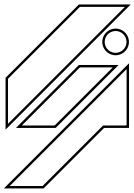
Retiming et parallélisation automatique.

Guillaume Huard



Laboratoire de l'Informatique du Parallélisme

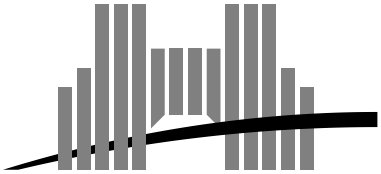
École Normale Supérieure de Lyon
Unité de recherche associée au CNRS n° 1398

Retiming et parallélisation automatique

Guillaume Huard
sous la direction d'Alain Darte

8 Juillet 1998

Research Report N° 98-33



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.00
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Retiming et parallélisation automatique

Guillaume Huard
sous la direction d'Alain Darte

8 Juillet 1998

Abstract

In this report, we study more deeply the retiming techniques that are useful both for automatic parallelization and architecture synthesis. We recall the formalism of retiming and the main results due to Leiserson and Saxe. We propose two new optimization results: the minimization and the maximization of the number of registerless edges of a synchronous circuit. These two optimizations appear for problems such as the software pipelining and the maximization of data locality.

Keywords: automatic parallelization, retiming, software pipelining, data locality.

Résumé

Nous nous proposons dans le cadre de ce rapport d'étudier plus avant les techniques de *retiming* utiles à la fois en synthèse d'architectures et en parallélisation automatique. Nous en présentons le formalisme et nous rappelons les principaux résultats obtenus par Leiserson et Saxe. Nous proposons deux nouveaux résultats d'optimisation sur cette technique : la minimisation et la maximisation du nombre d'arcs sans registres d'un circuit synchrone. Ces deux problèmes apparaissent notamment dans le cadre du pipeline logiciel et de la maximisation de la localité des données.

Mots-clés: parallélisation automatique, retiming, pipeline logiciel, localité des données.

Retiming et parallélisation automatique

Guillaume Huard sous la direction d'Alain Darte

8 Juillet 1998

Résumé

Actuellement, les techniques de parallélisation automatique comme le pipeline logiciel ou la parallélisation de boucles prennent une importance croissante. Elles visent aussi bien à exploiter les ressources disponibles dans les processeurs VLIW qu'à transformer du code séquentiel en code parallèle destiné à des langages comme HPF.

L'étude de ces techniques révèle que plusieurs d'entre elles mettent en avant un formalisme et un ensemble de problèmes connus sous le nom de *retiming* dans le cadre de la synthèse architecturale. Le *retiming*, introduit par Leiserson et Saxe en 1983, bien plus qu'une simple technique d'optimisation de circuits synchrones, s'avère un outil puissant pour la manipulation de graphes. Son utilisation dans le cadre de la parallélisation automatique a pu révéler sa généralité en réutilisant les résultats obtenus par Leiserson et Saxe. D'autre part la mise en avant de nouveaux problèmes solubles par *retiming* est venue confirmer son ouverture.

Nous nous proposons dans le cadre de ce rapport d'étudier plus avant les techniques de *retiming*. Nous en présentons le formalisme et nous rappelons les principaux résultats obtenus par Leiserson et Saxe. Nous proposons deux nouveaux résultats sur cette technique et nous étudions de quelle façon ils interviennent dans le cadre de la parallélisation automatique. Enfin nous présentons les nouveaux problèmes qui se posent, puis nous concluons sur les évolutions actuelles du *retiming* dans le cadre de la parallélisation automatique.

Table des matières

1 Introduction **3**

2 Introduction aux techniques de *retiming* **5**
2.1 Principe et formalisme 5
2.1.1 Circuits synchrones 5
2.1.2 *Retiming* 6
2.2 Optimisation des circuits synchrones 7
2.2.1 Minimisation de la période d'horloge 7
2.2.2 Minimisation du nombre de registres 7
2.2.3 Mise en œuvre 8
2.2.4 Variantes 8
2.3 Nouveaux critères d'optimisation 8

3 Minimisation du nombre d'arcs de poids nul **10**
3.1 Formulation du problème 10
3.2 Un algorithme naïf 11
3.3 Un algorithme primal-dual 11
3.3.1 Condition d'optimalité 12
3.3.2 Caractérisation des arcs 13
3.3.3 Algorithme 14
3.3.4 Complexité 15
3.4 Prise en compte de la période d'horloge 16
3.4.1 Interprétation des contraintes 16
3.4.2 Modification de l'algorithme 16

4 Maximisation du nombre d'arcs de poids nul **18**
4.1 Positionnement du problème 18
4.2 NP-complétude 18
4.2.1 Transformation 19
4.2.2 Réduction 19

5 Applications **20**
5.1 Graphes de dépendance 21
5.2 Ordonnancements et problèmes de *retiming* 22
5.2.1 Pipeline logiciel 22
5.2.2 Maximisation de la localité 24
5.2.3 Exemple 24
5.3 Alignement de données et *retiming* 27
5.4 Autres problèmes de *retiming*, extensions 27
5.4.1 Parallélisation de boucles 27
5.4.2 Recherche d'ordonnancements K-périodiques 27
5.4.3 *Retiming* multidimensionnel 28

6 Conclusion **28**

1 Introduction

De nos jours, l'incessante montée en fréquence des composants et leur complexité croissante s'approchant pas à pas des limites physiques des matériaux utilisés imposent la mise en œuvre d'optimisations à tous les niveaux de la conception d'un circuit.

C'est dans ce cadre que Leiserson et Saxe introduisirent en 1983 une technique souple et robuste pour l'optimisation de circuits qu'ils désignèrent sous le nom de *retiming* (voir [LS91]). D'une mise en œuvre simple, le *retiming* se fonde sur la mobilité des éléments de synchronisation (ou registres) d'un circuit afin d'en modifier les propriétés sans affecter son comportement fonctionnel.

Consistant donc en une relocalisation temporelle des parties fonctionnelles du circuit, le *retiming* affecte en conséquence plusieurs caractéristiques de celui-ci.

La première et la plus importante est désignée sous le nom de période d'horloge et possède une influence directe sur la fréquence maximale de fonctionnement d'un circuit. Le problème consistant à atteindre la période d'horloge minimale (c'est-à-dire la fréquence maximale) réalisable par *retiming* a été résolu efficacement par Leiserson et Saxe. Ils ont montré qu'il pouvait s'exprimer sous forme d'arcs supplémentaires dans le graphe représentant le circuit, permettant ainsi de le combiner à d'autres critères.

La seconde caractéristique généralement considérée dans la mise au point d'un circuit est son nombre total de registres, influant de façon évidente sur sa consommation et sa taille. Là encore, le *retiming* permet de modifier cette quantité et ainsi de gagner en espace et en puissance. Leiserson et Saxe ont également proposé une résolution efficace de ce problème, avec ou sans contrainte sur la période d'horloge.

Enfin, de nombreuses recherches ont été consacrées à l'intégration des techniques de *retiming* dans le cadre complexe de la synthèse de circuits. Selon les cas, des optimisations ([MS98]) ont pu être trouvées ou au contraire des preuves de complexité sont venues confirmer la difficulté de la conception d'un circuit optimal vis-à-vis de multiples critères ([dFAK+96]).

Pourtant il semble légitime de se demander si toutes les possibilités du *retiming* ont été exploitées. En effet, d'un point de vue plus global, le retiming apparaît plus comme une technique de graphe que comme une méthode d'optimisation spécifique aux circuits synchrones. De plus nous pouvons constater qu'il intervient effectivement hors de son cadre d'origine, entre autres dans de nombreuses techniques de parallélisation automatique. C'est pourquoi nous nous intéressons ici à deux nouveaux critères d'optimisation de graphes par *retiming*, tous deux dépourvus de sens dans le cadre de la synthèse d'architectures, mais qui se révèlent tout a fait intéressants en parallélisation automatique.

Tout d'abord nous étudierons le problème de la minimisation du nombre d'arcs de poids nul d'un graphe par *retiming*. Nous verrons que ce problème déjà évoqué dans [CDR96] était connu comme polynomial mais, jusqu'alors, n'était pas résolu autrement que par programmation linéaire. Nous proposerons alors un algorithme de graphe efficace pous la résolution de ce problème, permettant d'obtenir une meilleure complexité que celle de la solution précédente. Nous verrons également qu'il est possible d'ajouter à ce problème une contrainte sur la période d'horloge du graphe après *retiming* tout en restant efficace.

Nous étudierons ensuite le problème inverse, c'est-à-dire la maximisation du nombre d'arcs de poids nul par retiming. Une variante de ce problème autorisant un retiming non légal est proposée dans [DR93] ainsi qu'une preuve de NP-complétude au sens faible de ce sous-problème. Nous prouvons ici que le problème est NP-complet au sens fort dans tous les cas (*retiming* légal ou non) généralisant et complétant ainsi le résultat de Darte et Robert.

Nous verrons enfin un certain nombre d'applications en parallélisation automatique faisant

apparaître un de ces deux problèmes et illustrant leur intérêt.

Nous commencerons par le pipeline logiciel, qui fait partie d'un ensemble de techniques visant à exploiter le parallélisme contenu dans le corps des boucles. A l'heure actuelle, avec l'apparition de processeurs superscalaires ou plus récemment VLIW (Very Long Instruction Word), ces techniques prennent une importance croissante. Le but ici est de déterminer un ordre d'exécution des opérations permettant de conserver la sémantique du résultat tout en maximisant le nombre d'opérations exécutées à la même date (c'est-à-dire en parallèle).

La sémantique est préservée en procédant à une analyse des dépendances entre instructions, c'est-à-dire de l'ordre partiel d'exécution imposé par leurs effets de bord afin de garantir le résultat. L'ensemble de ces dépendances peut être représenté par un graphe appelé graphe de dépendance dont la manipulation servira à déterminer un ordonnancement valide des opérations (un ordre d'exécution respectant la sémantique du code original).

Il s'avère que la représentation choisie ainsi que la formulation d'un ordonnancement font apparaître l'expression d'un *retiming* agissant sur la valeur des dépendances. Ainsi il est possible d'en déduire de nouveaux critères d'optimisation permettant la mise en parallèle du maximum d'opérations par *retiming*. Ces critères d'optimisation partagent certains résultats utilisés pour les circuits synchrones (minimisation de la période d'horloge), mais mettent également en avant de nouveaux objectifs à atteindre (comme la minimisation du nombre total d'arcs de poids nul).

Nous considérerons ensuite le cas de la maximisation du nombre d'arcs de poids nul qui intervient dans plusieurs techniques de maximisation de la localité des données. Cette localité est d'une importance cruciale pour deux raisons : d'une part elle permet de maximiser l'utilisation de la mémoire cache et ainsi d'éviter de coûteux accès mémoire, d'autre part dans le cadre du parallélisme sur machines à mémoire distribuée elle est d'autant plus importante qu'un accès non local correspond à une communication, c'est-à-dire à une perte dépassant largement celle des accès mémoire.

Enfin nous parlerons de récentes recherches qui semblent s'être intéressées à l'extension des techniques de *retiming*. Nous pouvons citer, entre autres, [PS96] où le *retiming* devient multidimensionnel afin de saturer un nombre de ressources plus important, ou [SC93] où le *retiming* est combiné à un déroulage du graphe de dépendance afin de minimiser l'ordonnancement final.

Finalement, le *retiming* apparaît comme un point de jonction entre plusieurs domaines plus ou moins éloignés les uns des autres. D'un point de vue global il semble intervenir dans un ensemble de techniques de manipulation du temps relatif entre diverses tâches.

Nous tenterons donc dans le cadre de notre étude de présenter ses différentes facettes en donnant un aperçu des divers domaines dans lesquels il intervient. En résumé, ce rapport sera organisé suivant le plan suivant. Nous commencerons par rappeler en section 2 les bases historiques et le formalisme du *retiming*, ainsi que les principaux objectifs qu'il permet d'atteindre. Nous présenterons alors deux nouveaux résultats sur cette technique, à savoir un algorithme de graphe efficace pour la minimisation du nombre d'arcs de poids nul par *retiming* dans la section 3, et une preuve de forte NP-complétude du problème inverse, la maximisation du nombre d'arcs de poids nul par *retiming*, dans la section 4. Nous verrons alors, en section 5, comment le *retiming* s'est étendu à d'autres domaines et nous illustrerons l'utilité des nouveaux problèmes mis en évidence dans les sections précédentes, enfin nous compléterons notre étude par une discussion sur l'état actuel de cette technique et sur les perspectives de recherches qu'elle continue d'offrir.

2 Introduction aux techniques de *retiming*

Notre objectif dans cette section est de présenter, de manière générale et si possible synthétique, l'ensemble des techniques de *retiming* intervenant dans l'optimisation de circuits et la parallélisation automatique. Nous souhaitons également donner des pistes plus détaillées destinées au lecteur intéressé, ainsi que les voies de recherche encore ouvertes.

Les techniques de *retiming* ont été introduites par E. Leiserson et J.B. Saxe ([LS91]) dans le cadre de la synthèse architecturale et plus précisément de l'optimisation de circuit, elles sont fondées sur la mobilité des éléments de synchronisation, ou registres, et permettent l'amélioration d'un circuit rapidement conçu offrant ainsi un meilleur compromis temps de développement / performance.

La simplicité et l'efficacité du *retiming* ont permis son évolution rapide et son intégration dans la synthèse réelle de circuit ([MS98]) grâce à un certain nombre d'optimisations des techniques initiales.

Nous présentons donc dans un premier temps le principe et le formalisme utilisés dans la mise en œuvre du *retiming*. Nous verrons ensuite quels sont les critères d'optimisation des circuits synchrones et comment les satisfaire par *retiming*. Enfin nous présenterons deux problèmes de *retiming* encore ouverts dont nous apporterons la résolution dans les sections suivantes.

2.1 Principe et formalisme

Nous nous intéressons ici au principe du *retiming*, et nous introduisons les notations utilisées par la suite. Nous commençons par une présentation des circuits synchrones et de leur représentation telle qu'elle apparaît dans [LS91], sur lesquels nous appliquerons le *retiming* proprement dit.

