Ordonnancements périodiques dans les réseaux de processus : Application à la conception insensible aux latences

Jean-Vivien Millo

▶ **To cite this version:**

Jean-Vivien Millo. Ordonnancements périodiques dans les réseaux de processus : Application à la conception insensible aux latences. Réseaux et télécommunications [cs.NI]. Université Nice Sophia Antipolis, 2008. Français. NNT : . tel-00507498

HAL Id: tel-00507498
https://theses.hal.science/tel-00507498

Submitted on 30 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Nice - Sophia Antipolis – UFR Sciences
École Doctorale STIC

THÈSE

Présentée pour obtenir le titre de :

Docteur en Sciences de l'Université de Nice - Sophia Antipolis

Spécialité : INFORMATIQUE

par

Jean-Vivien MILLO

Équipe d'accueil : AOSTE – INRIA Sophia Antipolis – I3S

ORDONNANCEMENTS PÉRIODIQUES DANS LES RÉSEAUX DE PROCESSUS: APPLICATION À LA CONCEPTION INSENSIBLE AUX LATENCES.

Thèse dirigée par Robert de SIMONE

Soutenance à l'INRIA le 15 décembre 2008, à 15h12 devant le jury composé de :

Président :	Michel	AUGUIN	Université de Nice Sophia-Antipolis/ LEAT
Directeur :	Robert	de SIMONE	INRIA Sophia-Antipolis
Rapporteurs :	Marc	POUZET	Université de Paris sud/ INRIA Saclay
	Bruno	GAUJAL	INRIA Rhône Alpes
Examinateurs :	Gaël	CLAVÉ	Texas Instruments / Villeneuve Loubet

THÈSE

ORDONNANCEMENTS PÉRIODIQUES DANS LES RÉSEAUX DE PROCESSUS: APPLICATION À LA CONCEPTION INSENSIBLE AUX LATENCES.

STATIC SCHEDULING IN PROCESS NETWORKS; APPLICATION TO LATENCY INSENSITIVE DESIGN

JEAN-VIVIEN MILLO

Décembre 2008

A Anne, Michèle et Louis.

Remerciements

Tout d'abord, merci à ma famille et à ma femme de m'avoir supporté et soutenu jusqu'à présent.

Merci à la Région PACA et à la société ST Microelectronics pour avoir financé mes recherches.

Merci à l'Université de Nice Sophia-Antipolis pour la gestion de mon dossier et pour les formations dispensées par le collège des Etudes Doctorales auxquelles j'ai participé.

Merci aux collaborateurs du Centre Intégré de Microélectronique de la région PACA et plus particulièrement aux membres du projet Sys2RTL de la plateforme Conception pour m'avoir écouté et conseillé pendant nos réunions trimestrielles.

Merci à l'INRIA Sophia-Antipolis de m'avoir accueilli et d'avoir mis à ma disposition l'ensemble de ses ressources.

Merci à notre assistante de projet, Patricia Lachaume, pour sa sollicitude.

Merci à chacun des membres du projet AOSTE pour m'avoir consacré un peu de leur temps et pour leur soutien. Merci à Julien Boucaron avec qui j'ai beaucoup travaillé pendant mon stage de DEA et ma première année de thèse. Merci à Benoit Ferrero pour m'avoir aidé sur les aspects les plus techniques de mon travail. Merci à Anthony Coadou, Aamir Mehmood Khan et à Jean-François Le Tallec pour avoir partagé avec moi des discussions animées et leur bureau. Merci à Frédéric Mallet, Marie-Agnes Peraldi Frati et à Charles André pour m'avoir éclairé sur certains thèmes de recherches connexes à mon travail.

Et surtout, merci à Robert de Simone, mon directeur de thèse, pour avoir cru en moi.

Merci à Bruno Gaujal et à Marc Pouzet d'avoir accepté d'être mes rapporteurs. Merci à tous les deux pour les informations que vous m'avez transmises et les corrections que vous m'avez suggérés. Votre regard critique et constructif m'a été utile.

Pour finir, voici le nom de quelques personnes que je tiens à remercier en nom propre: Vanessa Wallet, Gaël Clavé, Yves Leduc, Pierre Nepomiastchy, Suzanne Reynders et surtout Obi Wan Kenobi.

Résumé

Du fait de la miniaturisation grandissante des circuits électroniques, la conception de système sur puce actuelle, se heurte au problème des latences sur les fils d'interconnexions traversant tout le circuit. Un système sur puce est un ensemble de blocs de calculs (les composants IP) qui s'échangent des données. Alors que la communication à l'intérieur de ces blocs de calculs peut toujours se faire de manière synchrone, c'est à dire s'abstraire comme une action instantanée, la communication d'un bloc de calculs à un autre prend un temps qui n'est pas négligeable. Il s'écoule plusieurs cycles d'horloge entre l'émission d'une donnée sur un fil d'interconnexion et sa réception.

La théorie du *Latency Insensitive Design* créée par Luca Carloni et Alberto Sangiovanni-Vincentelli permet entre autre de résoudre ce problème en implantant un protocole de communication basé sur la segmentation des fils d'interconnexions et sur le principe de rétroaction en cas d'embouteillage.

Dans un premier temps, nous avons donné un fondement théorique à cette théorie en la rapprochant formellement d'une modélisation par *Marked/Event graph* (Sous ensemble sans conflit des Réseaux de Pétri) et avec des places de capacité 2 ; ce qui génère naturellement le protocole de contrôle de flux.

Cette modélisation nous amène à la problématique principale de cet ouvrage : comment, et sous quelles conditions, peut on minimiser la taille des ressources de mémorisation utilisées comme tampons intermédiaires au long de ces fils d'interconnexions ? Car leur nombre et leur position peuvent se révéler critique à l'implantation matérielle.

Nous allons ensuite étudier cette question sous une hypothèse naturelle de déterminisme, ce qui permet d'obtenir des régimes de fonctionnement périodiques et réguliers.

Le but de cette thèse est de modifier le protocole mis en place dans la théorie du *Latency Insensitive Design* en prenant en compte cette hypothèse. L'étude des systèmes déterministes et des résultats existant nous a permis une première phase de modification appelée : *égalisation*.

L'étape suivante consiste à ordonnancer statiquement ces systèmes. Pour cela, nous avons choisi de représenter explicitement l'ordonnancement de chacun des éléments du système comme un mot binaire périodique où les "1" représentent les instants d'activités et les "0" d'inactivités tel que M. Pouzet et al. l'ont introduit dans le "N-synchronous Kahn network".

Une étude approfondie des différentes classes de mots binaires existants (mot de Sturm, de Christoffel, de Lyndon ou encore mécaniques) a précédé leur association à la théorie du *Latency Insensitive Design* et au processus d'*égalisation* pour obtenir des systèmes déterministes ordonnancés statiquement.

Abstract

Due to the increasing scaling of digital system, System-on-Chip (SoC) design deals with latencies problem on long wire interconnection through the whole chip. A SoC is a set of IP components communicating together. While the communication inside the IP component can still be considered synchronous (abstracted as instantaneous action), the communication between IP components should not. Many clock cycles occur between sending and reception of a data on an interconnexion wire.

The theory of *Latency Insensitive Design* (LID) created by L. Carloni and A. Sangiovanni-Vincentelli solves this problem by implementing a communication protocol based on segmentation of interconnection wire and back pressure in case of local traffic jam.

In the first time, we'll give theoretical basis of LID theory by formally linking it to the deterministic model of *Marked/Event graph* (conflict free subset of Petri net), and limiting the capacity of places by 2; which naturally implement the back pressure protocol.

This model drive us to the main problem of this work: How to minimize the size of memory resources used as buffer through the interconnection wires? Because their quantity and location should become critical at implementation.

Then we'll study this problem with the natural hypothesis of determinism. This allow the system to have regular and periodic behaviour.

The goal of this work is to modify the LID theory by taking care of this hypothesis. the study of deterministic systems and previous results lead us to a first modification step called: *Equalization*.

Next step consist in statically schedule these system. We chose to explicitly represent schedule of each element of the system using periodic binary word (1 for activity, 0 for stalling) such M. Pouzet and al. introduce it in "N-synchronous Kahn network".

A study of different classes of binary word (Sturm, Christoffel, Lyndon, Mechanical words) was prior to their association to LID theory and *Equalization* process. We obtained statically scheduled systems which answered to the main problem of this work.

Table des matières

1 Introduction **7**

1.1 Notre approche par l'exemple . . . 9

1.1.1 Abstraction . . . 10

1.1.2 Expansion de latences . . . 10

1.1.3 *Marked Graph* . . . 11

1.1.4 Systèmes insensibles aux latences . . . 12

1.1.5 Pré-égalisation et égalisation de latences . . . 13

1.1.6 Ordonnancement k-périodique balancé . . . 14

1.1.7 Implantation de l'emplacement secondaire de la place de gauche. . . . 15

1.1.8 Résultat . . . 15

1.2 Plan . . . 17

2 Mots binaires infinis périodiques **18**

2.1 Introduction . . . 18

2.2 Motivation . . . 19

2.3 Références bibliographiques . . . 19

2.4 Définitions préliminaires . . . 19

2.4.1 Mots binaires finis et infinis . . . 19

2.4.2 Operations : rotation et transposition . . . 20

2.5 Mots binaires balancés . . . 21

2.5.1 Définitions et propriétés de base . . . 21

2.6 Propriétés structurelles . . . 23

2.6.1 Construction de mots balancés . . . 23

2.6.2 Préservation de mots balancés . . . 24

2.6.3 Primalité des orbites . . . 26

2.6.4 Ordre dans l'ensemble $\mathbb{S}_p^k$ . . . 27

2.6.5 Transposition . . . 27

2.7 Relation entre transposition et rotation . . . 28

2.7.1 Le coefficient α . . . 29

2.7.2 Relation entre transposition and rotation . . . 29

2.8 Création de mots balancés utilisant le coefficient α 30
2.9 Conclusion 31

3 Réseaux de processus concurrents **33**
3.1 *Marked Event Graphs* 33
3.2 Sémantique dynamique 36
3.2.1 Règle d'exécution des nœuds de calculs 36
3.2.2 Propriétés 39
3.2.3 Retard 39
3.3 Modèle avec latence 40
3.4 Modèle avec capacité de stockage borné 42
3.4.1 Débit de graphe avec capacité 43
3.4.2 Capacité instantanée 45
3.5 Conclusion 46

4 Conception insensible aux latences (LID) **47**
4.1 La théorie du "Latency Insensitive Design" (LID) 47
4.1.1 Les problèmes de conception 48
4.1.2 Présentation générale 48
4.1.3 Shell-Wrapper 48
4.1.4 Relay-Station 50
4.1.5 Channel 50
4.1.6 Protocole de rétroaction (*Back-Pressure Protocol*) 51
4.1.7 Propriétés de correction 51
4.2 Formal Latency Insensitive Design 52
4.2.1 Description formelle de la *Relay-Station*(RS) 52
4.2.2 Vérification formelle du bon comportement de la *Relay-Station* 54
4.2.3 Description formelle du *Shell-Wrapper* 55
4.2.4 Vérification formelle du bon comportement du *Shell-Wrapper* 55
4.3 Vue d'ensemble des contributions aux "Latency Insensitive Design" 57
4.3.1 Systèmes insensibles aux latences généralisés 57
4.3.2 Une nouvelle approche du LID 58
4.3.3 Circuits synchrones élastiques 60

5 égalisation **62**
5.1 Processus d'égalisation 63
5.1.1 Lister les cycles : 63
5.1.2 Calculer le débit : 64
5.1.3 Calculer la k-périodicité du graphe. 64
5.1.4 Ajouter les latences entières : 64

5.1.5 Déterminer les ordonnancements . . . 65
5.1.6 Placer les registres fractionnaires : . . . 66
5.1.7 Optimiser l'initialisation : . . . 67
5.2 Le *Registre Fractionnaire* . . . 67
5.2.1 Hold . . . 69
5.2.2 Vérifications formelles du comportement d'un *registre fractionnaire*. . . . 70
5.3 Problème d'optimisation . . . 71
5.3.1 Calcul du débit . . . 71
5.3.2 Ajout de latences . . . 71
5.4 Propriété de correction . . . 73
5.4.1 Validation du processus d'égalisation . . . 73
5.4.2 Modélisation des canaux de communication . . . 74
5.5 Conclusion . . . 75

