Les problèmes d'ordonnancement cyclique dans la synthèse des circuits numériques

Imed Eddine Bennour et El Mostapha Aboulhamid

Département d'informatique et recherche opérationnelle
Université de Montréal
CP 6128, Centre ville, H3C-3J7,PQ, Canada

Résumé. *Le problème d'ordonnancement cyclique consiste à ordonner dans le temps l'exécution répétitive d'un ensemble d'opérations liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Cet article étudie ce problème dans le contexte de la synthèse de haut niveau des circuits numériques synchrones. Il présente les principaux résultats relatifs à ce problème et expose certaines méthodes pour le résoudre.*

1. Introduction

L'objectif de la synthèse de haut niveau est de trouver un circuit numérique synchrone qui satisfait une spécification donnée sous forme d'un algorithme. L'une des principales tâches de la synthèse de haut niveau est l'ordonnancement dans le temps des opérations que le circuit doit réaliser. Cet article traite le problème d'ordonnancement qui apparaît dans la synthèse des circuits réalisant des traitements itératifs. Un traitement itératif (ou répétitif) est équivalent à une boucle infinie. Ce type de traitements est fréquent dans les applications de traitement du signal où un même traitement est répété indéfiniment sur des données différentes. D'une façon informelle, le problème d'ordonnancement cyclique consiste à ordonner dans le temps l'exécution répétitive d'un ensemble d'opérations liées par des contraintes de précédence, en utilisant un nombre limité de ressources. Ce problème est étudié aussi dans les domaines de la recherche opérationnelle et de la compilation où certains résultats peuvent être exploités dans le domaine de la synthèse des circuits numériques.

La synthèse de haut niveau est présentement un domaine de recherche très actif. Les livres classiques en synthèse de haut niveau [DEM 94, GAJ 92, MIC 92] n'effleurent les ordonnancements cycliques que superficiellement. Cet article:

- Donne une introduction à la synthèse de haut niveau.
- Résume le principaux résultats relatifs aux problèmes d'ordonnancement cyclique et qui sont utiles pour la synthèse des circuits.
- Expose certaines méthodes pour résoudre les problèmes d'ordonnancement cyclique.

L'article est composé de 7 sections. La section 2 décrit les principales étapes de la synthèse de haut niveau. La section 3 présente une modélisation graphique des traitements itératifs et les critères d'optimisation dans l'ordonnancement des traitements itératifs. La section 4 définit le problème d'ordonnancement cyclique. La section 5

présente une synthèse des méthodes de résolution du problème d'ordonnancement cyclique. La section 6 traite le problème d'allocation des ressources dans les ordonnancements cycliques. Finalement, la section 7 présente certaines directions de recherche.

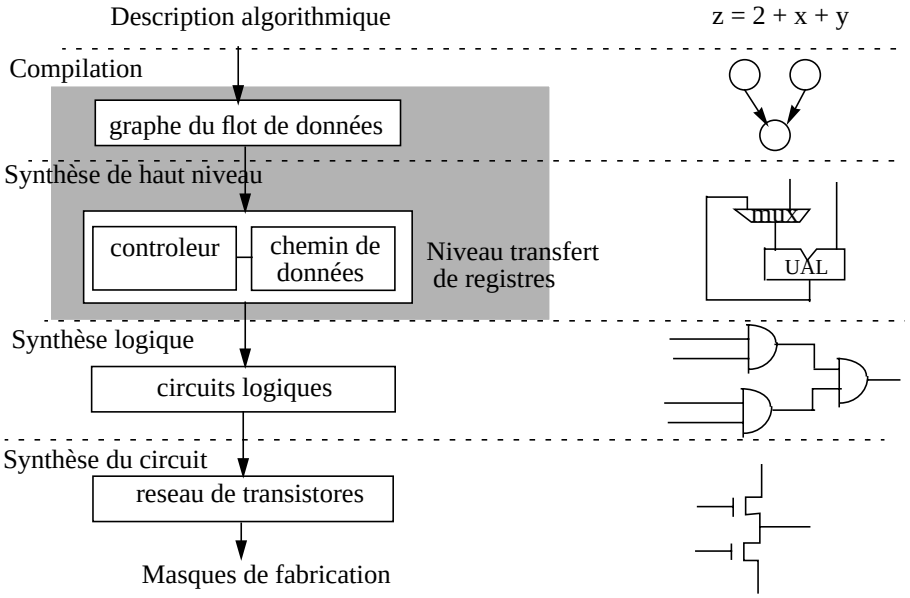


Figure 1. *Les différentes catégories de synthèse*

2. La synthèse de haut niveau

La figure 1 montre les principales étapes de conception d'un circuit. Chaque étape transforme la spécification du circuit en introduisant plus de détails. Au plus haut niveau d'abstraction, la spécification décrit la fonctionnalité du circuit en faisant abstraction de toute réalisation matérielle. La spécification est donnée dans un langage de description du matériel tel que VHDL. Elle est ensuite traduite en une représentation graphique appelée graphe du flot de données et de contrôle (GFD). Dans sa forme la plus simple, le GFD est un graphe orienté dont les noeuds représentent les opérations atomiques que le circuit doit réaliser, et les arcs représentent les précédences entre les opérations qui sont dues aux dépendances de données.

Exemple: La figure 2.a montre le GFD correspondant à l'expression

$$z = (2 + x) \times (x + y) \times (y + 5)$$

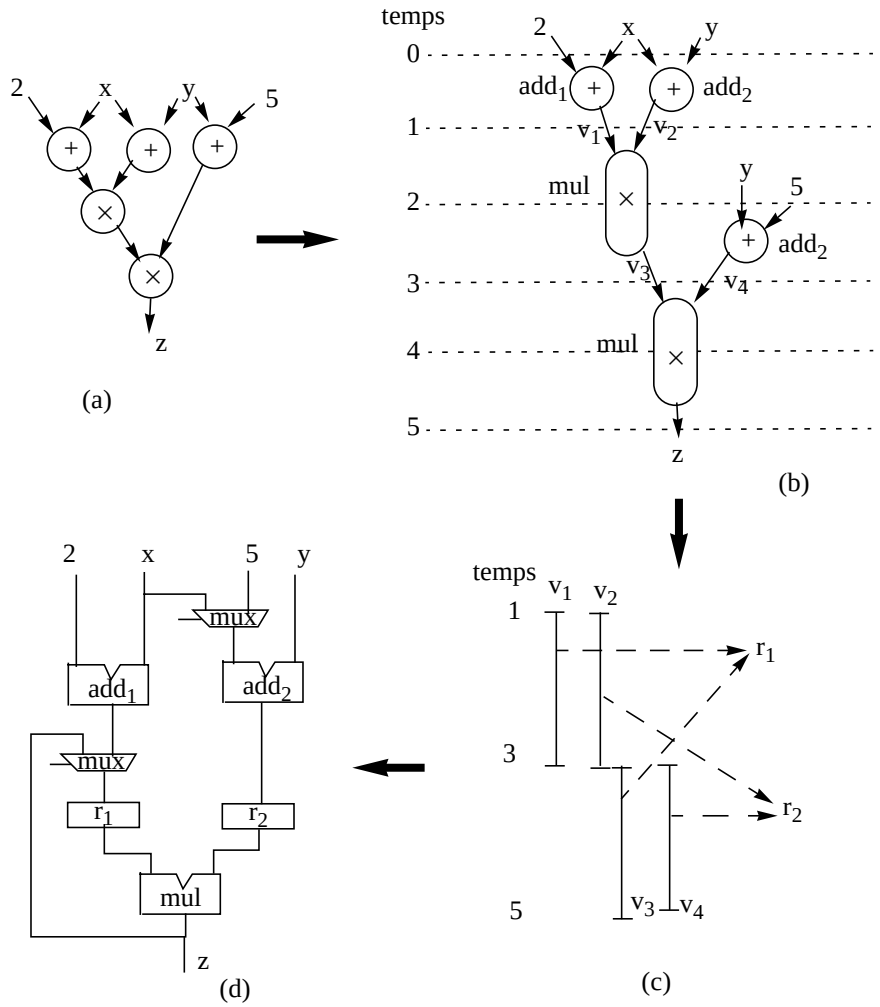


Figure 2. *Les différentes étapes de la synthèse de haut niveau:*
(a) Le GFD de l'expression $z = (2 + x) \times (x + y) \times (y + 5)$
(b) Ordonnancement (c) Les périodes de vie des variables avec une assignation à deux registres (d) Le chemin de données obtenu après la synthèse des connexions

La synthèse de haut niveau est une séquence de tâches qui transforment un GFD en une description plus détaillée appelée description au niveau transfert de registres. Cette dernière description donne les deux principales composantes formant un circuit, à savoir l'unité de traitement et l'unité contrôle. L'unité de traitement, appelée aussi *chemin de données*, contient les unités fonctionnelles et les registres. L'unité de contrôle détermine à chaque cycle d'horloge quelles opérations du GFD doivent être exé-

cutées et par quelles unités fonctionnelles.