2.1.1 Circuits synchrones

Un circuit synchrone peut être vu comme un ensemble d'unités fonctionnelles séparées par un certain nombre (éventuellement nul) de registres cadencés par une même horloge globale. Le terme synchrone désigne un circuit contenant un nombre non négatif de registres entre chaque unité fonctionnelle, et au moins un registre par cycle. Nous pouvons définir un registre comme un élément possédant une entrée et une sortie, fournissant en sortie un signal stable défini à chaque top d'horloge par son entrée courante. Les unités fonctionnelles sont des unités de calcul définissant leur sortie comme une fonction de leurs entrées, sous réserve que celles-ci soient stables durant un délai supérieur ou égal à leur délai de propagation.

Plus formellement nous modélisons un circuit par un graphe dirigé $G = (V, E, w, d)$, où V représente l'ensemble des unités fonctionnelles, $E \subset V^2$ l'ensemble des interconnections entre unités fonctionnelles, $w : E \longrightarrow \mathbb{N}$ une pondération des arcs de G indiquant le nombre de registres séparant deux unités fonctionnelles, telle que tout cycle ait un poids strictement positif, et $d : V \longrightarrow \mathbb{N}$ une pondération des sommets indiquant le délai de propagation d'une unité fonctionnelle. La figure (1) illustre ce formalisme.

Dans toute la suite, nous désignerons par $u \xrightarrow{e} v$ l'existence d'un arc e entre u et v et par $u \overset{p}{\rightsquigarrow} v$ l'existence d'un chemin p entre u et v. De plus, nous étendrons le délai (resp. le poids) aux chemins en définissant le délai (resp. le poids) d'un chemin comme la somme des délais (resp. des poids) de ses sommets (resp. de ses arcs).

Nous pouvons caractériser un circuit G par une quantité spécifique appelée période d'horloge et notée $\Phi(G)$. La période d'horloge d'un circuit définit le temps durant lequel les sorties des registres doivent être stables afin de garantir la stabilité de toutes les entrées, elle est égale par définition à

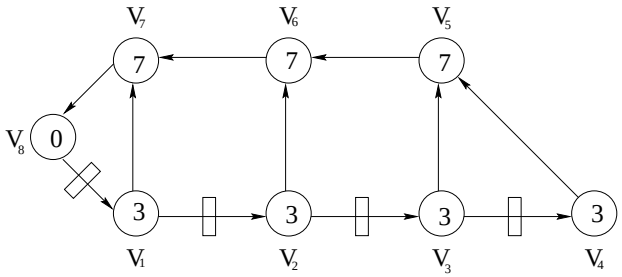


FIG. 1: Un exemple de circuit

la somme maximale des délais des unités fonctionnelles présentes sur un chemin sans registres du circuit. Nous verrons que la période d'horloge d'un circuit est d'une importance cruciale pour de nombreux problèmes de *retiming*, en particulier, pour les circuits synchrones, elle correspond à une borne à la fréquence maximale d'horloge pouvant être utilisée.

2.1.2 *Retiming*

Le *retiming* est une technique agissant sur les registres d'un circuit synchrone. Plus précisément, il définit une redistribution de l'ensemble des registres du circuit correspondant directement à un décalage dans le temps de l'exécution de ses différentes parties (le *retiming* s'avère équivalent à une technique connue sous le nom de dérive d'horloge permettant de mieux cerner cet aspect temporel, cf. [MS98]). Définir un *retiming* consiste donc à assigner à chaque unité fonctionnelle une valeur représentant la quantité de temps, en termes de tops d'horloge, par laquelle son exécution est retardée par rapport à l'exécution de l'ensemble du circuit (ou avancée dans le cas d'une valeur négative).

Formellement, nous définissons le *retiming* par une application q de l'ensemble des sommets du circuit dans $\mathbb{Z}$. Nous lui associons le poids des arcs après *retiming*, w_q, défini de la manière suivante : $\forall e = (u, v) \in E, w_q(e) = q(v) - q(u) + w(e)$. Afin de conserver une signification physique en tant que circuit au graphe, nous imposons $w_q(e) \geq 0$ pour tout arc $e \in E$. Un *retiming* vérifiant cette condition sera dit légal. La figure (2) illustre les effets du *retiming*.

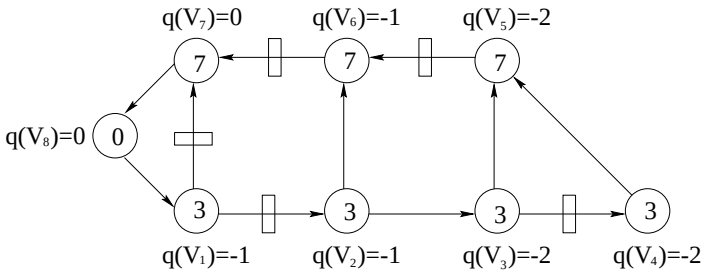


FIG. 2: Un *retiming* du circuit précédent

Nous pouvons constater que le *retiming* possède une influence évidente d'une part sur la période d'horloge d'un circuit et d'autre part sur son nombre total de registres. De plus, Leiserson et Saxe ont montré non seulement que le *retiming* ne modifiait pas le comportement fonctionnel d'un circuit, mais aussi qu'il s'agissait de la transformation la plus générale fondée sur une modification du nombre de registres par arc ne modifiant pas ce comportement.

2.2 Optimisation des circuits synchrones

Nous rappelons ici les principaux critères d'optimisation d'un circuit synchrone pouvant être atteints par le biais du *retiming*. Ces optimisations sont à l'heure actuelle largement intégrées dans la synthèse de circuits, mais nous verrons néanmoins que la combinaison du *retiming* avec d'autres techniques d'optimisation de circuit peut faire exploser la complexité de l'ensemble.

2.2.1 Minimisation de la période d'horloge

La première optimisation, et sans doute la plus importante, est la recherche d'un *retiming* minimisant la période d'horloge du circuit et donc permettant son utilisation à de plus hautes fréquences. Comme nous l'avons vu, le *retiming* agit sur le poids des arcs d'un graphe, il a donc une influence évidente sur la période d'horloge du circuit. La minimisation de celle-ci par *retiming* a été résolue de façon efficace par Leiserson et Saxe dans leur papier original ([LS91]), nous en rappelons ici brièvement le principe.

La minimisation de la période d'horloge d'un circuit se réalise par une recherche dichotomique d'une période réalisable parmi les périodes possibles. La première étape consiste donc à être capable de déterminer si une période d'horloge donnée est réalisable ou non, c'est-à-dire être capable de trouver un *retiming*, s'il en existe un, tel que le graphe après *retiming* ait la période d'horloge souhaitée. Ceci s'obtient en remarquant deux faits :

- le nombre total de registres sur un chemin ne varie que suivant le *retiming* de ses extrémités. Nous notons par $W(u,v) = \min\{w(p) | u \overset{p}{\rightsquigarrow} v\}$ le poids minimal d'un chemin entre deux sommets u et v, nous avons alors $W_q(u,v) = W(u,v) + q(v) - q(u)$;
- le délai induit par un chemin est invariant par *retiming* (en effet le *retiming* n'agit pas sur les délais des nœuds). Nous notons $D(u,v) = \max\{d(p) | u \overset{p}{\rightsquigarrow} v \text{ et } w(p) = W(u,v)\}$ le délai maximal induit par un chemin de poids minimal entre deux sommets u et v, nous avons alors $D_q(u,v) = D(u,v)$.

Il s'agit alors d'imposer un poids non nul sur tout chemin de poids minimal dont le délai maximal dépasse la période d'horloge. L'idée consiste à traduire ceci sous forme d'arcs supplémentaires.

Plus formellement, si c est la période d'horloge souhaitée, pour tout chemin $u \overset{p}{\rightsquigarrow} v \subset E$, tel que $D(u,v) > c$ nous imposons $W(u,v) \geq 1$ en ajoutant un arc de poids $W(u,v) - 1$ entre u et v.

Il est alors possible par un algorithme de type Bellman-Ford de déterminer s'il existe ou non un *retiming* achevant cette période d'horloge et, dans le cas positif, de fournir un tel *retiming* en $O(|V||E|)$. En d'autres termes la recherche d'un *retiming* satisfaisant une certaine période d'horloge se ramène à la recherche d'un *retiming* légal dans le graphe de départ muni d'arcs supplémentaires de contrainte d'horloge que nous appellerons arcs d'horloge.

Il est facile de voir que l'ensemble des périodes d'horloge possibles est inclus dans l'ensemble des délais de chemin possibles, plus précisément il est possible de limiter la recherche à l'ensemble des valeurs de $D(u,v)$. Pour trouver le *retiming* minimisant la période d'horloge, il suffit donc d'effectuer une recherche dichotomique dans cet ensemble, en utilisant la procédure précédente.

Finalement l'ensemble de l'algorithme détermine un *retiming* minimisant la période d'horloge en $O(|V||E|\log|V|)$ (où $|E|$ est en fait borné par $|V|^2$ après l'ajout des arcs d'horloge).

2.2.2 Minimisation du nombre de registres

Lors de la synthèse d'un circuit, minimiser le nombre total de registres revient à minimiser la taille de la mémoire nécessaire à ce circuit, et présente donc un certain intérêt. De même que la

période d'horloge, la minimisation du nombre de registres par *retiming* a été résolue par Leiserson et Saxe lorsqu'ils ont introduit les techniques de *retiming*.

La minimisation du nombre de registres par *retiming* revient à minimiser la somme de quantités locales à chaque sommet. Plus précisément, le nombre global de registres introduit par un *retiming* sur l'ensemble des arcs incidents à un sommet est égal à la valeur du *retiming* sur ce sommet multiplié par la différence entre le nombre d'arcs entrant et le nombre d'arcs sortant de ce sommet. Nous voulons donc :

$$\min \sum_{v \in V} q(v)(d^+(v) - d^-(v))$$

Le problème s'exprime alors sous forme de programme linéaire dont le dual révèle un problème de flot de coût minimum, soluble en temps polynomial. Il s'étend facilement au problème de minimisation du nombre de registres soumis à une contrainte sur la période d'horloge en ajoutant les arcs supplémentaires correspondants.

2.2.3 Mise en œuvre

La mise en pratique des algorithmes présentés précédemment implique souvent l'introduction d'un certain nombre d'optimisations. L'élimination de variables inutiles et de contraintes redondantes permet une nette amélioration des temps de calcul, comme le montrent Maheshwari et Sapatnekar dans [MS98]. En particulier la redondance de certains arcs d'horloge, que Leiserson et Saxe avaient déjà mise en avant, est détaillée dans cet article ainsi que les possibilités d'exploiter la mobilité limitée de certains registres.

En revanche, la combinaison du *retiming* à d'autres techniques d'optimisation de circuit, que nous pourrons appeler *retiming* généralisé, donne souvent naissance à des problèmes NP-complets, par exemple lors de la combinaison du *retiming* avec la répartition temporelle des opérateurs (ou *time folding*), ou encore avec le multiplexage (voir [dFAK+96]).

2.2.4 Variantes

Il semble que les besoins en *retiming* dans le cadre de la synthèse d'architectures se limitent, à l'heure actuelle, aux problèmes de minimisation de la période d'horloge et/ou du nombre total de registres.

Des variantes existent, là encore introduites par Leiserson et Saxe, consistant principalement à prendre en compte un coût de mise en place de registres par arc, ou encore des délais de propagation non uniformes au sein des opérateurs. Nous ne détaillerons pas ces autres points qui sortent du cadre de cette étude.

2.3 Nouveaux critères d'optimisation

Nous présentons dans cette section deux nouveaux problèmes de retiming encore ouverts dont nous proposons la résolution en sections 3 et 4.

Comme nous allons pouvoir le constater, ces critères d'optimisation semblent inutiles dans le cadre de l'optimisation des circuits synchrones, néanmoins nous verrons en section 5 qu'ils interviennent non plus en synthèse d'architectures mais en parallélisation automatique.

Considérons tout d'abord un exemple qui servira d'illustration à notre propos. Soit le graphe $G = (V, E, w)$, pondéré sur ses arcs seulement, décrit par la figure 3. Nous nous proposons d'étudier plus en détail les deux critères d'optimisation suivants :

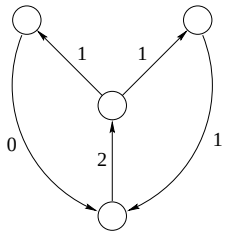


FIG. 3: Un exemple de graphe

– minimisation du nombre d'arcs de poids nul : ce critère intervient dans la mise en place d'heuristiques de pipeline logiciel comme nous le verrons en section 5.2.1. Le problème était connu comme polynomial (cf. [CDR96]), mais nécessitait le recours à la programmation linéaire en nombres entiers avec une matrice de contraintes totalement unimodulaire. Nous proposons en section 3 un algorithme de graphe efficace pour sa résolution ;

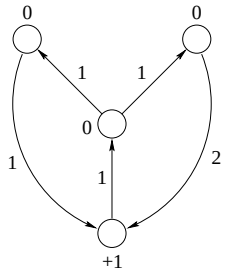


FIG. 4: La minimisation du nombre d'arcs de poids nul du graphe précédent

– maximisation du nombre d'arcs de poids nul : le problème encore peu connu intervient dans le cadre de la maximisation de la localité et de l'alignement des données comme nous le verrons en sections 5.2.2 et 5.3. Une variante faisant intervenir un *retiming* non légal apparaît dans le cadre de l'alignement de données et était connue comme NP-complète au sens faible ([DR93]). Nous prouvons en section 4 que le problème est NP-complet au sens fort dans les deux cas.