6 Régimes stationnaires balancés **76**
6.1 Bibliographie . . . 77
6.2 Méthodologie générale . . . 77
6.2.1 Ajouter les registres fractionnaires : . . . 77
6.2.2 Ordonnancer les nœuds du graphe : . . . 80
6.2.3 Ordonnancer les *registres fractionnaires* : . . . 81
6.2.4 Définir les états du régime permanent : . . . 82
6.2.5 Trouver une initialisation asynchrone : . . . 83
6.3 Propriétés de correction . . . 83
6.3.1 Ajout des *registres fractionnaires* . . . 83
6.3.2 Ordonnancements des nœuds de calculs . . . 86
6.3.3 Ordonnancements des *registres fractionaires* . . . 89
6.3.4 Les jetons . . . 91
6.3.5 Le régime permanent . . . 97
6.3.6 Initialisation du système . . . 98
6.3.7 Modélisation des canaux de communication . . . 99
6.4 Conclusion . . . 99

7 Mise en oeuvre **100**
7.1 Présentation . . . 100
7.2 Composants du logiciel . . . 101
7.2.1 Structure de données . . . 101
7.2.2 Algorithmique . . . 103
7.2.3 Simulation . . . 105

8 Conclusion **106**

A Valorisation des compétences des docteurs **122**

A.1 Cadre général et enjeux de ma thèse 122

A.1.1 Présentation succincte 122

A.1.2 Contexte de ma thèse 123

A.1.3 Moi dans ce contexte 124

A.2 Déroulement, gestion et coût de mon projet 125

A.2.1 Préparation et cadrage de mon projet 125

A.2.2 Conduite de mon projet 126

A.2.3 Evaluation et prise en charge du coût du projet 127

A.2.4 Ressources humaines 127

A.3 Compétences, savoir-faire, qualités professionnelles et personnelles 128

A.4 Résultats, impacts de la thèse 130

Chapitre 1

Introduction

Ce travail de thèse a été conduit (et financé) dans le cadre du projet Sys2RTL du Centre Intégré en Microélectronique régional PACA (CIM PACA). Le point de départ était d'étudier la conception de systèmes-sur-puce (SoC, pour "Systems-on-Chip") à partir de composants préexistants (dits blocs IP), et ce en considérant que les latences de communications entre ces composants ne pouvaient plus être considérées comme unitaires sur une horloge unique (communications synchrones). Les idées-clés évoquées pour motiver l'approche étaient "latency-insensitive design" (LID) et "timing closure", dont le sens technique précis restait à déterminer dans notre contexte.

Nos travaux se situent dans le cadre particulier où les latences des nœuds de calculs (les composants IP), ainsi que celles des canaux de communication (les "long global wires"), ont chacune une valeur fixe, connue et donnée en nombre entier de cycles sur une horloge virtuelle commune (le pas d'exécution). Ce modèle étend le modèle synchrone (où les latences sont unitaires sur une horloge globale de réaction), mais restreint le cas général (où les latences peuvent être variables dans le temps en fonction des données, des branchements, etc). Le modèle s'applique bien en particulier à certaines architectures matérielles embarquées pour le traitement de signal multimedia, où les calculs sont réguliers et répétitifs. La latence de chaque nœud de calculs composant est supposée obtenue par sa synthèse, indépendamment du système hôte : l'objectif est ici bien de pouvoir réutiliser des composants préexistants sans les modifier, et la seule hypothèse faite sera que l'horloge d'exécution de chaque composant peut être commandée par le système. La latence de chaque canal de communication entre composants correspondra en général à sa longueur ; elle pourra avoir été déterminée à partir de procédures de placement/routage. Dans tous les cas on fera l'hypothèse que ces latences de calculs et de communication sont des données du problème. L'objectif de l'approche sera d'exhiber des ordonnancements et de calculer des tailles de ressources (de bufférisassions essentiellement) pour optimiser le débit du fonctionnement du système et des composants, en respectant les contraintes temporelles et en minimisant l'empreinte physique de la mémoire.

La première partie du travail a été de proposer un modèle formel pour notre classe de SoCs. Il s'agit en particulier de modéliser finement les canaux de communication introduits pour assurer la conception insensible aux latences, c'est à dire une conception où on puisse construire ce système comme son modèle pour quelle que valeur de latence que ce soit, arbitraire mais donnée. A cette fin la théorie LID introduit deux

types de composants : d'une part des Shell Wrappers, encapsulant les composants de calculs de l'extérieur, et déclenchant leur exécution exactement quand les acheminements de données sur les canaux l'autorisent ; d'autre part les Relay-Stations, qui sont les éléments permettant de construire des tronçons de canaux de latence unitaire. De cette manière, un canal de latence n est modélisé par une chaîne de n Relay-Stations. Notre modèle se révèle étonnamment simple et direct : le système devient un Graphe d'Evénement (Marked/Event Graph) après expansion des latences, les nœuds de calculs en sont les transitions, et chaque Relay-Station est au départ une place bornée de capacité 2. A nouveau l'expansion de ces places procure exactement l'algorithme de contrôle de flux permettant d'éviter les congestions et de maintenir un certain flux d'exécutions locales et globales, tout en garantissant que l'implantation physique pourra en être faite.

Des travaux antérieurs [15] et parallèles à nos recherches [21] et [28] ont également été menés conduisant à des implantations matérielles efficaces (dites "élastiques" dans l'approche Cortadella/Kishinevsky). Néanmoins nos travaux sont les premiers qui ont relié aussi clairement la théorie LID à la modélisation par Marked/Event Graphs (MEGs) et autres Process Networks (PNs) définis en Théorie du Parallélisme, et aux propriétés mathématiques importantes existantes dans ces modèles formels. On peut montrer en particulier que dans certains cas le contrôle de congestion peut mener effectivement à des pertes de performance dues à des propagations amont. On peut caractériser exactement les conditions numériques de ce comportement à partir des valeurs des latences et le nombre de données.

Dans le cas le plus intéressant, quand les systèmes sont topologiquement fortement connexes et clos (ou quand on peut supposer que les données en entrée surviennent exactement au rythme attendu), les développements un peu plus anciens conduits sur l'ordonnancement statique de tels systèmes MEG par Carlier-Chrétienne [14] puis Cohen-Quadrat-Baccelli [8] en particulier permettent d'obtenir des résultats puissants sur leur ordonnancement dit k-périodique en phase stationnaire (après une période de stabilisation initiale). Ces ordonnancements peuvent être représentés comme des entités explicites au moyen de mots binaires infinis ultimement périodiques, qui seront manipulés dans le processus de synthèse et d'optimisation. L'objectif des optimisations sera principalement de réduire encore les besoins en ressources de bufférisassions, en étudiant les places exactes où le recours à un second registre s'avère nécessaire, et celles où une capacité 1 suffit. On cherche également à se dispenser de la logique combinatoire induite dans l'implantation matérielle pour y encoder le contrôle de flux et de congestion car cette logique ainsi que les registres sont ajoutés après que des informations de placement-routage aient donné les latences des canaux de communication. Cette exploitation des résultats d'ordonnancement statique appliqués et adaptés à notre contexte forme une seconde partie du document. Elle reprend également des principes de modélisation pour les langages dits N-synchrones (étendant les langages synchrones par des horloges multiples mais k-périodiques) de Pagetti-Pouzet-Cohen et al [25], où de tels mots binaires infinis sont utilisés pour le typage des sous-programmes concurrents.

Il apparaît alors que, même si les ordonnancements produits dans cette phase conduisent à une distribution relativement étalée des données, il existe des cas simples où une meilleure répartition pourrait encore optimiser la forme des ordonnancements et le gain des ressources, mais que ces solutions ne sont pas celles obligatoirement obtenues par une simulation simple et déterministe par exécution ASAP (chaque nœud s'exécute dès que possible, plusieurs possiblement simultanément). La troisième partie du document

se concentre sur l'étude de l'usage des mots infinis binaires balancés, c'est-à-dire dont les occurrences respectives entre les lettres 1 et 0 sont le mieux distribué possible. Ils seront utilisés pour construire des ordonnancements stationnaires le mieux réparti possible. Ces ordonnancements peuvent alors être construits de manière analytique, sans recours à une simulation éventuellement coûteuse. De plus ils permettent la meilleure optimisation possible des ressources de bufférisassions, et dispensent totalement de contrôle de flux dynamique. La représentation des instants d'activation de chaque nœud de calculs du système se fait par un simple offset initial sur le mot commun d'ordonnancement. Cette étude de l'utilisation des mots balancés pour des ordonnancements optimaux forme la troisième et dernière partie de ce mémoire. La faiblesse actuelle de cette approche est que, pour des raisons de temps, nous n'avons pas pu conduire entièrement l'étude de la phase d'initialisation menant d'un marquage arbitraire des positions de données dans le système à un marquage du régime stationnaire balancé. Nous savons que des initialisations optimales asynchrones existent mais les obtenir algorithmiquement reste un problème ouvert.

Par le passé, plusieurs personnes ont chercher à optimiser certains aspect d'un graphe où de son exécution. Nous pouvons citer Laurent Hardouin dont les résultats sont regroupés dans [36] et plus récement Olivier Marchetti qui dans sa thèse [42] cherche lui aussi à minimiser les ressources de stockage nécessaire à l'exécution de graphes d'événement.

Les mots binaires infinis balancés sont apparus dans les travaux mathématiques de E. B. Christoffel [24] et J. Bernouilli [10] , au XVIII^{eme} et XIX^{eme} siècles. Des travaux de E. Altman, B. Gaujal [5] les ont déjà appliqués à des questions de routage de paquet dans des réseaux. Dans nos travaux nous étudions spécifiquement leurs propriétés comportementales face à des opérations de rotation (qui correspondent à des décalages temporels dans la suite des instants d'exécution) ainsi qu'à des opérations de transposition élémentaire de type ...10... $\rightarrow$ ...01..., qui correspondent au retard élémentaire ("stall") d'une donnée pour un instant, par contrôle de flux. Les relations entre ces opérations sont systématiquement explorées, et un certain nombre de théorèmes de structure sont établis lors d'une partie initiale du présent document. Cette partie précède les chapitres sur la modélisation, l'ordonnancement et l'optimisation des Process Networks dans un contexte de conception LID de SoCs proprement dite.

1.1 Notre approche par l'exemple

Nous allons, dans cette section, utiliser l'exemple de la figure 1.1 pour introduire les principales étapes de notre approche. La figure 1.1 présente le bloc diagramme fonctionnel abstrait d'un système-sur-puce que l'on souhaite concevoir.

Les blocs "pré-traitement", "traitement" et post-traitement" sont des composants IPs que l'on s'est par exemple procuré chez différents revendeurs, et dont on ne dispose que comme boite noire. En fait, on suppose qu'on peut les exécuter à la demande en leur envoyant un signal d'activation (leur horloge), et que leur fonctionnement consomme une valeur sur chaque entrée au début de l'instant et produit une valeur sur chaque sortie à la fin de l'instant. Les éléments de mémorisation intéressants se trouvent donc sur les canaux.

De plus, on suppose que la liaison point à point de retour joignant les blocs de "post-traitement" et de

"pré-traitement" souffre d'un problème de latence (à droite sur la figure). Quand une donnée est émise sur ce canal par le bloc de "post-traitement", elle est reçue par le bloc de "pré-traitement" seulement trois cycles d'horloge plus tard du fait de la longueur de la connexion.

La resynchronisation du système nécessite l'ajout de ressource (de stockage, d'ordonnancement) sur les canaux et autour des composants. Notre approche consiste à trouver un ordonnancement qui minimise la quantité de ressource de stockage nécessaire.

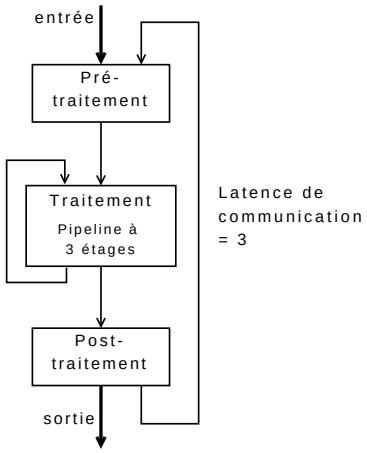


FIG. 1.1 – Système sur puce ayant des problèmes de délais de communication.