La synthèse de haut niveau est composée de quatre tâches dépendantes les unes des autres:
(1) la sélection des unités fonctionnelles,
(2) l'ordonnancement des opérations du GFD,
(3) l'allocation des registres aux variables, et
(4) l'allocation des bus aux transferts de données.
Dans ce qui suit, nous décrivons brièvement l'objectif de chaque tâche. Pour plus de détails, le lecteur peut se référer à [DEM 94, GAJ 92].

(1) *Sélection des unités fonctionnelles*. Cette tâche consiste à sélectionner à partir d'une bibliothèque de composantes matérielles les types d'unités fonctionnelles qui seront utilisées pour exécuter les opérations du GFD. Une unité fonctionnelle est caractérisée principalement par les types d'opérations qu'elle réalise, sa vitesse d'exécution, son coût et sa taille.

(2) *Ordonnancement des opérations du* GFD. L'objectif de cette tâche est de déterminer un ordre statique d'exécution des opérations du GFD par les unités fonctionnelles. L'ordre doit respecter les dépendances qui existent entre les opérations. On distingue deux catégories d'ordonnancements: l'ordonnancement sous contraintes de ressources et l'ordonnancement sous contraintes de performance. Dans la première catégorie, le nombre maximal d'unités fonctionnelles à utiliser dans le circuit est fixé d'avance, et l'objectif consiste à minimiser la durée de l'ordonnancement. Dans la deuxième catégorie, la durée de l'ordonnancement est fixée, et l'objectif est de minimiser le nombre d'unités fonctionnelles utilisées.

> **Exemple**: Supposons qu'on dispose de deux additionneurs {add_1, add_2} et d'un multiplicateur (mul). La durée d'une addition est une unité de temps et celle d'une multiplication est deux unités de temps. La figure 2.b montre un ordonnancement réalisable du GFD.

(3) *Allocation des registres aux variables*. Chaque variable intermédiaire (donnée) produite par une opération du GFD devrait être sauvegardée dans un registre tant que toutes les opérations utilisant cette donnée n'ont pas été exécutées. La période de vie d'une variable est l'intervalle défini par l'instant de sa création et l'instant de sa dernière utilisation. Les variables qui ont des périodes de vie disjointes peuvent partager le même registre. L'un des principaux critères d'optimisation dans cette tâche est la minimisation du nombre de registres.

> **Exemple:** Le GFD de la figure 2 contient 4 variables intermédiaires {v_1, v_2, v_3, v_2}. Leur périodes de vie sont respectivement [1,3], [1,3], [3,5] et [3,5]. Pour sauvegarder ces variables, il faut au moins deux registres. Une solution d'allocation serait d'assigner au premier registre r_1, les variables v_1 et v_3, et au deuxième registre r_2 les variables v_2 et v_4.

(4) *Allocation des bus aux transferts de données*. L'objectif de cette tâche est d'établir les interconnexions entre les unités fonctionnelles et les registres afin de former un chemin de données complet. Le principal critère d'optimisation dans cette tâche est la minimisation du nombre de connexions et de multiplexeurs.

Exemple: Le chemin de données de l'exemple de la figure 2.a obtenu après la synthèse des connexions est montré dans la figure 2.d.

3. Les traitements itératifs: modélisation et ordonnancement

Nous considérons maintenant le cas où le circuit à synthétiser effectue un traitement itératif. Dans ce type de traitements, deux catégories de dépendances peuvent exister entre les opérations: les dépendances entre les opérations d'une même itération et les dépendances entre des opérations appartenant à des itérations différentes. Cette section présente une modélisation graphique des traitements itératifs et les critères d'optimisation dans l'ordonnancement d'un traitement itératif.

3.1. Modélisation des traitements itératifs

Les deux modèles fréquemment utilisés pour représenter les opérations d'un traitement itératif et les contraintes de précédence qui les relient sont les réseaux de Petri temporisés et les graphes de précédence réduits [CAR 88, CHR 83]. Dans cet article, nous utilisons le deuxième modèle qui est une extension du GFD.

Un graphe de précédence réduit est un graphe orienté défini par le quadruplet (O, A, D, H) où:

- $O = \{o_i\}$ est l'ensemble de noeuds du graphe; les noeuds représentent les opérations que le circuit doit réaliser. Ces opérations sont dites génériques étant donné qu'elles peuvent être exécutées plusieurs fois sur des données différentes. Une itération correspond à une exécution de l'ensemble des opérations génériques.

- $A \subset O \times O$ est l'ensemble des arcs du graphe reliant toute paire d'opérations ayant une contrainte de précédence entre elles. On note par a_{ij} l'arc de o_i vers o_j.

- $H : A \rightarrow$ N (l'ensemble des entiers naturels) appelée fonction de hauteur. La valeur $H(a_{ij})$ indique que le résultat de l'opération o_i d'une itération n quelconque est utilisé par l'opération o_j de l'itération $(n + H(a_{ij}))$. On dit que l'opération o_i et liée à l'opération o_j par une contrainte de précédence intra-itération (resp. par une contrainte de précédence inter-itérations) si $H(a_{ij}) = 0$ (resp. si $H(a_{ij}) > 0$).

- $D: A \rightarrow$ N* appelée fonction de durée. La valeur $D(a_{ij})$ est égale à la durée d'exécution de l'opération o_i. Notons que la fonction de durée aurait pu être associée aux noeuds du graphes, mais par souci de conformité et de généralité nous l'avons associée aux arcs.

De n = 4 à ∞ faire

Debut

o_0: a[n] ← E[n] × d[n-2];

o_1: b[n] ← c_1 + a[n];

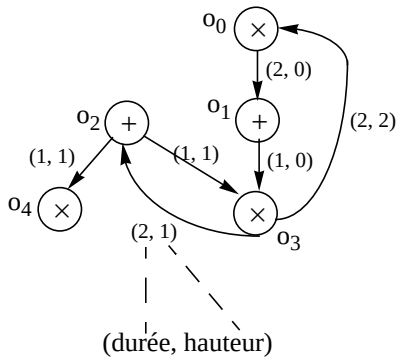
o_2: c[n] ← c_2 + d[n-1];

o_3: d[n] ← b[n] × c[n-1];

o_4: S[n] ← c_3 × c[n-1];

Fin

(a)



(b)

Figure 3. *Un exemple de traitement itératif. (a) Spécification (b) Graphe de précédence.*

Dans le reste de l'article on notera par $\langle o_i, n \rangle$ l'occurrence de l'opération o_i à l'itération n.

> **Exemple:** La figure 3 illustre une spécification d'un traitement itératif (a) et le graphe de précédence correspondant (b): la durée d'une addition est égale à une unité de temps et celle d'une multiplication est égale à deux unités de temps. Ce graphe de précédence sera utilisé tout le long de l'article.

La hauteur et la durée d'un circuit du graphe sont égales, respectivement, à la somme des hauteurs et à la somme des durées de ses arcs. Un graphe de précédence est cohérent s'il ne contient pas de circuit de hauteur nulle. Autrement certaines opérations ne pourront jamais être exécutées sans violer les contraintes de précédence.

Une opération générique est dite *réentrante* (resp. *non réentrante*) si les occurrences de cette opération peuvent (resp. ne doivent pas) être exécutées en parallèle. Dans le domaine de la synthèse de haut niveau ainsi que dans le reste de l'article on suppose par défaut que les opérations sont réentrantes. Notons que la non réentrance d'une opération o_i peut être modélisée dans le graphe de précédence par une boucle a_{ii} de hauteur égale à un.

3.2. Ordonnancement des traitements itératifs et critères d'optimisation

Le deux principales mesures de la performance d'un circuit effectuant un traitement itératif sont la fréquence et la latence. La *fréquence* correspond au nombre

moyen d'itérations exécutées par unité de temps; plus la fréquence d'un circuit est élevée plus sa performance est élevée. La *latence* est égale à la durée maximale d'exécution d'une itération. Dans les applications où une itération correspond au traitement d'un échantillon en entrée, la latence mesure la rapidité du circuit à traiter un échantillon.

L'objectif de l'ordonnancement d'un traitement itératif est de déterminer un ordre statique des exécutions répétitives des opérations génériques. L'ordre doit respecter les dépendances entre les opérations et les contraintes de ressources s'il y en a. Le premier critère d'optimisation de l'ordonnancement est la maximisation de la fréquence. Le second critère, valable uniquement pour certains types d'applications, est la minimisation de la latence. Pour atteindre des fréquences élevées, l'ordonnancement doit exploiter le deux niveaux de parallélisme présents dans les traitements itératifs: le parallélisme entre les opérations d'une même itération, et le parallélisme entre les opérations qui appartiennent à des itérations différentes.