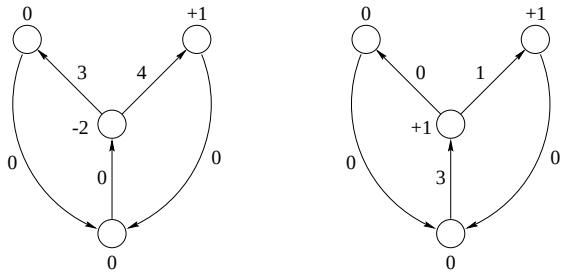


FIG. 5: La maximisation du nombre d'arcs de poids nul du graphe précédent (deux des solutions possibles)

Nous pouvons constater que ces deux critères n'ont apparemment aucun sens dans le cadre de la synthèse architecturale, de plus ils ne correspondent à rien de connu en matière de *retiming*, dans les deux cas ni la période d'horloge (ici les délais ne sont pas pris en compte), ni le nombre

maximal ou minimal de registres (resp. 4 et 7 sur l'exemple) ne sont impliqués dans la résolution des problèmes associés.

Il s'agit donc bien de deux problèmes nouveaux intervenant dans un cadre différent de la synthèse d'architecture que nous nous proposons de résoudre.

3 Minimisation du nombre d'arcs de poids nul

Nous traitons, dans cette partie, de l'utilisation des techniques de *retiming* dans le cadre de la minimisation du nombre d'arcs de poids nul d'un graphe. Comme nous le verrons en section 5.2.1, la résolution de ce problème s'avère utile à la mise en place d'heuristiques de pipeline logiciel.

Nous allons pouvoir constater que l'expression du problème sous forme de programme linéaire nous montre qu'il est polynomial (car la matrice du programme est totalement unimodulaire). Plus exactement, en écrivant astucieusement le dual du programme en question, nous découvrons un probleme de flot de coût minimal dont la resolution est connue. Néanmoins la fonction de coût particulière a prendre en compte dans notre cas complique le problème. En effet, comme nous allons le voir, une résolution directe du problème obtenu par un algorithme classique ne fournit pas la solution la plus efficace.

Nous présentons donc dans un premier temps un algorithme déduit directement du programme linéaire initial résolvant le problème en $O(|V||E|^2)$. Nous proposerons ensuite un algorithme de type primal-dual, moins naturel mais plus efficace, résolvant le problème en $O(|E|^2)$. Nous montrerons également comment combiner la minimisation de la période d'horloge et la minimisation du nombre d'arcs de poids nul du graphe.

3.1 Formulation du problème

Etant donné un graphe pondéré $G = (V, E, d, w)$, le problème consiste à trouver un *retiming* $q : V \to \mathbb{Z}$ tel que :

$$\forall u \xrightarrow{e} v \in E, \quad q(v) - q(u) + w(e) \geq 0 \quad (\textit{retiming} \text{ légal}) \tag{1}$$

$$\text{et} \quad |\{e \in E \mid w_q(e) = 0\}| \quad \text{soit minimal} \tag{2}$$

Notre but est ici de rendre la fonction objective (la quantité à minimiser) linéaire afin de pouvoir utiliser un certain nombre de résultats connus (programmation linéaire). Pour cela nous l'exprimons sous forme de coût dans lequel nous comptons les arcs de poids nul seulement. Nous définissons donc le coût d'un arc comme une application $v : E \to \mathbb{N}$ pour laquelle nous cherchons à obtenir une valeur $v(e)$ strictement positive sur les arcs e de poids nul et nulle sur les autres. En imposant :

$$\forall u \xrightarrow{e} v \in E, \qquad q(v) - q(u) + w(e) \geq 0 \tag{3}$$

$$q(v) - q(u) + w(e) + v(e) \geq 1 \tag{4}$$

et en minimisant $\sum_{e \in E} v(e)$, nous obtenons le résultat voulu (voir [CDR96]).

En effet, pour chaque arc, la contrainte (3) impose à q d'être légal, tandis que la contrainte (4) impose soit un poids non nul après *retiming*, soit une valeur non nulle de v. Trouver q et v de telle sorte que $\sum_{e \in E} v(e)$ soit minimal revient donc à trouver un *retiming* q rendant le nombre d'arcs de poids nul du graphe minimal.

Ici l'idée de simplifier les contraintes précédentes en :

$$\forall u \xrightarrow{e} v \in E, \quad q(v) - q(u) + w(e) + v(e) \geq 1 \tag{5}$$

$$0 \geq v(e) \geq 1 \tag{6}$$

comme nous pouvons le voir dans [CDR96] se révèle mauvaise. En effet dans ce cas, le passage au dual, que nous détaillons dans la suite, fait apparaître une expression de la fonction objective plus complexe dont la résolution s'avère ardue (les auteurs de [CDR96] étaient d'ailleurs restés bloqués sur ce problème).

L'ensemble des contraintes précédentes ((3) et (4)) ainsi que la fonction objective $\min \sum_{e \in E} v(e)$ constituent un programme linéaire pouvant s'écrire sous forme matricielle de la façon suivante :

$$\min \left\{ \begin{pmatrix} q \\ v \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} \middle| \begin{pmatrix} q & v \end{pmatrix} \begin{pmatrix} 0 & C & C \\ Id & Id & 0 \end{pmatrix} \geq \begin{pmatrix} 0 & 1-w & -w \end{pmatrix} \right\} \tag{7}$$

où C est la matrice de connexion de G, et q, v, w sont les vecteurs associés respectivement aux applications q, v, w. Le problème dual peut s'exprimer par ([dW90], p.33-34) :

$$\max \left\{ \begin{pmatrix} 0 & 1-w & -w \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \begin{pmatrix} 0 & C & C \\ Id & Id & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix}, \begin{pmatrix} x \\ y \\ z \end{pmatrix} \geq \mathbf{0} \right\} \tag{8}$$

ou encore

$$\min \{(w-1)y + wz \mid C(y+z) = 0,\ 0 \leq y \leq 1,\ z \geq 0\} \tag{9}$$

La contrainte $C(y+z) = 0$ impose à la quantité $f = y + z$ d'être un flot, c'est-à-dire :

$$\forall v \in V, \sum_{? \xrightarrow{e} v \in E} f(e) = \sum_{v \xrightarrow{e} ? \in E} f(e)$$

Les contraintes $y \geq 0$ et $z \geq 0$, limitent notre recherche aux flots positifs ou nuls que, par la suite, nous appellerons "flots admissibles".

Nous pouvons également remarquer que la connaissance du flux dans un arc e permet de définir sans ambiguïté $y(e)$ et $z(e)$ de telle sorte que le coût de f soit minimal : en effet, on montre sans difficultés que $f(e) > 0 \Rightarrow y(e) = 1$.

3.2 Un algorithme naïf

La formulation précédente du problème comme recherche d'un flot $f = y + z$ de coût minimal nous conduit à distinguer pour chaque arc deux flux $y(e)$ et $z(e)$ de coût différent, et soumis à des contraintes différentes sur leurs valeurs. En d'autres termes cela revient à rechercher un flot de coût minimal dans un graphe $G' = (V, E', d, w')$ déduit de G en doublant les arcs de E de la manière suivante :

$$\forall u \xrightarrow{e} v \in E, \begin{cases} e_1, e_\infty \in E' \\ w'(e_1) = w(e) - 1, w'(e_\infty) = w(e) \\ c(e_1) = 1, c(e_\infty) = \infty \end{cases}$$

où c représente la capacité d'un arc (c'est-à-dire la quantité de flux qu'il peut recevoir).

Le problème ainsi transformé se résoud de manière efficace par la plupart des algorithmes de recherche de flot de coût minimal avec capacité, nous en présentons un en $O(|V||E|^2)$ en annexe.

3.3 Un algorithme primal-dual

Nous présentons ici un algorithme de type primal-dual (jouant sur l'écart de coût d'un couple de solutions admissibles, l'une primale, l'autre duale). Cet algorithme est adapté d'un algorithme de

recherche de flot de coût minimal, connu sous le nom d'algorithme des arcs non conformes ([GM85], p. 152), et dû à Fulkerson (1961).

Soient $G = (V, E, w)$ un graphe pondéré sur ses arcs seulement, $q : V \to \mathbb{Z}$ un *retiming* légal de G, et $f : E \to \mathbb{N}$ un flot admissible de G.

Nous définissons pour chaque arc $u \xrightarrow{e} v \in E$ deux quantités :

$$w_q(e) = q(v) - q(u) + w(e) \quad (10)$$

$$v(e) = \begin{cases} 1 & \text{si } w_q(e) = 0 \\ 0 & \text{sinon} \end{cases} \quad (11)$$

La première, $w_q(e)$, correspond au poids d'un arc après *retiming* par q, la seconde, $v(e)$, est la valeur minimale vérifiant (4) et caractérisant le coût du *retiming* q.

Nous dirons que q est optimal lorsque $\sum_{e \in E} v(e)$ est minimal, c'est-à-dire lorsque q minimise le nombre d'arcs de poids nul de G.

3.3.1 Condition d'optimalité

Nous montrons ici qu'il existe une relation entre q et f, qui représentent respectivement une solution du problème primal et une solution du problème dual. Plus précisément, nous montrons que coût(q)$\geq$coût(f) et qu'il y a égalité à l'optimal.

Soit C_f le multicycle défini par f (tout flot peut se décomposer en combinaison linéaire des cycles élémentaires du graphe, [GM85], p. 135). Nous montrons que le coût d'un flot est indépendant d'un *retiming* du graphe :

proposition 1 $\sum_{e \in C_f} f(e)w(e) = \sum_{e \in C_f} f(e)w_q(e)$.

preuve : la preuve se trouve en annexe.

Ceci nous permet de montrer la proposition suivante, mettant en relation le coût de q et celui de f :

proposition 2 $\sum_{e \in E} v(e) \geq \sum_{e \in C_f} (1 - f(e)w(e))$.

preuve : la preuve se trouve en annexe.

Nous définissons alors pour tout arc $e \in E$ une quantité que nous désignerons sous le nom d'indice de conformité :

$$ic(e) = \begin{cases} v(e) & \text{si } f(e) = 0 \\ v(e) - 1 + f(e)w_q(e) & \text{sinon} \end{cases}$$

$ic(e)$ exprime pour chaque arc e l'écart de coût entre $q(e)$ et $f(e)$.

proposition 3 $\forall e \in E, ic(e) = 0 \Rightarrow$ *q est optimal.*

preuve : la preuve se trouve en annexe.

Nous obtenons donc une borne inférieure au coût du retiming, nous verrons par la suite, avec l'algorithme, que cette borne est atteinte. Cette caractérisation des solutions d'un problème par rapport à celles de son dual est également connue sous le nom de théorème des écarts complémentaires en programmation linéaire et peut être exprimée en termes de solutions admissibles des programmes (7) et (8).

3.3.2 Caractérisation des arcs

Comme nous l'avons vu, nous obtenons un *retiming* optimal lorsque tous les arcs ont un indice de conformité nul, étudions à présent sous quelles conditions cela se produit.

Si nous traçons pour un arc e quelconque l'ensemble des couples $(f(e), w_q(e))$ pour lesquels $ic(e) = 0$, nous obtenons le diagramme de la figure (6), que nous appellerons diagramme de conformité de e.

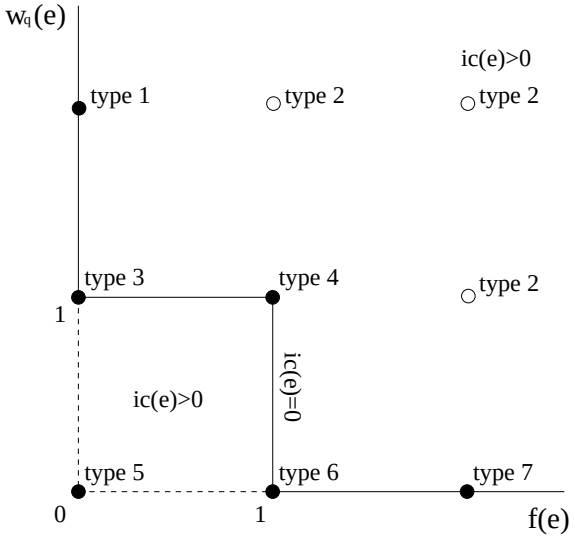


FIG. 6: Diagramme de conformité et différents types d'arcs

Comme illustré sur ce diagramme, nous assignons à chaque arc e un type suivant les valeurs de $w_q(e)$ et $f(e)$, et nous appellerons arcs conformes les arcs pour lesquels $ic(e) = 0$:

$$\left.\begin{array}{lrcl} \text{Type 1} : & w_q(e) > 1 & \text{et} & f(e) = 0 \\ \text{Type 3} : & w_q(e) = 1 & \text{et} & f(e) = 0 \\ \text{Type 4} : & w_q(e) = 1 & \text{et} & f(e) = 1 \\ \text{Type 6} : & w_q(e) = 0 & \text{et} & f(e) = 1 \\ \text{Type 7} : & w_q(e) = 0 & \text{et} & f(e) > 1 \end{array}\right\} \text{ arcs conformes}$$

$$\left.\begin{array}{lrcl} \text{Type 2} : & w_q(e) > 0 & \text{et} & f(e) > 1 \\ \text{ou} & w_q(e) > 1 & \text{et} & f(e) > 0 \\ \text{Type 5} : & w_q(e) = 0 & \text{et} & f(e) = 0 \end{array}\right\} \text{ arcs non conformes}$$

Nous en déduisons que si tous les arcs sont conformes, c'est-à-dire si $ic(e) = 0$ pour tout arc e, alors nous sommes à l'optimal.

