17: $\mathrm{T} \leftarrow \mathrm{N_{Clk_R}} \times \mathrm{Clk_R}$
18: **Fin tant que**
19: Fin

5.3 Respect de contrainte de temps d'exécution et utilisation minimale de code par retiming multidimensionnel décalé et le loop striping

5.3.1 Contexte

Notre travail s'intéresse à l'optimisation du temps d'exécution des nids de boucles. Dans cet objectif, le principe d'optimisation consiste à augmenter le parallélisme du traitement. Cette transformation entraîne la réduction du temps d'exécution en dépit d'une augmentation de la taille du code. En fait, le retiming multidimensionnel n'exploite que le parallélisme au niveau des instructions. Ce parallélisme est limité par l'atteinte de la période de cycle minimale. Cet objectif ne correspond pas à l'atteinte du temps d'exécution optimal. Prenons le cas du filtre à RII, le temps d'exécution minimal, que le retiming multidimensionnel décalé

peut atteindre, est d'une valeur de 623 unités de temps. Cette technique est incapable d'atteindre une contrainte de temps plus inférieure. Par ailleurs, plus la démarche d'optimisation réduit le temps d'exécution, plus le code de l'implémentation augmente. Malgré que la technique décalée permette de réduire la taille du code par rapport aux techniques de retiming multidimensionnel existantes, cette taille persiste à être toujours importante pour le cas des implémentations avec un niveau de parallélisme élevé. A titre d'exemple, le code de l'implémentation du filtre à RII avec la période de cycle minimale est d'une taille égale à 258 instructions. Dans le cas du parallélisme au niveau des itérations des GFDMs, les travaux de [5] décrivent une technique, intitulée « Loop Striping » permettant de sélectionner les itérations ayant des traitements indépendants dans le but de les exécuter en parallèle. Autant qu'une technique de parallélisme, le loop striping présente les mêmes inconvénients que le retiming multidimensionnel décalé : une contrainte de temps d'exécution n'est pas toujours atteinte ; De plus, elle génère des implémentations avec des tailles de codes importantes.

Cependant, plusieurs travaux ont décrit des démarches de parallélisme pour le respect de la contrainte du temps d'exécution [38, 39]. Ces démarches permettent d'exploiter les granularités du parallélisme que ce soit au niveau des instructions qu'au niveau des itérations. De ce fait, ils génèrent des solutions plus performantes que celles générées par le parallélisme à un seul niveau. De plus, ils permettent d'atteindre la contrainte en utilisant le moins de taille de code. Nous distinguons des démarches d'optimisation qui sont appliquées à des graphes de flot de données permettant le parallélisme d'une seule structure itérative [5, 8, 10, 31]. D'autres démarches de parallélisme ont été proposées à travers toutes les boucles du nid modélisées par la structure polyédrique [105, 3]. Pour le parallélisme du GFDM, le seul travail combinant le parallélisme au niveau des instructions et le parallélisme au niveau des itérations pour les GFDM vise à réduire le temps d'exécution moyen d'une itération. Ce travail ne prend pas en considération le temps d'exécution du aux prologues et aux épilogues. De ce fait, l'évolution du temps moyen d'une itération n'est pas similaire à l'évolution du temps d'exécution [35].

De ce fait, nous proposons dans cette partie une nouvelle approche d'optimisation permettant de combiner l'utilisation des techniques du retiming multidimensionnel décalé et du loop striping. Elle permet d'explorer l'espace des solutions proposées par les deux techniques, dans le but d'atteindre la contrainte de temps d'exécution tout en utilisant le minimum de code. Cette approche permettra d'atteindre des contraintes que chacune des techniques appliquées séparément ne le permet. De plus, même si la contrainte est atteinte, elle permet de générer une solution avec une taille de code inférieure à celles des codes générés par chacune des techniques. Dans ce contexte, nous présentons la théorie nécessaire pour la combinaison des deux techniques. Ensuite, nous proposons des algorithmes efficaces assurant la sélection du parallélisme en se basant sur l'évolution du temps d'exécution et de la taille du code.

5.3.2 Principes

Notre travail s'intéresse à la conception des systèmes temps réel embarqués, nécessitant le respect d'une contrainte de temps d'exécution. Nous prenons l'exemple du GFDM de la figure 4.1(a) dont nous visons à l'exécuter dans la limite de 550 unités de temps. Le retiming multidimensionnel décalé génère la solution finale qui s'exécute en 924 unités de temps, telle que schématisée dans la figure 4.3. Cette technique est incapable de respecter la contrainte, telle que soit la fonction du retiming choisie. La technique de loop striping incrémente le facteur $\boldsymbol{f}$ dans l'objectif de réduire le temps d'exécution de l'application. En respectant la valeur de l'offset $\boldsymbol{g} = \mathbf{2}$, cette technique doit collecter 5 itérations dans le même groupe pour

atteindre la contrainte du temps d'exécution. Cependant, la boucle interne demeure contenir 20 instructions. De plus, les itérations qui ne sont pas regroupées, sont intégrées en mont et en aval de la boucle interne autant que prologue et épilogue. Par conséquent, même si la technique du loop striping respecte la contrainte de temps, elle entraîne une augmentation de 8 fois de la taille du code.

En fait, le temps d'exécution est défini comme étant la multiplication de la période de cycle et le nombre des cycles. Donc, la réduction du temps d'exécution est similaire à la réduction de l'un de ces deux termes oubien la réduction des deux termes en même temps. Prenons le cas du GFDM après retiming multidimensionnel dont le GDC est affiché dans la figure 5.3(b). Nous constatons que les itérations $(1,0)$ and $(0,3)$ n'ont aucune dépendance de données en elles. Cette condition est vérifiée pour tous pairs d'itérations: $(i+1, j)$ et $(i, j+3)$, tel que $0 \leq i \leq n-3$ et $0 \leq j \leq m-1$. Nous procédons à appliquer le loop striping $LS(f=2, g=3)$ au GFDM après retiming dont les itérations sont collectées dans des partitions telles que affichées dans le l'espace d'itérations de la figure 5.8(a). Cette transformation implique la duplication des itérations de la boucle interne, et l'ajout du prologue et d'épilogue dont chacun contient 3 itérations de la boucle interne initial. Le GFDM généré dans la figure 5.9(a) est un graphe réalisable, dont l'exécution peut être ordonnancée en suivant le vecteur $s = (4,1)$. L'algorithme correspondant est structuré de 32 instructions tel que affiché dans la figure 5.9(b).

En se basant sur l'ordonnancement statique de la figure 5.8(b), l'algorithme nécessite 250 cycles pour être exécuté ; cependant, le période de cycle est toujours égale à $C(G) = 2$, d'où l'application est exécutée dans 500 unités de temps. Nous déduisons donc que l'application du retiming multidimensionnel décalé et le loop striping permet d'atteindre la contrainte de temps d'exécution. Ce résultat ne peut jamais être atteint par le retiming multidimensionnel décalé. De plus, l'implémentation présente un gain en matière de taille de code par rapport au loop striping d'une pourcentage de 41.06%.

Dans ce contexte, nous proposons une approche d'optimisation permettant d'utiliser les deux techniques de retiming multidimensionnel décalé et de loop striping, dont l'objectif d'atteindre la contrainte du temps d'exécution tout en utilisant le minimum de la taille du code. En fait, les deux techniques entraînent la même évolution en matière du temps d'exécution et de la taille du code. D'après l'étude bibliographique effectuée, il n'existe aucun travail affirmant qu'une des techniques est considérée autant qu'optimale par rapport à l'autre. Pour cela, notre approche d'optimisation assure l'exploitation de l'espace de solutions proposées par les deux techniques dans l'objectif de sélectionner celle offrant la meilleur performance. Nous décrivons dans les paragraphes suivants la théorie nécessaire pour le choix du niveau du parallélisme parmi ceux proposés par la technique de retiming multidimensionnel décalé et le loop striping. Même si le formalisme correspond au graphe flot de données bi-dimensionnel, il est directement extensible pour le cas général.

5.3.3 Ordre d'utilisation des techniques de parallélisme

Nous définissons dans ce paragraphe la théorie de l'application conjointe du retiming multidimensionnel décalé et du loop striping. Dans ce contexte, nous procédons à étudier explicitement la structure du GFDM avant et après l'application de chaque technique. En fait, le loop striping engendre la duplication des itérations, et ainsi la modification de la structure des boucles du GFDM. Nous introduisons dans le théorème 5.1 la structure du GFDM après le loop striping.

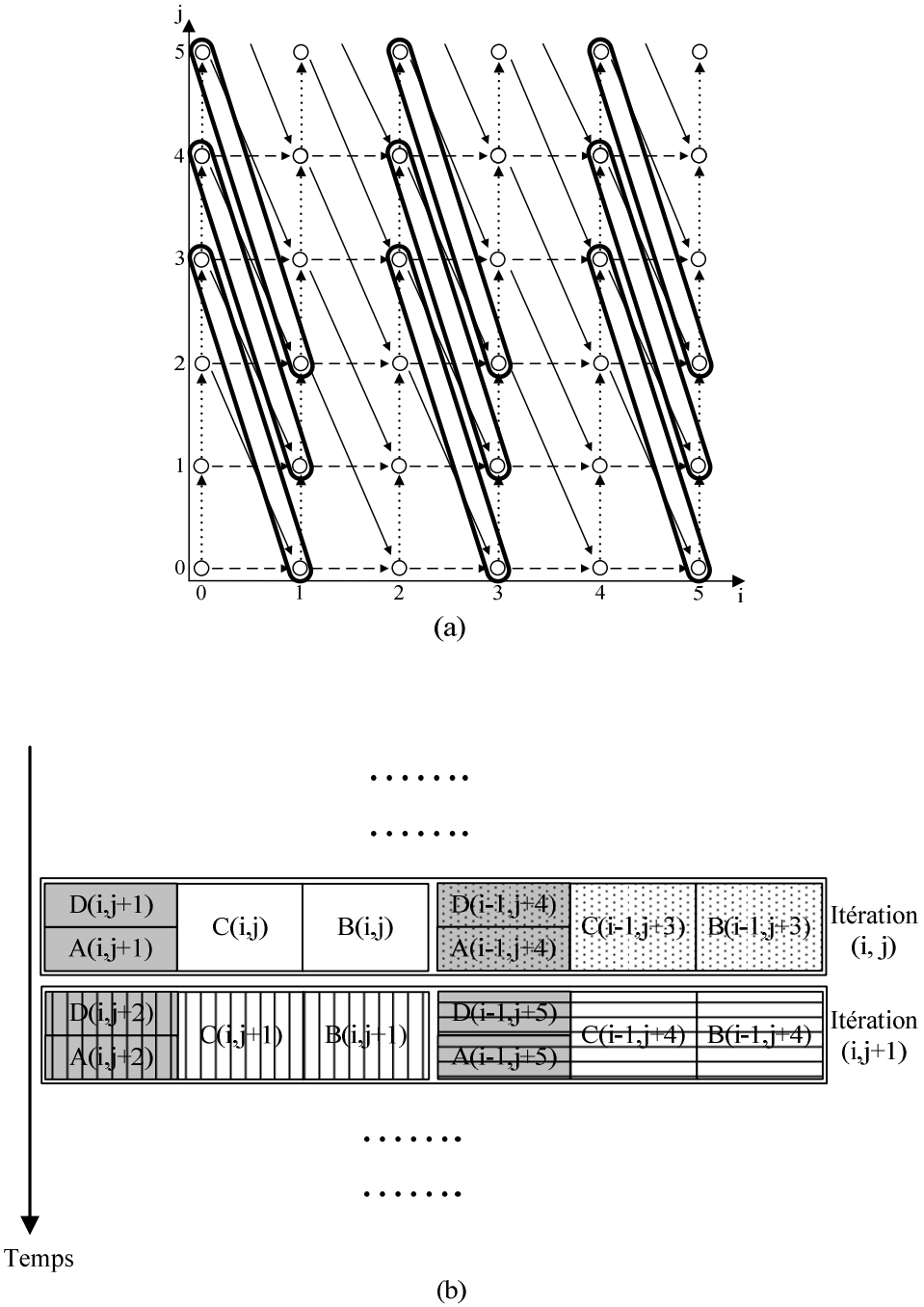
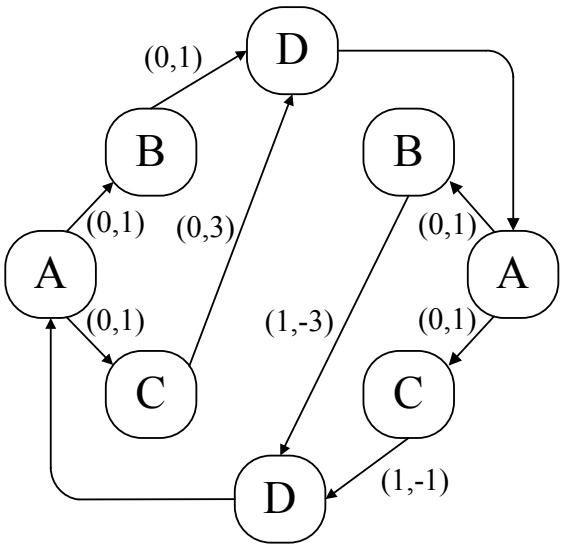