On distingue, principalement, deux catégories d'ordonnancements cycliques: (1) les ordonnancements périodiques et (2) les ordonnancements K-périodiques. Dans la catégorie (1) toutes les itérations ont le même ordonnancement, et la même durée sépare les débuts des exécutions de deux itérations successives. Dans la catégorie (2) l'ordonnancement de K itérations successives est fixe et se répète à un intervalle régulier.

> **Exemple:** La figure 4 montre un ordonnancement périodique de fréquence égale à 1/3 et de latence égale à 5. Un ordonnancement K-périodique du même graphe est donné dans la figure 5. La fréquence et la latence de cet ordonnancement sont respectivement de 2/5 et 5.

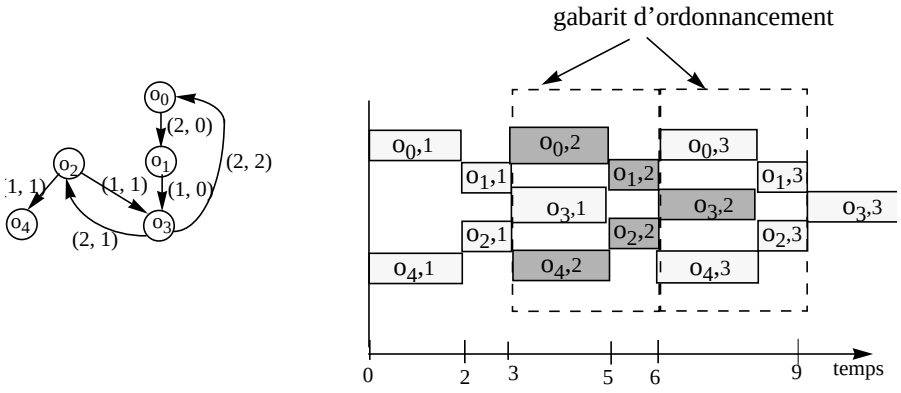


Figure 4. *Un ordonnancement périodique de fréquence 1/3 et de latence 5.*

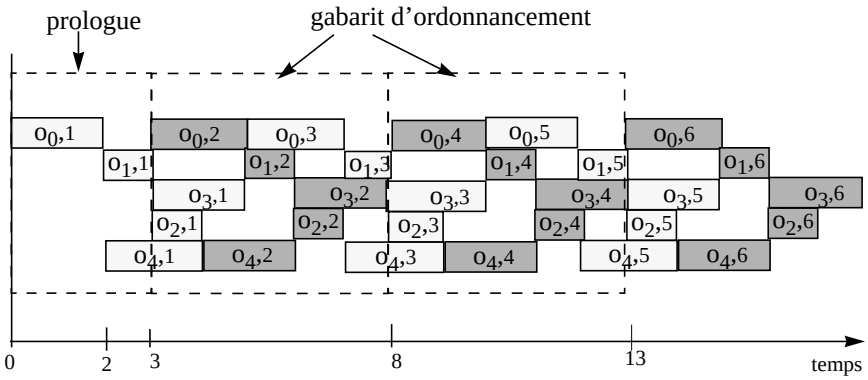


Figure 5. *Un ordonnancement K-périodique de fréquence 2/5.*

L'avantage des ordonnancements périodiques est qu'ils engendrent des architectures matérielles assez régulières. Cependant les ordonnancements K-périodiques permettent d'atteindre des fréquences plus élevées.

Un ordonnancement cyclique est *stable* s'il existe un instant à partir duquel il apparaît un *gabarit d'ordonnancement* qui se répète d'une façon périodique (voir la figure 5). Le *prologue* d'un ordonnancement stable est la partie de l'ordonnancement qui précède l'apparition du premier gabarit. Un ordonnancement stable est complètement défini par son prologue et son gabarit. Ainsi, pour réaliser matériellement un ordonnancement cyclique stable il suffit de mémoriser son prologue et son gabarit. La réalisation des ordonnancements qui ne sont pas stables nécessite une mémoire de taille exorbitante et donc ils ne sont pas intéressants.

3.3. Le dépliage du graphe de précédence

Le résultat de L dépliages d'un graphe de précédence est un nouveau graphe de précédence représentant les opérations de L itérations successives et les contraintes de précédence intra-itération et inter-itérations qui les relient. Le nombre de noeuds et le nombre d'arcs dans un graphe déplié L fois sont égaux, respectivement, à $L|O|$ et $L|A|$, où $|O|$ et $|A|$ sont, respectivement, le nombre de noeuds et d'arcs dans le graphe initial.

Exemple: La figure 6 montre le résultat de deux dépliages.

Notons qu'après un nombre élevé de dépliages, les contraintes de précédence deviennent uniquement entre des opérations d'une même itération ou entre des opérations appartenant au plus à deux itérations successives (autrement dit, les hauteurs des arcs deviennent égales à zero ou un).

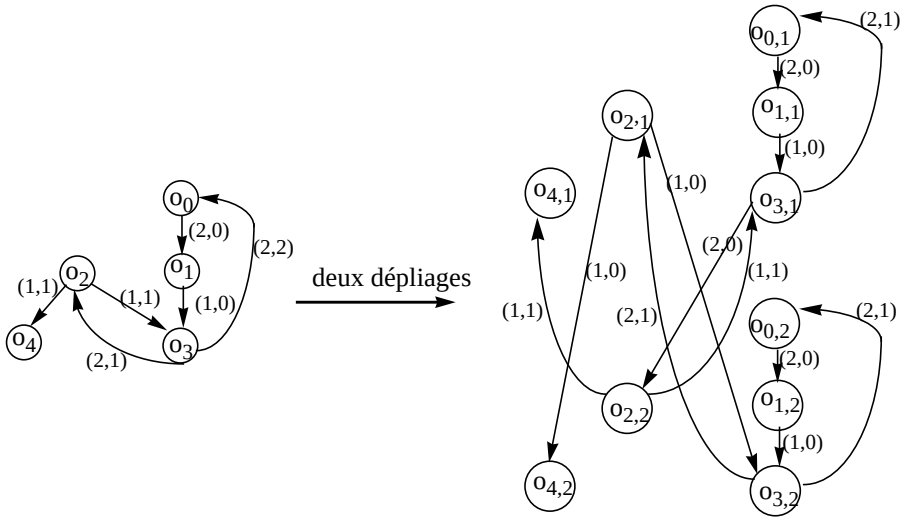


Figure 6. *Dépliage d'un graphe avec L=2.*

L'opération de dépliage a au moins deux utilités pour l'ordonnancement. D'une part, le dépliage rend plus explicite le parallélisme existant entre les opérations appartenant à des itérations différentes. Ce qui en général permet d'augmenter l'efficacité (en terme de fréquence) de certains algorithmes d'ordonnancement. D'autre part, certains algorithmes d'ordonnancement sont restreints à des graphes de précédence où les dépendances entre les opérations sont au plus entre deux itérations successives. Le dépliage permet d'utiliser ces algorithmes pour ordonner des graphes quelconques.

Voici un algorithme [PAR 91] qui permet de construire le graphe L fois déplié d'un graphe de précédence donné.

1: Pour chaque noeud o_i du graphe initial créer L noeuds $(o_{i,1},\ldots,o_{i,L})$.

2: Pour chaque arc a_{ij} du graphe initial faire:

-Si $H(a_{ij}) = 0$, alors ajouter dans le nouveau graphe des arcs $o_{i,q} \rightarrow o_{j,q}$ $(\text{de } q = 1 \text{ à } L)$ avec une hauteur nulle.

-Si $H(a_{ij}) < L$, alors ajouter des arcs $o_{i,q-H(a_{ij})} \rightarrow o_{j,q}$ $(\text{de } q = H(a_{ij}) + 1 \text{ à } L)$ avec une hauteur nulle. Et ajouter des arcs $o_{i,L-H(a_{ij})+q} \rightarrow o_{j,q}$ $(\text{de } q = 1 \text{ à } H(a_{ij}))$ avec une hauteur égale à un.

-Si $H(a_{ij}) \geq L$, ajouter des arcs $o_{i,\left\lceil \frac{H(a_{ij})-q+1}{L} \right\rceil L-H(a_{ij})+q} \rightarrow o_{j,q}$ avec une hauteur égale à $\left\lceil \frac{H(a_{ij})-q+1}{L} \right\rceil$ $(\text{de } q = 1 \text{ à } L)$.

3: Associer à chaque arc $o_{i,q} \rightarrow o_{*,*}$ $(\forall o_i, q = 1 \text{ à } L)$ une durée égale à $D(a_{ij})$.