Nous pouvons de plus constater que pour tout arc conforme e, il est possible de faire varier $w_q(e)$ ou $f(e)$ d'une unité tout en conservant la conformité de e, et pour tout arc non conforme, il est possible de faire décroître strictement son indice de conformité en faisant varier $w_q(e)$ ou $f(e)$ d'une unité. Plus précisément, nous avons les cas suivants :

- si $f(e)$ augmente : les arcs de type 3,6 et 7 deviennent respectivement de type 4,7 et 7, et restent conformes, les arcs de type 5 deviennent conformes (type 6), les autres arcs voient leur indice de conformité croître (strictement) ;
- si $f(e)$ diminue : les arcs de type 4 et 7 deviennent respectivement de type 3 et 6 ou 7, et restent conformes, les arcs de types 2 deviennent conformes (type 4 ou 1) ou voient leur indice

de conformité décroître (strictement), les autres arcs voient leur indice de conformité croître (strictement) ;

- si $w_q(e)$ augmente : les arcs de type 1,3 et 6 deviennent respectivement de type 1,1 et 4, et restent conformes, les arcs de type 5 deviennent conformes (type 3), les autres arcs voient leur indice de conformité croître (strictement) ;
- si $w_q(e)$ diminue : les arcs de type 1 et 4 deviennent respectivement de type 1 ou 3 et 6, et restent conformes, les arcs de type 2 deviennent conformes (type 4 ou 7) ou voient leur indice de conformité décroître (strictement), les autres arcs voient leur indice de conformité croître (strictement).

Nous allons maintenant voir comment exploiter ces possibilités afin de converger vers une solution optimale.

3.3.3 Algorithme

L'algorithme consiste à partir d'une solution initiale admissible et à la faire évoluer vers une solution optimale. Le *retiming* nul et le flot nul constituent respectivement un *retiming* légal et un flot admissible et pour ceux-ci, $ic(e) = v(e)$ pour tout arc $e \in E$, c'est-à-dire 1 sur les arcs de poids nul et 0 sur les autres. Remarquons que pour cette solution, tous les arcs sont de type 1,3 ou 5.

Tout le problème est donc de faire baisser l'indice de conformité des arcs de type 5, sans faire monter celui des autres arcs. Pour cela, nous assignons à chaque type d'arc une couleur représentative des degrés de liberté dont il dispose :

- noir pour les arcs de type 3,5 et 6 ($f(e)$ et $w_q(e)$ peuvent augmenter seulement) ;
- vert pour les arcs de type 2 et 4 ($f(e)$ et $w_q(e)$ peuvent diminuer seulement) ;
- rouge pour les arcs de type 7 ($f(e)$ peut augmenter ou diminuer, $w_q(e)$ ne peut pas varier) ;
- incolore pour les arcs de type 1 ($f(e)$ ne peut pas varier, $w_q(e)$ peut augmenter ou diminuer).

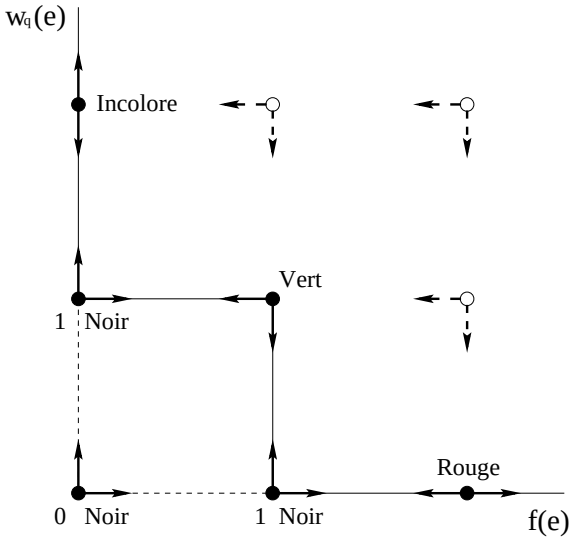


FIG. 7: Coloration des différents types d'arcs

note : Dans la suite nous ne considérerons pas les arcs verts de type 2, en effet nous allons partir d'une solution ne contenant pas de tels arcs, et nous ferons baisser l'indice de conformité des arcs noirs de type 5 sans faire monter celui des autres arcs, il n'y aura donc jamais de création d'arcs verts de type 2.

Nous allons maintenant, en utilisant le lemme des arcs colorés (ou lemme de Minty, [GM85], p.136-137), pouvoir résoudre notre problème. Considérons un arc non conforme e_0 (noir, comme tous nos arcs non conformes), deux cas peuvent se produire :

a) il existe un cycle passant par e_0, ne contenant pas d'arcs incolores, avec tous les arcs noirs orientés dans le sens de e_0, et tous les arcs verts orientés dans le sens contraire.

   En augmentant la valeur du flux de 1 pour les arcs du cycle dans le sens de e_0, et en la diminuant de 1 pour les autres, alors par définition des couleurs, le flot reste admissible (positif ou nul sur tous les arcs). De plus e_0 devient conforme, et l'indice de conformité des autres arcs n'augmente pas.

b) il existe un cocycle contenant e_0, ne contenant pas d'arcs rouges, avec tous les arcs noirs orientés dans le sens de e_0, et tous les arcs verts orientés dans le sens contraire.

   Le cocycle détermine une partition des sommets en deux ensembles. Notons A celui contenant l'extrémité terminale de e_0. En augmentant la valeur du *retiming* de 1 pour les sommets de A, alors par définition des couleurs, le *retiming* reste légal ($w_q(e)$ reste positif ou nul). De plus e_0 devient conforme, et l'indice de conformité des autres arcs n'augmente pas.

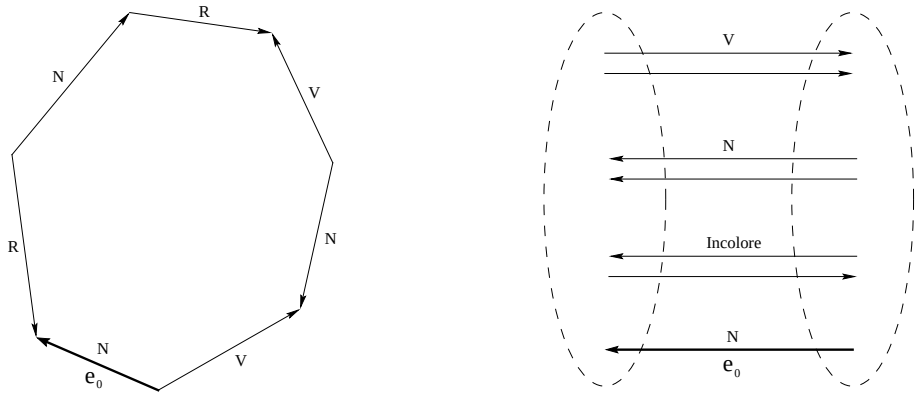


FIG. 8: Les deux éventualités du lemme de Minty

Nous obtenons donc au moins un arc non conforme de moins. En répétant l'opération (coloration, puis variation du flot ou du *retiming*), jusqu'à ce que tous les arcs soient conformes, nous parvenons au *retiming* optimal.

3.3.4 Complexité

La recherche du cycle ou du cocycle du lemme de Minty peut se faire par une procédure de marquage en $O(|E|)$ (voir la preuve dans [GM85], p.158-159). Comme au moins un arc non conforme devient conforme à chaque itération, le nombre d'itérations de l'algorithme est majoré par le nombre d'arcs de poids nul du graphe initial, que nous pouvons encore majorer par $|E|$, ce qui nous donne une complexité finale en $O(|E|^2)$.

note : L'implémentation de l'agorithme peut être optimisée en ne considérant pour la minimisation que les composantes fortement connexes du graphe. En effet si le graphe n'est pas fortement connexe, après avoir minimisé par *retiming* le nombre d'arcs de poids nul des composantes fortement connexes, il est toujours possible de rendre non nul le poids des arcs les reliant en ajustant par une constante les valeurs de retiming de chacune d'elles.

3.4 Prise en compte de la période d'horloge

Il s'agit maintenant de minimiser le nombre d'arcs de poids nul, pour une période d'horloge donnée. Rappelons que $\Phi(G)$ représente la période d'horloge de G, et étudions les modifications à apporter au programme linéaire initial.

3.4.1 Interprétation des contraintes

Aux contraintes (3) et (4) s'ajoute maintenant une nouvelle contrainte entre les paires de sommets du graphe entre lesquelles le délai du plus long chemin dépasse la période d'horloge. En effet un tel chemin devra contenir au moins un registre :

$$\forall (u,v) \in V^2, D(u,v) > \Phi(G) \Rightarrow q(v) - q(u) + W(u,v) \geq 1 \tag{12}$$

Cela correspond à la création de nouveaux arcs de poids $W(u,v) - 1$ entre toutes les paires de sommets (u,v) telles que $D(u,v) > \Phi(G)$. Si nous notons E_{clock} l'ensemble de ces nouveaux arcs, C_{clock} la matrice d'incidence de E_{clock}, $E' = E \cup E_{clock}$, et C' la matrice d'incidence de E', cela se traduit par la modification du programme linéaire initial (7) en :

$$\min \left\{ \begin{pmatrix} q \\ v \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} \middle| \begin{pmatrix} q & v \end{pmatrix} \begin{pmatrix} 0 & C & C & C_{clock} \\ Id & Id & 0 & 0 \end{pmatrix} \geq \begin{pmatrix} 0 & 1-w & -w & 1-W \end{pmatrix} \right\} \tag{13}$$

Ce qui, en passant au dual, donne :

$$\max \left\{ \begin{pmatrix} 0 \\ 1-w \\ -w \\ 1-W \end{pmatrix} \begin{pmatrix} x & y & z & t \end{pmatrix} \middle| \begin{pmatrix} 0 & C & C & C_{clock} \\ Id & Id & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix}, \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} \geq \mathbf{0} \right\} \tag{14}$$

c'est-à-dire

$$\min \{(w-1)y + wz + (W-1)t \mid C(y+z) + C_{clock}t = 0,\ 0 \leq y \leq 1,\ z \geq 0,\ t \geq 0\} \tag{15}$$

En notant y', z' et t' les prolongements respectifs de y, z et t dans E' par l'ajout d'un nombre suffisant de 0 (c'est-à-dire en considérant le flot dans le graphe de départ complété par les arcs d'horloge), cela revient à imposer à $f = y' + z' + t'$ d'être un flot positif de $G' = (V, E')$.

Nous pouvons remarquer que bien que les nouveaux arcs (que nous appellerons arcs d'horloge) n'aient pas d'incidence sur le coût du *retiming*, ils influent sur le coût du flot.

3.4.2 Modification de l'algorithme

Nous allons voir maintenant comment intégrer ces nouveaux éléments à l'algorithme de manière à obtenir pour les nouveaux arcs une coloration compatible avec les couleurs définies précédemment.

Le problème se pose désormais de la manière suivante : étant donné un graphe pondéré $G = (V, E, d, w)$ le problème consiste à trouver un *retiming* $q : V \to \mathbb{Z}$ et une application $v : E \to \mathbb{N}$,

tels que q soit valide dans $G' = (V, E', d, w)$, où $E_{clock} = \{(u,v) | D(u,v) > \Phi(G)\}$, $E' = E_{clock} \cup E$ et où w est redéfini sur E' de la façon suivante : $w(e) = \begin{cases} w(e) & \text{si } e \in E \\ W(u,v) - 1 & \text{si } e \in E_{clock} \end{cases}$.

En nous fondant sur les constatations précédentes, nous modifions l'indice de conformité des arcs afin de prendre en compte ces modifications :

$$\forall e \in E', ic(e) = \begin{cases} v(e) & \text{si } f(e) = 0 \text{ et } e \in E \\ v(e) - 1 + f(e) w_q(e) & \text{si } f(e) > 0 \text{ et } e \in E \\ f(e) w_q(e) & \text{si } e \in E_{clock} \end{cases}$$

Ce qui nous conduit à modifier les preuves des propositions 2 et 3 en tenant compte des nouveaux arcs de la manière suivante (nous noterons C_f, C_{fclock}, les ensembles constitués respectivement des arcs e de E et E_{clock} tels que $f(e) > 0$, et $C'_f = C_f \cup C_{fclock}$ le multicycle de G' défini par f) :

proposition 4 $\sum_{e \in E} v(e) \geq \sum_{e \in C_f} 1 - \sum_{e \in C'_f} f(e) w(e)$.

preuve : la preuve se trouve en annexe.

Nous en déduisons une nouvelle condition d'optimalité tenant compte des arcs d'horloge.

proposition 5 $\forall e \in E', ic(e) = 0 \Rightarrow q$ *est optimal, de plus* $\Phi(G_q) = \Phi(G)$.

preuve : la preuve se trouve en annexe.

Nous pouvons alors tracer le diagramme de conformité pour les arcs de E_{Clock}, et leur assigner un type de la façon suivante :

Type 1 :	$w_q(e) > 0$	et	$f(e) = 0$	arcs conformes
Type 3 :	$w_q(e) = 0$	et	$f(e) = 0$	
Type 4 :	$w_q(e) = 0$	et	$f(e) > 0$	
Type 2 :	$w_q(e) > 0$	et	$f(e) > 0$	arcs non conformes

Nous en déduisons une coloration compatible (avec les couleurs précédentes) de ces arcs :

- noir pour les arcs de type 3 ;
- vert pour les arcs de type 2 ;
- rouge pour les arcs de type 4 ;
- incolore pour les arcs de type 1.

note : Nous pouvons remarquer qu'initialement tous les arcs de E_{clock} sont conformes. Comme, par construction, l'algorithme ne crée jamais d'arcs non conformes, il est donc inutile de considérer les arcs verts de type 2 de E_{clock}.

Nous obtenons donc l'algorithme final, parvenant au *retiming* optimal, tout en conservant la période d'horloge de G. Le nombre d'étapes est toujours borné par $|E|$, puisque les arcs non conformes n'ont pas changé, en revanche le nombre d'arcs de G' étant borné par $|V|^2$ (le graphe devient au pire complet avec l'ajout des arcs d'horloge), la complexité de la procédure de marquage passe à $O(|V|^2)$, d'où une complexité totale en $O(|E||V|^2)$.