Figure 5.8 Filtre numérique d'ondelettes après retiming multidimensionnel et loop striping: (a) le GDC, (b) l'ordonnancement statique

Théorème 5.1 *soit un MDFG* $G = (V, E, d, t)$ *et* g *est un offset de* G. *Si* $g \neq 0$ *alors le GFDM après loop striping représente une structure de boucles imbriquées non-uniforme.*

Preuve. $g \neq 0$ *signifie que les iterations (1,0) et (0,g) composent la première partition. Les itérations* (i, j) *dont* $i < 1$ *et* $j < g - 1$ *doivent être exécutées avant les itérations regroupées. D'où,* $g \geq 1$ *implique l'ajout de boucle en amont de la boucle interne dans l'objectif d'exécuter les iterations (i,j). Par conséquence, le GFDM après loop striping est de structure de boucles imbriquées non-uniforme.*



(a)

Pour i de 0 m de pas 2 faire
D(i,0) = B(i-1 , 1) × C(i-1 , -1)
A(i,0) = D(i,0) × 5
D(i+1,0) = B(i , 1) × C(i , -1)
A(i+1,0) = D(i+1 ,0) × 5
Pour j de 0 à g-1 faire
D(i,j+1) = B(i-1 , j+2) × C(i-1 , j)
A(i,j+1) = D(i,j+1) × 5
B(i,j) = A(i,j) + 1
C(i,j) = A(i,j) + 2
Fin pour
Prologue
Pour j de g à n-1 faire
D(i,j+1) = B(i-1 , j+2) × C(i-1 , j)
A(i,j+1) = D(i,j+1) × 5
B(i,j) = A(i,j) + 1
C(i,j) = A(i,j) + 2
D(i+1,j-2) = B(i , j-1) × C(i , j-3)
A(i+1,j-2) = D(i+1,j-2) × 5
B(i+1,j-3) = A(i+1,j-3) + 1
C(i+1,j-3) = A(i+1,j-3) + 2
Fin pour
pour j de n-g-1 à n-1 faire
D(i+1,j-2) = B(i , j-1) × C(i , j-3)
A(i+1,j-2) = D(i+1,j-2) × 5
B(i+1,j-3) = A(i+1,j-3) + 1
C(i+1,j-3) = A(i+1,j-3) + 2
Fin pour
B(i,n) = A(i,n) + 1
C(i,n) = A(i,n) + 2
B(i+1 ,n) = A(i+1 ,n) + 1
C(i+1 ,n) = A(i+1 ,n) + 2
Epilogue
Fin pour

(b)

Figure 5.9 Filtre numérique d'ondelettes après retiming multidimensionnel et loop striping: (a) le GFDM, (b) l'algorithme

En se référant à l'exemple de la figure 5.8(b), le loop striping implique la duplication 3 fois de la boucle interne. Donc, pour le cas d'un GFDM généré par la technique du loop striping, un retiming multidimensionnel se limite à optimiser uniquement la boucle interne, ce qui minimise l'apport de son parallélisme.

Contrairement au loop striping, le retiming multidimensionnel préserve la structure des boucles imbriquées du GFDM initial. Ce critère est assuré telle que soit la fonction du retiming choisie. La figure 4.3 montre que l'algorithme après retiming multidimensionnel contient les mêmes structures des boucles, même après l'ajout des codes en amont et en aval des boucles. Nous étudions dans le théorème 5.2 la possibilité d'appliquer le loop striping après le retiming multidimensionnel.

Théorème 5.2 *soit GFDM* $G = (V, E, d, t)$*, un retiming multidimensionnel légal* R *de* G *et* G_R *le résultat du retiming de* G *par* R*. Si* g *est un offset de* G *alors il existe un offset* g' *de* G_R*.*

Preuve. g *est un offset de* G *qui signifie qu'il existe* $b = (x, 1)$ *et que* $b \times d > 0$ *pour tout* $d \in E$*. Cette condition prouve que* b *est un vecteur d'ordonnancement de* G*, et ainsi* b *et* R *sont orthogonaux [1].* R *est un retiming légal de* G*, donc il existe un vecteur d'ordonnancement* $s = (x, y)$ *tel que* $d_R \times s > 0$ *pour tout* $d_R \in E_R$*, dont* $G_R = (V, E, d_R, t)$*, ce qui vérifie l'existence d'un offset* g' *de* G_R*.*

Ce théorème prouve que le loop striping peut être appliquer à un GFDM déjà parallélisé par le retiming multidimensionnel r, tel que soit les valeurs des indices de la fonction r. En se basant sur les théorèmes 5.1 et 5.2, les solutions peuvent être générées en appliquant le loop striping, le retiming multidimensionnel ou les deux ensembles. En fait, le retiming multidimensionnel décalé applique plusieurs fonctions de retiming, pour atteindre la période de cycle minimale. En ce basant sur le théorème 5.2, le loop striping peut être appliqué sur un GFDM tel que soit le nombre de fonctions de retiming multidimensionnel y appliquées. Cette notion est introduite dans le lemme suivant.

Lemme 3.1 soit un GFDM $G = (V, E, d, t)$*, un retiming multidimensionnel légal* $R = (i, j)$ *de* G *et un GFDM* G_{DR} *après le retiming multidimensionnel décalé en utilisant* R*. Si* g *est un offset de* G*, alors il existe un offset* g_{DR} *de* G_{DR}*.*

Preuve. *Le retiming multidimensionnel décalé applique n fois la fonction* r *au même GFDM jusqu'à atteindre la période de cycle minimale. Donc, le lemme 5.1 est vérifié directement par la répétition n fois la vérification du théorème 5.2.*

Le lemme 5.1 offre plusieurs cas de retiming multidimensionnel, du premier GFDM généré après une seule fonction de retiming, à celui généré après l'ensemble des fonctions assurant l'atteinte la période de cycle minimale. Le loop striping peut être appliqué à chacun de ces GFDMs. D'où, l'espace des solutions des parallélisme possibles sont celles générées après le loop striping, après chaque fonction du retiming multidimensionnel, ou bien après chaque fonction du retiming multidimensionnel suivie par un loop striping.

Donc, l'approche d'optimisation sélectionne les paramètres de chaque technique. Ensuite, elle procède à choisir celle proposant le meilleur gain tel que décrit dans le paragraphe 5.3.4. Cette optimisation choisie est appliquer sur le GFDM courant. Si la contrainte n'est pas encore atteinte, l'approche reprend les étapes décrites itérativement, telles que illustrées dans le flot de l'approche de la figure 5.10.

5.3.4 Choix des techniques de parallélisme

Ce travail vise à générer une implémentation qui respecte la contrainte du temps d'exécution tout en utilisant le minimum de taille du code. Donc, le choix d'un parallélisme est basé sur son apport en matière de ces deux caractéristiques. Pour cela, après la sélection des paramètres d'optimisation de chaque technique, qui sont la fonction r pour le cas du retiming multidimensionnel et le facteur f et l'offset g pour le loop striping, l'approche procède à prédire l'amélioration du temps d'exécution et l'augmentation de la taille du code due à chaque technique. Ces valeurs prédites sont utilisées pour le choix du niveau parallélisme à appliquer entre le retiming multidimensionnel et le loop striping.

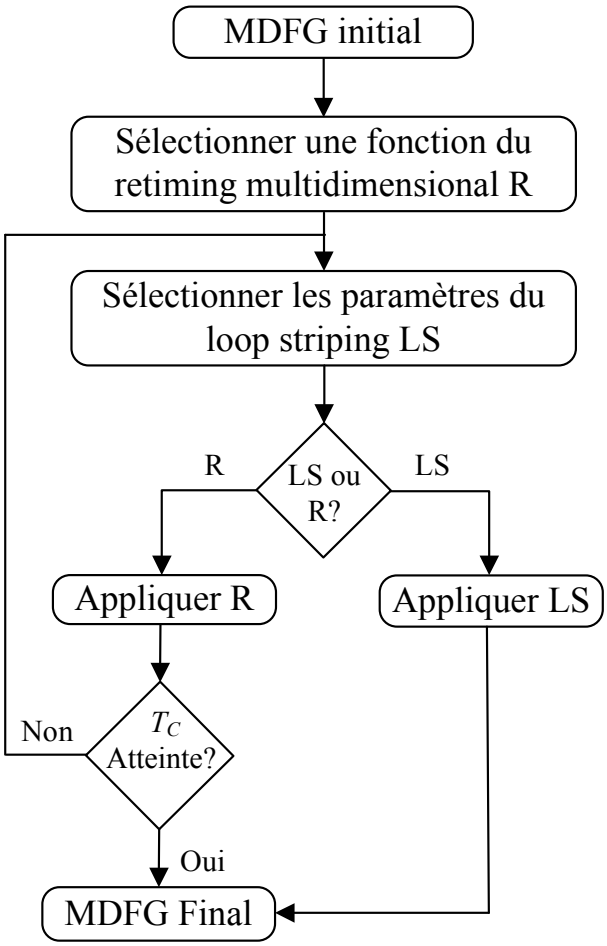


Figure 5.10 Flot de l'approche d'optimisation

En premier lieu, le choix d'un parallélisme entraînant une réduction du temps d'exécution peut conduire à une implémentation avec une taille de code importante. Pour le cas de l'optimisation décrite dans le paragraphe 5.3.1, le loop striping atteint la contrainte du temps d'exécution dont le retiming multidimensionnel est incapable de l'atteindre. Par contre, le code de l'implémentation générée par le loop striping est plus important que celle générée par l'application des deux techniques.

En deuxième lieu, l'approche ne peut pas choisir un parallélisme en se basant sur sa taille minimale, vue qu'une telle modification peut impliquer une amélioration négligeable en matière du temps d'exécution. Pour cela, le choix doit être basé à la fois sur le temps d'exécution et la taille du code. Dans ce contexte, nous proposons l'utilisation d'une fonction de coût telle que utilisée dans [36], dont l'équation est décrite dans 5.8.

$$f_{coût} = \Delta C / \Delta T \quad (5.8)$$

Dont ΔC représente l'augmentation de la taille du code et ΔT représente la réduction du temps d'exécution.

Une fonction de coût est calculée pour chaque parallélisme parmi le retiming multidimensionnel et le loop striping. Par la suite, notre approche choisit le parallélisme présentant le coût minimal afin de l'appliquer au GFDM courant. Les étapes de notre approche sont décrites dans l'algorithme 5.2. La démarche commence par la prédiction du

temps d'exécution T du GFDM et la période de cycle minimale. Par la suite, elle exécute une structure itérative qui sélectionne les paramètres du loop striping $LS = (f, g)$ et du retiming multidimensionnel $r = (d.x, d.y)$. Après, elle calcule le coût de chacune des techniques afin de choisir celle qui présente le coût minimale. Ces étapes sont répétées itérativement jusqu'à atteindre la contrainte du temps d'exécution ou jusqu'à atteindre la période minimale de cycle. Dans ce dernier cas, l'approche succède par l'application du loop striping.

Algorithme 5-2 Respect de la contrainte de temps d'exécution et utilisation de code de taille minimale par retiming multidimensionnel décalé et loop striping

Entrée : GFDM $G = (V, E, d, t)$, contrainte de temps d'exécution T_C, compteur de la boucle externe m et la boucle interne n
Sortie : GFDM respectant la contrainte T_C
Début
/ Calculer le temps d'exécution T et la période de cycle minimale C_{min} */*
Calculer la période de cycle $C = max\{t(p), d(p) = 0\}$
$N_C \leftarrow m \times n$
$T \leftarrow C \times N_C$
Calculer la période de cycle minimale $C_{min} = max\{t(v), v \in V\}$
/ Choisir la technique d'optimisation */*
Si $(T > T_C)$ alors
  Sélectionner une fonction de retiming multidimensionnel $R = (x, y)$
Tant que $(T > T_C)$ **faire**
  Calculer le coût du retiming f_R (tel que décrit dans l'algorithme 2)
  Sélectionner un loop striping $LS = (f, g)$
  Calculer le coût du loop striping f_{ls} (tel que décrit dans l'algorithme 3)
  Si $(C > C_{min})$ **et** $(f_R < f_{LS})$ **alors**
    Appliquer le retiming multidimensionnel R
    $T \leftarrow T_R$
  Sinon
    Appliquer le loop striping LS
    $T \leftarrow T_{LS}$
  Fin si
Fin tant que
Fin si
Fin

5.3.5 Calcul des fonctions du coût

5.3.5.1 Principes du calcul des fonctions du coût

Nous décrivons dans cette section notre démarche pour la prédiction des coûts du parallélisme. Le premier terme qui correspond à la réduction du temps d'exécution ΔT, représente la différence entre le temps d'exécution avant et après le parallélisme. Admettant que les deux valeurs du temps sont utilisées dans la démarche, elles doivent être prédites pour chaque technique du parallélisme. Pour le second terme, qui est l'augmentation de la taille du code ΔC, l'objectif de l'optimisation ne consiste pas à achever une valeur de taille de code bien définie. De plus, la taille du code ajouté en prologue et épilogue peut être calculée en fonction des paramètres du parallélisme que ce soit dans le cas du retiming multidimensionnel ou dans le cas du loop striping. Pour cela, nous procédons à prédire directement la différence du code ΔC à partir des paramètres de chaque technique.