4. Spécification du problème d'ordonnancement cyclique et calcul de la fréquence maximale

Dans la première partie de cette section nous donnons les définitions précises du problème d'ordonnancement cyclique et des ordonnancements K-périodiques et périodiques. Dans la seconde partie, nous étudions les facteurs qui limitent la fréquence d'un ordonnancement.

4.1. Modèle de ressources et définitions

Le modèle de ressources. On considère que chaque opération générique est affectée au préalable à un type d'unité fonctionnelle. Plusieurs opérations peuvent être affectées au même type d'unité fonctionnelle. Il peut y avoir plusieurs instances d'un même type d'unité fonctionnelle. Une unité fonctionnelle est dite *homogène* si elle peut exécuter n'importe quelle opération générique.

L'ordonnancement d'un graphe de précédence. Soit $G = (O, A, D, H)$ un graphe de précédence et soit Q^+ l'ensemble des rationnels positifs. Un ordonnancement cyclique de G est une fonction $\tau : O \times \mathrm{N}^* \rightarrow Q^+$ qui associe à chaque occurrence $\langle o_i, n \rangle$ l'instant de son début exécution. Un ordonnancement τ est réalisable ssi les deux contraintes suivantes sont vérifiées:

(1) Contraintes de précédence:

$$\forall a_{ij} \in A, \forall n \in \mathrm{N}^* . \ \tau\langle o_j, H(a_{ij}) + n\rangle \geq \tau\langle o_i, n\rangle + D(a_{ij}) \qquad [1]$$

(2) Contraintes de ressources: il existe une allocation des unités fonctionnelles aux opérations sans qu'il y ait un conflit de ressources.

La fréquence d'un ordonnancement cyclique τ est égale à

$$\lim_{n \to \infty} \frac{n}{\max_{o_i} \{ \tau\langle o_i, n\rangle + D(a_{ij}) \}}$$

Le problème d'ordonnancement cyclique (POC). Etant donné un graphe de précédence, une affectation des opérations génériques aux types d'unité fonctionnelles et un nombre d'instances de chaque type d'unité, le POC est de déterminer un ordonnancement réalisable de fréquence maximale.

Ordonnancement K-périodique. Un ordonnancement τ est dit K-périodique s'il existe deux entiers n_0, K et un rationnel positif P tel que:

$$\forall n \geq n_o, \forall o_i \in O . \ \tau\langle o_i, n + K\rangle = \tau\langle o_i, n\rangle + P$$

K et *P* sont appelés, respectivement, la périodicité et la période de τ. La fréquence de τ est égale à $\frac{K}{P}$.

Ordonnancement périodique. Un ordonnancement τ est périodique s'il existe un rationnel positif *P* tel que

$$\forall n \geq 2, \forall o_i \in O. \quad \tau\langle o_i, n\rangle = \tau\langle o_i, n-1\rangle + P \qquad [2]$$

La fréquence de τ est égale à $\frac{1}{P}$. Un ordonnancement périodique est un ordonnancement K-périodique de périodicité égale à un.

4.2. Facteurs affectant la fréquence

Deux facteurs limitent la fréquence maximale d'un traitement itératif: les contraintes de ressources et les dépendances entre les opérations. La valeur de la fréquence maximale due aux contraintes de ressources, notée F_{ress}, est donnée par l'expression suivante:

$$F_{ress} = \min_h \left\{ \frac{m_h}{\sum_{o_i \in O_h} D(a_{i*})} \right\} \qquad [3]$$

où m_h est le nombre d'unités fonctionnelles de type *h* disponibles, O_h est l'ensemble d'opérations génériques qui doivent être exécutées sur des unités de type *h*, et $D(a_{i*})$ est la durée de l'opération générique o_i.

La fréquence maximale due aux dépendances entre les opérations est appelée *fréquence critique* et notée F_{crit}. La valeur de F_{crit} est donnée par la formule suivante [CHR 83]:

$$F_{crit} = \min_{C_k} \left\{ \frac{\sum_{a_{ij} \in C_k} H(a_{ij})}{\sum_{a_{ij} \in C_k} D(a_{ij})} \right\} \qquad [4]$$

où C_k est un circuit élémentaire dans le graphe G. Cette formule découle de l'équation [1]. Si G ne contient pas de circuits, alors la fréquence critique est infinie et toutes les itérations peuvent s'exécuter en parallèle s'il y a suffisamment de ressources. Un circuit est dit critique si le rapport entre sa hauteur et sa durée est égale à la fréquence critique.

Exemple: Le graphe de précédence de la figure 3 contient deux circuits $C_1 = o_0 \rightarrow o_1 \rightarrow o_3 \rightarrow o_0$, $C_2 = o_2 \rightarrow o_3 \rightarrow o_2$
La valeur de la fréquence critique est égale à $\min\left(\frac{2}{5}, \frac{2}{3}\right) = \frac{2}{5}$. Le circuit C_1 est critique.

La fréquence maximale réalisable, notée F_{max}, est bornée par le minimum de F_{crit} et F_{ress}:

$$F_{max} \leq \min (F_{ress}, F_{crit}) \qquad [5]$$

4.3. Méthode de calcul de la fréquence critique

La fréquence critique d'un graphe de précédence peut être calculée en un temps polynômial. Cette section présente une méthode simple et efficace pour calculer la fréquence critique; d'autres méthodes sont décrites dans [KAR 78, ZAK 89, GER 92]. Notons que le calcul de la fréquence critique directement à partir de la formule [4] peut prendre théoriquement un temps exponentiel vu que le nombre de circuits élémentaires dans un graphe peut être exponentiel.

Bornes inférieure et supérieure de la fréquence critique: À partir de la formule [4], on peut déduire que

$$F_{crit} \in \left[\frac{1}{|O| \cdot d_{max}}, h_{max}\right]$$

où, $d_{max} = \max \{D(a_{ij})\}$, $h_{max} = \max \{H(a_{ij})\}$

La borne inférieure de F_{crit} correspond au cas où le graphe contient un circuit qui passe par tous les noeuds et que tous les arcs du circuit ont une durée égale à d_{max} et une hauteur nulle, à l'exception d'un arc qui a une hauteur égale à un. La borne supérieure correspond au cas où le graphe contient un circuit qui passe par tous les noeuds et que tous les arcs du circuit ont une hauteur égale à h_{max} et une durée égale à un.

La méthode de calcul de la fréquence critique est basée sur la propriété [DON 92] suivante qui découle de la contrainte de l'équation [1].

Soient $G = (O, A, D, H)$ un graphe de précédence et F un rationnel positif. F est une fréquence réalisable de G si le graphe $\hat{G} = (O, A, D, \hat{H})$ où

$$\hat{H}(a_{ij}) = D(a_{ij}) - \frac{1}{F} \cdot H(a_{ij})$$

ne contient pas de circuit de hauteur positive.

Ainsi, le calcul de la fréquence critique revient à la recherche de la plus grande valeur dans l' intervalle $[\frac{1}{|O| \cdot d_{max}}, h_{max}]$ qui vérifie la propriété précédente. La complexité de cette méthode en temps est de $O(|O||A| \log(|O|d_{max}))$ [DON 92].

5. Le problème d'ordonnancement cyclique: complexité et méthodes de résolution

La difficulté dans la résolution du problème d'ordonnancement cyclique (POC) est due principalement aux contraintes de ressources et à la présence de circuits dans le graphe de précédence. Dans cette section nous étudions le POC en distinguant les trois cas suivants: (1) cas où il n'y a pas des contraintes de ressources, (2) cas où le graphe de précédence est acyclique, (3) cas général où il y a des contraintes de ressources et des circuits dans le graphe de précédence. Les cas particuliers (1) et (2) ont été distingués car ils sont fréquents dans la synthèse des circuits numériques. En fait, pour certaines applications qui requièrent des fréquences élevées les contraintes sur les ressources sont secondaires, et plusieurs traitements itératifs ne contiennent pas des contraintes de précédence inter-itérations, et donc ils ont des graphes de précédence acycliques.

5.1. Le POC sans contraintes de ressources

Dans le cas où il n'y a pas de contraintes de ressources, le POC est calculable en un temps polynômial. La fréquence optimale est égale à la fréquence critique. D'autre part, il existe toujours des ordonnancements K-périodiques à valeur entière et de fréquence optimale. En plus, il existe toujours des ordonnancements périodiques, pas nécessairement à valeur entière, de fréquence optimale. On dit que les ordonnancements périodiques sont *dominants* dans le cas où il n'y a pas de contraintes de ressources. Dans ce qui suit, nous décrivons une méthode simple pour calculer ces deux types d'ordonnancements.