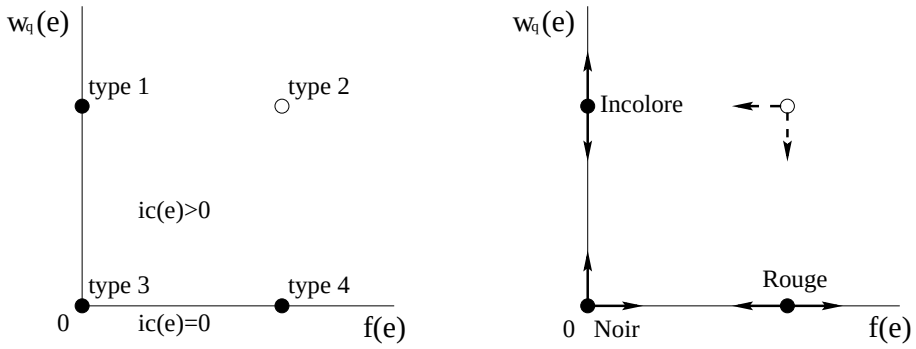


FIG. 9: Diagramme de conformité et coloration des nouveaux arcs

4 Maximisation du nombre d'arcs de poids nul

Nous traitons, dans cette partie, de l'utilisation des techniques de *retiming* pour la maximisation du nombre d'arcs de poids nul d'un graphe de dépendance. Nous pourrons constater l'utilité du problème dans le cadre de la maximisation de la localité et de l'alignement des données en sections 5.2.2 et 5.3. Nous prouvons que le problème est fortement NP-complet.

4.1 Positionnement du problème

Nous pouvons énoncer le problème de maximisation par *retiming* du nombre d'arcs de poids nul de la manière suivante :

NULL-WEIGHTED EDGES MAXIMIZATION :
INSTANCE : Un graphe dirigé $G = (V, E, w)$ et un entier positif K, où V est l'ensemble des sommets du graphe, $E \in V^2$, son ensemble d'arcs, et $w : E \longrightarrow \mathbb{N}$ une pondération des arcs.
QUESTION : Existe-t-il un *retiming* $q : V \longrightarrow \mathbb{Z}$ légal (*i.e.* tel que $\forall e = (u, v) \in E, w_q(e) = q(v) - q(u) + w(e) \geq 0$), tel que $|\{e \in E \mid w_q(e) = 0\}| \geq K$?

Le problème appartient à NP, en effet étant donné un *retiming* de G, il existe un *retiming* générant le même nombre d'arcs de poids nul dont les valeurs sont comprises entre 0 et le poids maximal d'un chemin du graphe (il suffit de considérer le graphe acyclique des arcs de poids nul à générer). Ce *retiming* constitue un certificat polynomial de l'instance (puisque ses valeurs sont bornées par la somme des poids du graphe).

4.2 NP-complétude

Nous démontrons la NP-complétude de NULL-WEIGHTED EDGES MAXIMIZATION en réduisant polynomialement NOT-ALL-EQUAL 3SAT (c.f. [GJ79], p.259) à ce problème. Nous en rappelons ici la formulation :

NOT-ALL-EQUAL 3SAT :
INSTANCE : Un ensemble $U = \{u_1, \ldots, u_n\}$ de variables booléennes et $C = \{c_1, \ldots, c_m\}$ de clauses sur U (pour toute variable $u \in U$, on appelle u et $\bar{u}$ des littéraux sur U, une clause sur U est un ensemble de littéraux sur U), tels que pour chaque clause $c \in C, |c| = 3$.
QUESTION : Existe-t-il une affectation de vérité pour U (*i.e.* une fonction $t : U \longrightarrow \{T, F\}$, en

posant $t(\bar{u}) = V$ si $t(u) = F$ et $t(\bar{u}) = F$ si $t(u) = V$) tel que chaque clause dans C ait au moins un littéral vrai (c'est-à-dire a, tel que $t(a) = V$) et un littéral faux (c'est-à-dire b, tel que $t(b) = F$) ?

4.2.1 Transformation

Soit une instance de NOT-ALL-EQUAL 3SAT, c'est-à-dire la donnée de U et C, nous définissons une transformation $f(U, C) = (G, K)$ de (U, C) en instance de NULL-WEIGHTED EDGES MAXIMIZATION de la manière suivante :

- nous partons de $G = (V, E)$, avec $V = \emptyset$ et $E = \emptyset$;
- pour chaque variable $u \in U$, $V = V \cup \{u, \bar{u}\}$, $E = E \cup \{(u, \bar{u}), (\bar{u}, u)\}$ et $w(u, \bar{u}) = w(\bar{u}, u) = 1$;

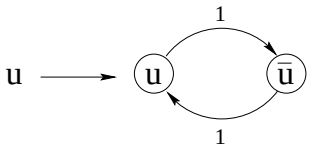


FIG. 10: Transformation d'une variable

- pour chaque clause $c = \{a, b, c\} \in C$, $E = E \cup \{(a, b), (b, a), (b, c), (c, b), (c, a), (a, c)\}$ et $w(a, b) = w(b, a) = \cdots = w(a, c) = 1$;

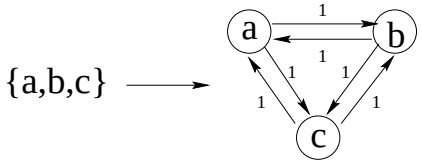


FIG. 11: Transformation d'une clause

- $K = 2m + n$.

Clairement, la transformation est polynomiale (deux sommets et deux arcs générés par littéral, six arcs générés par clause, d'où une transformation en $O(m + n)$).

4.2.2 Réduction

Soit une instance, (U, C), de NOT-ALL-EQUAL 3SAT, avec $(G, K) = f(U, C)$.

Si (U, C) est un instance positive de NOT-ALL-EQUAL 3SAT, soit une affectation de vérité, t, correspondant, et pour toute variable $u \in U$,

$$q(u) = \begin{cases} 1 & \text{si } t(u) = V \\ 0 & \text{si } t(u) = F \end{cases}$$
$$q(\bar{u}) = 1 - q(u)$$

alors f(U,C) est une instance positive de NULL-WEIGHTED EDGES MAXIMIZATION :

- q est un *retiming* légal de G : en effet pour tout arc $e = (u, v)$, $q(v) - q(u) \geq -1$, d'où $w_q(e) = q(v) - q(u) + w(e) \geq 0$ puisque tous les arcs de G ont pour poids 1 ;
- $|\{e \in E \mid w_q = 0\}| = K$:

- pour toute variable $u \in U$, $q(u) - q(\bar{u}) = \pm 1$ donc soit $w_q(u, \bar{u}) = 0$ et $w_q(\bar{u}, u) = 2$, soit $w_q(u, \bar{u}) = 2$ et $w_q(\bar{u}, u) = 0$, donc n arcs de poids nul générés par l'ensemble des variables ;
- pour toute clause $c = \{a, b, c\} \in C$, au moins un littéral est vrai et un faux, donc il existe deux littéraux, par exemple a et b tels que $q(a) = q(b)$ et $q(a) - q(c) = q(b) - q(c) = \pm 1$. Donc aucun arc nul entre a et b, un entre a et c, et un entre b et c, c'est-à-dire deux arcs nuls par clause, soit $2m$ pour l'ensemble des clauses.

au total $2m + n = K$ arcs de poids nul

Si $f(U, C)$ est une instance positive de NULL-WEIGHTED EDGES MAXIMIZATION, soit un *retiming*, q, correspondant, et pour tout littéral $u \in U$,

$$t(u) = \begin{cases} V & \text{si } q(u) \mod 2 = 1 \\ F & \text{si } q(u) \mod 2 = 0 \end{cases}$$

alors (U,C) est une instance positive de NOT-ALL-EQUAL 3SAT.

Remarquons tout d'abord que au plus un seul des deux arcs générés par une variable peut avoir un poids nul après *retiming* puisque le *retiming* conserve le poids des cycles.

D'autre part nous montrons qu'au plus deux des arcs générés par une clause peuvent avoir un poids nul. En effet la même remarque que précédemment nous permet dans un premier temps de limiter ce nombre à trois, c'est-à-dire un au plus sur chaque cycle de longueur deux entre deux littéraux. Supposons maintenant que pour une clause $c = \{a, b, c\}$ il y ait trois arcs de poids nul après *retiming*. Cela signifie que, par exemple, $q(a) = x$, $q(b) = x \pm 1$ (un arc de poids nul entre a et b, c'est-à-dire un des arcs du cycle de longueur deux), $q(c) = x \pm 1$ (un arc de poids nul entre a et c, ici aussi un des arcs du cycle de longueur deux), donc soit $q(b) = q(c)$ et il n'y a pas de troisième arc de poids nul, soit $q(b) = q(c) \pm 2$ et le *retiming* est non légal (de toutes façons un tel retiming génèrerait entre b et c un arc de poids -1 et un de poids 3, donc aucun de poids nul).

Ceci nous permet de déduire que si $|\{e \in E \mid w_q = 0\}| \geq K$, alors $|\{e \in E \mid w_q = 0\}| = K$, puisque $K = 2m + n$, et il y a exactement un arc de poids nul entre un littéral et son opposé, et deux dans chaque clause.

Il nous reste à montrer que t est une affectation de vérité et qu'il satisfait toutes les clauses avec au plus deux littéraux vrais, c'est-à-dire à montrer que :

- $t(u) \neq t(\bar{u})$: en effet il existe un arc de poids nul entre u et $\bar{u}$, donc $q(u) = q(\bar{u}) \pm 1$, d'où $q(u) \mod 2 \neq q(\bar{u}) \mod 2$;
- pour toute clause $c = \{a, b, c\}$, on a par exemple $t(a) \neq t(b)$ (au moins un littéral faux et un vrai) : en effet toute clause contient exactement deux arcs de poids nul, donc au moins un, par exemple entre a et b d'où $q(a) = q(b) \pm 1$, c'est-à-dire $t(a) \neq t(b)$.

□

5 Applications

Nous allons maintenant tenter d'exposer les liens existant entre *retiming* et parallélisation automatique, en détaillant certains problèmes dans lesquels apparaît l'expression d'un *retiming*, et nous verrons de quelle façon il est possible d'exploiter nos résultats précédents.

L'ensemble des techniques mises en œuvre dans les diverses approches de parallélisation automatique considère généralement un type de graphe, appelé graphe de dépendance, identique dans

sa forme aux graphes décrivant des circuits synchrones. Nous commençons donc dans cette section par présenter ces graphes de dépendance et nous détaillons leur fonction.

Nous verrons ensuite comment l'expression de certains problèmes d'ordonnancement fait apparaître une formulation des solutions en termes de *retiming* et de quelle façon l'utilisation d'un *retiming* peut influencer la localité des données. Nous établirons dans chaque cas un lien entre ces nouveaux problèmes et les résultats des sections 3 et 4, et nous évoquerons d'autres problèmes faisant intervenir l'expression d'un *retiming* mais pour lesquels les choses restent encore peu claires.

5.1 Graphes de dépendance

Nous introduisons ici brièvement le formalisme des graphes de dépendance, leurs principes et leur utilité en tant qu'outil théorique pour la parallélisation automatique.

La nécessité d'introduire un outil tel que les graphes de dépendance naît d'une simple constatation : dans un programme "régulier", le principal obstacle à la mise en parallèle des opérations qu'il exécute se résume à l'existence de dépendances entre celles-ci. En d'autres termes, il est possible qu'une opération donnée ne puisse s'exécuter qu'après l'exécution d'un certain nombre d'autres opérations. Ceci se produit par exemple lorsque plusieurs d'entre elles accèdent à un même endroit de la mémoire (et au moins une en écriture). Le but d'un graphe de dépendance est de représenter de façon simple et concise l'ensemble de ces contraintes.

Dans le cadre de la parallélisation automatique, nous nous limiterons à l'étude de la parallélisation des instructions contenues à l'intérieur d'un ensemble de boucles imbriquées (ou nid de boucles) et sans structures de contrôle (*while*, *goto*, etc.). Nous appelons dépendance une contrainte empêchant la mise en parallèle d'opérations, dans notre cas il ne s'agira que de dépendances liées aux lectures/écritures en mémoire. Nous appelons instruction une instance générique d'opération, indépendante des itérations (une instruction selon notre dénomination correspond donc à une instruction du code source). Un graphe de dépendance est alors la représentation des dépendances entre instructions.

Formellement, nous définissons un graphe de dépendance par un graphe dirigé $G = (V, E, w, d)$, où w est une pondération sur les arcs, et d une pondération sur les sommets, donc le même type de graphe que pour un circuit synchrone. Ici, d représente le temps d'exécution d'une opération et peut donc être assimilé au délai d'un opérateur dans un circuit synchrone. Pour ce qui est de w, le problème est moins évident. En effet w, représente une valeur de dépendance qui peut être exprimée :

- par un entier dans le cas par exemple de dépendances décrites par niveau ou de dépendances décrites par distance avec une seule boucle. Ce cas correspond directement au nombre de registres d'un circuit synchrone ;
- par un vecteur de dimension supérieure à un ou par un polyèdre dans les cas d'approximations plus précises des dépendances. Dans ces deux cas certains problèmes rencontrés font tout de même apparaître l'expression d'un *retiming* dans un graphe dérivé du premier.

Généralement, une dépendance e d'une instruction u vers une instruction v exprime le fait que l'instruction v d'une itération x ne peut s'exécuter qu'après l'instruction u de l'itération $x - w(e)$. Pour fixer les idées, voici un exemple de code, et le graphe de dépendance qui lui est associé (figure 12) :

```
Exemple 1 :
pour i=1 jusqu'à N
faire
   a[i] = a[i-1] + b[i-1]
   b[i] = a[i] + b[i-2]
finfaire
```

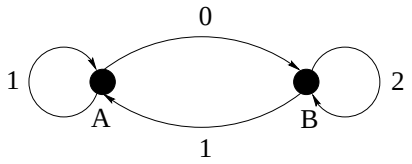


FIG. 12: Le graphe de dépendance de l'exemple 1

De façon plus intuitive, il est possible de voir la valeur d'une dépendance comme une distance (en nombre d'itérations) séparant les deux valeurs dépendantes (ou comme l'ensemble des distances possibles).

5.2 Ordonnancements et problèmes de *retiming*

La recherche d'ordonnancements constitue le cœur des techniques de parallélisation automatique. En effet, la donnée d'un ordonnancement correspond directement à la mise en évidence de parallélisme, les opérations exécutées à la même date pouvant potentiellement être exécutées en parallèle.