Quand ils sont actifs, les blocs de calculs consomment un jeton sur toutes leurs entrées et produisent un jeton sur chacune de leurs sorties. Le bloc median "traitement" a une structure pipelinée à trois étages et donc une latence de calcul. On suppose qu'initialement, il existe exactement une donnée à la sortie (où à l'entrée) de chaque bloc de calculs. Que les étages intermédiaires du pipeline du bloc "Traitement" sont aussi pleins. On considèrera également que les données en entrée globale du système sont disponibles exactement quand "Pré-traitement" s'exécute.

1.1.1 Abstraction

Dans notre approche, la fonction exacte calculée par les blocs de calculs ne nous importe pas, pas plus que la valeur d'une donnée : seulement sa présence ou son absence compte ainsi que le trafic qu'elle génère. Le système de la figure 1.1 est abstrait dans le graphe de la figure 1.2. Les blocs de calculs deviennent des nœuds de calculs. Les données deviennent des jetons. La nature des liaisons peut varier.

Les informations sur la structure pipelinée du bloc de "traitement" ainsi que sur le délai de communication sont conservées. Elles deviennent une *latence de calcul* associée au nœud "Traitement" et une *latence de communication* associée au canal de droite.

1.1.2 Expansion de latences

Pour de nombreuses raisons, il nous sera nécessaire d'expanser les latences (de calculs comme de communication) pour en exhiber explicitement les différentes étapes, et les positions progressives des jetons lors

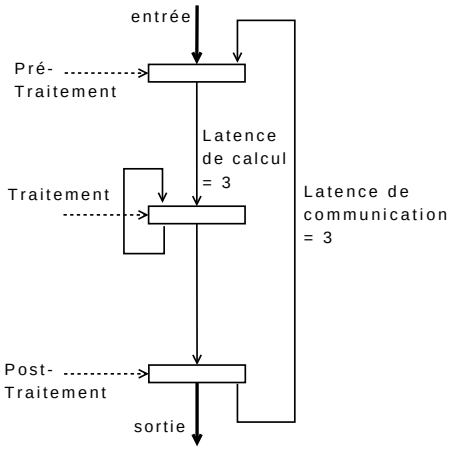


FIG. 1.2 – Transformation du système de la figure 1.1 en graphe.

des évolutions. Cette expansion se définit simplement (nos latences sont entières), et mènent au système de la figure 1.3

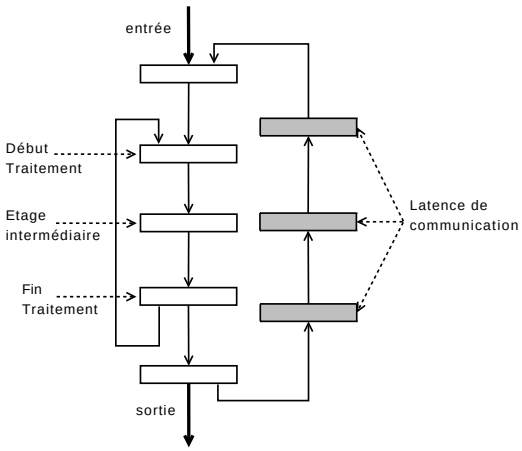


FIG. 1.3 – Découpage du pipeline du bloc de traitement en plusieurs blocs et représentation explicite du délai de communication.

1.1.3 *Marked Graph*

Si l'on considère maintenant que les canaux entre les transitions de calculs peuvent contenir des places de stockage non bornées pour les jetons, le lecteur aura reconnu un modèle dit *Marked graph* souvent présenté comme un sous-ensemble particulier de *Réseau de Petri* où chaque place comporte exactement une transition entrante et une transition sortante. Ce modèle est connu pour son absence de conflit et son déterminisme (si un nœud est exécutable, il le reste jusqu'à ce qu'il soit exécuté). On notera par ailleurs, que dans ce modèle le nombre de jetons pour tout cycle du graphe est constant au cours de l'exécution.

Sous notre hypothèse (les entrées globales surviennent à la demande $\Leftrightarrow$ le système est considéré clos),

on peut montrer que l'exécution "au plus tôt" des blocs de calculs c'est à dire dès que les jetons sont disponibles en entrée mène à un régime de fonctionnement dit "ultimement périodique". En d'autres termes, après une phase initiale d'ajustement, chaque bloc s'exécute suivant un schéma régulier et répétitif. Ces résultats sont dus à Carlier, Chretienne dans [14], puis raffinés pour le calcul des longueurs de période par Cohen, Quadrat, Baccelli et al. dans [8]. On peut alors représenter les schémas d'activations résultant d'un ordonnancement "au plus tôt" des *Marked graphs* par des mots binaires infinis k-périodiques (où **1** en i^{eme} position représente l'activité à l'instant i et **0** l'inactivité).

On a représenté l'ordonnancement des blocs dans la figure 1.4 (dans notre cas, la partie initiale est vide). On peut remarquer les phénomènes suivants généraux : les périodes de tous les blocs sont égales, et les mots contiennent tous le même nombre de **1** (dénommé périodicité). Ces nombres sont ceux de la latence et du nombre de jetons présents dans le cycle du graphe le plus lent. On note que sur notre exemple, il apparait des accumulations de jetons sur le canal de gauche. Il faut alors à priori prévoir une capacité de bufférisassions égale à la somme des jetons du cycle pour chacune de ces places.

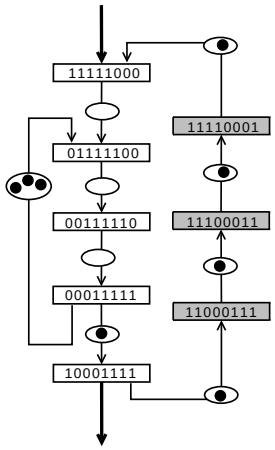


FIG. 1.4 – Représentation du système comme un *Marked Graph*. Les mots binaires inscrits dans les nœuds du graphe représentent leur ordonnancement. (Cette manière de représenter les ordonnancements ne fait pas partie des *Marked graph*, elle a été introduite dans [25]).

Nous allons maintenant chercher à raffiner ce modèle afin de borner la capacité de chaque place en dessous de cette borne haute pour obtenir une solution implantable sous forme de circuit.

1.1.4 Systèmes insensibles aux latences

La théorie du *Latency-Insensitive Design*(LID) vise à proposer une modélisation au niveau physique des canaux et de la logique déterminant les instants d'exécution des blocs de calculs IP. Chaque section de latence élémentaire est "implantée" par une *Relay-Station*, qui au niveau abstrait correspond à une place de capacité 2. La politique d'exécution devient : un bloc peut-être exécuté quand ses données sont disponibles en entrée **mais aussi** quand ses canaux de sortie l'autorisent (s'il ne sont pas pleins). Le *Shell-Wrapper* modélise cette politique (Dans la terminologie LID, les composants sont dénommés "Perles" et donc en-

capsulées dans une "coquille").

Cette restriction de capacité implémente de fait un protocole de contrôle de flux évitant les congestions : une place avertie d'un "embouteillage" à sa sortie ne peut avertir son prédécesseur qu'à l'instant suivant. D'où la nécessité d'une seconde cellule de mémoire (i.e. registre). A l'instant suivant, elle peut bloquer le trafic en amont donc une troisième cellule est inutile. Nous avons trouvé des exemples où par propagation arrière du signal de blocage, des phénomènes pathologiques apparaissent, contredisant certaines conclusions du LID. Sur l'exemple 1.5, le débit du graphe est de $4/7$ alors qu'il était de $5/8$ dans le *Marked graph* de la figure 1.4. Dans cet exemple, limiter les capacités des places à 2 réduit le débit du graphe.

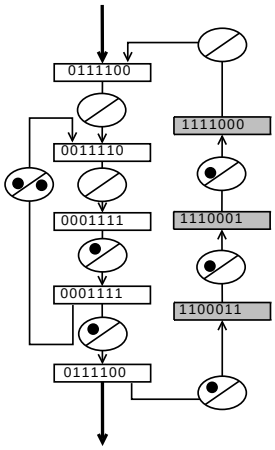


FIG. 1.5 – Simulation du fonctionnement du système insensible aux latences.

1.1.5 Pré-égalisation et égalisation de latences

Quand la différence de débit entre les cycles du graphe fait que, pour un cycle de fort débit, la seconde cellule de la station de relais est utilisée par toutes les données en transit, alors le système admettrait une latence supplémentaire à cet endroit. Cette latence "virtuelle" pourrait ensuite être utilisée par le designer, par exemple pour relaxer des conditions sur les composants eux-mêmes, mais ceci sort du cadre de cette thèse. La transformation de pré-égalisation consiste à ajouter autant de latence que possible sans pénaliser le débit global. Les cycles ont alors des débits trés proches et il ne reste plus que des écarts que nous nommerons *fractionnaires* ou *rationnels* entre ces débits. L'étude suivante, de pleine égalisation (fractionnaire), consistera justement à définir un registre partiellement transparent qui parfois retarde certaines valeurs d'un instant, et parfois les laisse progresser immédiatement.

A nouveau, par calcul d'un ordonnancement statique "au plus tôt", on peut déterminer les schémas temporels (des mots binaires infinis ultimement périodiques) selon lesquels les données doivent être retenues ou libérées.

Le graphe de notre exemple dans la figure 1.4 est composé de deux cycles. Le cycle de gauche formé par le triple nœud de "traitement" et le cycle de droite formé par les cinq nœuds de calculs et les trois nœuds de transport. Le cycle de gauche a un débit de $3/3$ (3 jetons pour 3 places). Et le cycle de droite a un débit de

$5/8$ (5 jetons pour 8 places). On peut rajouter une place (et un nœud de transport) dans le cycle de gauche afin de réduire son débit à $3/4$. Ce dernier est toujours supérieur au débit du cycle de droite. La figure 1.6 illustre cette modification.

La conséquence immédiate de cette modification est le retour à un débit de graphe de $5/8$ comme on peut le voir à la longueur et au nombre de 1 des ordonnancements inscrits dans les nœuds du graphe de la figure 1.6. De plus, l'ancienne place de gauche est moins souvent saturée.

L'ajout de latence augmente le coût d'implantation, ce qui est contraire à notre but. Par contre, celle-ci permet quasiment à elle seule de resynchroniser le débit des deux cycles du graphe. Ce changement "semble" améliorer le fonctionnement du système. Le chapitre 5 présente le processus d'*égalisation* qui formalise l'ajout de latence virtuelle comme outils de resynchronisation de graphe.

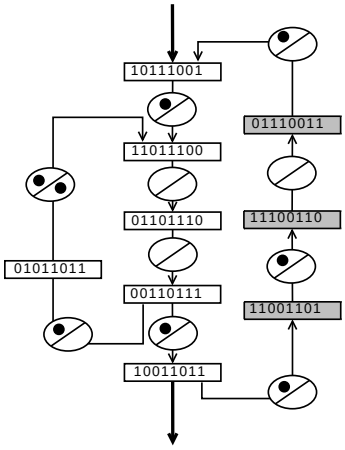


FIG. 1.6 – Simulation du fonctionnement du système pré-égalisé

1.1.6 Ordonnancement k-périodique balancé

On a vu que l'effet des limitations locales en capacité de bufférisassions était de mieux répartir les occurrences respectives de **0** et de **1** dans les mots binaires d'ordonnancement. On a néanmoins aussi vu que de telles contraintes seules, avec application d'une stratégie d'exécution "au plus tôt" des blocs de calculs, ne menaient malgré tout pas forcément à une solution optimale en allocation de ressources de bufférisassions. Nous allons donc maintenant prendre l'approche inverse, et définir des types d'ordonnancement par mots réguliers dit "balancés" (dus à Bernouilli père [10] puis Christoffel [24], et étudiés dans le cadre de l'ordonnancement/ Routage par Altman-Gaujal et al. [5]). Nous allons montrer maintenant comment (sur notre exemple) les ordonnancements balancés peuvent être propagés et préservés pour décrire un fonctionnement stationnaire admissible et efficace (i.e. respectant le débit du système).