D'après l'équation [2], les ordonnancements périodiques de fréquence maximale sont de la forme:

$$\forall n \geq 2, \forall o_i \in O \bullet \ \tau\langle o_i, n\rangle = \tau\langle o_i, n-1\rangle + \frac{1}{F_{crit}} \qquad [6]$$

Une fois la valeur de F_{crit} est calculée, les valeurs $\tau\langle o_i, 1\rangle$ peuvent être déduites facilement. Ces valeurs doivent satisfaire la contrainte de l'équation [1]. En combinant les équations [1] et [6], on obtient:

$$\forall a_{ij} \in A \bullet \ \tau\langle o_i, 1\rangle - \tau\langle o_j, 1\rangle \leq \frac{H(a_{ij})}{F_{crit}} - D(a_{ij})$$

Ce système est une version du problème du plus court chemin. Il peut être résolu par l'algorithme de Bellman-ford [COR 90] en un temps $O(|O||A|)$.

Une fois l'ordonnancement périodique calculé, il suffit d'appliquer le théorème suivant [HAN 94] pour obtenir un ordonnancement K-périodique de fréquence maximale. Ce théorème n'est valable qu'en l'absence de contrainte de ressources.

Si τ *est un ordonnancement périodique, alors* τ^* *défini comme suit:*

$$\forall o_i \in O, \forall n \geq 1 \,.\quad \tau^*\langle o_i, n\rangle = \lfloor \tau\langle o_i, n\rangle \rfloor$$

est un ordonnancement K-périodique et de même fréquence que τ

Dans le cas où il n'y a pas des contraintes de ressources, d'autres méthodes polynomiales ont été proposées dans [SCH 89, ZAK 89, IWA 90, CHA 93a, JEN 94] pour calculer des ordonnancements K-périodiques, à valeur discrète et de fréquence maximale. La méthode de Iwano *et al*. [IWA 90] est efficace car elle permet d'obtenir des ordonnancements de fréquence maximale et de gabarit de taille minimale.

L'ordonnancement cyclique au plus tôt. Considérons le mode d'ordonnancement qui consiste à ordonner les instances des opérations génériques au plus tôt: l'exécution d'une opération est activée dès que l'exécution de ses prédécesseurs est terminée. Un tel ordonnancement est dit *au plus tôt*. Carlier *et al*. [CHR 88] ont montré que dans le cas où les opérations sont non réentrantes et que le graphe de précédence est fortement connexe, les ordonnancements *au plus tôt* ont une structure K-périodiques; avec une périodicité *K* égale au produit des hauteurs des circuits critiques et une période égale à $K \cdot F_{crit}$. Ce résultat est aussi valable pour certains graphes faiblement connexes [CHR 88]. Les ordonnancements *au plus tôt* ont l'avantage d'avoir une latence minimale en plus de l'exécution au plus tôt des itérations. Par contre la taille du gabarit de ces ordonnancements peut être trop grande.

L'ordonnancement cyclique avec date limite. On suppose maintenant que la date de la fin d'ordonnancement de chaque itération *n* est soumise à une date limite Δ_n . Il est intéressant de savoir le moment le plus tard $t_\Delta\langle o_i, n\rangle$ de l'ordonnancement de l'opération $\langle o_i, n\rangle$ tel que les dates limites soient respectées. Ce problème a été étudié dans [CHR 91]. Il a été montré que dans le cas particulier où chaque Δ_n est égale au moment le *plus tôt* du fin d'exécution de l'itération *n*, l'ordonnancement $t_\Delta = \{t_\Delta\langle o_i, n\rangle\}$ à une structure K-périodique avec une périodicité *K* égale au produit des hauteurs des circuits critiques et une période égale à $K \cdot F_{crit}$.

5.2. Le POC avec des graphes de précédence acycliques

Munier a montré dans [MUN 91] que si le graphe de précédence ne contient pas de circuits alors le POC est polynômial. Notons que le problème de la minimisation de la latence pour une fréquence fixée est NP-complet même si le graphe de précédence est acyclique [GRO 92].

5.3. Le POC avec contraintes de ressources et graphe de précédence cyclique

Dans son cas général le POC est un problème NP-complet, même si le graphe de précédence est composé uniquement d'un circuit [HAN 94]. Un algorithme polynômial proche de l'optimum pour résoudre le problème à *m* unités fonctionnelles homogènes est donné dans [GAS 92]. La fréquence de l'ordonnancement obtenue par cet algorithme est égale dans le pire cas au deux tiers de la fréquence optimale.

Plusieurs méthodes heuristiques ont été proposées dans la littérature pour résoudre le POC. Nous décrivons dans ce qui suit les trois approches d'ordonnancement fréquemment utilisées.

1*ère approche*. Elle consiste à fixer un même ordonnancement pour toutes les itérations. Cet ordonnancement doit satisfaire les contraintes de précédence intra-itération et les contraintes de ressources. Puis les moments de début d'exécution des itérations sont calculées de sorte que les contraintes de précédence inter-itérations et les contraintes de ressources soient respectées. Cette approche est souvent utilisée avec le modèle des tables de réservation [KOG 81]. Les ordonnancements obtenus ont généralement une structure K-périodique. Des algorithmes qui utilisent cette approche sont décrits dans [SU 87, LIU 89, HAN 90].

2*ème approche*. Cette approches est itérative et consiste à fixer la fréquence à une certaine valeur cible inférieure ou égale à la borne supérieure donnée par l'équation [5], puis à chercher un ordonnancement périodique ayant cette fréquence. Si au bout d'un certain temps la recherche échoue, la fréquence cible est réduite et le processus est réitéré. Théoriquement, cette approche permet d'atteindre la fréquence maximale si le graphe de précédence est suffisamment déplié. Des algorithmes qui utilisent cette approche sont décrits dans [GRO 92, WAN 93, LEE 94].

3*ème approche*. Il s'agit d'une approche constructive. On commence avec une solution d'ordonnancement réalisable puis on la modifie dans le but d'améliorer la fréquence. Chao *et al*. ont montré dans [CHA 93b] que la technique de resynchronisation pouvait être utilisée pour déterminer des ordonnancements efficaces d'une manière constructive. Cette technique est détaillée dans la section suivante.

5.4. La resynchronisation et son utilisation dans l'ordonnancement

5.4.1. Définition de la resynchronisation

La technique de resynchronisation a été développée initialement pour réduire la période d'horloge dans les circuits synchrones [LEI 91]. Elle consiste à réorganiser le graphe de précédence en modifiant les hauteurs des arcs.

Soit $G = (O, A, D, H)$ un graphe de précédence et soit s une fonction de O dans Z (l'ensemble des entiers) dite fonction de resynchronisation. On appelle graphe resynchronisé de G par rapport à s le graphe $G_s = (O, A, D, H_s)$ où

$$\forall a_{ij} \in A \;.\; H_s(a_{ij}) = H(a_{ij}) + s(o_j) - s(o_i)$$

Exemple: La figure 7 montre le graphe obtenu après la resynchronisation: $s(o_0) = s(o_1) = s(o_4) = -2$; $s(o_2) = s(o_3) = -1$.

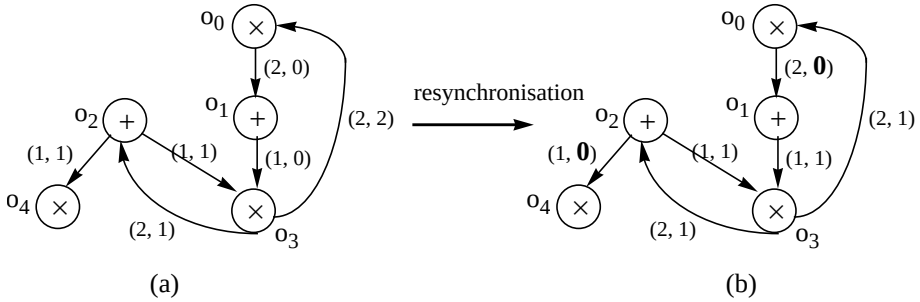


Figure 7. *Graphe de précédence après une resynchronisation.*

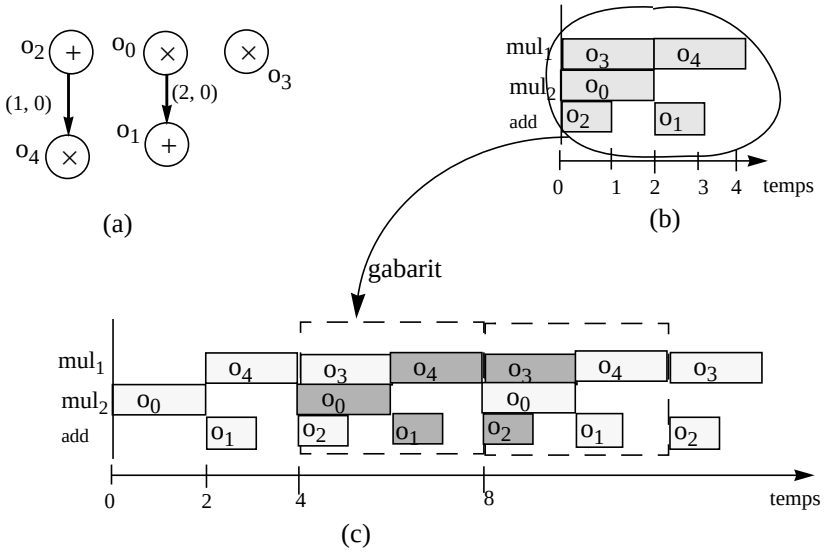


Figure 8. *(a) Le graphe resynchronisé de la figure 7.b avec seulement les dépendances intra-itération (b) Un ordonnancement non cyclique de ce graphe (c) L'ordonnancement cyclique dérivé à partir de l'ordonnancement non cyclique*