5.3.5.2 Algorithmes des fonctions de coût

Dans le cas du retiming multidimensionnel, l'optimisation modifie la période de cycle, le nombre des cycles et la taille du code, dont elles doivent être tous prédits. Le nombre de cycles est calculé tel que décrit dans la fonction 5.4. La taille du code ajouté est proportionnelle à la différence entre le nombre des cycles ajoutés, dont la première moitié contient les instructions de prologue et la deuxième contient les instructions d'épilogue. Le coût du retiming multidimensionnel est calculé tel que décrit dans l'algorithme 5.3.

Algorithme 5-3 Calcul du coût du retiming multidimensionnel décalé

Entrée : GFDM $G = (V, E, d, t)$, function de retiming $R = (x, y)$, le nombre des périodes de cycle N_C
Sortie : coût du retiming multidimensionnel f_R
0: **Début**
1: Définir les chemins des données à décaler
/ Calculer la période de cycle après le retiming multidimensionnel */*
2: Extraire les chemins critiques P_{ch} aprés retiming
3: Calculer la période du cycle après retiming $Clk_R = max\{t(p), d(p) = 0\}$ avec $p \in P_{ch}$
4: Calculer la taille du code de la boucle interne C_{in}
/ Calculer le nombre des periods de cycles et la taille du code ajoutée après le retiming multidimensionnel */*
5: **Si** $((x = 0)\ et\ (y \neq 0))$ **alors**
6: $N_{Clk_R} \leftarrow (N_{Clk} \times (n + |y|)\ ) / n$
7: $\Delta C \leftarrow C_{in} \times |y|$
8: **Sinon**
9: $N_{Clk_R} \leftarrow 2 \times |x| + (N_{Clk} \times (m + |x|) \times (n + |y|))/(m \times n)$
10: $\Delta C \leftarrow C_{in} \times (|x| + |y|)$
11: **Fin si**
/ Calculer la fonction du coût du retiming multidimensionnel */*
12: $f_R \leftarrow \Delta C/(T - (Clk_R \times N_{Clk_R}))$
13: **Fin si**
14: **fin**

Dans le cas du loop striping, l'optimisation consiste à la duplication f fois des instructions de la boucle interne et l'ajout des itérations non regroupées autant que prologue et épilogue. D'où, cette technique modifie uniquement le nombre de cycles et la taille du code. Nous débutons par définir le nombre des itérations qui seront exécutées dans chaque partie du prologue et d'épilogue. Ensuite, nous calculons le nombre des cycles en prenant en considération les nouvelles valeurs des itérateurs des boucles. L'augmentation du code ΔC représente l'ajout des instructions dans la boucle interne et celles du prologue et d'épilogue. Le coût du loop striping est calculé tel que décrit dans l'algorithme 5.4.

Notre approche génère une implémentation avec une taille de code minimale dont l'efficacité est prouvée dans le théorème 5.3.

Théorème 5.3 *Soit le GFDM $G = (V, E, d, t)$, Le nombre d'instructions de la boucle interne C_{in} et le nombre maximal d'itérations I, l'approche d'optimisation génère le GFDM final dont la complexité est $O(V^2 + E + I \times C_{in})$.*

Preuve. *La complexité de la technique de retiming multidimensionnel décalé est* $O(E + V^2)$. *Par la suite, le loop striping nécessite* $O(E)$ *pour la selection du facteur f et l'offset g et* $O(E + f \times C_{in})$ *pour générer le code final [5]. De plus, les fonctions de coût sont exécutées dans* $O(V)$ *dans le cas du retiming multidimensionnel décalé et dans* $O(C_{in})$ *dans le cas du loop striping dont chacune est associé à la complexité de la sélection. Admettant que la génération du code après le loop striping est exécutée une seule fois et* $f < I$, *la complexité de l'approche d'optimisation est de* $O(V^2 + E + I \times C_{in})$.

Algorithme 5-4 Calcul du coût du loop striping

Entrée : GFDM $G = (V, E, d, t)$, les parameters du loop striping (f, g), la période du cycle courante Clk

Sortie : coût du loop striping f_{LS}

0: **Début**

/ Calculer le nombre des periodes de cycles après loop striping */*

1: $iter \leftarrow 0$

2: **Pour** i de 0 à $(f - 2)$ **faire**

3: **Pour** j de 0 à $(g - 1)$ **faire**

4: $iter \leftarrow iter + 1$

5: **Fin pour**

6: **Fin pour**

7: $N_{Clk_{LS}} \leftarrow \left\lceil \frac{m}{f} \right\rceil \times \left(2 \times iter + (n - g) \right)$

/ Calculer la taille du code ajoutée après le loop striping */*

8: Calculer la taille des instructions de la boucle interne C_{in}

9: $\Delta C \leftarrow C_{in} \times iter + C_{in} \times f$

/ Calculer la fonction du coût du loop striping */*

10: $f_{LS} \leftarrow \frac{\Delta C}{\left(T - \left(Clk \times N_{Clk_{LS}}\right)\right)}$

11: **Fin**

5.3.6 Résultats expérimentaux

5.3.6.1 Principes

Dans cette section, nous évaluons l'apport de notre approche d'optimisation par rapport aux techniques du retiming multidimensionnel décalé et du loop striping. Dans une première étape, nous évaluons le respect des valeurs des contraintes du temps d'exécution. Par la suite, nous comparons les tailles des codes des implémentations générées par chacune des techniques, dans le cas où les contraintes sont respectées.

Notre expérimentation est guidée par l'intermédiaire de trois applications qui sont : le Filtre Numérique d'Ondelette (FNO) [2], le Filtre à Réponse Impulsionnelle Infinie (FRII) [1] et la Transformée de Walsh-Fourier (TWF) [26]. Pour valider la contribution de notre approche, nous comparons ses résultats à ceux générés par le loop striping et à ceux générés par le retiming multidimensionnel décalé. Cette étape consiste à appliquer les trois techniques d'optimisation sur les mêmes GFDMs. En fait, le retiming multidimensionnel décalé choisit le parallélisme en fonction des temps d'exécution des instructions. Ces derniers diffèrent en fonction des technologies et des cibles d'implémentations. De ce fait, nous procédons à modéliser chaque GFDM avec différent temps d'exécution des instructions. Après la génération des GFDMs résultats, nous en déduisons les codes optimisés des trois techniques

et nous déterminons leurs temps d'exécution en matière de périodes cycles et leurs tailles des codes en matières d'instructions, dont la démarche de mesure est décrite dans [13, 76].

5.3.6.2 Respect de la contrainte du temps d'exécution

La démarche de notre approche d'optimisation consiste à augmenter le parallélisme que ce soit au niveau des instructions qu'au niveau des itérations, dans l'objectif de respecter la contrainte du temps. Nous évaluons dans ce paragraphe l'apport de notre approche en matière de respect de la contrainte du temps d'exécution par rapport aux deux techniques d'optimisation. Pour le cas du filtre numérique d'ondelettes, nous varions les temps d'exécution des instructions et les contraintes des temps d'exécution telles que affichées dans la courbe de la figure 5.11. Les temps d'exécution des GFDMs optimisés sont schématisés dans l'histogramme de la même figure. Ces valeurs montrent les limites du retiming multidimensionnel et du loop striping par rapport à notre approche.

Prenons le cas de la transformée de Walsh-Fourrier dont les valeurs des temps d'exécution sont affichées dans la figure 5.12. Notre approche d'optimisation permet de respecter toutes les contraintes tandis que le loop striping permet de respecter une seule, et le retiming multidimensionnel n'a pu atteindre aucune des contraintes exigées. d'ou, elle respecte des contraintes d'optimisation dont les deux techniques d'optimisation ne le permettent pas. Par conséquent, notre approche d'optimisation permet d'atteindre des temps d'exécution minimaux du nid de boucles, dont le pourcentage d'amélioration de 35.21% par rapport au retiming multidimensionnel décalé et 20.38% par rapport à la technique de loop striping.

5.3.6.3 Minimisation des tailles des codes

Pour comparer l'apport en matière des tailles des codes, nous imposons des contraintes de temps d'exécution dont les trois techniques peuvent les respecter. Puis, nous déterminons les tailles des codes de leurs implémentations. Pour le cas du filtre numérique d'ondelettes, le nombre réduit des instructions implique un espace restreint des solutions de parallélisme. Nous avons constaté que l'approche d'optimisation choisie l'implémentation ayant le minimum du code parmi celles proposées du retiming multidimensionnel ou du loop striping, tel que les tailles des codes affichées dans l'histogramme de la figure 5.13.

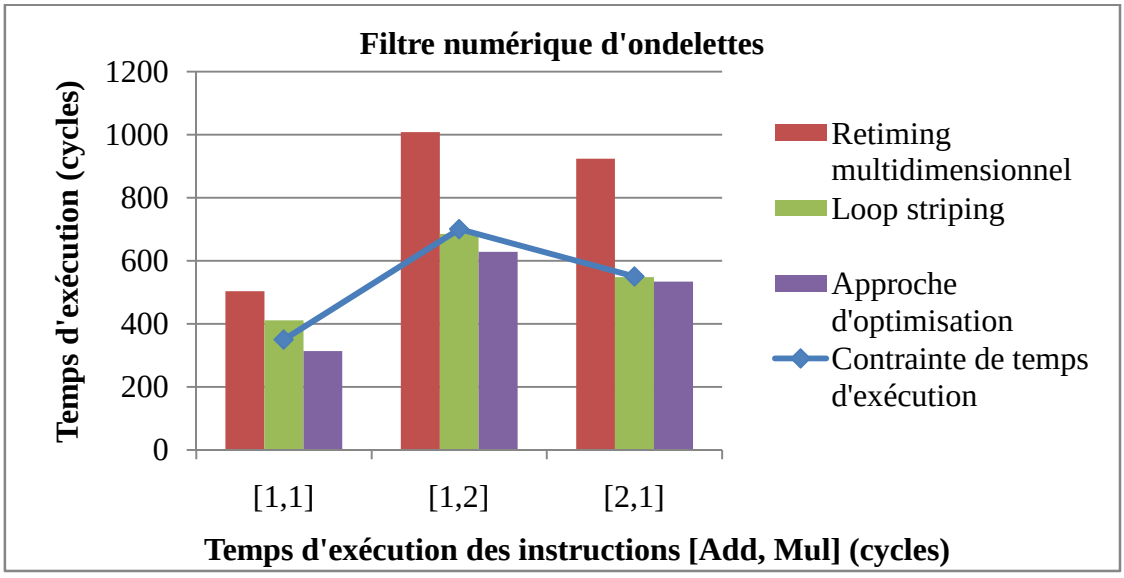


Figure 5.11 Temps d'exécution du filtre numérique d'ondelettes en fonction des techniques d'optimisation

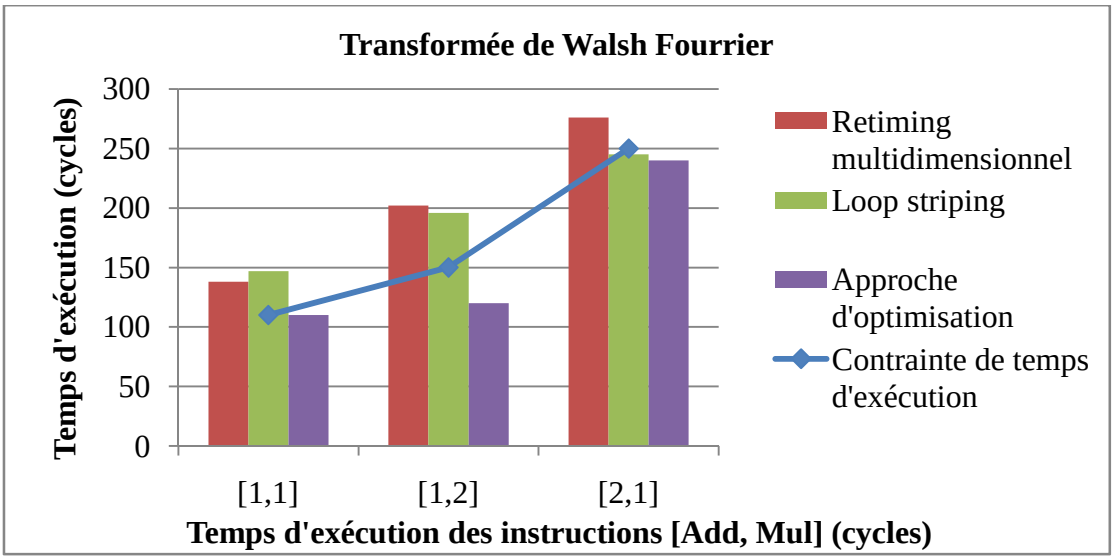


Figure 5.12 Temps d'exécution du transformée de Walsh Fourrier en fonction des techniques d'optimisation

Pour le cas du filtre numérique d'ondelettes, le nombre des nœuds des chemins critiques permettent d'appliquer une succession de fonctions de retiming multidimensionnel. Vu que chaque fonction de retiming est dans l'obligation de décaler les instructions à travers les deux boucles imbriquées, la taille du code augmente intensivement avec les niveaux du parallélisme. Dans cette expérimentation, les contraintes des temps d'exécution ont impliqué le recours à quatre fonctions de retiming, ce qui explique les tailles importantes de code dont les valeurs sont illustrées dans l'histogramme de la figure 5.14. Le choix de l'approche conduit toujours à l'implémentation ayant la taille du code minimale par rapport aux deux techniques.