L'un des objectifs de la parallélisation automatique étant justement de mettre en parallèle le maximum d'opérations, le problème consiste à déduire d'un graphe de dépendance un ordonnancement, c'est à dire une date d'exécution de chaque opération, minimisant le temps total d'exécution, et respectant bien entendu toutes les dépendances.

Nous présentons dans cette section deux approches possibles pour la recherche d'ordonnancements à partir d'un graphe de dépendance, et nous précisons de quelle manière elles mettent en évidence l'un des problèmes de *retiming* étudiés en sections 3 et 4. Nous évoquerons ensuite d'autres techniques pour la recherche d'ordonnancements mettant en évidence un problème de *retiming* différent ou généralisant les notions précédentes.

5.2.1 Pipeline logiciel

Le pipeline logiciel est une technique consistant à exploiter le parallélisme au niveau des instructions contenues dans une boucle, c'est-à-dire un parallélisme à grain très fin, utilisable par les processeurs superscalaires ou VLIW. En d'autre termes, cela revient à rechercher un ordonnancement des instructions du corps de la boucle permettant l'exécution la plus rapide possible de celui-ci.

Au précédent problème de recherche du parallélisme maximal entre instructions s'ajoute généralement une limitation sur le nombre de ressources utilisables. Autrement dit, il ne suffit plus de trouver l'ordonnancement de longueur minimale, mais il faut trouver l'ordonnancement de longueur minimale n'exécutant pas plus d'instructions en parallèle que le nombre de ressources disponibles.

Malheureusement la recherche d'un tel ordonnancement se heurte à de nombreux problèmes théoriques :

- La recherche d'ordonnancements ne peut raisonnablement se faire que parmi les ordonnancements K-périodiques (reproduisant un même motif toutes les K itérations), pour des raisons évidentes de temps et d'espace mémoire nécessaires à leur élaboration ;
- Bien que l'on conjecture la dominance des ordonnancements K-périodiques parmi les ordonnancements optimaux (autrement dit, il existerait un ordonnancement K-périodique optimal), la détermination du K en question reste de toutes façons un problème ouvert ;
- La simple détermination d'un ordonnancement périodique (c'est-à-dire 1-périodique) optimal sous contraintes de ressources est un problème NP-complet.

Nous présentons donc dans cette section une heuristique pour la recherche d'ordonnancements périodiques sous contraintes de ressources, due à Calland, Darte, et Robert ([CDR96]), et fondée sur plusieurs techniques de *retiming*. D'un point de vue global, elle tire parti de la structure de ces ordonnancements et de la mobilité du calcul de certaines instructions entre différentes itérations. Nous en résumons ici l'idée.

L'expression d'un ordonnancement périodique de période λ peut se faire par le biais d'une fonction σ associant à chaque instruction u d'une itération k donnée une date d'exécution de la manière suivante :

$$\sigma(u,k) = \lambda k + c_u$$

Afin d'identifier le schéma périodique, nous effectuons une division euclidienne de c_u par λ :

$$\sigma(u,k) = \lambda(k + q_u) + r_u, \text{ avec } 0 \leq r_u < \lambda$$

De plus, notre ordonnancement doit vérifier les contraintes exprimées par le graphe de dépendance, c'est-à-dire pour tout arc $u \xrightarrow{e} v$ de G :

$$\begin{aligned} \sigma(u,k) + d(u) &\leq \sigma(v, k + w(e)) \\ \lambda(k + q_u) + r_u + d(u) &\leq \lambda(k + w(e) + q_v) + r_v \\ (r_u - r_v) + d(u) &\leq \lambda(q_v - q_u + w(e)) \end{aligned}$$

En constatant que $r_u - r_v > -\lambda$, $d(u) \geq 0$, et que $(q_v - q_u + w(e))$ est entier, nous obtenons :

$$\begin{aligned} & q_v - q_u + w(e) \geq 0 \\ & q_v - q_u + w(e) = 0 \quad \Rightarrow \quad r_v \geq r_u + d(u) \end{aligned}$$

Ici $q_v - q_u + w(e)$ correspond directement au poids des arcs d'un graphe G_q obtenu à partir de G après *retiming* par q. Nous pouvons constater que seuls les arcs de poids nul après *retiming* imposent alors une contrainte supplémentaire sur l'ordonnancement final.

C'est ici que l'heuristique entre en jeu. Elle consiste à considérer les différents cycles de l'ordonnancement comme indépendants (c'est-à-dire ne se recouvrant pas), et à appliquer un ordonnancement par liste pour déterminer l'ordre d'exécution des opérations d'une itération, ainsi que sa période (λ).

Plus précisément les seuls arcs à considérer pour l'ordonnancement des instructions d'une itération sont les arcs de poids nul de G_q, puisque, comme nous l'avons vu, eux seuls imposent une contrainte à l'intérieur d'un cycle (les autres contraintes étant satisfaites par le découpage entre itérations). Le problème consiste donc à déterminer q de manière à minimiser le plus long

chemin de poids nul de G_q (c'est-à-dire la période d'horloge), constituant un chemin critique pour l'ordonnancement par liste.

Calland, Darte et Robert ont montré que cette heuristique était garantie (grâce aux bornes de l'ordonnancement par liste). Il proposèrent également un critère supplémentaire d'optimisation en choisissant le *retiming* minimisant le nombre total d'arcs de poids nul de G_q, permettant dans la majorité des cas de diminuer la période λ obtenue (le graphe considéré pour l'ordonnancement par liste ayant alors moins d'arcs, il semble probable que l'ordonnancement s'avèrera plus efficace).

Pour la minimisation du nombre d'arcs de poids nul, ils eurent alors recours à la programmation linéaire en nombres entiers. La matrice du programme étant totalement unimodulaire, sa résolution est donc théoriquement polynomiale.

Néanmoins, un des buts affichés pour l'amélioration de cette heuristique était d'éviter le recours à la programmation linéaire en nombres entiers pour déterminer un tel *retiming*. C'est ce que nous avons fait en section 3 en proposant un algorithme de graphe efficace pour la minimisation du nombre d'arcs de poids nul d'un graphe par *retiming*.

5.2.2 Maximisation de la localité

Les accès mémoire lors de l'exécution d'un programme sont bien souvent un frein à la vitesse de calcul des processeurs actuels. En parallélisme, où un accès à une donnée non locale (c'est-à-dire dans la mémoire d'un autre processeur) équivaut à une communication inter-processeurs, ce facteur prend une importance déterminante.

De plus la capacité des mémoires cache devenant de plus en plus grande, il devient très avantageux de tenter de maximiser leur utilisation. Là encore dans le domaine du parallélisme ce facteur est d'autant plus important qu'une "ferme" (*cluster*) de stations possède d'autant plus de mémoire cache qu'elle possède de machines (permettant parfois d'atteindre des accélérations superlinéaires).

C'est pourquoi la maximisation de la localité des données est d'un intérêt capital aussi bien en parallélisation automatique que dans le cas général. Par maximisation de la localité, nous entendons minimisation de la distance (dans le temps) entre le calcul d'une valeur et son utilisation. Cette notion est également connue sous la forme de durée de vie des variables. Minimiser la durée de vie d'une variable revient à augmenter sa probabilité de présence dans le cache, et à gagner ainsi en temps d'accès à la mémoire.

Si nous considérons le cas d'une simple boucle (ou d'un nid de boucles), nous dirons qu'une valeur dont dépend une instruction d'une itération donnée est locale si elle est calculée dans la même itération et non locale si elle a été calculée dans une itération précédente. Maximiser la localité reviendra donc à maximiser le nombre d'arcs de poids nul du graphe de dépendance, c'est-à-dire à maximiser le nombre d'opérations dépendant d'une opération de la même itération. Nous avons classé en section 4 ce problème qui se révèle NP-Complet au sens fort.

5.2.3 Exemple

Nous présentons ici la mise en œuvre de l'heuristique de pipeline logiciel de Calland, Darte et Robert sur un exemple. L'exemple considéré consiste en un calcul simple pour lequel la minimisation du nombre d'arcs de poids nul du graphe de dépendance se révèle fructueuse.

Considérons le calcul des éléments de la suite :

$$a_n = (a_{n-1})^2 + (a_{n-2})^4 + (a_{n-3})^8$$

Il est facile de voir que le calcul direct de toutes les exponentiations met en œuvre un trop grand

nombre de multiplications pour être efficace. Une approche plus intelligente serait de calculer les éléments de la suite par le code suivant :

```
pour n=3 jusqu'à N
faire
   b[n] = a[n-1]*a[n-1]
   c[n] = b[n-1]*b[n-1]
   d[n] = c[n-1]*c[n-1]
   a[n] = b[n]+c[n]+d[n]
finfaire
```

En prenant deux fois le temps de l'addition pour la multiplication, nous pouvons considérer que toutes les opérations ont un délai égal, ce qui nous donne le graphe de dépendance de la figure 13 dont la période d'horloge est déjà minimale (égale à deux).

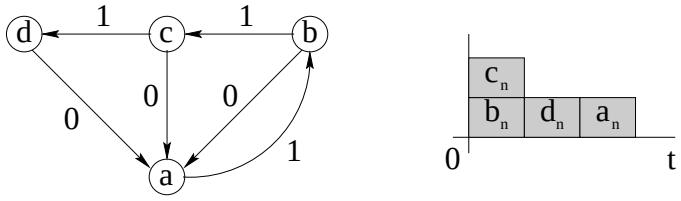


FIG. 13: Graphe de dépendance et ordonnancement périodique pour deux ressources

L'ordonnancement associé est obtenu en appliquant l'heuristique sans minimiser le nombre d'arcs de poids nul du graphe. Plus précisément, nous trouvons un ordonnancement périodique de latence minimale en procédant aux deux étapes suivantes :

- minimiser la période d'horloge du graphe de dépendance, ce qui est déjà fait dans notre exemple ;
- créer le motif répété à chaque itération en pratiquant un ordonnancement par liste sur le sous graphe des arcs de poids nul du graphe de dépendance.

Comme évoqué précédemment, dans cette heuristique il s'agit de satisfaire les dépendances indépendantes de la boucle, les autres étant naturellement satisfaites par la séparation entre itérations successives. La période d'horloge représente ici un chemin critique pour l'ordonnancement par liste.

Il semble alors naturel de penser que moins un graphe possède d'arcs, plus l'ordonnancement par liste se révélera efficace (malheureusement ce n'est pas toujours le cas). C'est dans cette idée que Calland, Darte et Robert proposent de minimiser le nombre d'arcs de poids nul avant d'ordonnancer le sous graphe qu'ils génèrent. En appliquant l'algorithme de la section 3 nous obtenons le graphe de la figure (15) après deux étapes détaillées dans la figure (14) (le choix de l'arc non conforme à traiter à une étape est fait de manière aléatoire, il est simple de vérifier sur cet exemple que cet ordre n'importe pas).

En ordonnançant le graphe de dépendance suivant la même méthode que précédemment nous obtenons cette fois un résultat meilleur (c'est-à-dire une latence inférieure, cf. figure 15). Nous pouvons de plus constater que le code correspondant au *retiming* trouvé reste simple (de façon générale un code produit après *retiming* ne change pas la structure de la boucle et reste donc lisible). Le code produit est le suivant (nous pouvons constater qu'un prologue et un epilogue supplémentaires ont été introduits, ce qui est classique en pipeline logiciel) :

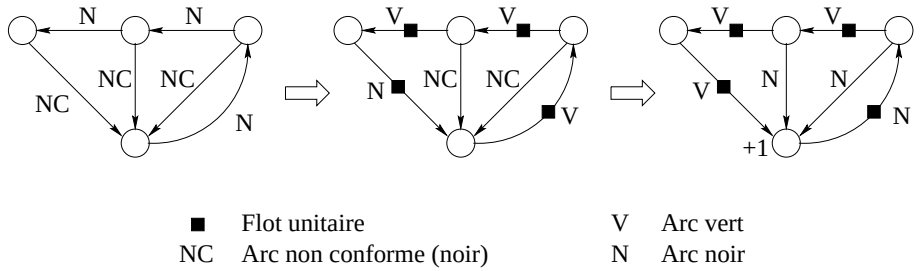


FIG. 14: Les étapes de la minimisation du nombre d'arcs de poids nul

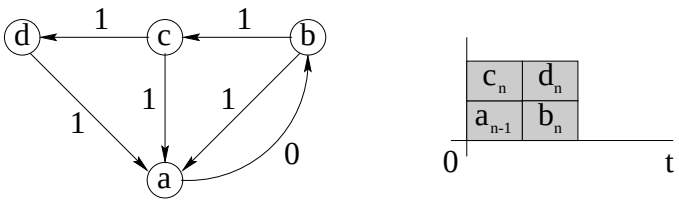


FIG. 15: Graphe de dépendance et ordonnancement périodique pour deux ressources après transformation

```
b[3] = a[2]*a[2]
c[3] = a[1]*a[1]
d[3] = a[0]*a[0]
pour n=4 jusqu'à N
faire
   En parallèle
      a[n-1] = b[n-1] + c[n-1] + d[n-1]
      c[n] = b[n-1]*b[n-1]
   En parallèle
      b[n] = a[n-1]*a[n-1]
      d[n] = c[n-1]*c[n-1]
finfaire
a[N] = b[N] + c[N] + d[N]
```

Nous avons donc pu vérifier l'intérêt de la méthode sur un exemple, et, bien que celle-ci ne soit qu'une heuristique (garantie, cf. [CDR96]), elle semble bonne intuitivement (moins d'arcs sur un graphe à ordonnancer). Néanmoins nous pouvons remarquer que cette technique n'est profitable que dans le cas de boucles dont le corps possède plusieurs instructions. Les techniques de pipeline logiciel intervenant généralement à un bas niveau (assembleur), il paraît probable d'obtenir de telles boucles (puisque la programmation à un bas niveau met souvent en œuvre un plus grand nombre d'instructions), sur lesquelles l'heuristique s'avérera efficace. Il est également possible d'associer cette heuristique à un pré-déroulage de la boucle (qui multiplie la taille de son corps).

