RAIRO. RECHERCHE OPÉRATIONNELLE

CLAIRE HANEN

Les tables de réservation numériques : un outil pour la résolution de certains problèmes d'ordonnancement cycliques

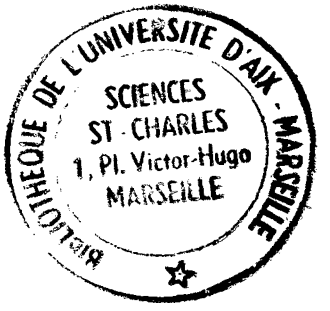
RAIRO. Recherche opérationnelle, tome 24, n° 2 (1990), p. 97-122

<http://www.numdam.org/item?id=RO_1990__24_2_97_0>

Recherche opérationnelle/Operations Research
(vol. 24, n° 2, 1990, p. 97 à 122)

LES TABLES DE RÉSERVATION NUMÉRIQUES : UN OUTIL POUR LA RÉSOLUTION DE CERTAINS PROBLÈMES D'ORDONNANCEMENT CYCLIQUES (*)

par Claire HANEN (1)



Résumé. — *Dans cet article, nous proposons une résolution exacte d'un problème d'ordonnancement cyclique avec contraintes de ressource issu d'une application informatique : l'exécution d'une boucle vectorielle récurrente par une architecture pipe-line. Le calcul d'une itération, préalablement découpée en tâches, est effectuée selon un ordonnancement générique fixé. Il s'agit alors d'enchaîner les itérations en maximisant le débit du pipe-line. Nous introduisons la notion de table de réservation numérique afin de gérer des ressources de même type de disponibilité quelconque et des dépendances entre les itérations. Nous exposons un algorithme fondé sur la recherche de circuits critiques dans un graphe d'état qui fournit le meilleur ordonnancement périodique pour une table donnée, ainsi qu'une nouvelle borne du débit. Nous en déduisons une méthode pour générer un microcode optimal prenant en compte toutes les contraintes de ressource.*

Mots clés : Ordonnancements cyliques; pipe-lines; Maximisation du débit.

Abstract. — *This paper tackles a cyclic scheduling problem with resource constraints issued from a particular application: mapping a vector recurrent loop on a pipe-lined architecture. Any iteration is processed according to the same fixed generic schedule of its elementary tasks. Iterations must be initiated so that the throughput is maximized. We define the numeric reservation table in order to manage general shared resource constraints and data dependences between the iterations, that were not taken into account in usual models. We develop an algorithm based on the search for critical cycles in a state graph, that produces the optimal periodic schedule for a given table. Moreover, a new upper bound of the throughput is stated.*

Keywords : Cyclic Scheduling; pipe-lined architectures; throughput maximization.

(*) Reçu octobre 1989.
(1) Laboratoire M.A.S.I., Université Pierre-et-Marie Curie, 4, place Jussieu, 75252 Paris Cedex 05.

Recherche opérationnelle/Operations Research, 0399-0559/90/02 97 26/$ 4.60

INTRODUCTION

Depuis quelques années, les ordinateurs scientifiques utilisent de plus en plus les pipe-lines microprogrammables. Ces architectures sont composées de modules parallèles spécialisés appelés unités fonctionnelles qui échangent des données au travers des registres. Leur contrôle est assuré au moyen d'un microprogramme qui gère l'activation de toutes les unités fonctionnelles ainsi que les chemins de données pendant chaque unité de temps. L'utilisation de tels pipe-lines pour exécuter des instructions de haut niveau pose des problèmes d'ordonnancement complexes, en raison de la diversité des ressources de la machine. Ils sont en général traités heuristiquement à l'aide d'algorithmes de liste [1, 6, 9, 15].

Le cas des calculs répétitifs (boucles vectorielles avec ou sans récurrences) mérite une attention particulière. En effet, il faut résoudre alors un problème d'ordonnancement cyclique : on considère un nombre infini d'itérations, calculées au moyen d'un même ensemble de tâches génériques et l'on maximise le débit du pipe-line, c'est-à-dire le nombre moyen d'itérations effectuées par unité de temps. La plupart des auteurs [4, 5, 10, 11, 12], [13, 14] imposent en outre un même ordonnancement des tâches génériques, appelé ordonnancement générique. Le but de cette hypothèse, que nous conservons, est de simplifier l'expression du problème et d'obtenir un microprogramme de taille raisonnable.

Ce problème a tout d'abord été traité [10, 11, 12] pour les pipe-lines classiques et les boucles vectorielles non récurrentes (à itérations indépendantes) restreignant ainsi la nature des contraintes considérées. Une table de réservation, déterminée au moyen d'heuristiques, représente l'allocation des ressources, unité par unité, pendant la durée de l'ordonnancement générique. La recherche de circuits dans un graphe d'état fournit un ordonnancement périodique optimal pour cette table. Cette approche ne permet notamment pas de représenter le partage de ressources de même type, comme les registres d'un même banc, ou des opérateurs identiques. Les solutions obtenues peuvent en conséquence être éloignées de l'optimum.

Certains auteurs ont cherché à déterminer un ordonnancement générique optimal. Les algorithmes d'insertion de délai [11, 12] fournissent une solution exacte en un temps polynômial dans un cadre très restreint : ni les registres ni les récurrences ne sont pris en compte. Ces algorithmes servent de base heuristique aux méthodes développées ces dernières années [4, 5, 14], qui calculent un ordonnancement global fondé sur un « bon » ordonnancement générique pour les boucles récurrentes. Cependant les registres sont toujours

supposés en nombre suffisant, ce qui peut amener à des solutions non réalisables.

Dans cet article, nous définissons les tables de réservation numériques qui permettent de gérer des ressources de même type (par exemple les registres) et la présence de dépendances entre les itérations. Nous supposons donc que les ressources du pipe-line sont partitionnées en plusieurs ensembles de ressources identiques, qui ne seront individuellement allouées qu'après la phase d'optimisation. Partant d'un ordonnancement générique fixé, une table de réservation numérique représente le nombre des ressources utilisées pour chaque unité de temps de cet ordonnancement. Nous exposons un algorithme fondé sur la recherche de circuits critiques dans un graphe d'état qui fournit le meilleur ordonnancement pour cette table, ainsi qu'une nouvelle borne du débit.

Dans le premier paragraphe, nous présentons le problème d'optimisation ainsi que le cadre des applications de notre étude. Le paragraphe 2 introduit la notion de table de réservation numérique. Le paragraphe suivant est consacré à la représentation des ordonnancements réalisables grâce à un graphe d'état fini. Enfin, dans le dernier paragraphe, nous résolvons le problème d'optimisation.

I. SPÉCIFICATION DU PROBLÈME D'ORDONNANCEMENT CYCLIQUE

Dans ce paragraphe, nous définissons le problème général d'optimisation auquel notre travail se rattache. Nous en présentons tout d'abord une application pratique (exécution d'une boucle vectorielle sur un pipe-line) puis nous introduisons les principaux paramètres du problème (ressources, tâches, mode d'exécution et critère d'optimisation).

I.1. Exemple

Cet exemple est destiné à présenter l'origine appliquée du problème d'ordonnancement que nous détaillons dans le paragraphe suivant. Partant d'une architecture et d'une boucle vectorielle particulières, nous spécifions les principales contraintes du problème.

Considérons un pipe-line microprogrammable constitué de quatre unités fonctionnelles (U1, U2, U3, U4) et de quatre «mémoires» (B1, B2, R1, R2). La figure 1 décrit les connexions entre ces éléments.

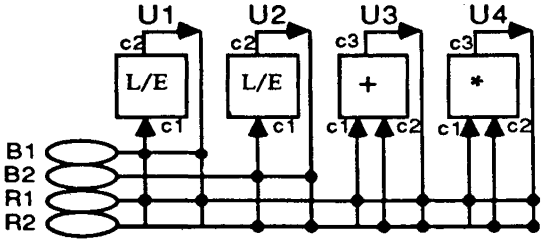


Figure 1. – Un pipe-line.