Tout d'abord, on peut supprimer les registres de capacité 2 du cycle de droite car ils ne sont jamais utilisés (figure 1.7).

Si on regarde l'ordonnancement des nœuds du graphe sur les figures 1.4 et 1.6, on a, pour le nœud le plus en haut, respectivement 11111000 et 10111001. Ces deux d'ordonnancement alternent des phases

de travail intensif (3 à 5 instants d'activités consécutifs) et des phases de relâche totale (2 à 3 instants d'inactivités consécutifs). Dans la figure 1.7, on a cherché à répartir le plus possible les instants d'activités et d'inactivités. On a "balancé" l'ordonnancement.

Résultat : le nœud de gauche ne se retrouve jamais bloqué alors que ses entrées sont valides parce que sa place en sortie est pleine. Les ordonnancements *balancés* et les mots *balancés* sont présentés dans le chapitre 2. En plus de réduire la quantité de places de capacité 2 nécessaires dans un graphe quand ses nœuds sont ordonnancés en utilisant des mots *balancés*, ils permettent aussi de construire un ordonnancement atteignable et optimal quand à la quantité de places de capacité 2 nécessaires. La construction d'un ordonnancement dit *balancé* est le sujet du chapitre 6.

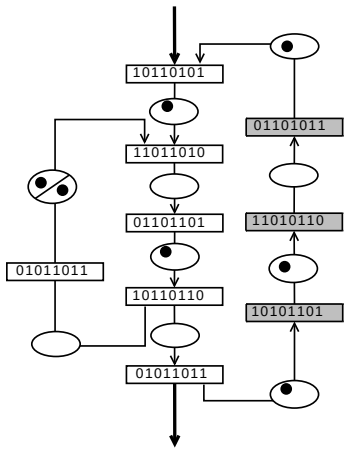


FIG. 1.7 – Simulation du fonctionnement du système pré-égalisé avec un ordonnancement balancé.

1.1.7 Implantation de l'emplacement secondaire de la place de gauche.

L'ajout de la latence sur le canal de gauche n'a pas suffi à resynchroniser totalement les deux branches du système. Il arrive que certains jetons du canal de gauche se présentent en entrée du nœud "début traitement" avec un instant d'avance. Dans la figure 1.8, durant une période, quatre jetons arrivent en avance au nœud "début traitement". La simulation nous a permis de savoir à quel instant ces jetons arrivent. On a donc rajouté un élément capable de capturer ces jetons et uniquement eux et de les relacher quand il le faut : le *Registre-Fractionnaire*. La section 5.2 présente le *registre fractionnaire* comme un automate à état fini et comme un circuit.

1.1.8 Résultat

A partir du graphe de la figure 1.8, on peut revenir à notre système initial en ayant résolu le problème de latence. On a rajouté trois registres sur le canal de droite afin de découper le fil d'interconnexion en section synchrone. On a ajouté un registre virtuel sur le canal de gauche pour freiner le débit du cycle. Le *registre fractionnaire* permet de gérer les dernières asynchronies entres les deux branches du système.

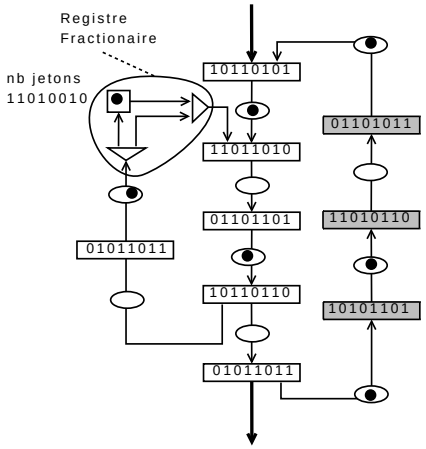


FIG. 1.8 – Une analyse statique du système nous permet de gérer localement la resynchronisation de la branche de droite avec celle de gauche grâce à notre *registre fractionnaire*.

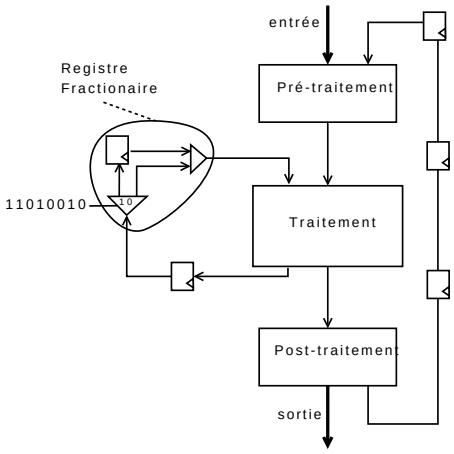


FIG. 1.9 – Le système égalisé et réordonnancé est muni d'un registre fractionnaire afin d'être implanté. Il n'a plus de problème de latence.

L'étude topologique du système nous a permis d'obtenir une solution équivalente à la solution historique (Latency Insensitive Design) mais avec beaucoup moins de ressources.

1.2 Plan

Tous nos résultats d'ordonnancement statique reposent sur une représentation explicite des instants d'activité au moyen de mots infinis binaires ($1 =$ Actif ; $0 =$ Inactif). En conséquence, le chapitre 2 contient essentiellement des rappels et définitions sur ces mots, mais aussi des théorèmes de structures sur les mots binaires balancés et leurs réactions à des opérations reliées ensuite à des sémantiques d'exécution aux chapitres suivants (principalement au chapitre 6).

Le chapitre 3 donne un cadre général à la théorie des Réseaux de Processus (Process Networks) **sans conflits** (où chaque place/canal possède exactement un bloc de calculs en entrée et en sortie). Les différentes variations possibles sur la nature des canaux, la prise en compte de latences et les propriétés de safety/livennes y sont considérées.

Le chapitre 4 définit la théorie de la conception insensible aux latences (*Latency Insensitive Design*/LID) due à L. Carloni et al [15]. Elle l'aborde d'un point de vue abstrait, en modélisant ses *Relay-Stations* comme place de capacité 2 par exemple.

Les prémices de la théorie du LID, et sa modélisation au moyen de composants hardware spécifiques synchrones ou asynchrones, ont également été considérés par Cortadella, Kishinersky et al [37] comme modèles "'synchrones élastiques"'.

Le chapitre 5 traite des questions d'égalisation, liées à l'optimisation de l'allocation des registres (entiers et fractionnaires) dédiés à la bufférisassions dans les canaux dans le cadre d'un ordonnancement statique k-périodique. Ces développements sont principalement la base algorithmique d'un logiciel *K-Passa*, décrit plus tard au chapitre 7. Le chapitre 6 aborde la question des ordonnancements statiques balancés, et de la présentation d'un même patron d'activité pour représenter l'activité de tous les blocs de calculs du système. Il utilise abondamment des résultats théoriques du chapitre 2.

Enfin, le chapitre 7 traite du logiciel *K-Passa* et de l'implantation des techniques des chapitres 4 et 5 dans un outil effectif. Les techniques du chapitre 6 ne sont pas encore soumises à implantation.

Le chapitre 8 conclut le document en mentionnant certaines pistes pour l'extension de ces travaux, dont certaines sont déjà amorcées (l'extension à des modèles avec choix internes sur les trafics de données).

Chapitre 2

Mots binaires infinis périodiques

2.1 Introduction

Nous allons parler de mots binaires infinis périodiques pour lesquels les occurences de **0** et de **1** sont les plus espacées possible.(pour un ratio de chaque lettre donné). Ces mots seront appelés *balancés*. Dans la littérature, les mots de *Sturm*[38, 3] sont un exemple de mots balancés infinis et apériodiques. Les mots de *Christoffel* [24, 39], eux le sont dans le cas périodique fini ou infini. Les mots de *Christoffel* sont maintenant attribués à Jean Bernouilli [10].

Nous allons définir formellement les mots balancés puis établir plusieurs résultats simples mais utiles les concernant Par exemple, pour un ratio donné k et n, l'ensemble des mots balancés est l'orbite (l'ensemble des rotations) du mot de *Christoffel* défini pour le même ratio. Ce dernier est le plus petit mot balancé au sens de l'ordre lexicographique.

Ensuite, nous allons nous intéresser à l'opération de transposition élémentaire qui consiste à substituer le motif 10 par le motif 01 à une position particulière du mot. Nous allons établir que concernant les mots balancés, pour un certain ratio k et n, il existe pour chaque mot un autre mot balancé obtenu par transposition élémentaire à partir du précédent.

Ce résultat est essentiel pour construire analytiquement le régime permanent d'un réseau de processus (graphe d'événement/*Event Graph*) dans la théorie de l'ordonnancement k-périodique, comme c'est le cas dans le domaine du *software pipelinning* et du *Jobshop scheduling*. La construction effective de cet ordonnancement est le thème du chapitre 6 de ce mémoire.

Le but de ce chapitre est de présenter des résultats préliminaires sur les mots balancés. Le premier théorème (théorème1) donne la taille et la composition de l'ensemble des mots balancés pour une longueur et un ratio donné. Le second théorème (théorème 2), concerne deux opérations définies dans la section 2.4 : La rotation et la transposition. Ces deux théorèmes nécessitent beaucoup de résultats intermédiaires, lemmes et autres propriétés présentés dans ce chapitre.

2.2 Motivation

Ce travail est motivé par des problématiques en théorie de l'ordonnancement k-périodique. On considère un réseau de processus où les nœuds de calculs sont interconnectés par des canaux de communications qui transportent des données (abstraits comme des jetons) de la sortie vers l'entrée d'un autre nœud de calculs ; de plus, l'exécution simultanée de plusieurs nœuds de calculs est permise. On peut représenter l'état d'activité d'un nœud de calculs instant après instant en utilisant **1** pour l'activité et **0** pour l'inactivité. Cette notation à été définie dans [25]. La séquence d'activité d'un nœud peut donc être représentée comme un mot binaire infini et la collection de ces mots pour tous les nœuds nous fournit l'ordonnancement du système.

Bien sûr l'ordonnancement, et à fortiori le mot binaire qui commande l'activation doit se conformer aux exigences strictes du jeu et de la disponibilité des jetons. Dans plusieurs cas spécifiques, comme les *Marked graph* fortement connexes, les techniques d'ordonnancement donnent un mot binaire ultimement périodique comme ordonnancement. Mais certains travaux concernant l'optimisation de ressource, principalement concernant la taille des canaux de communication, demandent en plus à ce que la répartition des jetons dans le flot soit la plus régulière possible. Ceci nous mène à l'utilisation de mot balancé comme classe de mot d'ordonnancement.

Un gros avantage à utiliser les mots balancés dans ce contexte vient du fait que contrairement au cas où les mots d'ordonnancement sont obtenus par simulation (construction de l'espace d'état atteignable), i ci, l'ordonnancement de chaque nœud de calculs est obtenu de manière analytique à partir d'un seul. Ceci nécessite d'utiliser des opérations particulières sur les mots infinis périodiques comme la rotation (retarder tout le monde d'un instant) ou la transposition élémentaire (retarder un jeton un instant) et leur interaction algébrique.

2.3 Références bibliographiques

La plupart des résultats présentés dans ce chapitre ont déjà été publié dans [40]. Gwénaël Richomme a aussi apporté sa contribution à l'étude des mots balancés dans [50]. La nouveauté de ce chapitre concerne la relation de transposition présenté à la section 2.4. Nous allons traiter de l'influence de cette relation sur les mots balancés à la section 2.6.5.

2.4 Définitions préliminaires

2.4.1 Mots binaires finis et infinis

Notation 1 (Mot binaire)**.** *Un mot binaire est une séquence finie ou infinie de valeurs binaires (ou* bits*).*

On note $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$*, pour booléen,* $\mathbb{B}^*$ *le monoïd libre de mot binaire fini,* $\mathbb{B}^+$ *l'ensemble de mot binaire fini non vide,* $\mathbb{B}^\omega$ *l'ensemble des mots binaires infinis, et* ε *est le mot vide.*
On note $\mathbb{B}^\infty = \mathbb{B}^* \cup \mathbb{B}^\omega$*. l'ensemble des mots bianires finis et infinis*

Pour $u \in \mathbb{B}^\infty$ on note $|u|$ la longueur de u (avec $|u| = \infty$ quand $u \in \mathbb{B}^\omega$).
De même, on note $|u|_1$ et $|u|_0$ le nombre d'occurrences de la lettre **1** *et* **0** *dans u respectivement.*
De plus, pour $u \in \mathbb{B}^+$ on note $rate(u)$ le rapport $|u|_1/|u|$. On note $\mathbb{B}_p^k$ le sous ensemble de $\mathbb{B}^*$ formé des mots u avec $|u| = p$ et $|u|_1 = k$.