Notre méthode semble donc s'avérer profitable, et le recours à un algorithme de graphe est un gain indéniable par rapport à la programmation linéaire.

5.3 Alignement de données et *retiming*

La mise en œuvre de calculs parallèles sur des machines à mémoire distribuée implique souvent une répartition des données (données du problème et valeurs calculées) entre les différents processeurs.

Le problème de l'alignement de données consiste à choisir cette répartition de manière à minimiser le nombre de communications interprocesseurs, c'est-à-dire à minimiser le nombre de calculs faisant intervenir une donnée non locale au processeur le réalisant.

Ce problème a été formalisé dans [DR93] en termes de graphe (graphe de communication) et de *retiming* (vecteur de translation). Le problème se révèle équivalent au problème de maximisation par *retiming* du nombre d'arcs de poids nul d'un graphe, cependant, dans ce cas, nous sommes autorisés à choisir un *retiming* non légal, c'est-à-dire sans contraintes sur le poids des arcs après *retiming*. Darte et Robert ont prouvé que le problème est NP-complet au sens faible.

Nous pouvons constater que notre démonstration de NP-complétude de la section 4 s'applique également dans ce cas puisque la preuve est toujours valide si nous autorisons un *retiming* non légal. Nous avons donc généralisé la démonstration de Darte et Robert et montré que le problème est NP-complet au sens fort.

5.4 Autres problèmes de *retiming*, extensions

Nous évoquons dans cette section d'autres problèmes posés en termes de *retiming* intervenant sur d'autres critères que ceux vus précédemment. Ils constituent pour la plupart des voies de recherche prometteuses et laissent entrevoir de nombreuses perspectives d'extension des techniques de retiming.

5.4.1 Parallélisation de boucles

Certains algorithmes de parallélisation de boucles, comme par exemple celui de Darte-Vivien ([DV97]) ou de Feautrier, permettent la reconstruction d'un nid de boucles en un nouveau nid composé d'un certain nombre de boucles séquentielles entourant un ou plusieurs nids de boucles parallèles.

Comme nous pouvons le constater dans [DSV96], cette technique peut conduire, suivant les vecteurs d'ordonnancement trouvés pour les boucles séquentielles, à la création d'une suite séquentielle de nids parallèles. Ceci n'est pas souhaitable à plusieurs titres, d'une part la séparation en suite séquentielle de nids parallèles requiert l'utilisation de synchronisations entre les nids, d'autre part cela diminue la localité des données dont nous avons déjà souligné l'importance.

L'approche proposée par Darte, Silber et Vivien permet d'éviter ce problème en construisant un minimum de nids parallèles dont le corps sera séquentiel. Ceci se réalise en autorisant un certain nombre de dépendances indépendantes de la boucle dans le nid final, et nécessite deux étapes : d'abord l'utilisation de techniques d'ordonnancement, puis celle d'un retiming afin d'obtenir les dépendances indépendantes de la boucle.

5.4.2 Recherche d'ordonnancements K-périodiques

Comme nous l'avons déjà mentionné, il semblerait que les ordonnancements K-périodiques soient optimaux, cependant la détermination du K en question, qui pourrait *a priori* être exponentiellement grand par rapport à la taille du graphe de dépendance, reste un problème ouvert.

C'est pourquoi Sánchez et Cortadella proposent dans [SC93] une heuristique pour la recherche d'ordonnancements K-périodiques optimaux avec ou sans contraintes de ressources. Le principe

de leur méthode combine plusieurs techniques afin de déterminer un ordonnancement maximisant l'utilisation des ressources (c'est-à-dire la moyenne du nombre de ressources utilisées par unité de temps). Plus précisément les techniques suivantes sont mises en œuvre :

- déroulage du graphe de dépendance, afin de ramener la recherche d'un ordonnancement K-périodique à la recherche d'un ordonnancement périodique dans le graphe déroulé K fois ;
- retiming (présenté sous une autre forme), afin de déterminer une fois le graphe déroulé un ordonnancement périodique associé (un peu à la façon de Calland, Darte et Robert, cf. section 5.2.1) ;
- suites de Farey et équations diophantiennes linéaires, afin de déterminer le nombre de déroulages.

Cependant afin de mettre en œuvre cette heuristique, il est nécessaire de borner le K de l'ordonnancement cherché (ce qui conduit à la perte d'ordonnancements peut-être meilleurs). De plus, elle requiert un grand nombre de calculs et peut donc se révéler coûteuse.

5.4.3 *Retiming* multidimensionnel

Les processeurs ayant tendance à permettre l'exécution simultanée d'un nombre d'instructions de plus en plus important, les techniques de pipeline logiciel ne parviennent pas toujours à saturer l'ensemble des ressources disponibles. C'est pourquoi l'extension de celles-ci aux nids de boucles comportant plus d'une boucle par la recherche de *retiming* multidimensionnel peut se révéler intéressante.

C'est ce que proposent Passos et Sha dans [PS96]. Leur technique consiste à reconstruire le nid de boucles de telle sorte que le corps soit totalement parallèle. Ils réalisent ceci en utilisant les techniques classiques d'ordonnancement afin de faire porter par la boucle externe toutes les dépendances non nulles, puis à appliquer un *retiming* multidimensionnel sur les dépendances restantes (c'est à dire un *retiming* modifié, déterminant pour chaque sommet non plus une constante, mais un vecteur de dimension supérieure à un).

Néanmoins cette approche présente deux inconvénients, d'une part elle produit un code complexe (car elle reconstruit le nid de boucles à partir des vecteurs d'ordonnancement), d'autre part elle détruit une éventuelle permutabilité totale entre les boucles, interdisant l'utilisation de techniques de *tiling* sur le résultat produit. De plus, l'usage du *retiming* dans le cas multidimensionnel est encore peu mature et des améliorations semblent possibles.

6 Conclusion

Nous avons présenté dans cette étude un ensemble de techniques dites de *retiming*, introduites par Leiserson et Saxe dans le cadre de la synthèse d'architectures et destinées à l'optimisation de circuits synchrones. Nous avons rappelé leurs principaux résultats, et nous avons mentionné les avancées réalisées depuis dans ce domaine.

Nous avons alors proposé la résolution de deux nouveaux problèmes de *retiming* en considérant la technique comme un outil de graphe et non plus une méthode réservée à l'optimisation de circuits.

Tout d'abord, nous avons résolu le problème de la recherche d'un *retiming* minimisant le nombre d'arcs de poids nul d'un graphe. Le problème s'avérait connu comme polynomial ([CDR96]) en tant que cas particulier de la programmation linéaire en nombres entiers (matrice totalement unimodulaire), et le but était ici de fournir un algorithme de graphe permettant sa résolution efficace. Nous avons proposé un algorithme résolvant le problème en $O(|E|^2)$, adapté d'un algorithme de

recherche de flot de coût minimal dû à Fulkerson. Nous avons également montré comment résoudre ce problème sous contrainte d'horloge tout en restant efficace ($O(|E||V|^2)$, c'est-à-dire légèrement plus que la minimisation de la période d'horloge, elle même en $O(|V||E|\log|V|)$).

Nous avons ensuite montré la NP-complétude du problème inverse, à savoir la maximisation du nombre d'arcs de poids nul par *retiming*. Plus précisément nous avons montré que le problème s'avérait NP-complet au sens fort pour tout type de *retiming* (légal ou non), généralisant et complétant ainsi un ancien résultat de Darte et Robert ([DR93]) pour l'alignement de données.

Enfin, nous avons illustré l'utilité de ces nouveaux résultats en présentant un ensemble de techniques de parallélisation automatique faisant intervenir un *retiming*.

Nous avons vu que la minimisation du nombre d'arcs de poids nul intervenait dans le cadre d'une heuristique de pipeline logiciel due à Calland, Darte et Robert. Le problème était ici de minimiser la période d'horloge, puis le nombre d'arcs de poids nul tout en conservant cette période d'horloge. Nous avons pu illustrer les apports de la minimisation du nombre d'arcs de poids nul sur un exemple.

Nous avons ensuite pu constater l'utilité de la maximisation du nombre d'arcs de poids nul, dans le cadre de la maximisation de la localité des données, aussi bien par réordonnancement que par alignement. Malheureusement comme nous l'avons déjà vu le problème est NP-complet au sens fort. Néanmoins le cas particulier des graphes acycliques semble intéressant à étudier et pourrait s'avérer polynomial.

Finalement nous avons évoqué les nombreux problèmes restant à résoudre. Nous pouvons citer le cas du pipeline logiciel, pour lequel les techniques de *retiming* multidimensionnel, c'est-à-dire appliquées à un nid de boucles, ou les techniques de déroulage de graphe et de recherche d'ordonnancements K-périodiques sont encore à l'état embryonnaire. De nombreuses approches restent encore inexplorées, comme par exemple un *retiming* multidimensionel ne reconstruisant pas un nouveau nid de boucles par la détermination d'hyperplans séparants comme dans [PS96].

De plus l'utilisation du *retiming* dans d'autres domaines est toujours possible. En effet la simplicité et la généralité de la technique en font un outil de graphe supplémentaire plus qu'une technique réservée à l'optimisation architecturale. En conclusion, le *retiming* apparaît donc comme une technique générale, efficace (résolution de la plupart des problèmes en temps polynômial), simple (il s'agit d'une simple pondération des sommets d'un graphe), et flexible (adapté à plusieurs domaines éloignés les uns des autres).

Remerciements : A Alain Darte pour sa direction inspirée, à Antoine Fraboulet et Fabien Rico pour leurs critiques pertinentes et leurs précieux conseils, et à Valentina Tadé pour son soutien.

Références

[CDR96] P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *IEEE transactions on parallel and distributed systems*, 9(1) :24–35, 1996.

[dFAK+96] B. de Fluiter, E.H.L. Aarts, J.H.M. Korst, W.F.J. Verhaegh, and A. van der Werf. The Complexity of Generalized Retiming Problems. *IEEE Transactions on computers-aided design of integrated circuits and systems*, 15(11) :1340–1352, 1996.

[DR93] A. Darte and Y. Robert. A Graph-Theoric Approach to the Alignement Problem. Technical Report 93-20, Laboratoire de l'Informatique du Parallélisme, 1993.

[DSV96] A. Darte, G.A. Silber, and F. Vivien. Combining retiming and scheduling techniques for loop parallelisation and loop tiling. Technical Report 96-34, Laboratoire de l'informatique du parallélisme, 1996.

[DV97] A. Darte and F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *International Journal of Parallel Programming*, 25(6) :447–496, 1997.

[dW90] D. de Werra. *Eléments de programmation linéaire et application aux graphes.* Mathématiques. Presses polytechniques romandes, 1990.

[GJ79] M.R. Garey and D.S. Johnson. *A Guide to the Theory of NP-Completeness.* W.H.Freeman and company, 1979.

[GM85] M. Gondran and M. Minoux. *Graphes et algorithmes*, volume 37 of *Collection de la direction des études et recherches d'électricité de france*. Eyrolles, 2^e edition, 1985.

[LS91] C.E. Leiserson and J.B. Saxe. Retiming Synchronous Circuitry. In *Algorithmica*, volume 6, pages 5–35. Springer-Verlag, 1991.

[MS98] N. Maheshwari and S. Sapatnekar. Efficient Retiming of Large Circuits. *IEEE Transactions on very large scale integration (VLSI) systems*, 6(1) :74–83, 1998.

[PS96] N.L. Passos and E.H.-M. Sha. Achieving Full Parallelism Using Multidimensional Retiming. *IEEE Transaction on Parallel and Distributed Systems*, 7(11) :1150–1163, 1996.

[SC93] F. Sánchez and J. Cortadella. RCLP : A Novel Approach for Resource-Constrained Loop Pipelining. Technical Report RR-93/06, Departament d'Arquitectura de Computador, Universitat Politècnica de Catalunya, 1993.

Annexes

Algorithme naïf

Voici l'algorithme naif évoqué en section 3.2 :

Recherche d'un flot de coût minimal dans G'

L'algorithme 1 p.252 de [dW90] se propose de déterminer un flot de coût minimum dans un graphe G en partant d'un flot admissible f de G et en construisant à chaque étape un graphe auxiliaire $R^*(f)$. L'existence d'un circuit élémentaire de poids négatif dans $R^*(f)$ nous permet alors de déterminer une circulation x^* dans $R^*(f)$ telle que $f+x^*$ soit admissible dans G et de coût strictement inférieur à celui de f. L'algorithme s'arrête lorque $R^*(f)$ ne contient plus de circuit élémentaire de poids négatif, f est alors optimal.

Nous allons détailler ici la construction de $R^*(f)$ dans le cas de la recherche de flot de coût minimum dans G'. Comme nous le verrons, il est possible dans ce cas de réduire le nombre d'arcs construits par l'algorithme.

Soit f' un flot admissible de G'. Nous pouvons remarquer que 0 constitue un flot admissible (il n'y a pas de capacité inférieure dans G') ce qui résoud le problème d'initialisation de l'algorithme.

Posons pour tout arc $y(e) = f'(e_1)$ et $z(e) = f'(e_\infty)$. Nous avons alors $f(e) = y(e) + z(e)$ admissible dans G (par construction), $0 \leq y(e) \leq 1$ (capacité de e_1) et pour tout f' optimal, $y(e) = 0 \Rightarrow z(e) = 0$: en effet sinon $y'(e) = y(e) + 1$ et $z'(e) = z(e) - 1$ définissent un flot f^* admissible et de coût strictement inférieur à f' (car $w'(e_1) < w'(e_\infty)$).