5.4.2. Relation entre resynchronisation et ordonnancement

Soit G un graphe de précédence et soit G_s le graphe obtenu par une resynchronisation de G. On peut démontrer [CHA 93b] que tout ordonnancement des opérations de G_s qui satisfait uniquement les contraintes de précédence intra-itération (arcs de hauteurs nulles) et les contraintes de ressources est un gabarit d'un ordonnancement périodique de G. Cette propriété montre que la solution d'un problème d'ordonnancement cyclique peut être obtenue en résolvant un problème d'ordonnancement simple (non cyclique) précédé d'une resynchronisation. La difficulté réside dans le choix de la resynchronisation à appliquer.

> **Exemple:** Le graphe de la figure 8.a est le graphe resynchronisé de la figure 7.b avec seulement les dépendances intra-itération. Supposons qu'on dispose de deux multiplicateurs et d'un additionneur. Un ordonnancement qui satisfait les dépendances intra-itération et les contraintes de ressources est donné dans la figure 8.b. Cette ordonnancement est le gabarit de l'ordonnancement cyclique périodique de la figure 8.c.

5.4.3. Une heuristique d'ordonnancement utilisant la resynchronisation

Dans ce qui suit on présente une heuristique illustrant l'utilisation de la resynchronisation dans un algorithme d'ordonnancement. L'heuristique consiste à effectuer d'une façon répétitive la resynchronisation suivante

$$\forall o_i \in O.\ s(o_i) = \begin{cases} -1 \text{ si } \forall a_{ji}, H(a_{ji}) > 0 \\ 0, \text{autrement} \end{cases}$$

suivie d'un ordonnancement, jusqu'à l'obtention d'une fréquence satisfaisante. L'heuristique est composée de trois étapes:

(a) Appliquer sur le graphe de précédence courant G la resynchronisation s. Le graphe résultant est G_s.

(b) Trouver un ordonnancement des opérations de G_s qui satisfait uniquement les contraintes de précédence intra-itération et les contraintes de ressources. Le critère d'optimisation est la latence de l'ordonnancent. À ce niveau, on peut utiliser les algorithmes d'ordonnancement existants pour résoudre les problèmes d'ordonnancement non cycliques.
Rappelons que l'ordonnancement obtenu est le gabarit d'un ordonnancement cyclique de G.

(c) Si la latence obtenue à l'étape (b) n'est pas satisfaisante, reprendre l'étape (a) avec $G \leftarrow G_s$.

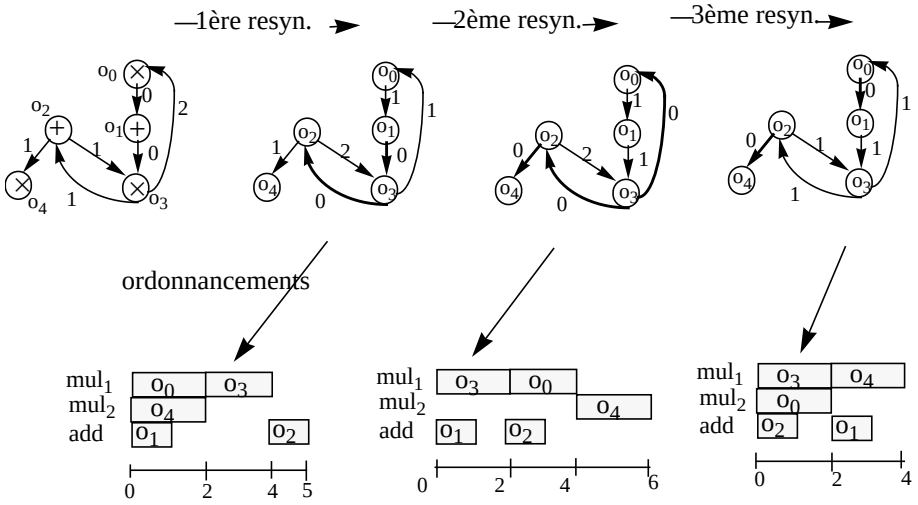


Figure 9. *Illustration de l'heuristique d'ordonnancement (nous avons omis* les durées des arcs).
1*ère resyn.* $s(o_0) = s(o_2) = s(o_4) = -1$; $s(o_1) = s(o_3) = 0$;
2*ème resyn.* $s(o_0) = s(o_1) = s(o_4) = -1$; $s(o_2) = s(o_3) = 0$;
3*ème* resyn. $s(o_1) = s(o_3) = -1$; $s(o_0) = s(o_2) = s(o_4) = 0$;

Illustration de l'heuristique. La figure 9 montre les trois itérations (une resynchronisation suivie d'un ordonnancement) effectuées avant l'obtention du gabarit de taille minimale. L'ordonnancement cyclique de fréquence maximale dérivé à partir du gabarit de taille minimale est celui de la figure 8.c.

6. Le problème d'allocation des ressources dans les ordonnancements cycliques

Dans la synthèse de haut niveau, les deux tâches qui viennent après l'ordonnancement sont l'allocation des unités fonctionnelles aux opérations et l'allocation des registres aux variables. Ces deux tâches sont discutées dans cette section.

6.1. Allocation des unités fonctionnelles

L'allocation des unités fonctionnelles consiste à assigner à chaque opération l'unité fonctionnelle qui doit l'exécuter. Pour un même ordonnancement, il peut exister plusieurs solutions d'allocation qui utilisent le même nombre d'unités fonctionnelles. Nous présentons dans ce qui suit certaines catégories d'allocation simples à réaliser matériellement.

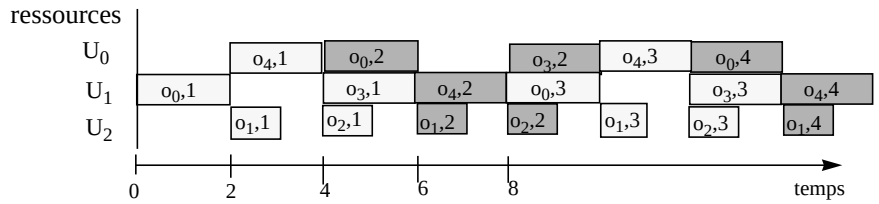


Figure 10. *Un ordonnancement périodique avec une allocation de type permutation.*

Notons par $U = \{U_0, U_1, \ldots, U_{m-1}\}$ l'ensemble des unités fonctionnelles disponibles, et par $r: O \times \mathrm{N}^* \rightarrow U$ la fonction d'allocation qui assigne à chaque opération $\langle o_i, n\rangle$ une unité fonctionnelle.

Une allocation r est de type permutation, s'il existe une permutation $p: U \rightarrow U$ telle que

$$\forall o_i \in O,\ \forall n > 1 .\quad r\langle o_i, n\rangle = p\,(r\langle o_i, n-1\rangle)$$

L'allocation r est complètement définie par la permutation p et les valeurs initiales de $r\langle o_i, 1\rangle$. Par conséquent, pour réaliser matériellement r il suffit de mémoriser la permutation p et l'ensemble $\{r\langle o_i, 1\rangle\}$

Exemple: La figure 10 montre un ordonnancement périodique qui utilise trois ressources et une allocation de type permutation. L'allocation est définie par:

$p\,(U_0) = U_1$, $p\,(U_1) = U_0$, $p\,(U_2) = U_2$

$r\langle o_0, 1\rangle = r\langle o_3, 1\rangle = U_1$, $r\langle o_1, 1\rangle = r\langle o_2, 1\rangle = U_2$, $r\langle o_4, 1\rangle = U_0$

Voici trois permutations particulières:

- La permutation identité:

$$\forall U_i \in U .\quad p\,(U_i) = U_i$$

Les allocations définies par la permutation identité sont simples à réaliser car la même unité fonctionnelle est allouée à toutes les instances d'une opération générique. Par contre, elles limitent la fréquence d'exécution des itérations.