Pour $u \in \mathbb{B}^\infty$ on définit son conjugué $\overline{u} \in \mathbb{B}^\infty$ comme le mot pour lequel les lettres 0 et 1 sont interchangées. Formellement : $\overline{\epsilon} = \epsilon$, $\overline{0.u} = 1.\overline{u}$ et $\overline{1.u} = 0.\overline{u}$.

Pour $i \leq |u|$ on note $u(i)$ ls i^{th} lettre de u.
*On utilise l'ordre lexicographique sur les mots : Pour $u, v \in \mathbb{B}^\infty$, $u < v$ ssi $\exists i \in \mathbb{N}, \forall j < i, u(j) = v(j)$ soit $u(i) =$ **0** et $v(i) =$ **1** soit $|u| = i$ et $|v| > i$. Cet ordre est total. Pour tout sous-ensemble V de $\mathbb{B}^\infty$, on note $inf(V)$ et $sup(V)$ respectivement son plus petit et son plus grand élément pour cet ordre.*
On utilise aussi l'ordre de précédence sur les mots : Pour $u, v \in \mathbb{B}_p^k$, $u \prec v$ ssi $\forall i \in [1, k]$, $u \uparrow i \geq v \uparrow i$ (où $u \uparrow i$ donne l'index du i^{th} **1** *dans u). Bien sûr, $(u \prec v) \Rightarrow (u < v)$, mais l'inverse n'est pas vrai : $0110 < 1001$ mais 0110 et 1001 ne peuvent être comparés par l'ordre de précédence ; la relation de précédence est un ordre partiel.*

Définition 1 (Mot binaire périodique infini)**.** *Un mot binaire infini t est dit* ultimement périodique *s'il est de la forme $u.v^\omega$, avec $u \in \mathbb{B}^*$ et $v \in \mathbb{B}^+$.*
Il est appelé simplement périodique *si en plus, $u = \epsilon$*
Pour $t = u.v^\omega$ un mot ultimement périodique, on appelle u sa partie initiale, *v sa partie* periodique, *$|v|_1$ sa* périodicité, *et $|v|$ sa période. On étend aussi la définition du* rate *par $rate(t) = rate(v)$, C'est le rapport de sa périodicité sur sa période. On appelle $\mathbb{P}$, (respectivement $\mathbb{P}_p^k$) l'ensemble des mots binaires infinis ultimement périodiques. (Respectivement de périodicité k et de période p).*

Contre avis excepté et suivant la tradition de la théorie du *modulo scheduling*, on va usuellement appeler k et p respectivement la périodicité et la période d'un mot binaire infini (ultimement périodique) donné.

Exemple 1. *$11.(01101)^\omega$ est 3-périodique avec une période 5, il appartient à $\mathbb{P}_5^3$.*

Notation 2. *Pour $u \in \mathbb{B}^*$ et $v \in \mathbb{B}^\infty$, u est un sous mot de v (noté $v \subset u$) si $\exists u_1 \in \mathbb{B}^*, u_2 \in \mathbb{B}^\infty$ telque $v = u_1.u.u_2$.*

2.4.2 Operations : rotation et transposition

Définition 2 (Rotation unitaire avant)**.** *On définit la rotation unitaire avant $\rho : \mathbb{B}^* \to \mathbb{B}^*$, comme $\rho(\epsilon) = \epsilon$, et $\forall u \in \mathbb{B}^+, \forall a \in \mathbb{B}$, $\rho(u.a) = a.u$.*

Remarque 1. *La définition est faite de telle manière que, si u représente l'ordonnancement d'un nœud de calculs, $\rho(u)$ est l'ordonnancement du nœud placé juste en dessous dans le graphe.*

Définition 3 (Rotation)**.** *Soit $u \in \mathbb{B}_p^k$. On peut générer le groupe cyclique formé de toutes les rotations de u à partir de rotation unitaire. Ce groupe s'appelle l'orbite de u, et est noté $O(u)$.*
$\rho^n(u) = \rho \circ \rho^{n-1}(u)$.
$\rho^{-n}(u) = \rho^{p-n}(u)$.
$\rho^n(w)$ peut être noté $w \circlearrowleft n$, or $w \circlearrowleft -n$.

Exemple 2.
$\rho^3(11010) = 11010 \circlearrowleft 3 = 01011.$
$\rho^{-3}(11010) = 11010 \circlearrowright 3 = 10110.$
$\rho^p(t) = \rho^0(t) = t$

Définition 4 (Transposition)**.** *Soit u, $v \in \mathbb{B}^*$. On appelle v le transpose unitaire de u (ou simplement le transpose pour faire court dans la suite), noté $v = \tau(u)$, ssi $\exists u_1, u_2 \in \mathbb{B}*$, $u = u_1.1.0.u_2$ et $v = u_1.0.1.u_2$. (ou $u = 0.u_1.1$ et $v = 1.u_1.0$) On peut raffiner la définition en une fonction de u vers v en posant que v est le transpose de u selon la position i noté $v = \tau(u, i)$, où $i = |u_1|$ (ou $i = |u_1| + 1$).*

Remarque 2. *La transposition élémentaire est faite pour représenter les effets d'un retard unitaire d'un seul jeton à un point donné dans le schéma d'ordonnancement (où 1 représente l'activité et 0 l'inactivité).*

Exemple 3.
$\tau(10101, 3) = 10011.$
$\tau(11010, 3)$ *n'est pas défini.*
$\tau(011, 3) = 110.$
$\tau((10101)^\omega, 3) = 10011.(10101)^\omega.$
$(\tau(10101, 3))^\omega = (10011)^\omega$

2.5 Mots binaires balancés

2.5.1 Définitions et propriétés de base

Nous allons appeler un mot balancé (dans la définition 5) quand ses occurences de **0** et de **1** sont les plus étalées possible. Cela signifie que deux de ses sous-mots de même longueur ont une différence minimum dans le décompte de chacune de ses lettres. En fait, on va chercher à caractériser cette propriété sur l'expansion finie du générateur d'un mot périodique infini. (voir lemme 1).

Définition 5 (Mot binaire balancé)**.** *Un mot binaire fini $u \in \mathbb{B}^+$ est appelé* balancé *si $\forall v, t$, deux sous-mots de u^ω telque $|v| = |t|$ on a*
$||v|_1 - |t|_1| \leq 1.$

Exemple 4. *Alors que 1010010100 est balancé, 1010100100 ne l'est pas (parce que n'importe quel sous-mot de longueur 5 doit contenir exactement 2 occurrences de la lettre* **1** *. La première moitié 10101 en contient 3 et la second 00100 seulement 1).*

on note $\mathbb{S}_p^k$ l'ensemble des mots finis balancés de longueur p et contenant k occurrences de **1** .
De plus, on dit que $u \in \mathbb{S}_p^k$ est simple quand k et p sont relativement premier. Par extension, un mot ultimement périodique est dit *balancé* si sa partie périodique l'est. (On a choisie la lettre $\mathbb{S}$ pour *Smooth* qui signifie "lisse" ou "régulier").

Remarque 3. *Si $u \in \mathbb{S}_p^k$, $\overline{u} \in \mathbb{S}_p^{p-k}$*

On note ici que la définition 5 serait différente si "sous-mots de u^ω" avait été remplacé par "sous-mots de u". Voici un contre-exemple, $u = 10101001$ n'est pas balancé alors que $\forall v, t$, deux sous-mots de u (et non pas u^ω) avec $|v| = |t|$ on a $||v|_1 - |t|_1| \leq 1$.

Cependant, le lemme 1 nous montre que l'on obtient une définition équivalente si "u^ω" est remplacé par "u^2" ou $\forall i \in \mathbb{N}$, $\rho^i(u)$ dans la définition des mots binaires balancés (def 5).

Lemme 1. *(1), (2), et (3) sont équivalents :*
Soit $u \in \mathbb{B}^*$.
(1) u *est balancé.*
(2) $\forall v, t$, *deux sous-mots de* u^2 *tel que* $|v| = |t|$ *on a* $||v|_1 - |t|_1| \leq 1$.
(3) $\forall v, t$, *deux sous-mots de* $\rho^i(u)$, $\forall i \in \mathbb{N}$ *tel que* $|v| = |t|$ *on a* $||v|_1 - |t|_1| \leq 1$.

Démonstration. $(1) \Rightarrow (2)$ et $(1) \Rightarrow (3)$ se vérifient trivialement : si $||v|_1 - |t|_1| \leq 1$ se vérifie pour n'importe quel couple de sous-mots de u^ω avec la même longueur (voir déf 5), c'est aussi vrai pour u^2 ou $\rho^i(u)$.

(2) implique (1) : Pour $u \in \mathbb{B}^+$, tous les différents sous-mots de u^ω avec une longueur $|v| \leq |u|$ sont présents dans u^2. $\forall v$ sous-mot de u^ω avec une longueur $|v| > |u|$, on a $v = v_1.u^n.v_2$ avec $v_1.v_2$ sous-mots de u^2. De plus, $|v|_1 = |v_1.v_2|_1 + n * |u|_1$. ($n \in \mathbb{N}$).

(3) implique (2) : $\forall i \in \mathbb{N}$, si u est un sous-mot de $\rho^i(u)$, il peut être réécrit comme $v = v_1.u^n.v_2$ avec $v_1.v_2$ sous-mot de u^2. □

Nos résultats principaux du chapitre sont les deux théorèmes suivants :

Théoréme 1. $\forall k, p$, *l'ensemble* $\mathbb{S}_p^k$ *est une seule orbite.* $\forall u, v \in \mathbb{S}_p^k$, $\mathbb{S}_p^k = O(u) = O(v)$.

Démonstration. Le fait que, l'orbite complète de $u \in \mathbb{S}_p^k$ est incluse dans $\mathbb{S}_p^k$, est évident depuis le lemme 1. L'inclusion inverse nécessite des corollaires et des lemmes techniques sur la structure profonde des mots binaires balancés définis dans la suite du chapitre. La preuve est donc reportée dans la section 2.6.2. □

Le théorème suivant explique que pour n'importe quel mot balancé, il n'existe qu'une seule position telle que la transposition élémentaire appliquée à cet endroit retourne un mot balancé. De plus, le rank de la rotation équivalente à cette opération est calculable..

Théoréme 2. *Soit* k *et* $p \in \mathbb{N}$ *mutuellement premier. Il existe* $\alpha \in \mathbb{N}$,*qui peut être calculé à partir de* k *et* p *tel que :* $\forall u \in \mathbb{S}_p^k$, *il existe* $\Delta \in [1, p]$ *tel que,* $\tau(u, \Delta) \in \mathbb{S}_p^k$ *et* $\tau(u, \Delta) = \rho^{-\alpha}(u)$.

Démonstration. La section 2.7.1 est dédiée au coefficient α. Le fait que $\tau(u, \Delta) \in \mathbb{S}_p^k$ sera démontré à part dans la section 2.6.5. La deuxième partie du théorème : $\tau(u, \Delta) = \rho^{\alpha}(u)$ nécessite le lemme présenté dans la section 2.7.2. En conséquence, la preuve du théorème est reportée à la section 2.7.2. □

2.6 Propriétés structurelles

2.6.1 Construction de mots balancés

Le lemme suivant rend plus explicite le lien entre débit et nombre de **1** dans les sous-mots d'un mot balancé.

Lemme 2. *(1) est équivalent à (2).*
(1) $u \in \mathbb{B}^+$ *est* balancé
(2) Le nombre de **1** *dans tout sous-mot de* u^ω *de longueur* l *est soit* $\lfloor l * |u|_1/|u| \rfloor$ *soit* $\lceil l * |u|_1/|u| \rceil$.