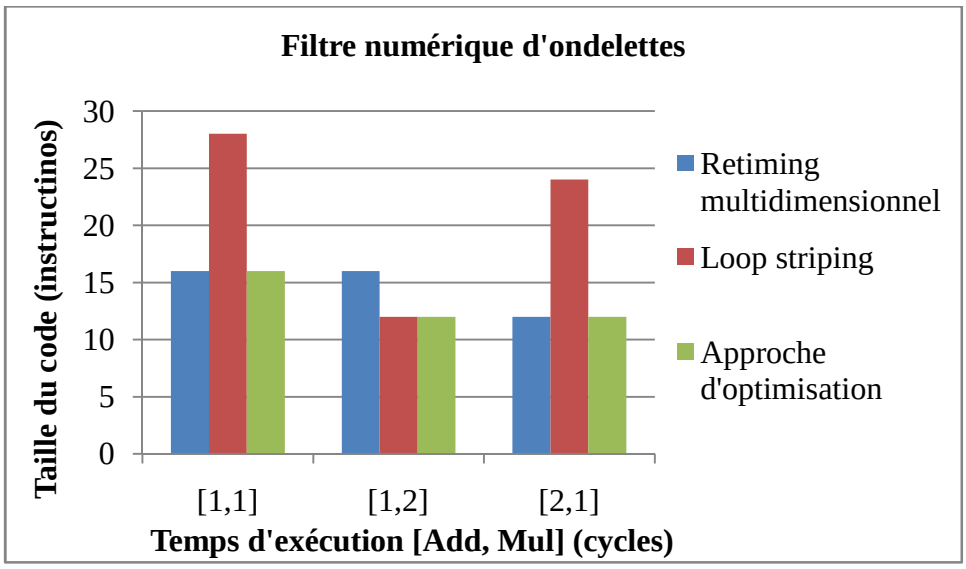


Figure 5.13 Tailles des codes du filtre numérique d'ondelettes en fonction des techniques d'optimisation

Pour la transformée de Walsh-Fourrier dont les tailles des codes sont illustrées dans l'histogramme de la figure 5.15, les dépendances des données inter-itérations à travers les deux boucles du nid impliquent une augmentation importante du code des solutions générées par chacune des deux techniques. Pour le cas de cette application, nous avons imposé des contraintes de temps d'exécution impliquant différents niveaux de parallélisme par chacune

des trois techniques. Pour le cas d'une contrainte proche du temps d'exécution initial du GFDM, notre approche choisie la solution générée par le retiming multidimensionnel. Pour les deux autres contraintes, elle combine les deux niveaux de parallélisme.

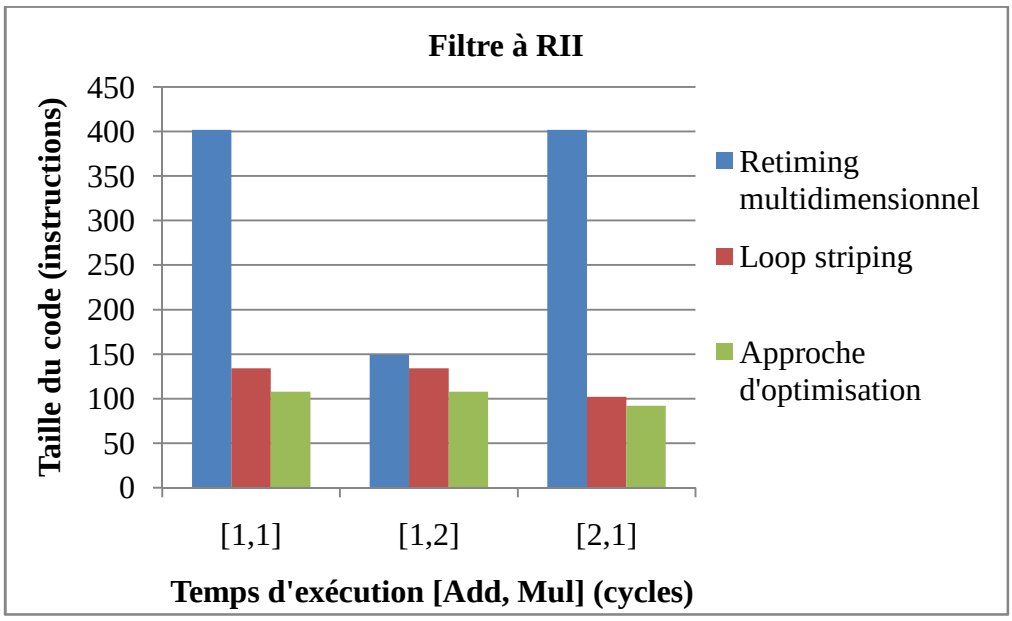


Figure 5.14 Tailles des codes du filtre à RII en fonction des techniques d'optimisation

Par conséquent, nous constatons que notre technique d'optimisation génèrent des implémentations dont les tailles des codes sont inférieures aux celles correspondant aux codes générés par chacune des autres techniques, avec des pourcentages d'amélioration de 35.21% par rapport au retiming multidmensionnel décalé et 16.38% par rapport à la technique de loop striping.

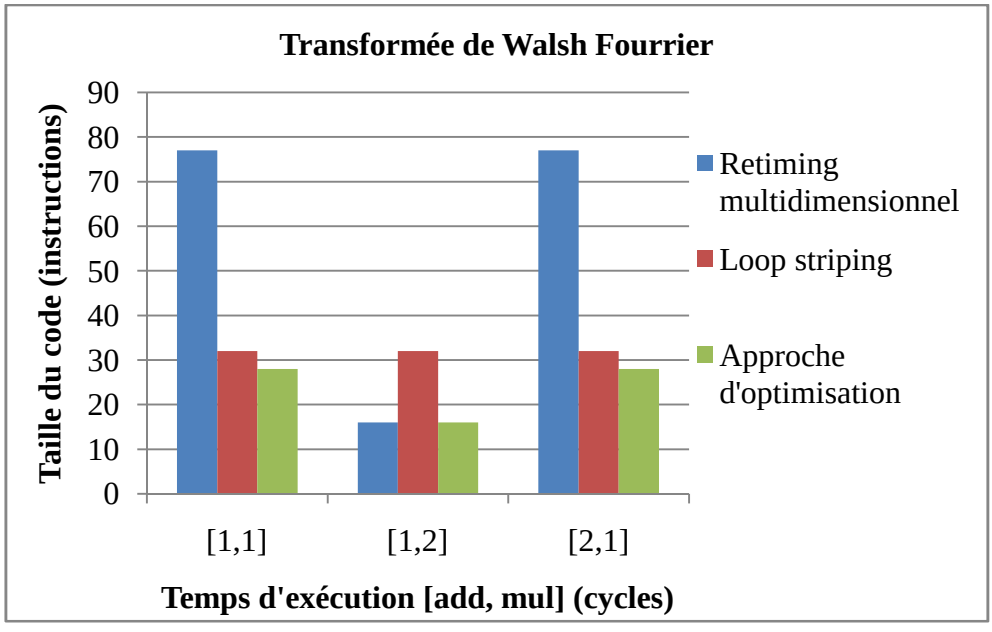


Figure 5.15 Tailles des codes du transformée de Walsh Fourrier en fonction des techniques d'optimisation

5.4 Conclusion

Dans ce chapitre, nous avons mis en pratique la technique du retiming multidimensionnel décalé dans le contexte des implémentations temps réel embarquées des nids de boucle. En

première lieu, nous avons proposé une démarche d'optimisation utilisant la technique « retiming multidimensionnel décalé » pour déterminer le niveau minimal du parallélisme, dans l'objectif de respecter la contrainte du temps d'exécution tout en utilisant un code de taille minimale. Elle permet de prédire les temps d'exécution et les tailles du code en fonction du retiming multidimensionnel, puis de sélectionner le niveau minimal permettant de respecter la contrainte du temps d'exécution. Par la suite, nous avons décrit une deuxième démarche permettant de combiner les parallélismes au niveau des instructions et au niveau des itérations, en utilisant notre technique et le « loop striping », pour respecter une contrainte de temps d'exécution tout en générant une taille de code minimale.

La validation expérimentale a montré que l'utilisation conjointe de notre technique et le « loop striping » permet de respecter des contraintes de temps d'exécution, que chacune des techniques appliquées séparément ne le permettait. De plus, même si la contrainte est satisfaite, elle permet de générer une solution avec une taille de code inférieure à celles générées par chacune de ces deux techniques. Le résultat de cette validation est à confirmer par des implémentations des codes optimisées sur des architectures parallèles.

CHAPITRE 6 : CONCLUSION GÉNÉRALE ET PERSPECTIVES

6.1 Conclusion

Nous nous sommes intéressés dans ce travail à l'optimisation du temps d'exécution des nids de boucles. Ces structures algorithmiques requièrent une puissance de calcul de plus en plus importante, en vue d'exécuter les traitements itératifs. La réduction du temps d'exécution est réalisée par l'augmentation du niveau du parallélisme des traitements des nids. Dans ce contexte, nous avons décrit les différents modèles de représentation des boucles imbriquées et les techniques du parallélisme qui y sont associées. A partir de cette description, nous avons constaté que le GFDM permet une représentation explicite des granularités au niveau des itérations et au niveau des instructions. Cette étape a été poursuivie par une étude détaillée du GFDM et ces techniques de parallélisme. Cette deuxième partie de l'état de l'art nous a permis de déduire que les techniques du parallélisme au niveau des instructions visent à atteindre un parallélisme total, entraînant une augmentation importante des tailles du code, et empêchant d'atteindre des temps d'exécution optimaux.

Dans ce contexte, nous avons proposé une nouvelle technique de parallélisme des instructions du nid de boucles, intitulée « retiming multidimensionnel décalé ». Elle permet d'ordonnancer le nid de boucles avec une période de cycle minimale, sans atteindre un parallélisme total. Cette technique assure le parallélisme des chemins de données entiers appartenant aux traitements internes des itérations. Elle explore les dépendances de données et les temps d'exécution des différents chemins de données, pour les analyser et déterminer ceux à paralléliser. Par la suite, elle extrait un vecteur de parallélisme adéquat aux chemins sélectionnés. Cette technique a été étendue en utilisant itérativement une fonction optimale du parallélisme pour l'atteinte de la période de cycle minimale. La technique de « retiming multidimensionnel décalé » a été validée par l'implémentation d'applications en traitement du signal et d'images, sur les architectures parallèles multi-GPUs. Les résultats expérimentaux ont montré que la technique de « retiming multidimensionnel décalé » génère des implémentations avec des temps d'exécution et des tailles de codes inférieurs à ceux des implémentations générées par les techniques du retiming multidimensionnel progressif et enchaîné.

Dans la deuxième contribution, nous avons mis en pratique notre technique dans le contexte des implémentations temps réel embarquées des nids de boucle. L'objectif était de respecter la contrainte du temps d'exécution tout en utilisant un code de taille minimale. Dans ce contexte, une première démarche a été proposée permettant l'utilisation du retiming multidimensionnel décalé pour déterminer le niveau du parallélisme minimal assurant le respect de la contrainte du temps d'exécution. Par la suite, une deuxième démarche a été décrite permettant de combiner les parallélismes au niveau des instructions et au niveau des itérations, en utilisant notre technique et celle du « loop striping », pour respecter l'objectif fixé. Les résultats expérimentaux de la validation ont montré que l'utilisation conjointe de notre technique et le « loop striping » permet de respecter des contraintes du temps d'exécution, que chacune des techniques séparément ne le permet. De plus, même si la contrainte est satisfaite, elle permet de générer une solution avec une taille de code inférieure à celles générées par chacune de ces deux techniques.