B1 et B2 sont deux bancs mémoire destinés à stocker les variables des programmes de haut niveau exécutés par le pipe-line. Deux bancs de registres R1 et R2 contenant trois registres chacun forment les ressources locales de mémoire. Nous supposerons que chaque registre peut stocker une donnée indéfiniment, et peut être lu puis écrit une fois pendant une unité de temps.

U1 est une unité de lecture écriture sur le banc B1. Pendant une unité de temps, il peut, soit lire un mot sur B1 et le stocker dans un registre de R1, soit effectuer l'opération inverse. U2 est de même nature mais fonctionne avec B2 et R2.

U3 est un additionneur; il peut, en une unité de temps, lire son premier (resp. second) opérande sur l'un des registres de R1 (resp. R2), les additionner et ranger le résultat aussi bien sur R1 que sur R2.

Nous considérons alors la boucle FORTRAN suivante que nous devons exécuter sur l'architecture ci-dessus avec un débit maximal lorsque le nombre d'itérations N tend vers l'infini.

```
DOI=1,N
B(I+2)=(A(I)+B(I)*(A (I)*B(I))+B(I)
CONTINUE
```

Si le tableau A est stocké dans le banc B1 et B dans le banc B2, chaque itération peut être décomposée en un ensemble de sept tâches élémentaires décrit ci-dessous :

Tâche 1 : *Lire A(I)* est exécutée par U1, son résultat est stocké dans R1;

Tâche 2 : *Lire B(I)* est exécutée par U2, son résultat est stocké dans R2;

Tâche 3 : *Additionner A(I)* et *B(I)* est exécutée par U3, son résultat est stocké dans R1;

Tâche 4 : *Multiplier A(I) et B(I)* est exécutée par U4, son résultat est stocké dans R2;

Tâche 5: *Multiplier les résultats des tâches* 3 *et* 4 est exécutée par U4, son résultat est stocké dans R1;

Tâche 6: *Ajouter* $B(I)$ *au résultat de la tâche* 5 est exécutée par U3, son résultat est stocké dans R2;

Tâche 7: *Écrire le résultat de la tâche* 6 *sur* $B(I+2)$ est exécutée par U2.

Les transmissions de données entre les tâches induisent naturellement des contraintes sur l'ordonnancement et nous amènent à distinguer deux types de dépendances.

Si l'on note $\langle k,p \rangle$ l'occurrence de la tâche k dans la p-ième itération, on peut noter que, pour tout p, la tâche $\langle 2,p \rangle$ doit être effectuée avant $\langle 4,p \rangle$ puisqu'il y a transmission d'un résultat, via un registre, de l'unité fonctionnelle qui exécute la tâche 2 à celle qui exécute la tâche 4. On dit alors que la dépendance entre les tâches 2 et 4 est interne. Par ailleurs, pour tout p, $\langle 2,p+2 \rangle$ lit sur le banc mémoire externe B2 la valeur de $B(p+2)$ que $\langle 7,p \rangle$ modifie. Afin de respecter la sémantique du programme, pour tout p la tâche d'écriture $\langle 7,p \rangle$ doit précéder la tâche de lecture $\langle 2,p+2 \rangle$. On dit alors que la dépendance entre les tâches 7 et 2 est externe d'ordre 2. Ces informations peuvent être résumées dans un graphe des dépendances que montre la figure 2. L'ordre de la dépendance correspond à la valuation de l'arc.

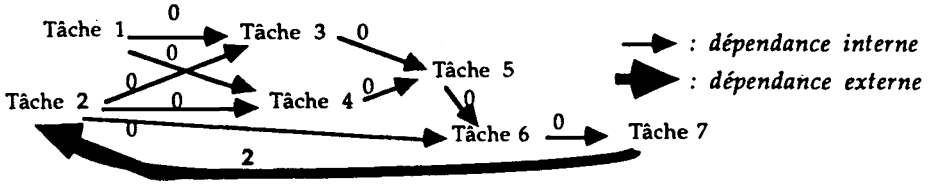


Figure 2. – Graphe des dépendances.

On cherchera à ordonnancer toutes les occurrences des tâches génériques, en respectant les contraintes suivantes:

– Précédences liées aux dépendances de données.

– A chaque instant, une unité fonctionnelle exécute au plus une tâche.

– Lorsqu'une tâche émet un résultat intermédiaire, il y a un registre disponible susceptible de le stocker. De plus, la valeur de ce registre reste inchangée tant qu'elle peut être utilisée comme opérande d'une autre tâche.

I.2. Spécification du problème

Le problème d'ordonnancement cyclique que nous étudions est spécifié par un ensemble de ressources, un ensemble de tâches génériques devant être exécutées une infinité de fois et un ensemble de contraintes de dépendance décrites sous la forme d'un graphe. Un mode d'exécution particulier des tâches est imposé: un ordonnancement générique, donné à l'avance, est périodiquement répété dans le temps.

I.2.1. *Ressources*

Nous considérons trois types de ressource: les unités fonctionnelles, les mémoires et les connexions entre ces éléments.

Les unités fonctionnelles $UF = \{U1, \ldots, Uq\}$ sont des opérateurs spécialisés séquentiels (additionneurs, multiplieurs, unité de lecture/écriture, ..., etc.) qui peuvent être actifs simultanément. Leur temps de calcul, qui mesure l'intervalle de temps entre la lecture des opérandes et l'écriture des résultats sur des ressources mémoire, est indépendant des données traitées.

On s'intéresse ici aux contraintes liées à un seul type de ressource mémoire: les ressources dites locales (registres), destinées à stocker les résultats intermédiaires émis par les unités fonctionnelles. Les mémoires externes contenant les données d'entrée et de sortie des fonctions exécutées par le pipe-line (variables de programme) sont supposées en quantité suffisante. Les ressources locales, en nombre limité, sont organisées en bancs de registres, $REG = \{R1, \ldots, Rs\}$, le banc Ri contenant r_i registres identiques. La bande passante (nombre de lectures/écritures simultanées) de ces bancs est éventuellement limitée.

Enfin, un réseau d'interconnexion relie les canaux d'entrée et de sortie des unités fonctionnelles aux bancs de registres. Des ressources de type bus peuvent être utilisées dans ce cadre, imposant un nombre limité de transferts de données simultanées entre un groupe d'unités fonctionnelles et un groupe de bancs de registres. La contrainte correspondante sera modélisée par une ressource fictive supplémentaire.

I.2.2. *Tâches*

On se donne alors un ensemble de tâches génériques, $T = \{1, \ldots, n\}$ destinées à être exécutées une infinité de fois. On note $\langle k, p \rangle$ l'occurrence de la tâche k dans la p-ième itération. Une tâche $\langle k, p \rangle$ est exécutée par une unité fonctionnelle. L'allocation des tâches aux unités fonctionnelles est générique (indépendante de l'itération) et fixée *a priori*. On a donc une

fonction d'allocation $A : T \to UF$: la tâche $\langle k,p \rangle$ est exécutée par l'unité fonctionnelle $A(k)$. De ce fait, elle a une durée d_k indépendante de p.

I.2.3. *Le graphe de dépendance*

Deux types de dépendances entre les tâches sont prises en compte, comme l'illustrait l'exemple ci-dessus:

– Les dépendances internes qui lient les tâches d'une même itération et qui nécessitent des registres;

– les dépendances externes (récurrences) qui lient les tâches d'itérations différentes, mais de manière uniforme.

Ces dépendances sont représentées à l'aide d'un graphe valué $G = (T, Di \cup De, v)$ comme suit:

– un arc $(i,j) \in Di$ valué 0 (dépendance interne) correspond à la transmission d'une donnée entre l'unité fonctionnelle qui exécute $\langle i,p \rangle$ et celle qui exécute $\langle j,p \rangle$, via un banc de registres spécifié *a priori* et indépendant de l'itération p. Cette transmission utilise donc une ressource mémoire qui doit contenir le résultat de $\langle i,p \rangle$ tant que $\langle j,p \rangle$ n'a pas été exécutée; elle est donc indispensable entre le début d'exécution de $\langle i,p \rangle$ et le début d'exécution de $\langle j,p \rangle$.