Démonstration. (2)⇒ (1) : Pour $u \in \mathbb{B}^*$ si le nombre d'occurrence de la lettre **1** dans tout sous-mot de u^ω de longueur l est soit $\lfloor l * |u|_1/|u| \rfloor$ soit $\lceil l * |u|_1/|u| \rceil$, on a $\lceil l * |u|_1/|u| \rceil - \lfloor l * |u|_1/|u| \rfloor \leq 1$. Donc u est balancé.

(1)⇒ (2) : Pour u mot binaire balancé (conformément à la définition 5),

Pour $l \in \mathbb{N}$, on considère les mots : u^l, $(\rho(u))^l$, $(\rho^2(u))^l$,...,$(\rho^{|u|-1}(u))^l$. Tous ces mots ont une longueur $|u| * l$ et contiennent $|u|_1 * l$ occurrences de la lettre **1** .

Pour $i \in [0, p[$, nous supposons qu'il existe un sous-mot de $(\rho^i(u))^l$ avec une longueur l contenant seulement $\lfloor l * |u|_1/|u| \rfloor - 1$ occurrences de la lettre **1** . Comme u est balancé, les autres sous-mots de longueur l contiennent au moins $\lfloor l * |u|_1/|u| \rfloor$ occurrences de la lettre **1** . Dans ce cas, le nombre maximum d'occurrences de la lettre **1** dans $(\rho^i(u))^l$ est $|u| * \lfloor l * |u|_1/|u| \rfloor - 1 < l * |u|_1$. Pour conclure, tout sous-mot de $(\rho^i(u))^l$ contient au moins $\lfloor l * |u|_1/|u| \rfloor$ occurrences de la lettre **1** .

Pour $i \in [0, p[$, nous supposons qu'il existe un sous-mot de $(\rho^i(u))^l$ avec une longueur l contenant $\lceil l * |u|_1/|u| \rceil + 1$ occurrences de la lettre **1** . Comme u est balancé, les autres sous-mots de longueur l contiennent au moins $\lceil l * |u|_1/|u| \rceil$ occurrences de la lettre **1** . Dans ce cas, le nombre minimum d'occurrences de la lettre **1** dans t est $|u| * \lceil l * |u|_1/|u| \rceil + 1 > l * |u|_1$. Pour conclure, tout sous-mot de $(\rho^i(u))^l$ contient au plus $\lceil l * |u|_1/|u| \rceil$ occurrences de la lettre **1** .

□

On peut maintenant fournir un algorithme itératif simple de construction de mot balancé. Les lemmes 2et 3 sont dus à E. Altman, B. Gaujal and A. Hordijk [5].

Lemme 3 (Construction de mot balancé)**.** *Soit* $k, p \in \mathbb{N}$ *Avec* $0 < k \leq p$. *Soit* $u \in \mathbb{B}^\omega$, *Tel que* $\forall i \in \mathbb{N}$

$$u(i) = \lfloor i * k/p \rfloor - \lfloor (i-1) * k/p \rfloor$$

Alors u *est un mot balancé. (Tous les autres mots balancés peuvent être obtenus par rotation).*

Démonstration. Soit $u \in \mathbb{B}^\omega$ un mot binaire généré par l'algorithme du lemme 3.

Soit v un sous-mot de u commençant à l'indice x avec une longueur l. Le nombre d'occurences de la lettre **1** dans v, noté $|v|_1$ est
$|v|_1 = \sum_x^{x+l-1} \lfloor i * k/p \rfloor - \lfloor (i-1) * k/p \rfloor$. On peut simplifier l'expression en :
$|v|_1 = \lfloor (l + x - 1) * k/p \rfloor - \lfloor (x-1) * k/p \rfloor$.
Maintenant on regroupe les deux parties entières en :

$\lfloor l * k/p \rfloor \leq |v|_1 \leq \lfloor l * k/rp \rfloor + 1 \Leftrightarrow$
$\lfloor l * k/p \rfloor \leq |v|_1 \leq \lceil l * k/p \rceil$ Si p ne divise pas $l * k$.
Si p divise $l * k$, $|v|_1 = l * k/p = \lfloor l * k/p \rfloor = \lceil l * k/p \rceil$ □

Pour un nombre rationnel positif $q = k/p < 1$, L'algorithme suit la définition arithmétique classique des *mots de Christoffel* [24, 39]. (Originellement dû à Jean Bernoulli, "le père", qui définit pour la première fois cette classe de mot en 1771 [10]). La définition est aussi celle du *mot mécanique inférieur* qui, associé avec le *mot mécanique supérieur* ($w(i) = \lceil i * k/p \rceil - \lceil (i-1) * k/p \rceil$), forment les bases de la *théorie des mots mécaniques*(*mechanical word theory*) [13]. Si q est irrationnel, w devient un mot apériodique connu comme un *mot de Sturm* [38, 3].

Corollaire 1. *Soit $u \in \mathbb{S}_p^k$ un mot binaire généré par l'algorithme du lemme 3. $u = inf(\mathbb{S}_p^k)$*

Démonstration. Soit v un prefixe de u de longueur l.
$|v|_1 = \Sigma_{i=1}^{l} \lfloor i * k/p \rfloor - \lfloor (i-1) * k/p \rfloor$. Après développement et simplification, $|v|_1 = l * k/p \rfloor$. Le nombre de **1** dans v est toujours minimal conformément au lemme 2. Il n'existe pas de mot balancé plus petit, w est le plus petit. □

Mot balancé non simple Quand k et p ne sont pas relativement premier, tout mot balancé non simple peut être décomposé comme une répétition de mot balancé simple plus petit. (Leur longueur est relative au $pgcd(k, p)$). Le lemme 4 formalise ce résultat.

Lemme 4. *Soit $k, p \in \mathbb{N}$, $0 < k \leq p$ et $pgcd(k, p) = x$.*
$\forall u \in \mathbb{S}_p^k$. Il existe $v \in \mathbb{S}_{p/x}^{k/x}$ tel que $u = v^x$

Démonstration. Soit t un sous-mot de u qvec une longueur p/x. Le nombre de **1** dans t est :
$\lfloor (p * k)/(x * p) \rfloor \leq |t|_1 \leq \lceil (p * k)/(x * p) \rceil$
$k/x \leq |t|_1 \leq k/x$ avec $k/x \in \mathbb{N}$.
$|t|_1 = k/x$. u est k/x périodique de période p/x. $\forall i \in \mathbb{N}\ u(i + p/x) = u(i)$. □

2.6.2 Préservation de mots balancés

Si on applique certaines opérations à des mots balancés, l'opération nous retourne un mot de longueur ou de $rate$ différent mais toujours balancé. Nous allons ici étudier le cas de l'opération de réduction.

Motifs réguliers dans les mots balancés Un mot balancé simple est construit à partir de deux motifs différents en longueur d'exactement 1. On va appeler ces deux motifs $\pi_l(u)$ et $\pi_h(u)$ respectivement (l pour low(le plus petit) et h pour high(le plus grand)).

Lemme 5. *Si $rate(u) < 1/2$, $u = u_1.....u_k$ avec $\forall i$, $u_i \in \{\pi_l(u), \pi_h(u)\}$ Alors $|\pi_l(u)| = \lfloor p/k \rfloor$ et $|\pi_h(u)| = \lceil p/k \rceil$ et les deux motifs sont dans $0^*.1.0^*$.*
Si $rate(u) > 1/2$, $u = u_1.....u_{p-k}$ avec $\forall i$, $u_i \in \{\pi_l(u), \pi_h(u)\}$ alors $|\pi_l(u)| = \lfloor p/(p-k) \rfloor$ et $|\pi_h(u)| = \lceil p/(p-k) \rceil$ et les deux motifs sont dans $1^*.0.1^*$.

Remarque : si $rate(u) = 1/2$, u n'est pas simple. De plus, si $u = sup(\mathbb{S}_p^k)$ (resp. $u = inf(\mathbb{S}_p^k)$),alors les deux motifs $\pi_l(u)$ et $\pi_h(u)$ sont dans $1 * .0*$ (resp. $0 * .1*$).

Si on considère les motifs comme des symboles de base du mot (ex. 1.0^n et 1.0^{n+1} au lieu de 0 et 1), le mot balancé est toujours formé de deux motifs, et ce récursivement

Exemple 5.
$u = 1001010100101001010100101001010010$ *est composé de* $X = 100$ *et de* $Y = 10$, $u = XYYXYXYYXYXY$. *Maintenant, si on note* $a = XYY$ *et* $b = XY$, $u = ababb$. *Et encore, si on note* $m = ab$ *et* $n = abb$, $u = mn$.

Dans notre exemple, dans les couples (X,Y), (a,b) et (m,n), la différence de taille entre les deux éléments est de 1, de plus le plus grand des deux peut être obtenu à partir du plus petit en répétant son dernier symbole une fois.

Réduction de mots balancés

On définit une opération qui, à partir d'un mot balancé, retourne un autre mot balancé plus court.

Définition 6. *$R : \mathbb{P}_p^k \to \mathbb{P}_{p'}^k, p > p'$, est une opération qui pour un mot binaire $u \in \mathbb{P}_p^k$, supprime r bits successifs ayant une valeur* **0** *devant chaque bit ayant la valeur* **1** *. r est le nombre minimal de* **0** *devant chaque* **1** *dans tout u.*

Lemme 6. *La fonction de réduction R est une surjection.*

Démonstration. Pour un r donné, tout élément de l'ensemble des images $\mathbb{P}_{p'}^k$ a au moins un antécédent dans $\mathbb{P}_p^k$. □

Lemme 7. *Soit $u \in \mathbb{S}_p^k$. Si u est balancé, la réduction de u est aussi balancée.*

Démonstration. Cette propriété à vérifier est équivalente à : Si la réduction de u n'est pas balancée, u n'est pas balancé. On suppose que la réduction de u n'est pas balancée. Il existe deux sous-mots v et v' de la réduction de u avec une taille l tel que $|v|_1 = |v'|_1 + 2$, $v(1) = 1$, $v(l) = 1$, $v'(1) = 0$, et $v'(l) = 0$. (Cette prévision peut être vérifiée en énumérant tous les cas).

Maintenant, si on identifie par w et w', les deux sous-mots de u qui deviendront resp. v et v' après reduction de u. $|w|_1 = |v|_1 = |w'|_1 + 2 = |v'|_1 + 2$ mais la taille de w et w' est, resp., $l + r * |v|_1$ et $l + r * (|v|_1 - 2)$. w peut être réécrit comme $w = x.0^r$ comme $|x| = l + r * (|v|_1 - 1)$. et w' peut aussi être réécrit comme $0^r.w' = x'$ avec $|x'| = l + r * (|v|_1 - 1)$. Donc dans u, on a deux sous mots x et x' tel que $|x| = |x'|$ et $|x|_1 = |x'|_1 + 2$, u n'est pas balancé. □

Lemme 8. *Soit $u \in \mathbb{S}_p^k$, si $|u|_0 > |u|_1$, après réduction, $|u|_0 < |u|_1$:*

Démonstration. Tant que u est balancé, u a devant chaque **1** au moins $\lfloor p/k \rfloor - 1$ bit avec la valeur **0** car $p = \lfloor p/k \rfloor * k + reste$. $reste$ **est le reste de la division euclidienne de** p **par** k. Si on supprime $\lfloor p/k \rfloor - 1$ bits avec la valeur **0** devant chaque **1** , le mot résultant $u' \in \mathbb{S}_{k+remainder}^k$. On conclut par $|u|_0 = reste < |u|_1 = k$. □

Cette dernière preuve éclaire un point important, si on applique l'opération de réduction à un mot de l'ensemble $\mathbb{S}_p^k$, Le mot résultant appartient à $\mathbb{S}_{k+(p \mod k)}^k$. Ce résultat sera important dans la section suivante (2.6.3) pour prouver le théorème 1

2.6.3 Primalité des orbites

Nous avons maintenant tous les outils nécessaires à la preuve du théorème 1

Théorème 1. *Soit $u \in \mathbb{S}_p^k$ un mot balancé simple. L'ensemble $\mathbb{S}_p^k$ est le groupe cyclique généré par les rotations de u.*

Démonstration. On limite notre étude au cas $k < p/2$, Sinon, l'unicité dépend directement de l'unicité de $\overline{w}$.