Détaillons maintenant la construction de $R^*(f')$: pour chaque arc $u \xrightarrow{e} v \in E$, trois cas se présentent :

- si $y(e) = 0$ et $z(e) = 0$ (c'est-à-dire $f(e) = 0$) : l'algorithme de base crée deux arcs, e_1^* et e_2^* de u vers v dans $R^*(f')$ de poids respectifs $w(e)$ et $w(e) - 1$. Dans notre cas seul e_2^* est nécessaire : en effet s'il existe un circuit élémentaire de poids négatif passant par e_1^*, il est de la forme $(e'_1, \ldots, e'_i, e_1^*, e'_{i+1}, \ldots, e'_n)$ et ne contient pas e_2^* (puisqu'élémentaire), e_1^* étant un arc de u vers v, $(e'_1, \ldots, e'_i, e_2^*, e'_{i+1}, \ldots, e'_n)$ est alors lui aussi un circuit élémentaire de poids négatif (puisque $w'(e_2^*) < w'(e_1^*)$) ;

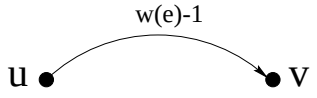


FIG. 16: Arc de $R^*(f')$ correspondant à un arc e de E tel que $f(e) = 0$.

- $y(e) = 1$ et $z(e) = 0$: l'algorithme de base crée deux arcs, $u \xrightarrow{e_1^*} v$ et $v \xrightarrow{e_2^*} v$ dans $R^*(f')$ de poids respectifs $w(e)$ et $-w(e) + 1$. Dans ce cas, les deux sont à conserver puisqu'ils sont de direction opposée ;
- $y(e) = 1$ et $z(e) > 0$: l'algorithme de base crée trois arcs, $u \xrightarrow{e_1^*} v$, $v \xrightarrow{e_2^*} u$ et $v \xrightarrow{e_3^*} u$ dans $R^*(f')$ de poids respectifs $w(e)$, $-w(e)$ et $-w(e) + 1$. De même que précédemment, il est possible de supprimer e_3^* sans altérer l'existence de circuit élémentaire de poids négatif dans $R^*(f')$.

Considérons enfin le nombre d'itérations de l'algorithme : pour chaque arc e de G, $w(e) \geq 0$, donc par construction, $w'(e_1) \geq -1$ et $w'(e_\infty) \geq 0$. De plus $c(e_1) = 1$, il y a donc au plus $|E|$ arcs de poids -1 dans G'. Chacun d'eux laissant passer au plus une unité de flot, la valeur minimale

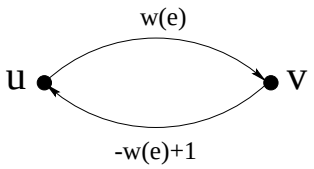


FIG. 17: Arcs de $R^*(f')$ correspondant à un arc e de E tel que $f(e) = 1$.

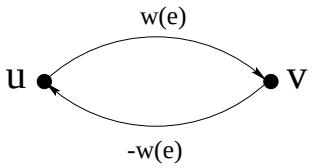


FIG. 18: Arcs de $R^*(f')$ correspondant à un arc e de E tel que $f(e) > 1$.

d'un flot admissible est donc bornée inférieurement par $-|E|$, ce qui borne le nombre d'itérations de l'algorithme (en effet à chaque étape le coût du flot diminue strictement, le coût initial étant 0).

La procédure de recherche d'un circuit de poids négatif pouvant être réalisée en $O(|V||E|)$ par un algorithme de type Bellman-Ford, nous obtenons une complexité de $O(|V||E|^2)$ pour la recherche d'un flot de coût minimal.

Détermination du retiming

L'algorithme de la section 6 se termine par la détermination d'un flot $f = y + z$ tel que $R^*(f)$ ne contienne pas de cycle de poids négatif. Déterminons, par une recherche de plus court chemin à partir d'un point arbitrairement choisi dans $R^*(f)$, une valeur $r(u)$ pour chaque sommet u.

Comme nous l'avons vu, à chaque arc $u \xrightarrow{e} v$ de G correspondent au plus deux arcs dans $R^*(f)$, trois cas se présentent :

- $y(e) = 0$ (et $z(e) = 0$), cf. fig. 16 :
  un seul arc, de poids $w(e) - 1$ correspond à e dans $R^*(f)$

$$r(v) \leq r(u) + w(e) - 1$$

- $y(e) = 1$ et $z(e) = 0$, cf. fig. 17 :
  deux arcs, de poids $w(e)$ et $-w(e) + 1$ correspondent à e dans $R^*(f)$

$$\begin{cases} r(v) \leq r(u) + w(e) \\ r(u) \leq r(v) - w(e) + 1 \end{cases} \iff r(u) + w(e) - 1 \leq r(v) \leq r(u) + w(e)$$

- $y(e) = 1$ et $z(e) > 0$, cf. fig. 18 :
  deux arcs, de poids $w(e)$ et $-w(e)$ correspondent à e dans $R^*(f)$

$$\begin{cases} r(v) \leq r(u) + w(e) \\ r(u) \leq r(v) - w(e) \end{cases} \iff r(v) = r(u) + w(e)$$

Nous en déduisons un *retiming* en posant

$$q(u) = -r(u) \text{ et } v(e) = \begin{cases} 0 & \text{si } f(e) = 0 \\ 1 - (q(v) - q(u) + w(e)) & \text{sinon} \end{cases}$$

et nous vérifions que $\begin{pmatrix} q & v \end{pmatrix}$ est une solution admissible de (7) en s'assurant des conditions (3), (4) et $v \geq 0$:

- nous avons dans tous les cas

$$r(v) \leq r(u) + w(e) \iff q(v) - q(u) + w(e) \geq 0$$

  ce qui vérifie (3) ;
- si $y(e) = 0$:

$$r(v) \leq r(u) + w(e) - 1 \iff q(v) - q(u) + w(e) \geq 1$$

  et $v(e) = 0$ vérifient (4) et $v \geq 0$;
- si $y(e) = 1$:

$$r(u) + w(e) - 1 \leq r(v) \leq r(u) + w(e) \iff 1 \geq q(v) - q(u) + w(e) \geq 0$$

  d'où

$$\begin{cases} v(e) = 1 - (q(v) - q(u) + w(e)) \geq 0 \\ q(v) - q(u) + w(e) + v(e) = 1 \geq 1 \end{cases}$$

  donc (4) et $v \geq 0$ sont vérifiées.

Nous avons donc un flot de coût minimal, et une solution admissible de (7), nous vérifions qu'elle est optimale en appliquant le théorème des écarts complémentaires (p. 39, [dW90], problème sous forme standard) :

$\begin{pmatrix} q & v \end{pmatrix}$ et $\begin{pmatrix} x & y & z \end{pmatrix}$ sont optimales ssi

$$\begin{aligned} vx &= 0 \\ (qC + v - 1 + w)y &= 0 \\ (qC + w)z &= 0 \end{aligned}$$

Nous vérifions les trois cas de la manière suivante :

1. on a $x = 1 - y$:
   - si $y(e) = 0$, $v(e) = 0$, d'où $v(e)x(e) = 0$;
   - si $y(e) = 1$, $x(e) = 0$, d'où $v(e)x(e) = 0$.

   donc $vx = \sum_{e \in E} v(e)x(e) = 0$;
2. - si $y(e) = 0$, $(q(v) - q(u) + v(e) - 1 + w(e))y(e) = 0$;
   - si $y(e) = 1$, $q(v) - q(u) + v(e) - 1 + w(e) = 0$ par définition de $v(e)$, d'où $(q(v) - q(u) + v(e) - 1 + w(e))y(e) = 0$.

   donc $(qC + v - 1 + w)y = \sum_{u \xrightarrow{e} v \in E} (q(v) - q(u) + v(e) - 1 + w(e))y(e) = 0$;
3. - si $z(e) = 0$, $(q(v) - q(u) + w(e))z(e) = 0$;
   - si $z(e) > 0$, $q(v) - q(u) + w(e) = 0$ par définition de q (à partir de r), d'où $(q(v) - q(u) + w(e))z(e) = 0$.

   donc $(qC + w)z = \sum_{u \xrightarrow{e} v \in E} (q(v) - q(u) + w(e))z(e) = 0$.

Nous obtenons donc le retiming optimal en appliquant une recherche de type Bellman-Ford sur le graphe auxiliaire $R^*(f')$ final. La complexité totale de l'algorithme est donc $O(|V||E|^2 + |V||E|)$, c'est-à-dire $O(|V||E|^2)$. Nous allons voir dans la section suivante que nous pouvons faire mieux en abordant le problème d'une manière plus détournée.

Preuves des propositions pour la minimisation

Les preuves des propositions de la section 3 mettant en relation l'indice de conformité et l'optimalité du retiming.

proposition 1 $\sum_{e\in C_f} f(e)w(e) = \sum_{e\in C_f} f(e)w_q(e)$.

preuve :

$$\begin{aligned}
\sum_{e\in C_f} f(e)w(e) &= \sum_{e\in C_f} f(e)w(e) + \sum_{v\in V} q(v)\left(\sum_{?\xrightarrow{e} v\in E} f(e) - \sum_{v\xrightarrow{e} ?\in E} f(e)\right)\\
&= \sum_{e\in C_f} f(e)w(e) + \sum_{v\in V}\left(\sum_{?\xrightarrow{e} v\in E} f(e)q(v) - \sum_{v\xrightarrow{e} ?\in E} f(e)q(v)\right)\\
&= \sum_{e\in C_f} f(e)w(e) + \sum_{e=(u,v)\in C_f} (f(e)q(v) - f(e)q(u))\\
&= \sum_{e=(u,v)\in C_f} f(e)(w(e) + q(v) - q(u))\\
&= \sum_{e\in C_f} f(e)w_q(e)
\end{aligned}$$

□

proposition 2 $\sum_{e\in E} v(e) \geq \sum_{e\in C_f}(1 - f(e)w(e))$.

preuve :

$$\begin{aligned}
\sum_{e\in E} v(e) - \sum_{e\in C_f}(1 - f(e)w(e)) &\geq \sum_{e\in C_f} v(e) - \sum_{e\in C_f}(1 - f(e)w_q(e))\\
&\geq \sum_{e\in C_f}((v(e) - 1) + f(e)w_q(e))\\
&\geq \sum_{e\in C_f}(f(e)(v(e) - 1) + f(e)w_q(e))\\
&\quad \left.\begin{array}{ll}\text{car} & f(e) > 0\\ & v(e) - 1 \leq 0\end{array}\right\} \text{(par définition)}\\
&\quad \text{d'où, } f(e)(v(e) - 1) \leq v(e) - 1\\
&\geq \sum_{e\in C_f}(f(e)(-w_q(e)) + f(e)w_q(e))\\
&\quad \text{car } v(e) - 1 \geq -w_q(e)\\
&\quad \text{(par définition de v(e))}\\
&\geq 0
\end{aligned}$$

□

proposition 3 $\forall e \in E, ic(e) = 0 \Rightarrow$ *q est optimal.*

preuve : si $ic(e) = 0$ pour tout arc de G, alors :

$$\begin{aligned} 0 &= \sum_{e \in E} ic(e) \\ &= \sum_{e \in E \setminus C_f} v(e) + \sum_{e \in C_f} (v(e) - 1 + f(e) w_q(e)) \\ &= \sum_{e \in E} v(e) - \sum_{e \in C_f} (1 - f(e) w(e)) \end{aligned}$$

donc

$$\sum_{e \in E} v(e) = \sum_{e \in C_f} (1 - f(e) w(e))$$

et d'après la proposition 2, $\sum_{e \in E} v(e)$ est minimale. □

proposition 4 $\sum_{e \in E} v(e) \geq \sum_{e \in C_f} 1 - \sum_{e \in C'_f} f(e) w(e)$.

preuve :

$$\begin{aligned} \sum_{e \in E} v(e) - \sum_{e \in C_f} 1 + \sum_{e \in C'_f} f(e) w(e) &\geq \sum_{e \in C_f} v(e) - \sum_{e \in C_f} 1 + \sum_{e \in C'_f} f(e) w_q(e) \\ &\quad \text{d'après la proposition 1} \\ &\geq \sum_{e \in C_f} (v(e) - 1) + \sum_{e \in C_f} f(e) w_q(e) \\ &\quad \text{car } f(e) \geq 0, w_q(e) \geq 0, \text{ et } C_f \subset C'_f \\ &\geq \sum_{e \in C_f} (f(e)(v(e) - 1) + f(e) w_q(e)) \\ &\quad \left. \begin{array}{ll} \text{car} & f(e) > 0 \\ & v(e) - 1 \leq 0 \end{array} \right\} \text{(par définition)} \\ &\quad \text{d'où, } f(e)(v(e) - 1) \leq v(e) - 1 \\ &\geq \sum_{e \in C_f} (f(e)(-w_q(e)) + f(e) w_q(e)) \\ &\quad \text{car } v(e) - 1 \geq -w_q(e) \\ &\quad \text{(par définition de v(e))} \\ &\geq 0 \end{aligned}$$

□

proposition 5 $\forall e \in E', ic(e) = 0 \Rightarrow q$ *est optimal, de plus* $\Phi(G_q) = \Phi(G)$.

preuve : De manière analogue à la preuve de la proposition 3, si $ic(e) = 0$ pour tout arc de G', alors :

$$\begin{aligned} 0 &= \sum_{e \in E'} ic(e) \\ &= \sum_{e \in E \setminus C_f} v(e) + \sum_{e \in C_f} (v(e) - 1 + f(e) w_q(e)) + \sum_{e \in E_{clock}} f(e) w_q(e) \\ &= \sum_{e \in E \setminus C_f} v(e) + \sum_{e \in C_f} v(e) - \sum_{e \in C_f} 1 + \sum_{e \in C_f} f(e) w_q(e) + \sum_{e \in C_{fclock}} f(e) w_q(e) \\ &= \sum_{e \in E} v(e) - \sum_{e \in C_f} 1 + \sum_{e \in C'_f} f(e) w_q(e) \end{aligned}$$

donc

$$\sum_{e \in E} v(e) = \sum_{e \in C_f} 1 - \sum_{e \in C'_f} f(e) w(e)$$

et d'après la proposition 4, $\sum_{e \in E} v(e)$ est minimale. De plus comme le *retiming* est légal, (12) est vérifiée par construction de E_{clock}, et donc $\Phi(G_q) = \Phi(G)$. □