– un arc $(i,j) \in De$ valué $r \geqslant o$ (dépendance externe d'ordre r) correspond à une récurrence portant sur les variables du programme stockées en mémoire externe. Elle induit la contrainte suivante: pour toute itération p, l'exécution de $\langle i,p \rangle$ doit être terminée avant le début de la tâche $\langle j,p+r \rangle$.

La cohérence de ces contraintes est assurée par l'absence de circuits de valuation nulle.

I.2.4. *Mode d'exécution et critère d'optimisation: Ordonnancements k-périodiques statiques de débit maximal*

Le problème de la maximisation du débit compte tenu des contraintes de dépendance et de ressources ci-dessus a été étudié dans [7, 8]. Une modélisation en termes de réseaux de Petri temporisés permet la recherche de la solution optimale. En raison de sa complexité, l'algorithme proposé ne peut toutefois traiter que des problèmes de petite taille. C'est pourquoi de nombreux auteurs [1, 4, 10, 11, 12, 14] introduisent des hypothèses supplémentaires portant sur le mode d'exécution des tâches.

La première de ces hypothèses consiste à imposer une structure statique à l'ordonnancement. On limite la recherche aux ordonnancements pour lesquels chaque itération s'effectue selon le même ordonnancement générique des tâches.

Notons $t(i,p)$ la date d'exécution de la tâche $\langle i,p \rangle$. Pour tout entier p, pour tout couple de tâches génériques (i,j),

$$t(i,p) - t(j,p) = t(i,p+1) - t(j,p+1).$$

Un ordonnancement est alors entièrement décrit d'une part à l'aide d'un ordonnancement des tâches génériques $g : T \to \mathbb{N}$, dit ordonnancement générique, et d'autre part au moyen d'une suite croissante infinie $S = \{ t_0 = 0, t_1, \ldots, t_p, \ldots \}$ de dates de lancement des itérations : Pour toute tâche i, et tout entier p, $t(i,p) = t_p + g(i)$.

Remarquons que, dès que les dépendances internes sont satisfaites au niveau de l'ordonnancement générique, elles le seront aussi dans tout ordonnancement statique.

La seconde hypothèse concerne la structure de la suite des dates de lancements. Pour des raisons de facilité d'implémentation, il est en effet nécessaire d'imposer une certaine périodicité à la suite S. On a recours à la notion de K-périodicité qui généralise la notion de suite arithmétique en permettant une plus grande souplesse de l'ordonnancement tout en préservant l'existence mathématique du débit. Nous montrerons dans le paragraphe IV que les solutions K-périodiques sont d'une certaine manière dominantes pour le critère d'optimisation.

DÉFINITION : La suite $\{ t_p \}$ est *K-périodique de période P* si et seulement si, à partir d'un certain rang p_0, $t_{p+K} = t_p + P$ dès que $p \geqslant p_0$. K est appelé *facteur de périodicité* de la suite. Une suite arithmétique est 1-périodique.

Le problème a été abordé selon deux directions principales. Dans la première, ([4, 11, 12, 14]) on recherche simultanément un ordonnancement générique et une suite *arithmétique* de dates de lancements. Lorsque le problème est peu contraint (pas de dépendances externes, ni de ressources mémoire), un algorithme polynômial exact a été déterminé. Dans les autres cas, seules des heuristiques ont été proposées.

La seconde direction, à laquelle se rattache cet article, consiste à rechercher, à partir d'un ordonnancement générique fixé (obtenu par exemple au moyen d'une heuristique), une suite K-périodique de dates de lancements (K non fixé). Jusqu'à présent, seules des contraintes extrêmement limitées (pas de dépendances externes ni de ressources de disponibilité initiale supérieure à 1) avaient été prises en compte [1, 10].

Dans la suite, nous supposerons donc donné un ordonnancement générique $g : T \to \mathbb{N}$. Le débit du pipe-line, que l'on cherche à maximiser est le nombre moyen d'itérations calculées par unité de temps. Toutefois, nous préférons

formuler le problème d'optimisation comme la *minimisation* de *la moyenne asymptotique*, qui est l'inverse du débit:

Déterminer une séquence croissante K-périodique $S=\{t_0=0, t_1, \ldots, t_n, \ldots\}$ *de dates de lancement des itérations, qui respecte les contraintes de dépendance et de ressource, et qui minimise la moyenne asymptotique*:

$$L(S)=\lim_{p\to\infty}(t_p/p).$$

Remarque: Dans le cadre d'une suite K-périodique de période P, la moyenne asymptotique n'est autre que le rapport P/K.

II. TABLES DE RÉSERVATION NUMÉRIQUES

Dans ce paragraphe, nous montrons sur un exemple comment les contraintes du problème, y compris les dépendances, s'expriment en termes de contraintes de ressource. Nous présentons ensuite la notion de table de réservation numérique associée à un ordonnancement générique ainsi que quelques notations utiles pour la résolution du problème d'ordonnancement.

II.1. Exemple

Les contraintes du problème d'ordonnancement peuvent être formalisées de la manière suivante: chaque tâche générique consomme et restitue des quantités entières de ressources de différentes classes. Le schéma d'utilisation des ressources est alors reproduit, tout comme l'ordonnancement générique, à chaque itération. La définition des classes de ressources se fait suivant le type de contrainte. Afin d'alléger la formalisation nous utilisons comme support l'exemple précédent. Toutefois, une description plus rigoureuse peut être trouvée dans [7].

Considérons donc à nouveau la boucle et l'architecture de l'exemple précédent. Soit g l'ordonnancement générique défini par:

$$g(1)=g(2)=0, \qquad g(3)=1, \qquad g(4)=2,$$
$$g(5)=3, \qquad g(6)=4, \qquad g(7)=5.$$

Il est clair que toute unité fonctionnelle peut être considérée comme une classe de ressource de disponibilité initiale 1. Chaque tâche exécutée sur U_i nécessite une unité de la classe associée pendant la durée de son exécution.

A chaque banc de registres nous associons une classe de ressource de disponibilité initiale 3. A chaque itération, l'occurrence de la tâche 1 écrit son résultat sur l'un des registres de R1. Un tel registre doit donc être disponible à cet instant. Ce registre restera indisponible jusqu'à ce que les tâches 3 et 4 de la même itération l'aient lu, c'est-à-dire deux unités de temps plus tard. Par conséquent, la tâche 1 mobilise une unité de R1 pendant les deux premières unités de temps d'une itération. Le même raisonnement conduit à la description suivante:

La tâche 2 mobilise une unité de R2 pendant les quatre premières unités de temps;

La tâche 3 mobilise une unité de R1 pendant les unités de temps un et deux;

La tâche 4 mobilise une unité de R2 pendant l'unité de temps deux;

La tâche 5 mobilise une unité de R1 pendant l'unité de temps trois;

La tâche 6 mobilise une unité de R2 pendant l'unité de temps quatre.

Enfin, la dépendance externe d'ordre 2 entre la tâche 7 et la tâche 2 se modélise au moyen d'une classe de ressource fictive, appelée REC, de disponibilité initiale 2. Une unité de cette classe est mobilisée du début de la tâche 2 à la fin de la tâche 7 c'est-à-dire ici pendant toute la durée de l'ordonnancement générique. Ainsi la tâche $\langle 7, p \rangle$ précède toujours la tâche $\langle 2, p+2 \rangle$.

La table de réservation numérique décrit l'utilisation de ces ressources par l'ordonnancement générique comme le montre la figure 3:

	0	*1*	*2*	*3*	*4*	*5*	*Disponibilité initiale*
U1	T1	0	0	0	0	0	1
U2	T2	0	0	0	0	T7	1
U3	0	T3	0	0	T6	0	1
U4	0	0	T4	T5	0	0	1
R1	1	2	1	1	0	0	3
R2	1	1	2	1	1	0	3
REC	1	1	1	1	1	1	2

Pour plus de lisibilité, le nom des tâches est représenté à la place de l'entier 1 dans les lignes correspondant aux unités fonctionnelles

Figure 3. – Table de réservation numérique A.