6.2 Perspectives

Nous étendons les travaux de ce manuscrit dans trois axes différents. Le premier consiste à enrichir l'exploration des spécifications algorithmiques du nid de boucles dans l'objectif d'améliorer les performances des implémentations générées. Dans Le deuxième axe, nous nous intéresserons à la prise en considération des paramètres de l'architecture pour améliorer

la précision de l'estimation. Le troisième consiste à automatiser les démarches d'optimisation que nous avons proposée.

En premier lieu, le formalisme de notre technique permet de l'étendre pour prendre en considération les structures des nids de boucles non-uniformes, intégrant des instructions à travers les boucles. De plus, la démarche d'utilisation conjointe de notre technique avec le « loop striping » fera l'objet d'une extension par le recours à d'autres techniques d'optimisation au niveau des itérations, tel que la technique du « loop tiling » et la technique du « loop distribution and fusion ».

En deuxième lieu, nos travaux de future s'intéresseront à l'amélioration des démarches d'estimation des performances en fonction du parallélisme. Cet objectif implique la prise en considération des paramètres technologiques et architecturaux des cibles d'implémentation tels que le temps d'accès aux mémoires, la communication inter-processeurs, etc. De plus, nous visons à étudier l'évolution de la taille de la mémoire en fonction du parallélisme proposé par le retiming multidimensionnel décalé. Cette étape d'évaluation permettra par la suite d'orienter notre démarche d'optimisation dans l'objectif de respecter des contraintes en matière de taille maximale de mémoire ou de nombre d'unités de traitement. Par ailleurs, certaines implémentations temps réel exigent des contraintes temporelles en termes de latence ou cadence, dont les valeurs sont calculées à partir des données d'entrées/sorties. De ce fait, la démarche d'estimation doit prédire, non seulement le temps d'exécution des unités de traitement, mais le temps total de la latence. Pour le cas des implémentations sur des architectures multi-GPUs, l'estimation de la latence implique l'estimation du temps du lancement du code à partir de l'unité centrale de traitement et le temps d'accès mémoire aux données, en plus du temps d'exécution des GPUs.

En troisième lieu, nous nous intéresserons à concrétiser nos démarches de parallélisme dans des outils logiciels. Cette étape consiste à générer ou étendre des compilateurs d'architectures parallèles. Ces extensions permettront de générer des implémentations temps réel dont le compilateur intercepte les contraintes (temps d'exécution, nombre des processeurs élémentaires…) de la part du concepteur afin de choisir le parallélisme et de générer l'implémentation adéquate.

BIBLIOGRAPHIE

[1] N.L. Passos, E.H.M. Sha, “Achieving full parallelism using multi-dimensional retiming”, J. IEEE Trans. Par. Dist. Syst., Volume 7, Issue 11, 1150-1163, Nov. 1996.

[2] Sheliga, Passos, E.H.M. Sha, “Fully parallel hardware/software codesign for multidimensional DSP applications”, In Proceedings of the 4th International Workshop on Hardware/Software Co-Design CODES’96, Pennsylvania (USA), March 18-20, 1996.

[3] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen, John Cavazos, “Iterative optimization in the polyhedral model: Part II, multidimensional time”, in the ACM sigplan conference on programming language design and implementation (PLDI ’08), USA, Pages 90-100, 2008.

[4] Zhuge, Xue, Qiu, Hu, E.H.M. Sha, “Timing Optimization via Nest-Loop Pipelining Considering Code Size”, J. Microproc. Microsyst. Volume 32, Issue 7, Pages 351-363 , October 2008.

[5] C. Xue, E.H.-M. Sha, “Maximize parallelism minimize overhead for nested loops via loop Striping”, J. of VLSI Sig. Proc., vol. 47, pp. 153–167, December 2006.

[6] Q. Zhuge, C. Xue, Z. Shao, M. Liu, M. Qiu, E.H.-M. Sha, “Design optimization and space minimization considering timing and code size via retiming and unfolding”, J. Microproc. Microsyst., vol. 30,pp. 173–183, 2006.

[7] O’Neil, Tongsima, and E.H.M. Sha, “Extended retiming: Optimal scheduling via a graph-theoretical approach”, In Proceeding the Acoustics, Speech, and Signal Processing, volume 4, pages 2001–2004, Arizona (USA), Mar. 15-19, 1999.

[8] T.W. O’neil, and E.H.-M. Sha, “Combining extended retiming and unfolding for rate-optimal graph transformation”, J. of VLSI Sign. Process., vol. 39, iss. 3, pp: 273–293, March 2005.

[9] T. Grosser , A. Cohen , P. H. J. Kelly , J. Ramanujam , P. Sadayappan, S. Verdoolaege, “Split tiling for GPUs: automatic parallelization using trapezoidal tiles”, Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units, Pages 24-31, March 16, 2013.

[10] Q. Zhuge, B. Xiao, Z. Shao, and E.H.-M. Sha, C. Chantrapornchai, “Optimal code size reduction for software-pipelined and unfolded loops”, ISSS, pp. 144-149, 2002.

[11] Q. Zhuge, Z. Shao, B. Xiao, and E.H.-M. Sha, “Design space minimization with timing and code size optimization for embedded DSP”, CODES+ISSS, California (USA), pp. 144-149, October 2003.

[12] C. J. Xue, E.H.M. Sha, Z. Shao, M. Qiu, “Effective Loop Partitioning and Scheduling

under Memory and Register Dual Constraints, Design", Automation and Test in Europe, 2008. DATE '08, Munich (Germany), Page(s):1202 – 1207, 10-14 March 2008.

[13] C. J. Xue, J. Hu, Z. Shao, E.H.M. Sha, "Iterational Retiming with Partitioning: Loop Scheduling with Complete Memory Latency Hiding", ACM Transactions on Embedded Computing Systems, Vol. 9, No. 3, Article 22, February 2010.

[14] D. Liu, Z. Shao, M. Wang, M. Guo, J. Xue, "Optimal Loop Parallelization for Maximizing Iteration-Level Parallelism", CASES'09, Grenoble (France), October 11–16, 2009.

[15] A. Qasem, K. Kennedy, "Model-guided empirical tuning of loop fusion", International Journal of High Performance Systems Architecture, Volume 1 Issue 3, Pages 183-198, December 2008.

[16] H. Rong, Z. Tang, R. Govindarajan, A. Douillet, G.R. Gao, "single-dimension software pipelining for multidimensional loops", ACM transactions on architecture and code optimization, Vol. 4, No.1, Article 7, March 2007.

[17] M. Liu, E. H. M. Sha,Q. Zhuge, Y. He, M. Qiu, " Loop Distribution and Fusion with Timing and Code Size Optimization", J Sign. Process. Syst., 62:325–340, 2011.

[18] M. Fellahi, A. Cohen, "Software Pipelining in Nested Loops with Prolog-Epilog Merging", High Performance Embedded Architectures and Compilers Lecture Notes in Computer Science, Volume 5409, pp 80-94, 2009.

[19] M. A. Khan, "Improving performance through deep value profiling and specialization with code transformation", journal of Computer Languages, Systems & Structures 37, pages :193–203, 2011.

[20] A. Morvan, S. Derrien, P. Quinton, "Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion", in International Conference on Field-Programmable Technology (FPT), Page(s):1 – 10, New Delhi (india), 12-14 Dec. 2011.

[21] K. Turkington, G. A. Constantinides, K. Masselos, P. Y. K. Cheung, "Outer Loop Pipelining for Application Specific Datapaths in FPGAs", IEEE Transaction on very large scale integration (VLSI) systems, VOL. 16, NO. 10, pp:1268-1280, OCTOBER 2008.

[22] K. Muthukumar, Doshi, "Software pipelining of nested loops". Lecture Notes in Computer Science 2027, 165–181, 2001.

[23] M. Fellahi, A. Cohen, S. Touati, "Code-Size Conscious Pipelining of Imperfectly Nested Loops", in Proceedings of the 2007 workshop on MEmory performance: Dealing with Applications, systems and architecture (MEDEA), 2007.

[24] E. Leiserson, B. Saxe, "retiming synchronous circuitry", algorithmica, Springer New York, volume 6, numbers 1-6, juin 1991.

[25] N.L. Passos, E.H.M. Sha, "Full Parallelism In Uniform Nested Loops Using Multi-Dimensional Retiming", International Conference on Parallel Processing, 1994.

[26] N. L. Passos , D. C. Defoe, R. J. Bailey, R. H. Halverson, R. P. Simpson, “Theoretical Constraints on Multi-Dimensional Retiming Design Techniques", Proceedings of the AeroSense-Aerospace/Defense Sensing, Simulation and Controls, Orlando, FL, April, 2001.

[27] Q. Zhuge, Z. Shao, E. H. M. Sha, “timing optimization of nested loops considering code size for DSP applications”, Proceeding of the 2004 international conference on parallel processing, 2004.

[28] Q. Zhuge, E.H. M. Sha, C. Chantrapornchai, “CRED: code size reduction technique and implementation for software-pipelined applications”, in Proceedings of the IEEE Workshop of Embedded System Codesign (ESCODES), pp. 50–56, September, 2002.

[29] T. C. Denk, K. K. Parhi, "Two-Dimensional Retiming," in the IEEE Transactions on VLSI, Vol. 7, No. 2, pp. 198-211, June, 1999.

[30] N. L. Passos, E. H.-M. Sha, "Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints", in the IEEE Transactions on VLSI Systems, Volume 6, Number 4, pages 719-730, December 1998.

[31] L. F. Chao, E. H. M. Sha, “scheduling data flow graphs via retiming and unfolding”, IEEE, 1997.

[32] C. Xue, Z. Shao, M. Liu, M. Qiu, E. H. M. Sha, “Loop Scheduling with Complete Memory Latency Hiding on Multi-core Architecture”, the 12th international conference on parallel and distributed systems (ICPADS), 2006.

[33] C. J. Xue, Z. Shao, M. liu, M. K. Qiu, E.H.-M. Sha, “Optimizing parallelism for nested loops with iterational and instructional retiming”, J. Embed. Comput., vol. 3, iss.1, pp. 29-37, January 2009.

[34] T.W. O’neil, E.H.-M. Sha, “Combining extended retiming and unfolding for rate-optimal graph transformation”, J. of VLSI Sign. Process., vol. 39, iss. 3, pp: 273–293, March 2005.

[35] C. J. Xue, Z. Shao, M. liu, M. K. Qiu, E.H.-M. Sha, “Optimizing parallelism for nested loops with iterational and instructional retiming”, J. Embed. Comput., vol. 3, iss.1, pp. 29-37, January 2009.

[36] L. Kaouane, M. Akil,T. Grandpierre, Y. Sorel, “ A methodology to implement real-time applications onto reconfigurable circuits”, J. Supercomp., vol. 30, iss. 3, pp. 283-301, December 2004.

[37] K.K. Parhi, D.G. Messerschmitt, “Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding”, IEEE Transact. Comp., vol. 40, iss. 2, pp. 178-195, February 1991.

[38] O. Lobachev, M. Guthe, R. Loogen, “Estimating parallel performance”, J. Par. Dist. Comp., January 2013.

[39] G. Romanazzi, P.K. Jimack, C.E. Goodyer, “Reliable performance prediction for multigrid software on distributed memory systems”, J. Adv. Engin. Softw., vol. 42, pp.

247–258, May 2011.

[40] O. Sinnen. “Task Scheduling for Parallel Systems”. Wiley, 2007.

[41] P. Y. Calland, A. Darte, Y. Robert, F. Vivien. “On the removal of anti and output dependencies. Int. Journal of Parallel Programming”, 26(2):285–312, 1998.

[42] F. Sanchez, J. Cortadella, “Time-Constrained Lopp Pipelining“, I3C, 1995.

[43] J. P. Elloy. “Systèmes réactifs synchrones et asynchrones“. In Applications, Réseaux et Systèmes – École d’été temps réel’99, pages 43–51, Futuroscope, Septembre 1997.

[44] P. Richard, F. Cottet, C. Kaiser. “Précédences généralisées et ordonnançabilité des tâches de suivi temps réel d’un laminoir“, Journal Européen des Systèmes Automatisés, 35 (9) :1055–1071, 2001.

[45] C. Kaiser. “Description et critique d un système temps réel pour le suivi d’un laminoir : Robustesse et potentiel d’évolutivité“, volume 20(1). Hermes Science, 2001.

[46] D. Isovic, G. Fohler, L. Steffens. “Real-time issues of mpeg-2 playout in resource constrained systems“. International Journal on Embedded Systems, 1, issue 2(ISSN 1740-4460), 6 2004.

[47] "Real time faq", 1998. http:www.realtime-info.be/encyc/techno/publi/faq/rtfaq.htm.

[48] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, “Ordonnancement temps réel”, Hermes, pp. 207, 2000.

[49] Turing (A.M.). – “On computable numbers, with an application to the entscheindungs problem”. In : Proc. London Math. Soc.