Soit u, v deux mots balancés $\in \mathbb{S}_p^k$. Si on applique en parallèle l'opération de réduction (Déf 6) à u et v, on obtient deux mots de $\mathbb{S}_{k+(p \mod k)}^k$ (lemme 7). si on prend le conjugué de ces mots, on obtient des mots de $\mathbb{S}_{k+(p \mod k)}^{p \mod k}$ (lemme 8).

On note que le reste de la division euclidienne de $k + (p \mod k)$ par $(p \mod k)$ est le même que le reste de la division euclidienne de k par $(p \mod k)$.

Si on se réfère à l'algorithme usuel pour calculer le pgcd de deux nombres et le fait que $pgcd(k, p) = 1$, si on applique l'opération de réduction, suivie de l'opération du conjugé, un certain nombre de fois, on va finalement obtenir pour u et pour v un mot de $\mathbb{S}_{remainder}^1$ ($remainder \in \mathbb{N}$). Il existe seulement un mot et ses rotations contenant un seul **1** avec une taille $remainder$. Donc, u et v sont équivalents par rotation. □

La figure 2.1 donne l'ensemble (a) $\mathbb{S}_7^3$ et (b) $\mathbb{S}_6^2$.

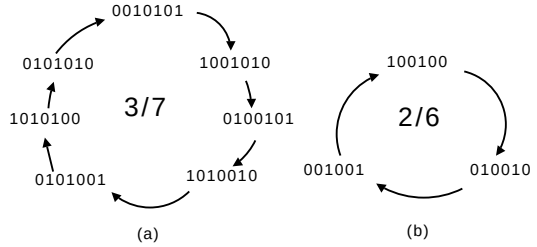


FIG. 2.1 – L'orbite d'un mot balancé u pour (a) $u \in \mathbb{S}_7^3$ and (b) $u \in \mathbb{S}_6^2$

Corollaire 2. *Si $pgcd(k, p) = 1$, $|\mathbb{S}_p^k| = p$.*

Démonstration. Soit $u \in \mathbb{S}_p^k$. On suppose qu'il existe $i \in [1, p]$, tel que $u = \rho^i(u)$. Dans ce cas, $u = \rho^{2*i}(u) = ... = \rho^{n*i}(u)$. Si $pgcd(p, i) = 1$, $u = 1^p$ or $u = 0^p$. Donc $pgcd(k, p) = k = p$ # ou $k = 0$ (la fonction $pgcd$ n'est pas définie en 0) #. Autrement $pgcd(k, i) = n$ donc $u = v^n$ et $|v|_1 = k/n$. Comme $pgcd(k, p) = 1$, k/n ce n'est pas un nombre entier. #. □

Corollaire 3. *Si $pgcd(k, p) = x$, $|\mathbb{S}_p^k| = p/x$.*

Démonstration. Suivant le lemme 4, $\forall u \in \mathbb{S}_p^k$, $\exists v \in \mathbb{S}_{p/x}^{k/x}$, tel que $u = v^x$. Comme $|\mathbb{S}_{p/x}^{k/x}| = p/x$ (théorème 1), $|\mathbb{S}_p^k| = p/x$. □

2.6.4 Ordre dans l'ensemble $\mathbb{S}_p^k$

Le lemme 9 montre comment $inf(\mathbb{S}_p^k)$ et $sup(\mathbb{S}_p^k)$ sont proches. Plus précisement, $sup(\mathbb{S}_p^k) = \tau(inf(\mathbb{S}_p^k), p)$

Lemme 9. *Soit $u \in \mathbb{B}^{p-2}$. Soit $v = inf(\mathbb{S}_p^k)$ Soit $w = sup(\mathbb{S}_p^k)$. si $v = 0.u.1$ alors $w = 1.u.0$.*

Démonstration. Soit v' un préfixe de $v = inf(\mathbb{S}_p^k)$ de longueur l. $|v'|_1 = \lfloor l * k/p \rfloor$. Soit w un préfixe de $1.u.0$ de longueur l. $|w|_1 = |v'|_1 + 1$. (excepté pour $l = p$ où $|w|_1 = |v'|_1$). Donc $|w|_1 = \lceil l * k/p \rceil$. Le nombre de **1** dans w est toujours maximum tel qu'on le voit dans le lemme 2. Donc $1.u.0$ est le $sup(\mathbb{S}_p^k)$. □

Lemme 10 (symetrie centrale entre sup et inf de $\mathbb{S}_p^k$)**.**
$sup(\mathbb{S}_p^k)(i) = inf(\mathbb{S}_p^k)(p + 1 - i)$.

Démonstration. $inf(\mathbb{S}_p^k)(p+1-i) = \lfloor (p+1-i)k/p \rfloor - \lfloor (p-i)k/p \rfloor$
$= \lfloor (1-i)k/p \rfloor - \lfloor (-i)k/p \rfloor$
$= -\lceil (i-1)k/p \rceil + \lceil i * k/p \rceil$
$= \lceil i * k/p \rceil - \lceil (i-1)k/p \rceil$
$= sup(\mathbb{S}_p^k)(i)$.(lemme 9) □

On peut aussi établir que l'ordre de précédence est complet sur $\mathbb{S}_p^k$.

Lemme 11 (Ordre de précédence sur $\mathbb{S}_p^k$)**.** *Soit $u, v \in \mathbb{S}_p^k$, $u \prec v$ ou $v \prec v$ ou $u = v$. L'ordre de précédence est total sur l'ensemble $\mathbb{S}_p^k$.*

Démonstration. Supposons qu'existe u, $v \in \mathbb{S}_p^k$ qui ne peuvent pas être comparés par la relation de précédence. Il existe i, $j \in [1, p]$, tel que $u \uparrow i < v \uparrow i$ et $u \uparrow j > v \uparrow j$. Soit w un sous-mot de u à l'indice suivant $u \uparrow i$ à l'indice juste avant $u \uparrow j$. Soit w' un sous-mot de v commençant et finissant au même indice que w dans v. On a $|w|_1 = |w'|_1 - 2$. Mais u et v sont équivalents par rotation (théorème 1) donc w et w' sont tous les deux des sous-mots de u et de v. Pour conclure, u et v ne sont pas balancés. □

2.6.5 Transposition

Lemme 12 (Transposition dans $\mathbb{S}_p^k$)**.** $\forall u \in \mathbb{S}_p^k$, $\exists! i$ *tel que* $\tau(u, i) \in \mathbb{S}_p^k$

Démonstration. Soit $v \in \mathbb{S}_p^k$, on s'intéresse à la rotation de v qui est $w = inf(\mathbb{S}_p^k)$. si on applique l'opération de transposition à n'importe quel **1** de w, on obtient un mot plus petit (ordre lexicographique) qui n'est donc pas balancé. à l'exception du dernier bit de w. si on se réfère au lemme 9, on obtient le plus grand élément de $\mathbb{S}_p^k$ □

Définition 7 (Le bit Δ)**.** *Il y a seulement un bit dans $u \in \mathbb{S}_p^k$ où l'opération de transposition peut être appliqué et retourne un mot balancé. Ce bit s'appel Δ ou $\Delta(u)$.*

Comme on peut le voir dans le lemme 12, le bit Δ d'un mot balancé est le dernier bit de $inf(\mathbb{S}_p^k)$. A partir de cette information,on peut retrouver le bit Δ dans n'importe quel élément de $\mathbb{S}_p^k$.

Corollaire 4. $\Delta(inf(\mathbb{S}_p^k) \circlearrowleft i) = i$.

Démonstration. $\Delta(inf(\mathbb{S}_p^k)) = 0$. $\Delta(inf(\mathbb{S}_p^k) \circlearrowleft i) = i$. □

Le lemme suivant indique que le bit Δ n'est jamais au même indice dans deux rotations différentes du même mot.

Corollaire 5. $\forall i, j \in [1, p]$, $i \neq j$, *soit* $u \in \mathbb{S}_p^k$, $\Delta(\rho^i(u)) \neq \Delta(\rho^j(u))$

Démonstration. $\Delta(\rho^i(u)) = \Delta(u) + i$ et $\Delta(\rho^j(u)) = \Delta(u) + j$ □

Jusqu'à maintenant, la transposition était une relation car un mot binaire a plusieurs transposés. Maintenant, on peut définir une fonction de transposition qui est toujours appliquée au bit Δ d'un mot balancé et retourne un et un seul mot balancé.

Définition 8 (τ^x)**.** *On définit l'opération* $\tau^x : \mathbb{S}_p^k \rightarrow \mathbb{S}_p^k$. *On associe à un mot balancé* $u \in \mathbb{S}_p^k \rightarrow \tau^x(u)$. *On applique* x *transpositions successives aux bits* Δ. *Dans sa notation complète, (voir déf 4 pour des détails), le second argument de la fonction* τ *est l'index du* **1** *transposé. Ici, ce paramètre est toujours l'index du bit* Δ. *On ne le fait plus apparaitre.*

Exemple 6.
$\tau^1(11010) = 10110$
$\tau^2(10101) = \tau^1(01101) = 01011$
$\tau^p(w) = w$

La fonction de tranposition est définie sur les mots balancés seulement pour le bit Δ. Mais la relation de transposition est aussi définie pour les autres bits. En revanche, si la relation de transposition est appliquée à n'importe quel autre bit valide que le bit Δ, elle ne retourne jamais de mot balancé.

Propriété 1 (Ordonner $\mathbb{S}_p^k$ en utilisant τ)**.** *Pour* $u = sup(\mathbb{S}_p^k)$, *l'ordre global lexicographique et de précédence* $\mathbb{S}_p^k$ *est :*
$\tau^{p-1}(u) < \tau^{p-2}(u) < \tau^{p-3}(u) < ... < \tau^1(u) < u$

Démonstration. Soit $w \in \mathbb{S}_p^k$, $\tau^1(w) < w$ (si w n'est pas le plus petit élément). Suivant le lemme 11, comme la différence entre w et $\tau^1(w)$ apparaît dans seulement 2 bits (une transposition), les deux mots se suivent dans l"ordre lexicographique. □

2.7 Relation entre transposition et rotation

Nous allons maintenant faire la preuve du théorème 2. Elle nécessite la définition du coefficient α tel qu'une rotation de rang $-\alpha$ d'un mot balancé retourne le même mot balancé que la fonction de transposition unitaire.

2.7.1 Le coefficient α

La définition formelle de α peut sembler contre-intuitive, elle sera justifiée plus tard.

Définition 9 (p^{-1}, k^{-1}, α)**.** *Soit k, p deux nombres entiers relativement premier, $0 < k < p$. On note p^{-1} (resp. k^{-1}) l'unique entier tel que $0 < p^{-1} < k$ et $p^{-1} * p \equiv 1 \mod k$ (resp. $0 < k^{-1} < p$ et $k^{-1} * k \equiv 1 \mod p$).*
On note α l'unique solution de $(\alpha = p^{-1} * p - 1)/k$, $0 < \alpha < p$. (α est un entier car $p^{-1} * p - 1 \equiv 0 \mod k$).

Remarque 4. *Par conséquence, et suivant le théorème de Euler-Fermat, une des solutions de l'équation $p * x - k * y = 1$ est (x,y)=(p^{-1},α). (Où k et p sont les mêmes que dans la definition 9)*

Lemme 13. *$\alpha * (p - k) \equiv 1 \mod p$. En d'autres mots, $p - k$ est l'inverse de $\alpha \mod p$.*

2.7.2 Relation entre transposition and rotation

Le lemme 14 compare un mot balancé u et sa rotation de rang α. Ces deux mots sont les mêmes à l'exception de deux bits consécutifs où la séquence 01 dans u devient 10 dans la rotation de u de rang α.

Lemme 14 ($u \circlearrowleft \alpha$)**.**
Soit $u \in \mathbb{S}_p^k$. $\exists i \in [1, p]$ tel que $\forall j \in [1, p]/\{i, i+1 \mod p\}$,
$u(j) = u \circlearrowleft \alpha(j)$,
$u(i) = 0$, $u(i+1 \mod p) = 1$
et $u \circlearrowleft \alpha(i) = 1$, $u \circlearrowleft \alpha(i+1 \mod p) = 0$

Démonstration. Soit $u \in \mathbb{S}_p^k$.