II.2. Table de réservation numérique et séquences réalisables: définitions

Dans ce paragraphe, nous supposons que les contraintes du problème d'ordonnancement ont été formalisées comme des contraintes de ressource génériques portant sur r ensembles de ressources, appelés $\mathrm{RCL}_1, \ldots, \mathrm{RCL}_r$, de disponibilités initiales $M_1, \ldots, M_r$.

DÉFINITION : Une table de réservation numérique T (abv. T.R.N.) est une matrice $r \times D$ d'entiers. Ses lignes sont indexées par les r classes de ressource, tandis que ses colonnes sont indexées par les unités de temps. $T(i,j)$ est la quantité de ressource de RCL_i utilisée pendant la j-ième unité de temps par l'ordonnancement générique.

On doit donc avoir: $T(i,j) \leqslant M_i$, $1 \leqslant i \leqslant r$; $0 \leqslant j \leqslant D-1$.

Supposons donnée une table de réservation numérique T. Soit $S = \{ t_0 = 0, t_1, \ldots, t_n, \ldots \}$ une suite croissante d'entiers. La suite S provoque un conflit au temps t pour la classe de ressource RCL_i si plus de M_i unités de RCL_i sont mobilisées par les itérations actives à la date t. Nous formalisons maintenant ces notions élémentaires.

DÉFINITIONS: Soit $S = \{ t_0 = 0, t_1, \ldots, t_n, \ldots \}$ une suite croissante, $t \in \mathbb{N}$ une date et T une table de réservation numérique de durée D.

Nous définissons tout d'abord l'ensemble des *iétrations actives à la date t*:

$$\mathrm{IACT}(S, t) = \{ p / t_p \in S, t_p \leqslant t < t_p + D \}.$$

Soit RCL_i une classe de ressource. RCL_i est dite *surchargée à la date t* si:

$$\sum_{p \in \mathrm{IACT}(S, t)} T(i, t - t_p) > M_i.$$

S provoque un conflit si et seulement s'il existe t et i tels que RCL_i soit surchargée à la date t. S est réalisable si et seulement si elle ne provoque pas de conflit.

Remarques: 1. Dans les problèmes réels, il existe toujours une classe de ressource de disponibilité initiale 1 (unité fonctionnelle). Dans ce cas deux itérations ne peuvent être lancées à la même date. Nous considérons donc uniquement les séquences strictement croissantes.

2. Comme l'on minimise la moyenne asymptotique l'on peut se contenter d'étudier l'ensemble dominant des séquences pour lesquelles deux lancements successifs sont espacés d'au plus D unités de temps.

En effet si, pour un certain p, $l_p = t_{p+1} - t_p > D$ et si S est réalisable alors la séquence $S' = \{t'_0, \ldots, t'_n, \ldots\}$ définie par: $t'_k = t_k$ if $k \leqslant p$, sinon $t'_k = t_k - (l_p - D)$ l'est aussi. Comme $t'_k \leqslant t_k$, S' est meilleure que S.

III. REPRÉSENTATION DES SÉQUENCES RÉALISABLES

Comme pour les tables de réservation, notre approche consiste à construire un graphe d'état fini à partir de la T.R.N. dont les sommets correspondent aux quantités de ressource mobilisées par les itérations actives à un instant donné. Ses chemins sont associés aux séquences réalisables. Nous définissons tout d'abord les états instantanés d'une séquence. Nous utilisons cette notion pour déterminer une condition nécessaire et suffisante de faisabilité d'une séquence. Puis nous définissons le graphe d'état, et enfin nous utilisons les vecteurs de collision [10] pour le simplifier.

III.1. États instantanés

Soit $S = \{t_0 = 0, t_1, \ldots, t_n, \ldots\}$ une suite d'entiers strictement croissante, $t > 0$ une date et T une T.R.N. de durée D. Comme un vecteur de collision pour les tables classiques, l'état instantané de S au temps t résume l'influence du passé de la séquence sur les D unités de temps à venir.

DÉFINITION : Soit $S_t = \{t_p \in S / t_p \leqslant t\}$ la suite des dates de lancements inférieures à t.

L'état instantané IST_t de S *au temps* t est la matrice $r \times D$ définie par:

$$\forall s, \quad 0 \leqslant s \leqslant D-1, \quad \forall i \in \{1, \ldots, r\},$$
$$\mathrm{IST}_t(i,s) = \sum_{p \in \mathrm{IACT}(S_t,\, t+s)} T(i, t+s-t_p).$$

$\mathrm{IST}_t(i,s)$ est la quantité de ressource de la classe RCL_i mobilisée au temps $t+s$ par les itérations lancées dans l'intervalle de temps $[0,t]$. On remarque que $\mathrm{IST}_0 = T$.

Nous cherchons maintenant à exprimer les relations de récurrence entre les états instantanés d'une séquence. Pour cela, nous introduisons quelques notations.

NOTATIONS: On note τj l'opérateur sur les matrices qui décale ses éléments de j colonnes vers la gauche en complétant par des zéros à droite. Si M est

une matrice

$$n \times m, \qquad \forall s,\ 0 \leqslant s \leqslant n-1, 0 \leqslant t \leqslant m-j-1, \qquad \tau j(M)(s,t) = M(s, t+j);$$
$$\forall s,\ 0 \leqslant s \leqslant n-1, m-j \leqslant t \leqslant m-1, \qquad \tau j(M)(s,t) = 0.$$

Pour une séquence $S = \{t_p\}$ on note 1_p la p-ième *latence* de S: $1_p = t_{p+1} - t_p$.

LEMME 1 : *Soit* $S = \{t_p\}$ *une suite strictement croissante. Les états instantanés* IST_t *de S vérifient la relation de récurrence suivante*:

$\forall t$, si $\exists p : t_p = t+1$ alors $\mathrm{IST}_{t+1} = \tau_1(\mathrm{IST}_t) + T$, sinon $\mathrm{IST}_{t+1} = \tau_1(\mathrm{IST}_t)$.

Démonstration: D'après la définition de IST_i on a: $\mathrm{IST}_{t+1}(i,s) = \sum_{p \in \mathrm{IACT}(S_{t+1},\, t+1+s)} T(i, t+1+s-t_p)$ si $s < D-1$, alors

$$\mathrm{IST}_{t+1}(i,s) = \sum_{p \in \mathrm{IACT}(S_t,\, t+1+s)} T(i, t+1+s-t_p) + \sum_{p/t_p = t+1} T(i,s)$$

et

$$\mathrm{IST}_{t+1}(i, D-1) = \sum_{p/t_p = t+1.} T(i, D-1).$$

Mais si $s < D-1$,

$$\sum_{p \in \mathrm{IACT}(S_t,\, t+1+s)} T(i, t+1+s-t_p) = \mathrm{IST}_t(i, s+1)$$

d'où le résultat.

Q.E.D.

Ce lemme induit une relation de récurrence entre les états $\mathrm{IST}_{t_{p+1}}$ et IST_{t_p} que nous exprimons dans le lemme suivant.

LEMME 2 : *Si* $S = \{t_p\}$ *est une suite strictement croissante, la suite d'états* IST_{t_p}, $p \geqslant 0$, *satisfait la relation de récurrence*:

$$\forall p: \quad \mathrm{IST}_{t_{p+1}} = \tau_{1_p}(\mathrm{IST}_{t_p}) + T.$$

Démonstration: D'après le lemme 1, si $t \in [t_p, t_{p+1}[$, alors $\mathrm{IST}_t = \tau_{t-t_p}(\mathrm{IST}_{t_p})$. Comme une itération est lancée à la date t_{p+1}, on a:

$$\mathrm{IST}_{t_{p+1}} = \tau_1(\mathrm{IST}_{t_{p+1}-1}) + \mathrm{T} = \tau_1(\tau_{(t_{p+1}-t_p-1)}(\mathrm{IST}_{t_p})) + \mathrm{T}.$$

Par conséquent, $\mathrm{IST}_{t_{p+1}} = \tau_{(1_p)}(\mathrm{IST}_{t_p}) + \mathrm{T}$

Q.E.D.