[50] C. Kaiser, G. Stoffel. “Système d’acquisition et d analyse en temps réel des signaux d un laminoir“. Rapport scientifique CEDRIC, 1999.

[51] F. Sanchez, J. Cortadella, “Maximum-Throughput Software Pipelining”, ICCP 1998.

[52] C. Jesshope, “Scalable instruction-level parallelism”, in Proc. Computer Systems: Architectures, Modeling and Simulation, 3rd and 4th Int. Workshops, SAMOS 2004.

[53] B. R. Rau, J. A. Fisher, “Instruction-level parallel processing: History, overview, and perspective”, The Journal of Supercomputing, May 1993, Volume 7, Issue 1-2, pp 9-50.

[54] D. K. Arvind , V. E. F. Rebello, “Instruction-Level Parallelism In Asynchronous Processor Architectures”, proceedings of the 3RD international workshop on algorithms and parallel VLSI architecture.

[55] K. W. Rudd, “Vliw Processors: Efficiently Exploiting Instruction Level Parallelism”, Stanford University, 1999.

[56] K. Lodaya , P. Weil, “A Kleene iteration for parallelism”, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science Volume 1530, 1998, pp 355-366.

[57] M. Lam, “Software pipelining: An effective scheduling technique for VLIW machines”, PLDI '88 Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation , Pages 318-328.

[58] D. Liu, Y. Wang, Z. Shao, M. Guo, J. Xue, “Optimally Maximizing Iteration-Level Loop Parallelism”, IEEE Transactions on parallel and distributed systems, volume 23Issue 3, March 2012, pages 564-572.

[59] J. M. A. Roy, “Exploiting iteration-level parallelism in declarative programs”, Doctoral Dissertation.

[60] “Nested Loops Scheduling”, Encyclopedia of Parallel Computing, 2011, p 1283.

[61] S. Shekhar, H. Xiong, “Nested-Loop, Blocked”, Encyclopedia of GIS, 2008, p 787.

[62] “Nested Loops”, Cost-Based Oracle Fundamentals, 2006, pp 307-318.

[63] C. Beeri, M. Y. Vardi, “The implication problem for data dependencies”, Automata, Languages and Programming Lecture Notes in Computer Science Volume 115, 1981, pp 73-85.

[64] L. Qiao, W. Huang, Z. Tang, “Coping with Data Dependencies of Multi-dimensional Array References”, Network and Parallel Computing Lecture Notes in Computer Science Volume 3779, 2005, pp 278-284.

[65] M. Heffernan, K. Wilken, “Data-Dependency Graph Transformations for Instruction Scheduling”, Journal of Scheduling October 2005, Volume 8, Issue 5, pp 427-451.

[66] N. Vasilache, C. Bastoul, A. Cohen, “Polyhedral Code Generation in the Real World”, Compiler Construction Lecture Notes in Computer Science Volume 3923, 2006, pp 185-201.

[67] S. Derrien, S. Rajopadhye, P. Quinton, T. Risset, “High-Level Synthesis of Loops Using the Polyhedral Model”, High-Level Synthesis, 2008, pp 215-230.

[68] M. W. Benabderrahmane, L. N. Pouchet, A. Cohen, C. Bastoul, “The Polyhedral Model Is More Widely Applicable Than You Think”, Compiler Construction Lecture Notes in Computer Science Volume 6011, 2010, pp 283-303.

[69] Z. Huang, S. Malik, “Exploiting Operation Level Parallelism through Dynamically Reconfigurable Datapaths”, DAC 2002, June 10-14, 2002.

[70] L. Wang, Z. Wang, K. Dai, “Cycle Period Analysis and Optimization of Timed Circuits”, Advances in Computer Systems Architecture Lecture Notes in Computer Science Volume 4186, 2006, pp 502-508.

[71] G. Tu, F. Yang, Y. Lu, “Scheduling algorithms based on weakly hard real-time constraints”, Journal of Computer Science and Technology November 2003, Volume 18, Issue 6, pp 815-821.

[72] S. Leonardi, A. Marchetti-Spaccamela, A. Vitaletti, “Approximation Algorithms for Bandwidth and Storage Allocation Problems under Real Time Constraints”, FST TCS

2000: Foundations of Software Technology and Theoretical Computer Science Lecture Notes in Computer Science Volume 1974, 2000, pp 409-420.

[73] N. Ge, M. Pantel, X. Crégut, “Formal Specification and Verification of Task Time Constraints for Real-Time Systems”, Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies Lecture Notes in Computer Science Volume 7610, 2012, pp 143-157.

[74] “Latency Hiding”, Encyclopedia of Parallel Computing, 2011, p 1006.

[75] T. Sato, I. Arita, “Execution Latency Reduction via Variable Latency Pipeline and Instruction Reuse”, Euro-Par 2001 Parallel Processing Lecture Notes in Computer Science Volume 2150, 2001, pp 428-438.

[76] C. Xue, Z. Shao, M. Liu, E. H. M. Sha , "Iterational Retiming: Maximize Iteration-Level Parallelism", CODES+ISSS’05,Sept. 19–21, 2005, Jersey City, New Jersey, USA.

[77] F. M. Ciorba, “Algorithms Design for the Parallelization of Nested Loops”, doctoral dissertation national technical university of Athens, February 2008.

[78] A. Cohen, “program analysis and transformation: from the polytope model to formal languages”, University of Versailles, December 2009.

[79] D. Petkov, R. Harr, S. Amarasinghe, “Efficient pipelining of nested loops: unroll-and-squash”, Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, April 2001, ISBN 0-7695-1573-8.

[80] T. W. O’Neil, E. H. M. Sha, “Time-Constrained Loop Scheduling with Minimal Resources”, Journal of Embedded Computing - Embeded Processors and Systems, Volume 2 Issue 1, January 2006, Pages 103-117.

[81] J. L. Lo, S. J. Eggers, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, “Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading”, ACM Transactions on Computer Systems, August 1997.

[82] F. Balasa, P. G. Kjeldsberg, A. Vandecappelle, M. Palkovic, Q. Hu, H. Zhu, F. Catthoor, "Storage Estimation and Design Space Exploration Methodologies for the Memory Management of Signal Processing Applications", Journal of Signal Processing Systems November 2008, Volume 53, Issue 1-2, pp 51-71.

[83] Y. Elloumi, M. Akil, M. H. Bedoui, “Timing and Code Size Optimization on Achieving Full Parallelism in Uniform Nested Loop”, Journal of computing, VOLUME 3, ISSUE 7, Juillet 2011, ISSN 2151-9617, pages : 68-77.

[84] Y. Elloumi, M. Akil, M. H. Bedoui, “Execution Time Optimization Using Delayed Multidimensional Retiming”, à “16th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (IEEE/ACM DS-RT)”, 25-27 Octobre 2012, à Dublin (Ireland), ISBN: 978-0-7695-4846-3, Pages 177-184.

[85] Y. Elloumi, M. Akil, M. H. Bedoui, “Execution Time and Code Size Optimization using Multidimensional Retiming and Loop Striping”, à “16th EUROMICRO Conference on Digital System Design(DSD 2013)”, 04-06 Septembre 2013, à Santander (Espagne). (En

cours d'hébergement).

[86] Y. Elloumi, M. Akil, M. H. Bedoui, "Execution time and code size optimization using delayed multidimensional retiming", soumis au journal "ACM Transaction on architecture and code optimization" (TACO).

[87] L. Kaouane, "Formalisation et optimisation d'applications s'exécutant sur architecture reconfigurable", thèse de doctorat, Université de Marne-La-Vallée, Décembre 2004.

[88] L. Kaouane, M. Akil, T. Grandpierre, "A methodology to implement real-time applications on reconfigurable circuits", The Journal of Supercomputing" December 2004, Volume 30, Issue 3, pp 283-301.

[89] Y. Elloumi, M. Akil, T. Grandpierre, M. H. Bedoui, "Latency and Power Optimization in AAA Methodology for Integrated Circuits", à "17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2010)", 12-15 décembre 2010, à Athènes (Grèce), Pages 639–642, ISBN: 978-1-4244-8155-2.

[90] S. Kurra, N. K. Singh, P. R. Panda, "The Impact of Loop Unrolling on Controller Delay in High Level Synthesis", DATE '07 Proceedings of the conference on Design, automation and test in Europe, Pages 391-396, ISBN: 978-3-9810801-2-4

[91] V. Sarkar, "optimized unrolling for nested loop", International Journal of Parallel Programming, October 2001, Volume 29, Issue 5, pp 545-581.

[92] Y. Dong, J. Zhou, Y. Dou, L. Deng, J. Zhao, "Impact of Loop Unrolling on Area, Throughput and Clock Frequency for Window Operations based on a Data Schedule Method", 2008 Congress on Image and Signal Processing.

[93] X. Liu, M. C. Papaefthymiou, E G. Friedman, "Retiming and Clock Scheduling for Digital Circuit Optimization", IEEE Transaction on computer-aided design of integrated circuits and systems, VOL. 21, NO. 2, FEBRUARY 2002.

[94] M. Gao, J. Huang, S. Zhang, Z. Qian, S. Voros, D. Metaxas, L. Axel, "4D Cardiac Reconstruction Using High Resolution CT Images", Lecture Notes in Computer Science, Volume 6666, 2011, pp 153-160.

[95] M. Ioannides, A. Hadjiprocopis, N. Doulamis, and al., " Online 4D reconstruction using multi-images available under open access", ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume II-5/W1, 2013, 2–6 Septembre 2013, Strasbourg, France.

[96] P. Crowley, J. L. Baer, "Worst-Case Execution Time Estimation for Hardware-assisted Multithreaded Processors", In Proc. of the 2nd Workshop on Network Processors.

[97] V. Blanco, J.A. Gonzalez, C. Leon, C. Rodrıguez, G. Rodrıguez, M. Printista, "Predicting the performance of parallel programs", Parallel Computing 30 (2004) 337−356.

[98] O. Lobacheva, M. Guthe, R. Loogen, "Estimating Parallel Performance", J. Parallel Distrib. Comput. 22 January 2013.

[99] V. Bandishti, I. Pananilath, U. Bondhugula, "Tiling Stencil Computations to Maximize

Parallelism”, SC12, November 10-16, 2012, Salt Lake City, Utah, USA.

[100] L. Renganarayanan, D. Kim, S. Rajopadhye, M. M. Strout, “Parameterized Tiled Loops for Free”, PLDI '07 Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation, pages 405-414, June 11–13, 2007, San Diego, California, USA.

[101] G. Goumas, N. Drosinos, M. Athanasaki, N. Koziris, “Automatic Parallel Code Generation for Tiled Nested Loops”, SAC 2004, Nicosia, Cyprus.

[102] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan, "Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model", Lecture Notes in Computer Science, Vol. 4959, pages : 132-146, 2008.

[103] Hui Liu · Zili Shao · Meng Wang · Junzhao Du·Chun Jason Xue · Zhiping ia"Combining Coarse-Grained Software Pipelining with DVS for Scheduling Real-Time Periodic Dependent Tasks on Multi-Core Embedded Systems", J. Sign. Process. Syst., novembre 2008.

[104] S. Simon, E. Bernard, M. Sauer, J.A. Nossek, “A new retiming algorithm for circuit design“, IEEE International Symposium on Circuits and Systems, ISCAS '94., 35 - 38 vol.4.

[105] L. N. Pouchet, C. Bastoul, A. Cohen, N. Vasilache, “Iterative Optimization in the Polyhedral Model: Part I, One-Dimensional Time”, CGO '07 Proceedings of the International Symposium on Code Generation and Optimization, pages 144-156, ISBN:0-7695-2764-7.

[106] P. Lieverse, P. V. DerWolf, E. Deprettere, K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems ", IEEE Workshop on Signal Processing Systems, pages : 181 – 190, Oct 1999.

[107] M. Matrice, W.O. Timothy, E.H.M. Sha, "Retiming Synchronous Data-Flow Graphs to ReduceExecution Time",IEEE transaction on signal processing, ISSN : 1053-587X, volume 49, issue 10, page : 2397-2407, octobre 2001.

[108] Rachid SEGHIR, " Méthodes de dénombrement de points entiers de polyèdres et applications à l'optimisation de programmes", thèse de doctorat, Universit´e Louis Pasteur-Strasbourg I, décembre 2006.

[109] Y.H. Lee, C. Chen, "An effective and efficient code generation algorithm for uniform loops on non-orthogonal DSP architecture", The Journal of Systems and Software, page 410–428, july 2006.

Annexe A : Implémentation des nids de boucles sur architectures NVIDIA

A.1. Environnement logiciel

CUDA est une plate-forme de programmation introduite par NVIDIA dans l'objectif d'exécuter des calculs en parallèle. Le modèle de programmation assure l'exécution d'un traitement donné sur un périphérique physiquement séparé (appelé Device), et qui fonctionne comme un coprocesseur pour l'hôte exécutant le programme C (appelé Hot). CUDA est livrée avec un environnement logiciel qui permet aux développeurs d'utiliser C comme un langage de programmation de haut niveau. Cet environnement assure la configuration de l'exécution du code sur le Device en matière d'accès aux mémoires, d'allocation des traitements aux GPUs et de synchronisation de l'exécution des GPUs.