-La permutation circulaire:

$$\forall U_i \in U .\quad p\,(U_i) = U_{(i+1) \bmod m}$$

où m est le nombre d'unités fonctionnelles. Hanen *et al*. [HAN 95] ont montré que les allocations définies par une permutation circulaire sont dominantes dans les or-

donnancements périodiques à opérations non réentrantes. En d'autres termes il existe toujours un ordonnancement périodique de fréquence maximale avec une allocation de ressources circulaire. Notons que dans ce type d'allocation on suppose que toutes les unités fonctionnelles sont homogènes.

- *La permutation cyclo-statique* [SCH 85]:

$$\forall U_i \in U . \quad p(U_i) = U_{(i+d) \bmod m}$$

où *d* est une constante positive, appelée facteur de déplacement des ressources. La permutation circulaire est un cas particulier de la permutation cyclo-statique.

Comme direction de recherche, il serait intéressant d'étudier le problème d'ordonnancement cyclique sous des contraintes relatives au mode d'allocation des unités fonctionnelles aux opérations en plus des contraintes sur le nombre d'unités disponibles.

6.2. Allocation des registres aux variables

Exemple: Le graphe de la figure 11.a contient 4 variables intermédiaires. Les durées de vie de ces variables obtenues après l'ordonnancement des opérations sont représentées par des segments dans la figure 11.c. À titre d'exemple, la variable v_2 est modifiée par l'opération $<o_2,1>$ à l'instant 5, et elle est utilisée une itération plus tard par l'opération $<o_3,2>$ dont l'exécution se termine à l'instant 10. Remarquons qu'il y a toujours deux occurrences de la variable v_2 qui sont en vie en même temps, ce qui nécessite au moins deux registres pour les sauvegarder. Le nombre total de registres nécessaires est 4.

Etant donnés un graphe de précédence et son ordonnancement, le problème d'allocation des registres consiste à trouver une assignation des variables aux registres qui minimise le nombre total de registres. Ce problème est NP-complet [DEM 94]; il est polynômial dans les ordonnancements non-cycliques. Des heuristiques d'allocation sont données dans [HEN 92, RAU 92, ALO 94].

Le nombre de registres peut être aussi donné comme une contrainte et le problème serait de trouver un ordonnancement de fréquence maximale sous cette contrainte. Peu de travaux [HAN 90] ont traité ce problème.

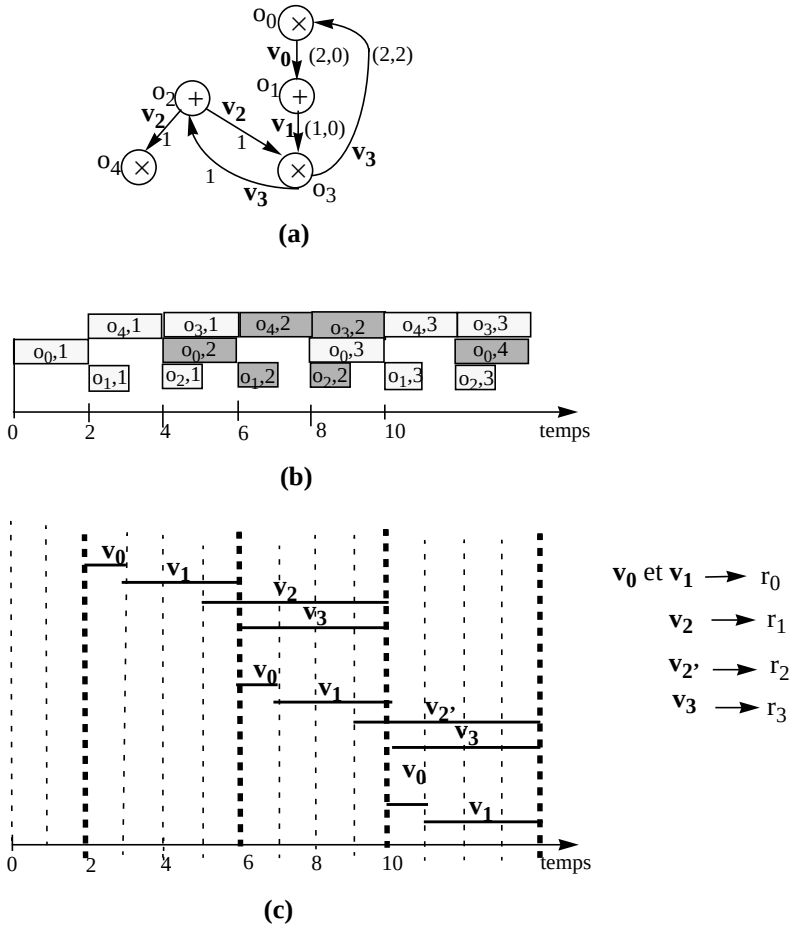


Figure 11. *Un exemple d'allocation des registres aux variables. (a) Les variables du graphe de précédence (b) Un ordonnancement (c) Durée de vie des variables et allocation des registres.*

Tableau 1 : Complexité des problèmes liés aux ordonnancements cycliques

Problèmes	Contraintes de ressources	Circuits dans le graphe de précédence	Complexité	Référ. (algorithmes exacts ou heuristiques)
Calcul de la fréquence critique	X†	1	Polynomiale	[KAR 78, ZAK 89, GER 92]
Maximisation de la fréquence (POC)	1	1	NP-complet	[SU 87, LAM 88, LIU 89, GOO 90, HAN 90, AIK 91, GAS 92, CHA 93b]
Maximisation de la fréquence	1	0	Polynomiale	[MUN 91]
Maximisation de la fréquence	0	1	Polynomiale	[SCH 89, ZAK 89 IWA 90, PAR 91 DON 92, CHA 93a, JEN 94]
Minimisation de la latence pour une fréquence donnée	1	X	NP-complet	[PAR 88, GRO 92, HWA 91, WAN 93, LEE 94]
Minimisation des registres pour un ordonnancement donné	X	1	NP-complet	[HEN 92, RAU 92, ALO 94]

† 1= contrainte présente, 0 = contrainte absente, X= 0 ou 1.

7. Conclusion et directions de recherche

Dans cette étude nous avons présenté les résultats que nous avons considérés pertinents pour les problèmes de l'ordonnancement cyclique dans le cadre de la synthèse de haut niveau. Le tableau 1 résume la complexité de ces problèmes et donne les références aux méthodes existantes pour les résoudre. Ces résultats ont été empruntés aux trois domaines où ce problème surgit, à savoir la compilation, la recherche opérationnelle (RO) et la synthèse de haut niveau (SHN). Ce dernier domaine possède ses caractéristiques propres qui le différencient des deux autres. Alors que la compilation suppose des processeurs séquentiels ou à la rigueur des multiprocesseurs homogènes, la SHN va s'intéresser à une granularité plus fine du parallélisme. Le domaine de la RO foisonne de résultats intéressants mais certaines hypothèses telles que la non-réen-

trance limitent l'utilisation immédiate de ces résultats, il serait profitable de réadapter ces résultats au domaine de la SHN. D'autre part, les boucles ne sont qu'un cas particulier des structures de contrôle qui peuvent exister dans un traitement itératif, en effet ce dernier peut avoir des énoncés conditionnels ainsi que des boucles imbriqués. L'accélération des boucles imbriqués pose un problème d'ordonnancement dit multidimensionnel, c'est un champ encore très peu exploré [WOL 91, PAS 94]. Le problème d'optimisation des connexions est particulier à la SHN et peut être primordial pour des circuits tels que les FPGAs. Le problème d'allocation de registres pose des défis particuliers différents du domaine de la compilation, on peut par exemple décider d'utiliser des files au lieu de banques de registres, ce qui nécessite moins de matériel tout en s'adaptant au traitement itératif [BEN 96b, ALO 94]. L'optimisation de l'interconnexion des différents modules est très importante.

Afin que la SHN fasse partie intégrante du processus de conception de circuits il faudra que chacun de ces problèmes soit résolu de manière satisfaisante pour le concepteur. Etant donné la difficulté inhérente à ces problèmes, les concepteurs recherchent aussi des algorithmes polynômiaux pour calculer des estimateurs qui les aiderait dans un processus de conception semi-automatique, où plusieurs alternatives peuvent être évaluées efficacement, ainsi les paramètres suivants sont intéressants à calculer:

- Borne supérieure de la fréquence réalisable étant donné un ensemble de ressources. L'équation 5 donne déjà une première borne.
- Borne inférieure de la latence réalisable pour une fréquence cible [HU 93, BEN 96a].
- Borne inférieure sur le nombre de registres nécessaires au stockage des variables étant donné un ordonnancement.

De tels algorithmes sont très utiles pour (1) explorer rapidement la performance et le coût de plusieurs réalisations matérielles d'une même application sous différentes contraintes de ressources, et (2) pour évaluer la qualité des solutions d'ordonnancement et d'allocation obtenues par les méthodes heuristiques.