III.2. Séquences réalisables

La notion d'état instantané peut naturellement exprimer l'absence de conflits :

LEMME 3 : *Une suite croissante S est réalisable ssi* :
$\forall i \in \{1, \ldots, r\}$, $\forall t \in \mathbb{N}$, $\mathrm{IST}_t(i, 0) \leqq M_i$.

Démonstration : Par définition,

$$\mathrm{IST}_t(i,0) = \sum_{p \in \mathrm{IACT}(S_t, t)} T(i, t - t_p) \qquad \text{où} \quad S_t = \{ t_p \in S / t_p \leqq t \}.$$

Mais $\mathrm{IACT}(S_t, t) = \mathrm{IACT}(S, t)$.

Donc

$$\mathrm{IST}_t(i,0) > M_i \Leftrightarrow \sum_{p \in \mathrm{IACT}(S,t)} T(i, t - t_p) > M_i$$

$\Leftrightarrow$ RCL_i est surchargée au temps t dans la séquence S.

Q.E.D.

Exemple : *La figure* 4 *montre les états instantanés d'une séquence non réalisable pour la T.R.N. A de la figure* 3. *Cette séquence induit un conflit au temps* 4.

	100000		000000		100000		000000		100000
	100001		000010		100101		001010		110101
IST0 =	010010	*IST1*=	100100	*IST2*=	011010	*IST3* =	110100	*IST4* =	111010
	001100		011000		111100		111000		111100
	121100		211000		231100		311000		231100
	112110		121100		323110		231100		423110
	111111		111110		222211		222110		332211

Sequence S={0,2,4} *Conflits: R1 etREC sont surchargées au temps 4*

Figure 4. – États instantanés pour la T.R.N. A

Comme les valeurs des états instantanés croissent uniquement aux dates de lancement, on peut établir une condition nécessaire et suffisante de faisabilité d'une séquence basée sur la suite des états $\mathrm{IST}_{t_p} p = 0, 1, \ldots$

THÉORÈME 1 : *Une séquence* $S=\{t_0=0,\ldots,t_p,\ldots\}$ *est réalisable si et seulement si pour tout entier* $p\geqq 0$ *on a*:

$$\forall s\in\{0,\ldots,D-1\},\quad \forall i\in\{1,\ldots,r\};\qquad \mathrm{IST}_{t_p}(i,s)\leqq M_i.$$

Démonstration: Soit RCL_i une classe de ressource et $s\in\{0,\ldots,D-1\}$. On observe facilement que, pour tout p, $\mathrm{IST}_{t_p}(i,s)\leqq\mathrm{IST}_{t_{p+s}}(i,0)$, car le membre gauche de l'inégalité fait intervenir les lancements effectués jusqu'à t_p, et le membre droit ceux effectués jusqu'à t_p+s. Par conséquent si S est réalisable, d'après le lemme 3, $\mathrm{IST}_{t_p}(i,s)\leqq M_i$.

Réciproquement, soit t un entier tel que $t_p\leqq t<t_{p+1}$. D'après le lemme 1, $\mathrm{IST}_t=\tau_{t-t_p}(\mathrm{IST}_{t_p})$, et donc $\mathrm{IST}_t(i,0)=\mathrm{IST}_{t_p}(i,t-t_p)$ si $t-t_p\leqq D-1$; sinon $\mathrm{IST}_t(i,0)=0$. Ainsi $\mathrm{IST}_{t_p}(i,s)\leqq M_i\Rightarrow\mathrm{IST}_t(i,0)\leqq M_i$.

Q.E.D.

III.3. Le graphe d'état

Afin de représenter les séquences réalisables, nous développons un graphe d'état dont les sommets sont des états instantanés de la table de réservation numérique et dont les arcs représentent des intervalles de temps entre deux lancements. Chaque chemin de ce graphe sera donc associé à une séquence réalisable. Nous le construisons à partir de l'état instantané initial IST_0.

DÉFINITION : *Le graphe d'état*, noté $\mathrm{ISTG}=(V,E,w)$ est un multigraphe valué défini comme suit: V est l'ensemble des sommets, E celui des arcs, $w: E\to|N$ est leur valuation. $\mathrm{IST}_0\in V$.

Si $\mathrm{ST}\in V$, alors $(\mathrm{ST},\mathrm{IST}_0)\in E$ et $w(\mathrm{ST},\mathrm{IST}_0)=D$.

Pour tout s, $0\leqq s<D$, si la T.R.N. peut être lancée sans conflits de l'état ST à la date s, alors un arc d'origine ST et de valuation s est ajouté; son extrémité ST′ est l'état obtenu s unités de temps plus tard :

Donc $(\mathrm{ST},\mathrm{ST}')\in E$ et $w(\mathrm{ST},\mathrm{ST}')=s$ si et seulement si (∗) est satisfaite :

$$\mathrm{ST}'=\tau_s(\mathrm{ST})+\mathrm{IST}_0\quad\text{et}\quad\forall i\in\{1,\ldots,r\},\qquad\forall s'<D,\quad\mathrm{ST}'(i,s')\leqq M_i\quad(*)$$

Comme chaque ligne d'un sommet de ce graphe est nécessairement bornée par $\mathrm{Max}(M_i)$, ce procédé de construction est fini, ainsi que le graphe d'état.

Exemple: *La figure* 5 *montre le graphe d'état de la table de réservation numérique* A.

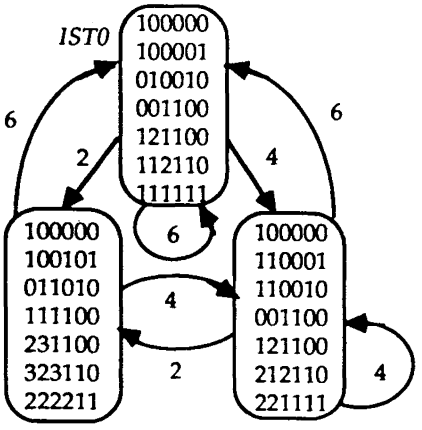


Figure 5. – Graphe d'état de A.

Le lemme suivant exprime une propriété du graphe d'état qui s'avère utile pour construire une séquence réalisable à partir d'un chemin de ISTG.

THÉORÈME 4 : *A tout chemin du graphe dont les arcs sont valués* $1_0, \ldots, 1_{n-1}$ *est associé un chemin de même valuations issu de l'état initial* IST_0.

Démonstration: Soit $\mathrm{ST}_0, \ldots, \mathrm{ST}_n$ un chemin de ISTG, dont les arcs sont valués $1_0, \ldots, 1_{n-1}$. Le procédé de construction du graphe d'état implique que, pour tout $p<n$,

$$\mathrm{ST}(p+1)=\tau_{1\,p}(\mathrm{ST}_p)+\mathrm{IST}_0.$$

Mais si $\mathrm{ST}_0 \neq \mathrm{IST}_0$, il existe un chemin de ISTG d'origine IST_0 et d'extrémité ST0. Par conséquent, il existe un sommet ST et une constante s tels que $\mathrm{ST0}=\tau_s(\mathrm{ST})+\mathrm{IST}_0$. On en déduit que $\mathrm{STO} \geqslant \mathrm{IST}_0$. [C'est-à-dire que pour tout couple (i,s), $\mathrm{STO}(i,s) \geqslant \mathrm{IST}_0(i,s)$.]

Si l'on définit une nouvelle suite de n états par:
$\mathrm{ST'}\,0=\mathrm{IST}_0$, $\mathrm{ST'}\,(p+1)=\tau_{1\,p}(\mathrm{ST'}\,p)+\mathrm{IST}_0$.