Le parallélisme en CUDA requiert la définition de tous les traitements à exécuter sur un processeur distinct, appelé thread. Pour bien maîtriser le nombre important des threads, CUDA procède à les ranger sous la forme d'une « Grille » indexée. De plus, la grille est répartie en « Blocs », qui sont définis par leurs positions dans la grille, et le nombre des threads qu'y sont inclus. De même, un thread est identifié par son bloc dans la grille, puis par sa position dans le bloc. Nous schématisons dans la figure A-1 la répartition des threads par blocs bi-dimensionnels, ainsi que la répartition bi-dimensionnel de la grille. Les threads d'un même bloc ne peuvent par être exécutés dans le même espace temporel. Pour cela, CUDA permet leurs exécutions par groupe de 32 threads, intitulé « wrap ». Un warp doit assurer l'exécution d'un ensemble de thread ayant le même calcul interne.

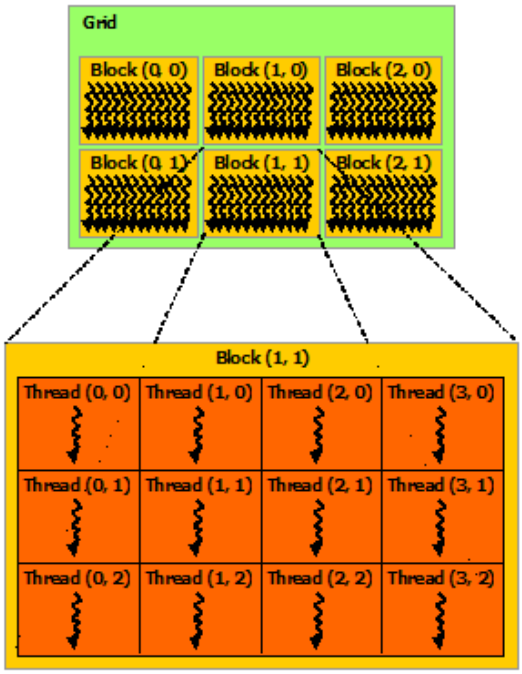


Figure A-1 Répartition des threads de la grille en blocs

Le traitement à exécuter sur le Device est à insérer dans une fonction en langage C intitulée « Kernel », en utilisant le prescripteur de déclaration « _global_ ». La plateforme CUDA exécute en parallèle plusieurs instances du traitement indiqué dans le Kernel, par opposition à l'exécution régulière des fonctions de C. Le nombre des instances N est spécifié lors de l'appel du Kernel dans le programme principal, en utilisant la syntaxe de configuration d'exécution $<<< x, N >>>$, avec x est le nombre des blocs, telle que affichée dans le code de la figure A-2.

```
/* Déclaration du kernel */
__global__ void Nom_du_kernel (parametres) {  }
/* Programme principal */
int main()
{
/* Appel du kernel */
Nom_du_kernel <<<1, N>>> (parametres);
}
```

Figure A-2 Déclaration et appel du Kernel

Le modèle de programmation CUDA suppose également que l'hôte et le périphérique disposent de leurs propres espaces de mémoire séparés dans la mémoire DRAM, dénommé respectivement mémoire de l'hôte et mémoire du périphérique. Lors du lancement d'un Kernel, CUDA consiste à réserver un espace mémoire pour y stocker les données d'entrée/sortie du Kernel. Cet espace mémoire est à libérer après l'exécution du Kernel.

A.2. Principe de l'ordonnancement des nids de boucles

CUDA propose une démarche d'ordonnancement des nids de boucles permettant d'exécuter en parallèle toutes les itérations (si le nombre des cores disponibles le permet). Chaque itération est allouée à un thread, appartenant à un même bloc. De ce fait, le bloc doit être définie de façon que le nombre des threads blockDim est supérieur ou égal au nombre des itérations. Prenons l'exemple d'un nid de boucles assurant l'addition des matrices A et B dans la matrice C, dont leurs dimensions sont $N \times N$. Les termes $(blockIdx.x \times blockDim.x)$ et $(blockIdx.y \times blockDim.y)$ identifient le bloc dans la grille. Les termes de $(threadIdx.x)$ et $(threadIdx.y)$ identifient les threads à l'intérieur du bloc. Pour allouer chaque itération à un thread, les itérateurs des cellules des matrices i et j sont exprimés en utilisant les identificateurs du bloc, et les identificateurs du thread dans le bloc, tels que affichés dans le code de la figure A-3. Suite à la déclaration des itérateurs, les instructions du traitement itératif sont appelées dans une structure conditionnelle testant les valeurs maximales des itérations.

Nous avons exécuté un nid de boucles avec des dépendances de données inter-itérations, dont le traitement est indiqué dans le code de la figure A-4. Ayant les matrices A et B de taille 10×10 et initialisées respectivement à 1 et à 0, le code incrémente les valeurs $B[i][j]$ en fonction de l'itérateur j : si $B[i][j] = X$ alors $B[i][j+1] = X + 1$.

Cependant, l'exécution du Kernel génère des résultats erronés : toutes des cellules de la matrice B sont égales à 1, tels que indiqués dans le terminal d'exécution de la figure A-5. Due à l'exécution parallèle des itérations, chaque cellule $B[i][j]$ est calculée par l'addition de la valeur initiale de $B[i][j] = 0$ et la valeur $A[i][j] = 1$. En fait, ce mode d'exécution consiste à affecter chaque itération à un core, afin de les exécuter en parallèle. D'où, elle ne prend pas en

considération les dépendances de données inter-itérations. Par conséquent, cette démarche n'est pas fonctionnelle pour tous les cas des structures des nids de boucles, en particulier pour les nids de boucles ayant des dépendances de données inter-itérations.

```
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
/* Définir les équations des itérateurs */
 int i = blockIdx.x * blockDim.x + threadIdx.x;
 int j = blockIdx.y * blockDim.y + threadIdx.y;
/* Appel du traitement itératif */
 if (i < N && j < N)
       /* le traitement itératif */
}
```

Figure A-3 Structure du nid de boucles par indexation des itérations

```
__global__ void gpu_matrixadd(int *B,int *A, int N)
{
int i = threadIdx.x + blockDim.x * blockIdx.x;
int j = threadIdx.y + blockDim.y * blockIdx.y;
If  (i < N && j < N)
       {
       B[i*N+j] = B[i*N+(j-1)] + A[i*N+j];
       }
}
```

Figure A-4 Code du nid de boucles par indexation des itérations

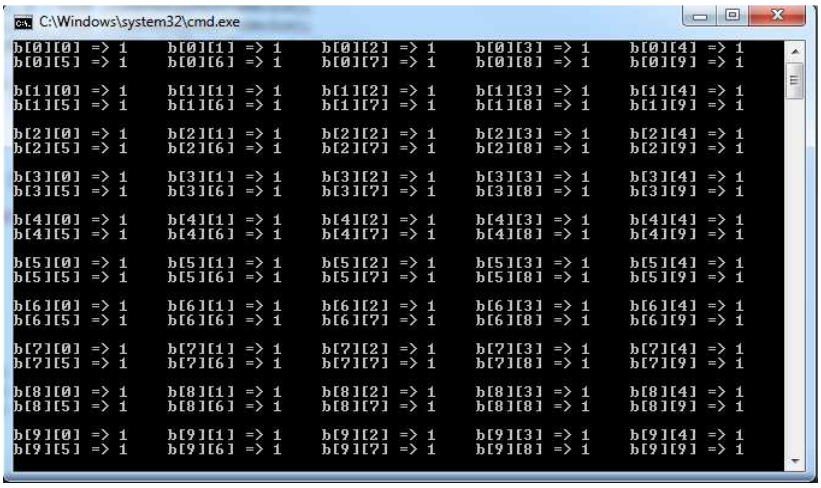


Figure A-5 Exécution du nid de boucles par indexation des itérations

Dans ce contexte, nous procédons à l'utilisation des structures « For {} » pour la modélisation des boucles imbriquées. Pour assurer une exécution dans l'ordre des dépendances de données, nous ajoutons l'instruction «__syncthreads() » dans l'objectif d'exécuter entièrement une itération, avant d'exécuter la suivante. Nous affichons dans la figure A-7 le terminal d'exécution du code de la figure A-6 utilisant les structures « For ». Elle valide l'exactitude des valeurs générées par le nid de boucles, prouvant ainsi la prise en considération des dépendances de données inter-itérations.

```
__global__ void gpu_matrixadd(int *B,int *A, int N)
{ for(int i=0; i<N; i++)
{    for(int j=0;j<N;j++)
     {
       B[i*N+j] = B[i*N+(j-1)] + A[i*N+j];
         __syncthreads() ;
     }
} }
```

Figure A-6 Code du nid de boucles en utilisant « For »

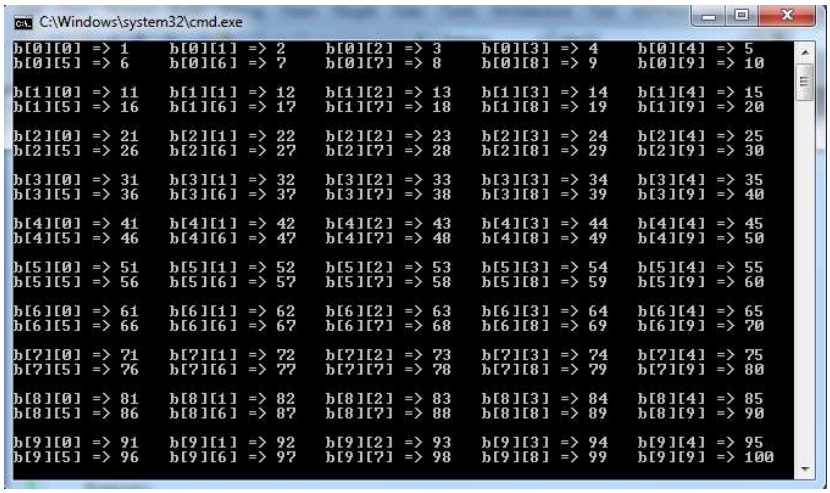


Figure A-7 Exécution du code du nid de boucles en utilisant « For »

A.3. Parallélisme des instructions

Notre travail consiste à identifier les traitements indépendants pour les exécuter en parallèle. Ces traitements diffèrent du point de vue des instructions de calcul, dont chacun doit être alloué à un cœur distinct. L'environnement CUDA consiste à exécuter en parallèle un ensemble de threads, regroupés dans un même *wrap*, ayant des traitements identiques. Les traitements à exécuter en parallèle ne doivent pas appartenir à un même *warp*. En fait, CUDA permet de définir des *wraps* ayant des traitements différents et pouvant être exécutés en parallèle. De ce fait, nous procédons à affecter chacun des traitements à exécuter en parallèle à un *warp* différent. Chaque traitement est alloué à un thread dont l'identifiant appartient à l'intervalle des 32 threads du *wrap* correspondant.

Prenons le cas du filtre numérique d'ondelette, dont nous visons à exécuter les instructions B et C en parallèle, telles que schématisées dans l'ordonnancement statique de la figure 3.2(c). Pour affecter chacune des instructions B et C à un *warp* distinct, on fait recours

à une structure conditionnelle testant les identificateurs des threads : l'instruction B est affecté au premier *warp* en exigeant que les identifiants des threads soient compris entre 0 et 31. Par la suite, nous affectons l'instruction C au deuxième *warp* dont les threads sont identifiés du 32 et 63, telles que décrites dans le code de la figure A-8.