Bien que les problèmes d'ordonnancement cyclique et d'allocation des registres soient NP-complets, il est utile d'avoir des algorithmes pour les résoudre d'une façon optimale. Puisqu'en pratique, ces algorithmes prennent des temps d'exécution raisonnables pour des problèmes de petite taille et peuvent servir comme étalon pour des heuristiques.

Bibliographie

[AIK 91] A. Aiken, A. Nicolau, "A realistic Resource-Constrained Software Pipelining Algorithm", *Advances in Languages and Compilers for Parallel Processing*, MIT Press, 1991.

[ALO 94] M. Aloqeely and C. Y. Roger Chen, "Sequencer-Based Data Path Synthesis of Regular Iterative Algorithms", *In proceedings of the* 31*st Design Automation Conference*, 1994.

[BEN 96a] I.E. Bennour and E.M. Aboulhamid, "Lower-bounds on the Iteration and Initiation Interval of functional Pipelining and Loop Folding," to appear in the journal *Design Automation for Embedded Systems*, Kluwer Academic Publisher.

[BEN 96b] I. E. Bennour and E. M. Aboulhamid, "Register Allocation using Circular FIFOs", in proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1996.

[CAR 88] P. Carlier, P. Chrétienne, *Les Problèmes d'Ordonnancement: Modélisation/ Complexité Algorithmes* , Masson, Paris 1988.

[CHA 93a] L.-F. Chao, E. H.-M. Sha, "Rate-Optimal Static Scheduling for DSP Data-Flow Programs", *In proceedings of* the *Third Great Lakes Symposium on VLSI*, March 1993.

[CHA 93b] L.-F. Chao, A. LaPaugh, "Rotation Scheduling: a Loop Pipelining Algorithm", *In proceedings of the Design Automation Conference* 1993.

[CHR 91] P. Chrétienne, "The Basic Cyclic Scheduling Problem With Deadlines", *Discrete Applied Mathematics*, Vol. 30, 1991.

[CHR 83] P. Chrétienne, *Les rèsaux de Petri Temporisés*, Thèse d'état, Université Pierre et Marie Curie, 1983.

[COR 90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press and Mc Graw Hill, 1990.

[DEM 94] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[DON 92] H. V. Dongen, G. R. Gao, Q. Ning, "A Polynomial Time Method for Optimal Software Pipelining", Parallel processing, CONPAR VAPPV 92, *Lectures Notes in Computer Sciences*, Vol. 634, 1992.

[GAJ 92] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis*, Kluwer Academic Publishers, Boston, 1992.

[GAS 92] F. Gasperoni, U. Schwiegelshohn, "Scheduling Loops on Parallel Processors: A Simple Algorithm with Close to Optimum Performance", Parallel Processing, CONPAR VAPPV 92, *Lectures Notes in Computer Sciences*, Vol. 634, 1992.

[GER 92] H. Gerez, S. M. H de Groot, O. E. Herrmann, "A polynomial-time algorithm for the computation of the iterative-period bound in recursive data-flow graphs", *IEEE Trans. on Circuits and Systems*, Vol. 39, No. 1, January. 1992.

[GOO 90] G. Goossens, J. Rabaey, J. Vandewalle, H. De Man, "An efficient Microcode Compiler for Application Specific DSP Processors", *IEEE Trans. on Computer-Aided Design*, Vol. 9, No. 9, September 1990.

[GRO 92] M. H de Groot, S. H. Gerez, O. E. Herrmann, "Range-chart-guided Iterative Data Flow Graph Scheduling", *IEEE Trans. on Circuits and Systems*, Vol. 39, No. 5, May 1992.

[HAN 90] C. Hanen, "Les Tables de Réservation Numériques: Un Outil pour la Résolution de Certains Problèmes D'ordonnancement Cycliques", *Recherche Opérationnelle*, Vol. 24, No. 2, 1990.

[HAN 94] C. Hanen, "Study of NP-hard Cyclic Scheduling Problem: the Recurrent Job-Shop", *European Journal of Operation Research*, Vol. 72, 1994.

[HAN 95] C. Hanen, A. Munier "A Study of the Cyclic Scheduling Problem on Parallel Processors", *Discrete Applied Mathematics* , Vol. 57, 1995.

[HEN 92] L.Hendre, G.R. Gao, E. Altman, C. Mukerji, "A Register Allocation Framework

Based on Hierarchical Cyclic Interval Graphs", *Lectures Notes in Computer Sciences* , Vol. 641, 1992.

[HU 93] Y. Hu, A. Ghouse, B. S. Carlson, "Lower bounds on the iteration time and the number of resources for functional pipelined data flow graphs" in proceedings of ICCD, 1993.

[HWA 91] C.-T. Hwang, Y.-C. Hsu, Y.-L. Lin, "Scheduling for Functional Pipelining and Loop Winding", *In proceedings of the Design Automation Conference* 1991.

[IWA 90] K. Iwano, S. Yeh, "An Efficient Algorithm for Optimal Loop Parallelization", Algorithms, *Lectures Notes in Computer Sciences*, Vol. 450, 1990.

[JEN 94] L.-G. Jeng, L.-G. Chen, "Rate-optimal DSP Synthesis by Pipeline and Minimum Unfolding", *IEEE Trans. on VLSI*, Vol. 2, No. 1, March 1994.

[KAR 78] R.M. Karp, " A characterization of the minimum cycle in a digraph" Discrete *Mathematics* Vol. 23, 1978.

[KOG 81] P. M. Kogge, "*The architecture of Pipe-lined Computers*", McGraw Hill, New York, 1981.

[LAM 88] M. Lam, "Software Pipelining: an Effective Scheduling Technique for VLIW Machines" *In proceedings of the SIGPLAN* 88 *Conference on Prog. Language Design and Implementation*, 1988.

[LEE 94] T.-F. Lee, A. C.-H. Wu, D. D. Gajski, "A Transformation-based Method for Loop Folding", *IEEE Trans. on Computer-Aided Design*, Vol. 13, No. 4, April 1994.

[LEI 91] C. E. Leiserson, J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, 1991.

[LIU 89] D. Liu, W. Giloi, "A Loop Optimization Technique Based on Scheduling Table", *In proceedings of* the 22nd *Annual int. Workshop on Microprogramming and Microarchitecture*, 1989.

[MIC 92] P. Michel, U. Lauther, P. Dusy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, Boston, 1992.

[MUN 91] A. Munier, "Résolution d'un Problème d'Ordonnancement Cylique à Itérations Indépendantes et Contraintes de Ressources", *Recherche Opérationnelle*, Vol. 25, No. 2, 1991.

[PAR 91] K. K. Parhi, D. G. Messerschmitt, "Static Rate-optimal Scheduling of Iterative Data-flow Programs Via Optimum Unfolding", *IEEE Trans. on Computers*, Vol. 40, No.2, Feb. 1991.

[PAR 88] N. Park, A. C. Parker, "Sehwa: a Software Package for Synthesis of Pipelines from Behavioral Specifications", *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 3, March 1988.

[PAS 94] N. L. Passos, E. H. Sha, S. C. Bass, "Loop Pipelining for Scheduling Multi-Dimensional System via Rotation", *In proceedings of the 31th Design Automation Conference*, 1994.

[RAU 92] B. R. Rau, M. Lee, P. P. Tirumalai, M. S. Schlansker, "Register Allocation for Modulo Scheduled Loops: Strategies, Algorithms and Heuristics", *In Proceedings of SIGPLAN* 92 *Conference on Programming Language Design and Implementation*, 1992.

[SCH 85] D. A. Schwartz, T. P. Barnwell , “Cyclo-Static Multiprocessor Scheduling For The Optimal Realization of Shift-Invariant Flow Graphs“, *In proceedings of* the *International Conference on Accoustics, Speech and Signal Processing*, 1985.

[SCH 89] U. Schwiegelshohn, F. Gasperoni, K. Ebcioglu, “On Optimal Loop Parallelisation”, 22nd *Annual int. Workshop on Microprogramming and Microarchitecture*, 1989.

[SU 97] B. Su, S. Ding, J. Wang, J. Xia, “GURPR--A Method for Global Software Pipelining”, *In proceedings of* 20*th Annual Workshop on Microprogramming*, 1987.

[WAN 93] C.-Y. Wang, K. K. Parhi, “Loop List Scheduler for DSP Algorithms Under Resource Constraints”, *In proceedings of ISCAS* 1993.

[WOL 91] M. E. Wolf, M. S. Lam, “A Loop Transformation Theory and an Algorithm to Maximize Parallelism”, *IEEE trans. on Parallel and Distributed Systems*, Vol. 2, No. 4, October 1991.

[ZAK 89] A. Zaky, P. Sadayappan, “Optimal Static Scheduling of Sequential Loops on Multiprocessors”, *In proceedings of* the *International Conference on Parallel Processing*, Vol. III, 1989.