On constate aisément que $\mathrm{ST'}\,p \leqslant \mathrm{ST}p$ pour tout p. Comme les $\mathrm{ST}p$ satisfont (*), les $\mathrm{ST'}p$ aussi. Par conséquent, $\mathrm{ST'}\,0, \ldots, \mathrm{ST'}\,n$ est un chemin de ISTG.

Q.E.D.

Le théorème suivant établit les relations entre les chemins de ISTG et les séquences de lancement réalisables d'une table de réservation numérique.

THÉORÈME 2 : *A chaque séquence réalisable est associé un chemin du graphe d'état ISTG issu de l'état initial* IST_0.

Réciproquement, à chaque chemin de ISTG est associé une séquence de lancements réalisable.

Démonstration: Soit $t_0=0, t_1, \ldots, t_n$ une séquence sans conflits, et notons $l_p=t_{p+1}-t_p$. Cette séquence est associée au chemin dont les sommets sont $IST_0, IST_{t1}, \ldots, IST_{tn}$, et dont les arcs sont valués $l_0, l_1, \ldots, l_{n-1}$.

Réciproquement, soit (STO, ..., STn) un chemin de ISTG, dont les arcs sont valués $l_0, l_1, \ldots, l_{n-1}$. D'après le lemme 4, on peut considérer que $ST0=IST_0$; d'où, en posant $t_0=0$, et $t_p=\sum_{0\leq j\leq p-1} l_j$, la faisabilité de $S=\{t_0, \ldots, t_n\}$ découle du fait que les sommets du chemin sont les états instantanés de la séquence.

Q.E.D.

III.4. Simplifications

Les résultats précédents ne tiennent pas compte du fait que certaines informations des états peuvent être résumées dans un seul vecteur booléen. En effet, les lignes de la table de réservation numérique correspondant aux ressources de disponibilité initiale 1 peuvent être considérées exactement comme des tables classiques [10]. Par conséquent, la notion de vecteur de collision peut être utilisée.

Si T est une table de réservation numérique, considérons l'ensemble B des classes de ressources RCL_i de disponibilité initiale 1. Notons T_B la sous-matrice de T constituée des lignes correspondantes. Rappelons brièvement la définition des vecteurs de collision :

DÉFINITION [10] : Soit $S=\{t_p\}$ une séquence de lancements pour la table T_B. Le vecteur de collision du temps t associé à S est un vecteur *booléen* CV_t de longueur D. Pour $s\in\{0, \ldots, D-1\}$, $CV_t(s)=1$ si et seulement si un lancement de T_B à la date $t+s$ provoquerait un conflit avec les lancements de la sous-séquence S_t. Le vecteur initial de collision CV_0 peut être aisément calculé à partir de la table de réservation: $CV_0(s)=1$ si et seulement si sur l'une des lignes deux « 1 » sont espacés de s unités de temps.

Grâce aux propriétés des vecteurs de collision [10], on peut alors considérer les états simplifiés pour la table de réservation numérique T:

DÉFINITION : L'état instantané simplifié de S au temps t SIS_t est une matrice $(r+1-|B|)\times D$.

(1) Sa première ligne (indexée par 0) est le vecteur de collision à la date t de T_B.

(2) Les autres lignes correspondent à la sous-matrice de IST_t de lignes indexées par B^c.

Le lemme 3 et le théorème 1 se prolongent en une relation de récurrence sur les états simplifiés et donc en une condition nécessaire et suffisante de faisabilité. Le théorème 1 *bis* résume ces propriétés, + désignant l'opérateur booléen OU sur la première ligne, et l'addition naturelle sur les autres.

THÉORÈME 1 *bis* : *Soit* $S=\{t_0=0, t_1, \ldots, t_n, \ldots\}$ *une suite croissante*

(*a*) $\forall p$, $SIS_{t_p+1}=\tau_{1\,p}(SIS_{t_p})+SIS_0$;

(*b*) *S est réalisable si et seulement si*:

- $\forall p \geqq 0$, $SIS_{tp}(0, 1_p)=0$.
- $\forall p \geqq 0$, $\forall s \in \{0, \ldots, D-1\}$, $\forall i \notin B$, $SIS_{tp}(i, s) \leqq M_i$.

Exemple : La figure 6 *montre les états simplifiés pour la table numérique A.*

Sequence $S=\{0, 2, 6, 8, 12, \ldots\}$

$$SIS0=\begin{matrix}110101\\121100\\112110\\111111\end{matrix} \quad SIS2=\begin{matrix}110101\\231100\\323110\\222211\end{matrix} \quad SIS6=\begin{matrix}110101\\121100\\212110\\221111\end{matrix} \quad SIS8=SIS2, \ldots$$

Figure 6. – États simplifiés.

Ce théorème nous permet de simplifier la construction du graphe d'état dont les états simplifiés seront les sommets: en remplaçant IST_0 par SIS_0 et la condition (*) par la suivante:

$$(ST, ST') \in E \quad \text{et} \quad w(ST, ST')=s \Leftrightarrow (**)\ ST'=\tau_s(ST)+SIS_0;$$

$$ST(0, s)=0; \quad \text{et} \quad \forall i \in \{1, \ldots, r\}, \quad \forall s' < D, \quad ST'(i, s') \leqq M_i.$$

Appelons STG ce graphe simplifié. On peut alors formuler les mêmes propriétés que pour ISTG, et en particulier le théorème suivant:

THÉORÈME 2 *bis* : *A chaque séquence de lancement réalisable est associé un chemin de* STG *issu du sommet initial* SIS_0. *Réciproquement, à tout chemin de* STG *est associé une séquence réalisable.*

Exemple: *La figure* 7 *montre le graphe d'état simplifié pour la T.R.N. A.*

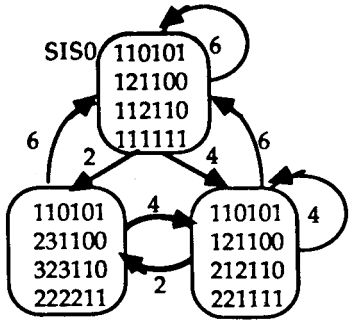


Figure 7. – Graphe d'état simplifié de A.

Remarque: On peut parfois ajouter de l'information (« 1 » logiques) à la première ligne de SIS_0. En effet, si dans la table de réservation numérique il existe une ligne i avec $M_i > 1$, s et s' tels que $T(i,s)+T(i,s') > M_i$, alors toute séquence contenant une latence $1_p = s'-s$ produira un conflit. On peut donc poser $SIS_0(1, s'-s) = 1$.

IV. ORDONNANCEMENT OPTIMAL

Le problème d'optimisation peut maintenant s'exprimer en termes de chemins du graphe d'état. Nous montrons que la séquence optimale K-périodique reste un cycle de lancement en présence de ces contraintes de ressources partagées et qu'un tel cycle est associé à un circuit critique [2, 3] du graphe d'état. Nous montrons que l'hypothèse de la K-périodicité de la séquence de lancements n'est pas contraignante dans la mesure où de telles solutions sont dominantes pour le critère. Nous donnons ensuite une nouvelle borne de la moyenne asymptotique d'une table de réservation donnée.

IV.1. Solution du problème d'optimisation

Introduisons tout d'abord quelques notations.

DÉFINITIONS: Soit $\mu = \{\mathrm{STO}, \ldots, \mathrm{ST}n\}$ un chemin de STG, dont les arcs sont valués $1_0, \ldots, 1_{n-1}$.

La *hauteur de* μ, notée $h(\mu)$, est le nombre de ses arcs $h(\mu) = n$.

La *longueur de* μ, notée $l(\mu)$, est la somme des valuations de ses arcs :

$$l(\mu) = \sum_{0 \leq p \leq n-1} l_p.$$

La *moyenne de* μ, notée $L(\mu)$, est le quotient $l(\mu)/h(\mu)$.