Nous rappelons que l'appel du Kernel engendre la déclaration du nombre des threads nécessaires. Pour le cas d'une exécution parallèle de N traitements, l'exécution du Kernel nécessite $N \times 32$ threads. Pour le cas du filtre numérique d'ondelettes, deux *warps* sont nécessaires pour l'exécution de ce code, et ainsi nécessite 64 threads à déclarer lors de l'appel du Kernel.

```
if ((tid>=0)&&(tid<=31))
/* warp numéro 1 */
{   B[i*N1+j]= A[i*N1+j]+ 1;  }
Else  If ((tid>=32)&&(tid<=63))
/* warp numéro 2 */
{   C[i*N1+j]= A[i*N1+j]+ 1;  }
```

Figure A-8 Exécution parallèle des instructions

A.4. Parallélisme d'un chemin de données

Un chemin de données peut être réparti en plusieurs fragments d'instructions en se basant sur ses dépendances de données. Ces fragments doivent être exécutés d'une façon séquentielle, tels que soit les traitements qu'ils assurent. De plus, leurs instructions sont à exécuter dans le même espace temporel. Pour respecter cette ordre d'exécution, nous insérons des instructions de synchronisation dans le Kernel en appelant la fonction « __syncthreads () ». Cette fonction agit comme une barrière à laquelle tous les threads en amont doivent être exécutés avant de lancer l'exécution des threads en aval. Reprenons l'exemple du filtre numérique d'ondelette, dont nous distinguons trois fragments de code à exécuter en séquentiel qui sont respectivement {D}, {A} et {B, C}. Pour assurer un ordre d'exécution, on insère deux instructions « __syncthreads() » telles que indiquées dans le code de la figure A-9.

```
D[i*N1+j]= B[i*N1+j]*C[i*N1+j];
 __syncthreads();
 A[i*N1+j]= D[i*N1+j]*5;
__syncthreads();
 if ((tid>=0)&&(tid<=31))
/* warp numéro 1 */
{   B[i*N1+j]= A[i*N1+j]+ 1;  }
Else  If ((tid>=32)&&(tid<=63))
/* warp numéro 2 */
{   C[i*N1+j]= A[i*N1+j]+ 1;  }
```

Figure A-9 Synchronisation des instructions

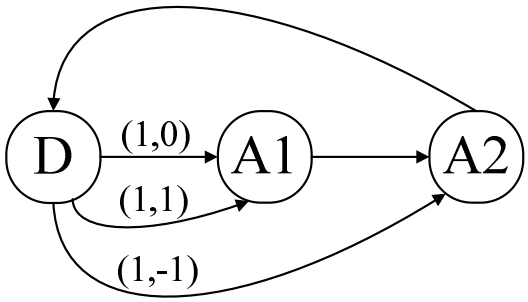
Annexe B : Génération des exécutables de l'algorithme de JACOBI

B.1. Ordonnancement de l'application initiale

L'algorithme de JACOBI, schématisé dans la figure B-1(a), est composé de deux boucles imbriquées. Les dépendances de données engendrent des délais dont les valeurs sont affichées dans le GFDM de la figure B-1(b).

Pour t de 1 à T-1 faire
Pour i de 2 à N−2 faire
A1 : X ← a[t -1,i] + a[t -1,i-1]
A2 : Y← X + a[t -1,i+1]
D : a[t,i] ← Y /3
Fin pour
Fin pour

(a)



(b)

Figure B-1 Application de JACOBI : (a) l'algorithme, (b) le GFDM, (c) l'ordonnancement statique

Un vecteur d'ordonnancement $s(x, y)$ est tout vecteur vérifiant $s \times d(e) > 0$ pour tous $e \in E$. Le vecteur s est à déterminer à partir des trois inéquations suivantes :

$$(1,0) \times (x, y) > (0,0)$$

$$(1,1) \times (x, y) > (0,0)$$

$$(1, -1) \times (x, y) > (0,0)$$

Ce qui signifie que :

$$x > 0$$

$$x > |y|$$

Par ailleurs, un vecteur d'ordonnancement doit être sélectionné de façon que la somme $(x + y)$ est minimale. D'où, le vecteur d'ordonnancement s du GFDM est égal à (1,0). L'algorithme est ordonnancé suivant l'ordre croissant de l'itérateur de la boucle interne, tel que affiché dans l'ordonnancement statique de la figure B-2. Le progranne du Kernel est affiché dans la figure B-3.

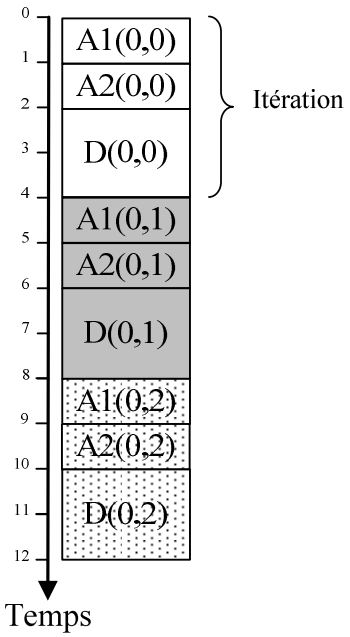


Figure B-2 Ordonnancement statique de l'application de JACOBI

```
__global__ void filtre( int *X, int *Y, int *a, int *H, float *myclock1)
{
clock_t str = clock();
for(int m=1;m< ROWS; m++)
{  for (int n=2; n< COLUMNS; n++)
  {                 X[m * N + n] = a[(m-1) * N + n] + a[(m-1) * N + (n-1)];
                    Y[m * N + n] = X[m * N + n] + a[(m-1) * N + (n+1)];
                    a[m * N + n] = Y[m * N + n] / H[1 * N + 1];
  }  }
 clock_t stp = clock();
*myclock1  = (((float)stp - (float)str)/(CLOCKS_PER_SEC))/1000000;
}
```

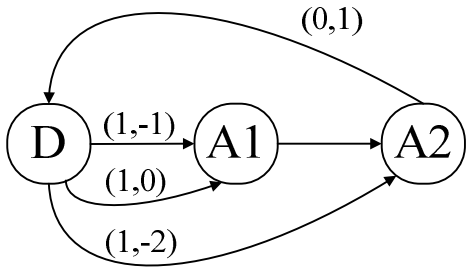
Figure B-3 Code de l'application de JACOBI

B.2. Ordonnancement après la technique du retiming décalé

Une fonctions de retiming multidimensionnel r est un vecteur orthogonale à s. D'où, nous choisissons la fonction $r = (0,1)$. En se basant sur les temps d'exécution des nœuds du GFDM tels que affichés dans le tableau 4-7, la technique sélectionne le chemin $A1 \rightarrow A2$ pour le décaler, dont le GFDM et l'ordonnancement sont affichés respectivement dans la figure B-4 et la figure B-5.

Pour t de 1 à T-1 faire
A1 : X ← a[t -1,2] + a[t -1,1]
A2 : Y← X + a[t -1,3]
Pour i de 2 à N−3 faire
A1 : X ← a[t -1,i+1] + a[t -1,i]
A2 : Y← X + a[t -1,i+2]
D : a[t,i] ← Y /3
Fin pour
D : a[t,N-2] ← Y /3
Fin pour

(a)



(b)

Figure B-4 Application de JACOBI retiming multidimensionnel décalé : (a) l'algorithme ; (b) le GFDM

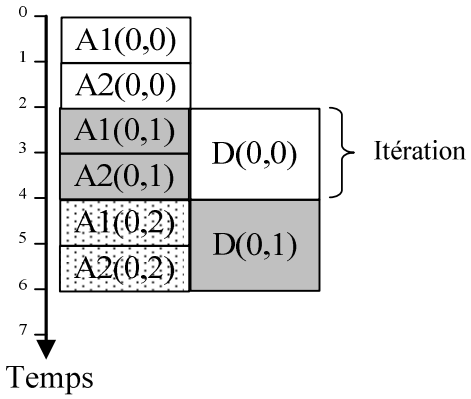


Figure B-5 Ordonnancement de l'application de JACOBI après retiming multidimensionnel décalé

```
__global__ void filtre( int *X, int *Y, int *a, int *H, float *myclock1)
{
int  j =   blockIdx.x * blockDim.x + threadIdx.x;
int  i =   blockIdx.y * blockDim.y + threadIdx.y;
int tid= i * N + j;   clock_t str = clock();
for(int m=1;m< ROWS; m++) {
       X[m * N + 1] = a[(m-1) * N + 1] + a[(m-1) * N + 0];
       Y[m * N + 1] = X[m * N + 1] + a[(m-1) * N + 2];
       for (int n=1; n< COLUMNS; n++)
       {     if ((tid>=0)&&(tid<=31)) {a[m * N + n] = Y[m * N + n] / H[1 * N + 1];}
              Else if ((tid>=32)&&(tid<=63))
              {  X[m * N + (n+1)] = a[(m-1) * N + (n+1)] + a[(m-1) * N + n];
                 Y[m * N + (n+1)] = X[m * N + (n+1)] + a[(m-1) * N + (n+2)];
              } __syncthreads();
       }
   a[m * N + b1] = Y[m * N + b1] / H[1 * N + 1]; }
clock_t stp = clock();
*myclock1  = (((float)stp - (float)str)/(CLOCKS_PER_SEC))/1000000;
}
```

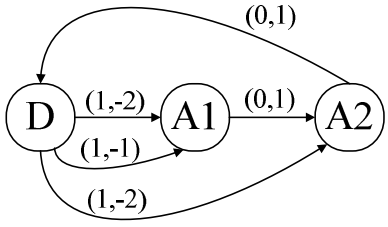
Figure B-6 Code de l'application de JACOBI après retiming multidimensionnel décalé

B.3. Ordonnancement après la technique du retiming enchaîné

Sachant que $r = (0,1)$ est une fonction de retiming optimale, la technique de retiming multidimensionnel enchaîné applique $(r \times 2)(A1)$ puis $(r \times 1)(A2)$, dont l'algorithme et le GFDM finaux sont affichés respectivement dans la figure B-7.

Pour t de 1 à T-1 faire
A1 : X ← a[t -1,2] + a[t -1,1]
A1 : X ← a[t -1,3] + a[t -1,2]
A2 : Y← X + a[t -1,3]
Pour i de 2 à N−3 faire
A1 : X ← a[t -1,i+2] + a[t -1,i+1]
A2 : Y← X + a[t -1,i+2]
D : a[t,i] ← Y /3
Fin pour
A2 : Y← X + a[t -1,N-2]
D : a[t,N-1] ← Y /3
D : a[t,N-2] ← Y /3
Fin pour

(a)



(b)

Figure B-7 Application de JACOBI après retiming multidimensionnel enchaîné : (a) l'algorithme ; (b) le GFDM

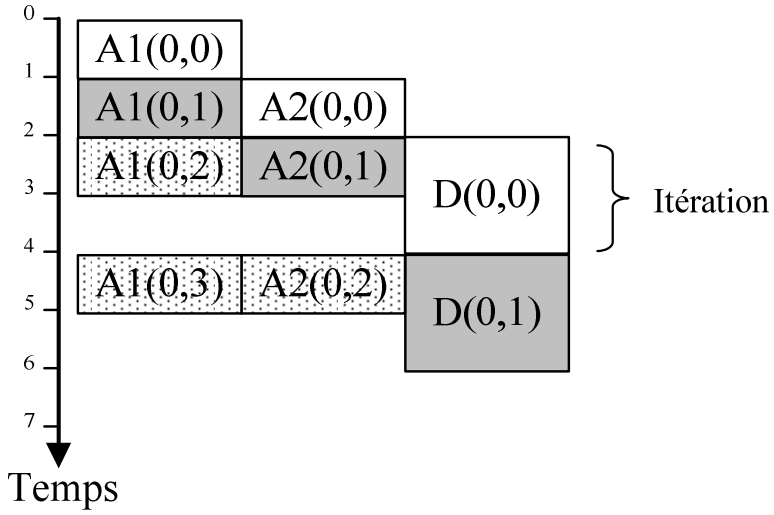


Figure B-8 Ordonnancement statique de l'application de JACOBI après retiming multidimensionnel enchaîné

```
__global__ void filtre( int *X, int *Y, int *a,int *H, float *myclock1)
{
int  j =   blockIdx.x * blockDim.x + threadIdx.x;
int  i =   blockIdx.y * blockDim.y + threadIdx.y;
int tid= i * N + j;
clock_t str = clock();
for(int m=1;m< ROWS; m++)
    {
    X[m * N + 1] = a[(m-1) * N + 1] + a[(m-1) * N + 0];
    __syncthreads();
    if ((tid>=0)&&(tid<=31))
    {
    X[m * N + 2] = a[(m-1) * N + 2] + a[(m-1) * N + 1];
    }
    else if ((tid>=32)&&(tid<=63))
    {
    Y[m * N + 1] = X[m * N + 1] + a[(m-1) * N + 2];
    }
    for (int n=1; n< COLUMNS; n++)
        {
          if ((tid>=0)&&(tid<=31))
           {
           X[m * N + (n+2)] = a[(m-1) * N + (n+2)] + a[(m-1) * N + (n+1)];
           }
          else if ((tid>=32)&&(tid<=63))
           {
           Y[m * N + (n+1)] = X[m * N + (n+1)] + a[(m-1) * N + (n+2)];
           }
          else if ((tid>=64)&&(tid<=95))
           {
            a[m * N + n] = Y[m * N + n] / H[1 * N + 1];
           }
                __syncthreads();
        }
    __syncthreads();
    if ((tid>=0)&&(tid<=31))
    {
    a[m * N + c1] = Y[m * N + c1] / H[1 * N + 1];
```

```
    }
    Else if ((tid>=32)&&(tid<=63))
    {
    Y[m * N + c1] = X[m * N + c1] + a[(m-1) * N + a1];
    }
    __syncthreads();
   a[m * N + b1] = Y[m * N + b1] / H[1 * N + 1];
   }
 clock_t stp = clock();
*myclock1  = (((float)stp - (float)str)/(CLOCKS_PER_SEC))/1000000;
}
```

Figure B-6 Code de l'application de JACOBI après retiming multidimensionnel enchaîné