Un *circuit* μ de STG est dit *critique* si sa moyenne est minimale :

$$\forall\, \nu \text{ circuit de STG}, \quad L(\mu) \leqslant L(\nu).$$

Si ν est un circuit de STG, de valuations $l_0, \ldots, l_{k-1}$, le k-uplet $C = (l_0, \ldots, l_{k-1})$ est appelé *cycle de lancement. $P = \Sigma l_p$ est sa période et k sa hauteur.*

A un tel cycle est associée la séquence de lancement infinie k-périodique de période P suivante :

$$t_n = sP + \sum_{0 \leq r \leq q} l_r \qquad \text{si} \quad n = s, k+q, \qquad 0 \leqslant q < k.$$

D'après le théorème 2, nous pouvons formuler le problème d'optimisation comme suit :

Déterminer un chemin infini $\mu = \{\mathrm{ST}0, \ldots, \mathrm{ST}n, \ldots\}$ *du graphe* STG *dont la séquence de lancements associée est K-périodique et tel que si* $\mu_n = \{\mathrm{ST}0, \ldots, \mathrm{ST}n\}$, $\lim_{n \to \infty} L(\mu_n)$ *existe et est minimale.*

Étudions tout d'abord la nature des chemins associés aux séquences K-périodiques réalisables.

LEMME 5 : *Si* $S = \{t_0 = 0, \ldots, t_p, \ldots\}$ *est une séquence K-périodique réalisable, alors il existe un chemin fini* α *et un circuit* ν *du graphe* STG *tels que S soit la séquence associée au chemin* $\mu = \alpha . \nu^{\infty}$,

De plus, le cycle de lancement associé à ν *à la même moyenne asymptotique que la séquence S.*

Démonstration: Par hypothèse S est K-périodique de période P à partir d'un certain rang p_0 : pour tout $p \geqslant p_0$, $t_{p+K} = t_p + P$.

Soit $\alpha 1$ le chemin associé à la sous-séquence $S' = \{t_p, p < p_0\}$ et soit ST le dernier sommet de ce chemin. Notons $l_p = t_{p_0+p} - t_{p_0-1}$, $p < K$. Puisque S est réalisable, d'après le théorème 2, il existe un chemin infini $\beta = \{\mathrm{ST} = \mathrm{ST}0, \ldots, \mathrm{ST}n, \ldots\}$ dont les arcs sont valués : $l_0, \ldots, l_{K-1}, l_0, \ldots, l_{K-1}, \ldots$

Le graphe étant fini, la suite des sommets $\{\mathrm{ST}n / n = p.K, p \geqslant 0\}$ possède au moins deux éléments égaux STi et STj. Appelons $\alpha 2$ le sous-chemin de β

constitué des sommets $\{\mathrm{ST}0, \ldots, \mathrm{ST}i\}$, et posons $\nu = \{\mathrm{ST}i, \ldots, \mathrm{ST}j\}$. ν est donc un circuit; de plus, le chemin $\alpha 2 . \nu^{\infty}$ a les mêmes valuations que le chemin β.

En posant $\alpha = \alpha 1 . \alpha 2$, on en déduit immédiatement que le chemin $\mu = \alpha . \nu^{\infty}$ est bien associé à la séquence S.

Enfin, si

$$n - h(\alpha) = q . h(\nu) + r, \qquad r < h(\nu),$$

alors

$$L(\mu_n) = (l(\alpha) + q . l(\nu) + c_1(r))/(h(\alpha) + q . h(\nu) + c_2(r))$$

avec $c_1(r) < l(\nu)$; $c_2(r) < h(\nu)$. Lorsque $n \to \infty$, $q \to \infty$ et donc $L(\mu) = \lim_{n \to \infty} L(\mu_n) = L(\nu)$.

Q.E.D.

Ce lemme a pour conséquence directe l'optimalité des cycles de lancements associés aux circuits critiques:

THÉORÈME 3 : *Si ν est un circuit critique du graphe d'état* STG, *valué $(l_0, \ldots, l_{k-1})$, alors le cycle de lancement $C = (l_0, \ldots, l_{k-1})$ est optimal.*

L'hypothèse de la K-périodicité des ordonnancements peut être justifiée en montrant que ceux-ci sont dominants pour la minimisation de la moyenne asymptotique. En effet, si $S = \{t_0 = 0, \ldots, t_p, \ldots\}$ est une séquence croissante non nécessairement K-périodique, même si la suite $m_p = t_p/p$ n'a pas de limite, on peut mesurer sa performance grâce à son plus petit point d'accumulation. La proposition suivante établit dans ce cadre la dominance des solutions K-périodiques.

PROPOSITION : *Si $S = \{t_0 = 0, \ldots, t_p, \ldots\}$ est une séquence infinie réalisable, et soit m le plus petit point d'accumulation de la suite t_p/p.*

Pour tout $\varepsilon > 0$ il existe une séquence K-péridoqiue réalisable de moyenne au plus égale à $m + \varepsilon$.

Démonstration: Soit $\mu = \{\mathrm{ST}0, \ldots, \mathrm{ST}n, \ldots\}$ le chemin du graphe d'état associé à S, et soit $S' = \{t_{pi}\}$ une suite extraite de S telle que t_{pi}/p_i converge vers m. Il lui correspond une suite de sommets du chemin $\mu : \mathrm{ST}0 = \mathrm{ST}_{p0}, \ldots, \mathrm{ST}p_i, \ldots$

Étant donné un $\varepsilon > 0$, on peut trouver un rang j à partir duquel

$$(t_{pi} + D)/(1 + p_i) < m + \varepsilon \quad \text{dès que } i \geqq j.$$

Soit alors v le circuit formé du sous-chemin de μ $ST p_0, \ldots, ST p_j$ et de l'arc retour valué D de $ST p_j$ au sommet ST0. Nous avons $l(v) = t_p + D$ et $h(v) = 1 + p_j$.

Par conséquent, le cycle de lancement associé à v est bien de moyenne au plus égale à $m + \varepsilon$, et sa séquence associée est par construction K-périodique.

Q.E.D.

Exemple: *Le cycle optimal de notre exemple est* $C = (2, 4)$. *Le circuit critique correspondant est représenté en gras dans la figure* 7. *Il est à noter que ce circuit n'est pas constant, et que la recherche habituellement proposée d'un ordonnancement* 1*-péridoqiue* [4, 14] *peut être améliorée par l'utilisation de ces techniques. En effet pour la table A le meilleur cycle constant est* $C' = (4)$.

IV.2 Une nouvelle borne inférieure

Nous déduisons de notre approche une nouvelle borne inférieure de la moyenne asymptotique, basée sur l'utilisation moyenne des classes de ressources. Soit donc $S = \{ t_0, \ldots, t_n, \ldots \}$ une séquence de lancements réalisable d'une table de réservation numérique T, et soit RCL_i une classe de ressource. Dans toute séquence réalisable, l'utilisation moyenne de RCL_i par unité de temps ne peut excéder 1. Le calcul de cette utilisation en fonction de la moyenne asymptotique de l'ordonnancement permet d'établir le théorème 4:

THÉORÈME 4 : *Soit T une table de réservation numérique de durée D, et C un cycle de lancement optimal de T. La moyenne asymptotique L(C) est bornée inférieurement par l'utilisation totale de chaque ressource dans une itération*:

$$L(C) \geqslant \operatorname{Max}_i \Big(\sum_{0 \leqslant s < D} T(i,s)/M_i \Big).$$

Démonstration: Chaque itération utilise $T(i,s)$ unités de RCL_i pendant la s-ième unité de temps de son exécution.

Au temps t, la séquence S utilise: $\sum_{p \in IACT(S,t)} T(i, t - t_p)$ unités de RCL_i.

Le pourcentage instantané est donc $(1/M_i) * \sum_{p \in IACT(S,t)} T(i, t - t_p)$.

Maintenant, si D_n est la durée de la séquence S_{t_n}, nous obtenons en moyenne jusqu'à l'itération n:

$$[(1/Dn) * \sum_{0 \leqslant t < Dn} \sum_{p \in IACT(S,t)} (T(i, t - t_p)/M_i)] \leqslant 1$$

En changeant l'ordre de sommation, on obtient:

$$[(1/D_n) * \sum_{0 \leqq p \leqq n} \sum_{t_p \leqq t < t_p + D} (T(i, t - t_p)/M_i)] \leqq 1.$$

Or $\sum_{t_p \leqq t < t_p + D} T(i, t - t_p)$ est exactement la somme des éléments de la ligne i de T, et donc ne dépend pas de p. En prenant la limite lorsque n tend vers l'infini, il vient:

$$\lim_{n \to \infty} [(n/D_n) * \sum_{0 \leqq s < D} (T(i, s)/M_i)] = (1/L(S)) * \sum_{0 \leqq s < D} (T(i, s)/M_i) \leqq 1.$$

Q.E.D.

Exemple: *Considérons un autre ordonnancement des tâches génériques pour notre exemple, et pour simplifier négligeons la contrainte de dépendance externe entre T2 et T7. On obtient alors la table de réservation numérique B de la figure* 8.

	0	1	2	3	4	5	6	7	Mi
U1	T1								1
U2	T2							T7	1
U3		T3					T6		1
U4			T4	T5					1
R1	1	2	1	1	1	1	0	0	3
R2	1	1	2	1	1	1	1	0	3

Figure 8. – Table numérique B.

Si les registres sont ignorés, ce qui est fréquent avec les techniques habituelles, le cycle de lancement constant $C=(2)$ *est optimal.*

Or, notre borne inférieure est $\text{Max}(2, 7/3, 8/3)$. *Comme* $8/3 > 2$, *on voit immédiatement que ce cycle n'est pas réalisable. La figure* 10 *montre le graphe d'état associé à B.* $C=(3)$ *en est le cycle optimal.*

CONCLUSION

La microprogrammation de boucles vectorielles récurrentes pose d'importants problèmes d'ordonnancement répétitifs. Ces problèmes sont caractérisés par un nombre fini de tâches devant être effectuées une infinité de fois.

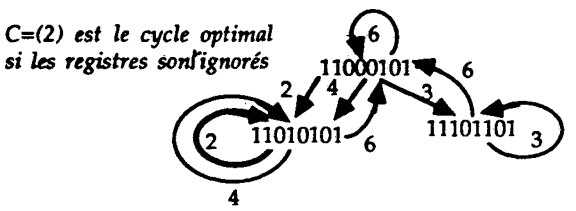


Figure 9. – Graphe d'état sans registres.

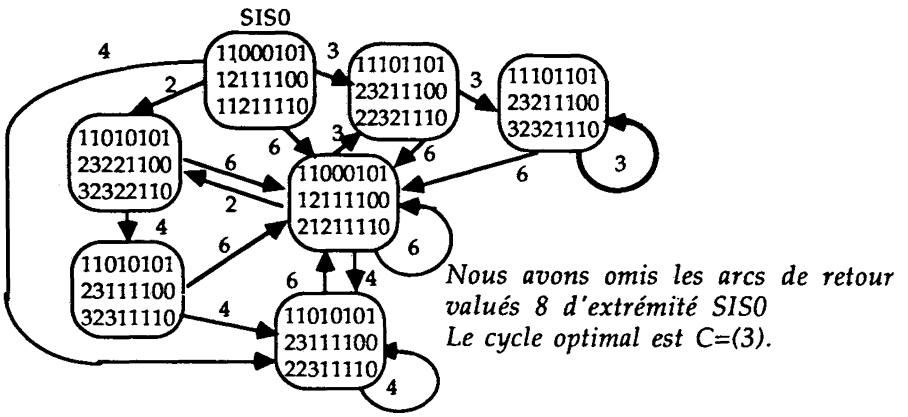


Figure 10. – Graphe d'état de B.

Le critère d'optimisation n'est plus la minimisation d'une durée mais la maximisation de la fréquence d'exécution des tâches, appelée débit.

Le problème de microprogrammation est habituellement traité au moyen d'heuristiques, fournissant en particulier un ordonnancement générique fixe des tâches d'une même itération. Les outils permettant d'optimiser le débit, principalement basés sur la notion de table de réservation, ne pouvaient jusqu'à présent pas prendre en compte des contraintes de ressources partagées suffisamment générales. Il en résultait des solutions soit non implémentables, soit éloignées de l'optimum, en raison de l'allocation statique des ressources, unité par unité.

Nous avons proposé une généralisation de la notion de table de réservation afin de remédier aux défauts du précédent modèle. Nous avons montré que les algorithmes de résolution pour les tables de réservation se généralisent

également et permettent d'améliorer la qualité des solutions. Enfin, une borne supérieure du débit a été établie.

Ce travail devrait donc permettre une amélioration des techniques d'optimisation utilisées dans les compilateurs de machines comme les calculateurs vectoriels ou les nouveaux super-ordinateurs. 1 doit cependant être poursuivi afin de proposer des méthodes pour choisir efficacement l'ordonnancement générique.

L'un des prolongements utiles de cet article serait d'étudier l'apport de ce type de technique à la microprogrammation des boucles comportant des sauts conditionnels. A la différence des boucles vectorielles traitées ici, les ordonnancements les concernant doivent être exprimés en termes stratégiques et non plus uniquement statiques, dans la mesure où les tâches effectuées dépendent des données.

En conclusion, les problèmes répétitifs ouvrent un champ d'étude riche en applications et pourtant très peu exploré jusqu'ici.

BIBLIOGRAPHIE

1. T. AGERAWALA, *Microprogram Optimization: a survey*, I.E.E.E. Trans. on Computers, vol. C-25, n° 10, octobre 1976.
2. J. CARLIER et P. CHRÉTIENNE, *Les problèmes d'ordonnancement : Modélisation/Complexité Algorithmes*, Masson, Paris 1988.
3. P. CHRÉTIENNE, *Chemins extrémaux d'un graphe doublement valué*, R.A.I.R.O., vol. 18, n° 3, p. 221-245, 1984.
4. C. EISENBEIS, *Optimisation automatique de programmes sur "Array-processors"*, Thèse de 3e cycle, Université de Paris-VI, 1986.
5. C. EISENBEIS, Optimization of Horizontal Microcode Generation for Loop Structures. *Proc. of the* 1988 *ACM Inté. Conf. on Super-computing*, Saint-Malo, France, juillet 1988, p. 453-465.
6. J. R. ELLIS, *Bulldog: A Computer for VLIW Architectures*, M.I.T. Press, 1986.
7. C. HANEN, *Problèmes d'ordonnancement des architectures pipe-lines : modélisation, optimisation, algorithmes.* Thèse d'université, Rapport M.A.S.I., n° 193, Univ. Paris-VI, septembre 1987.
8. C. HANEN, *Optimizing Microprograms for Recurrent loops on Pipe-lined Architectures using Timed Petri Nets.* Advances in Petri nets (à paraître) Springer-Verlag, 1989.
9. D. LANDSKOV, S. DAVIDSON, B. SHRIVER et P. W. MALLETT, *Local Microcode Compaction Techniques*, ACM Computing Surveys, vol. 12, n° 3, septembre 1980.
10. P. M. KOGGE, *The Architecture of Pipe-lined Computers*, New York, McGraw Hill, 1981.
11. J. H. PATEL et E. S. DAVIDSON, *Improving the Throughput of a Pipe-line by Insertion of delays*, I.E.E.E. 3rd annual symp. on Computer Architecture, janvier 1976.

12. B. R. RAU et C. D. GLAESER, *Some Scheduling Techniques and a Easily Schedulable Horizontal Architecture for High Performance Scientific Programming*, I.E.E.E./A.C.M. 14th annual microprogramming workshop, octobre 1981.
13. L. E. SHAR, *Design and Scheduling of Statically Configured Pipe-lines*, Stanford University, T. R. n° 42, 1972.
14. M. SING-LING LAM, *A systolic Array Optimizing Compiler*, Ph. D. Thesis, Carnegie Mellon, mai 1987.
15. M. TOKORO, E. TAMURA et T. TAZIZUKA, *Optimization of Microprograms*, I.E.E.E. Trans. on Computers, vol. C-30, n° 7, juillet 1981.