On va comparer $u(i)$ et $u(i-\alpha)$ pour $i \in [\![1, p]\!]$.

$u(i-\alpha) = \lfloor (i-\alpha) * k/p \rfloor - \lfloor (i-1-\alpha) * k/p \rfloor$.
$u(i) = \lfloor (i) * k/p \rfloor - \lfloor (i-1) * k/p \rfloor$.
on remplace α dans $u(i-\alpha)$ par sa valeur et on simplifie :
$u(i-\alpha) = \lfloor \frac{i*k+1}{p} \rfloor - \lfloor \frac{(i-1)*k+1}{p} \rfloor$.

Pour $i * k \in [\![p-1, k-1[\![\mod p$; $u(i-\alpha) = 1$.
Pour $i * k \in]\!]p-1, k-1]\!] \mod p$; $u(i) = 1$.
Pour $i * k \in [\![k-1, p-1[\![\mod p$; $u(i-\alpha) = 0$.
Pour $i * k \in]\!]k-1, p-1]\!] \mod p$; $u(i) = 0$.

Pour $i * k <> p-1$ et $i * k <> k-1$, $u(i) = u(i-\alpha)$. Pour $i * k = p-1$, $u(i-\alpha) = 1$, $u(i) = 0$ et pour $i * k = k-1$, $u(i-\alpha) = 0$, $u(i) = 1$. De plus, si $i * k = p-1$, donc $(i+1) * k = k-1$, les deux seuls bits ayant une différence sont successifs : $\rho(u, \alpha) > u$. □

Voici le second résultat majeur du chapitre. Dans l'ensemble des mots balancés, l'opération de transposition est équivalente à l'opération de rotation de rang $-\alpha$.

Théorème 2.
Soit $u \in \mathbb{S}_p^k$, on a $\tau^1(u) = \rho(u, -\alpha)$

Démonstration. Suivant le lemme 14, la différence entre $u \in \mathbb{S}_p^k$ et $\rho(u, -\alpha)$ intervient dans seulement deux bits successifs tels que $u = u_1.10.u_2$ devient $\rho(u, -\alpha) = u_1.01.u_2$. (où $u = 0.u_2.u_1.1$ devient $\rho(u, -\alpha) = 1.u_2.u_1.0$) Suivant le lemme 12, cette opération est équivalente à la fonction $\tau^1(u)$. □

2.8 Création de mots balancés utilisant le coefficient α

Cette section présente un algorithme qui génère un mot balancé en répartissant les lettres **1** tous les α bits modulo p dans le mot. La figure 2.2 l'illustre pour $(k, p) = (3, 5) \Rightarrow \alpha = 3$.

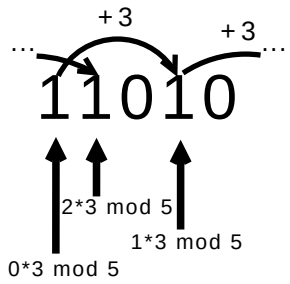


FIG. 2.2 – Algorithme de création de mot balancé utilisant α pour $(k, p) = (3, 5) \Rightarrow \alpha = 3$.

Lemme 15 (Algorithme de création de mot balancé utilisant α)**.** *Soit $m \in \mathbb{B}^*$, $|m| = p$ et $|m|_1 = k$.*
$\forall i \in [\![0, k-1]\!]$, $m(i * \alpha \mod p) = 1$ et 0 pour $i \in [\![k, p[\![$
Donc m est balancé.

Démonstration. Soit $k, p \in \mathbb{N}$, $0 < k \leq p$ et $pgcd(k, p) = 1$. on a le $pgcd(p, \alpha) = 1$ car α a une inverse modulo p.

la fonction qui associe $(i * \alpha) \mod p$ à i est une bijection de l'ensemble $[\![0, p-1]\!]$ vers $[\![0, p-1]\!]$.

NB : $-k$ est l'inverse de $\alpha \mod p$, donc $-k * \alpha = 1 \mod p$

Soit $w \in \mathbb{B}^*$ un mot construit suivant l'algorithme 15. Soit s un sous-mot de w commençant à l'indice $n+1$ ($n \in \mathbb{N}$) de longueur l. On veut calculer le nombre de **1** dans s.

On associe au i^{th} bit de w, $n \in \mathbb{N}$ tel que $n * \alpha = i$. n est appelé de rang de $w(i)$. Si le rang $n \in [0, k-1]$, $w(i) = 1$, et 0 sinon.

Le rang du $i - 1^{th}$ bit est n' tel que $n' * \alpha = i - 1 \Leftrightarrow n' * \alpha = n * \alpha + k * \alpha \Leftrightarrow n' = n + k$.
$n + x * k$ est le rang du $i - x^{th}$ bit de w
$\frac{n+x*k}{p} - \lfloor \frac{n+x*k}{p} \rfloor$ est le rang du $i - x^{th}$ bit de w raporté entre $[\![0, 1]\!]$

$k - p(\frac{n+x*k}{p} - \lfloor\frac{n+x*k}{p}\rfloor)$ est positif si $(n + x * k) \mod p < k$ et négatif si $(n + x * k) \mod p \geq k$
$\lceil\frac{k-p(\frac{n+x*k}{p}-\lfloor\frac{n+x*k}{p}\rfloor)}{p}\rceil$ est égale à $w(i - x)$.

On peut calculer le nombre de **1** dans s.
$|s|_1 = \Sigma_{x=l}^{1}\lceil\frac{k-p(\frac{n+x*k}{p}-\lfloor\frac{n+x*k}{p}\rfloor)}{p}\rceil$
Ceci peut être réécrit en :
$|s|_1 = \Sigma_{x=l}^{1}\lceil\frac{-(n+k*(x-1))}{p}\rceil + \lfloor\frac{n+x*k}{p}\rfloor$
La fin de la preuve est coupée en deux, chaque partie donne une borne à $|s|_1$.

Borne inférieure $|s|_1 = \Sigma_{x=l}^{1}\lfloor\frac{n+x*k}{p}\rfloor - \lfloor\frac{n+k(x-1)}{p}\rfloor$
Après développement et simplification, on obtient :
$|s|_1 = \lfloor\frac{n+l*k}{p}\rfloor - \lfloor\frac{n}{p}\rfloor$
so $\lfloor\frac{l*k}{p}\rfloor \leq |s|_1 \leq \lfloor\frac{l*k}{p}\rfloor + 1$

borne supérieure $|s|_1 = \Sigma_{x=l}^{1}\lceil\frac{-(n+k*(x-1))}{p}\rceil - \lceil\frac{-(n+x*k)}{p}\rceil$
Après développement et simplification, on obtient :
$|s|_1 = \lceil\frac{-n}{p}\rceil - \lceil\frac{-(n+l*k)}{p}\rceil$
so $\lceil\frac{l*k)}{p}\rceil - 1 \leq |s|_1 \leq \lceil\frac{(l*k)}{p}\rceil$

conclusion $\lfloor\frac{l*k}{p}\rfloor \leq |s|_1 \leq \lceil\frac{(l*k)}{p}\rceil$ L'algorithme est correct. □

Corollaire 6. *Soit $u \in \mathbb{B}_p^k$, $\exists! i$ tel que $\tau(\rho^\alpha u, i) = u$. alors $u \in \mathbb{S}_p^k$.*

Démonstration. $\exists! i$ tel que $\forall j \notin \{i, i+1\}$, $u(j) = \rho^\alpha u(j)$ et $u(i) = 0$, $u(i+1) = 1$, $\rho^\alpha u(i) = 1$, $\rho^\alpha u(i+1) = 0$.

Comme $u(j) = \rho^\alpha u(j)$, $u(j) = u(j-\alpha)$. On a donc les k **1** de u qui sont répartis tout les α bits modulo p. Conformément au lemme 15, u est balancé.

Si u est balancé, $i = \Delta(\rho^\alpha u)$ et $\tau(\rho^\alpha u) = \rho^{\alpha-\alpha} u = \rho^0 u = u$ (théorème 2). □

Cet algorithme génére un mot balancé pour une complexité linéaire en fonction de la taille du mot.

2.9 Conclusion

Dans ce chapitre, on définit formellement la notion de mot balancé (originellement appelé *smooth*) dans nos publications [46, 45]). Puis nous donnons la taille de l'ensemble contenant les mots balancés et nous définissons deux opérations équivalentes : la rotation et la transposition.

A partir de ces résultats, on va pouvoir regarder leur application dans le domaine de l'ordonnancement statique. Le chapitre 6 explique comment construire un régime permanent dont l'ordonnancement de tous les nœuds du graphe est un mot balancé.

De plus, on va pouvoir prouver que la quantité d'éléments de stockage nécessaire à l'ordonnancement balancé d'un graphe est minimal comparé au même graphe mais dont les nœuds de calculs ne sont pas ordonnancés par des mots balancés.

Chapitre 3

Réseaux de processus concurrents

Dans tout notre travail, nous considérons des variations autour d'un modèle générique de Réseaux de Processus (Process Network ou PN dans la suite). Dans ce modèle les blocs de calculs (nœuds de calculs) sont reliés par des canaux directionnels point-à-point. Un calcul s'effectue en consommant une donnée sur chaque canal d'entrée, pour en produire une sur chaque canaux de sortie. Le modèle de base sera donc assimilé à des *Marked Graph*, bien connu dans la littérature [26]. Les différentes variations proviennent d'hypothèses faites sur la capacité de stockage ("bufférisassions") et la nature des canaux, ainsi que sur les sémantiques d'exécutions (Synchrone, au plus tôt, asynchrone). De fait, notre travail consiste principalement à étudier comment le passage des modèles asynchrones à des modèles plus synchrones peut se faire harmonieusement en optimisant les ressources des connexions.

Il s'agit donc d'ordonnancement sous contraintes. Il existe des résultats classiques d'ordonnancement dans ce domaine pour les cas de système clos et connexes, qui admettent des solutions statiques régulières. Néanmoins ceci laisse encore une grande marge de manœuvre dans l'ensemble des solutions, dans lesquelles nous aurons à construire nos optimisations (ces travaux antérieurs font à priori l'hypothèse de capacité infinie de stockage des canaux).

Nous introduisons notre modèle et ses variations dans ce chapitre, ainsi que les rappels sur l'ordonnancement statique et ses résultats connus, et enfin des définitions (sur les cycles et leur débit potentiel, etc) qui nous seront utiles dans les chapitres ultérieurs.

3.1 *Marked Event Graphs*

Cette section présente notre modèle de base. A la source de notre modèle se trouvent les réseaux de Petri qui ont été inventés par Carl Adam Petri en 1962 dans [48].

Définition 10 (réseau de processus)**.** *Un réseau de processus est un graphe G composé de nœuds de calculs reliés entre eux par des canaux point à point (les arcs). Formellement, un réseau de processus est une structure $G =< N, C, M >$ où*

– *N est l'ensemble des nœuds du système.*

– $Chan \subset \{N * N\}$ *est l'ensemble des canaux du système.*
– M *le marquage de* G *est une fonction de* $Chan \rightarrow \mathbb{N}$ *qui à chaque canal lui associe le nombre de jetons qu'il contient.*

En l'absence d'ambiguïté, le réseau de processus est appelé réseau, graphe ou système. L'état d'un graphe est égal à son marquage. Dans la suite du chapitre, nous parlerons sémantique dynamique du graphe. Cette sémantique dynamique consiste à faire évoluer uniquement le marquage du graphe. L'évolution dynamique du graphe se fait au travers de l'exécution de ses nœuds. Plusieurs modèles (ou politiques) d'exécution existent (nous en reparlerons dans la suite) mais une règle reste toujours vrai : Pour s'exécuter, un nœud consomme un jeton sur chacune de ses entrées et produit un jeton sur chacune de ses sorties.

La figure 3.1 représente un réseau de processus.

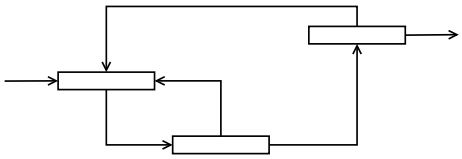


FIG. 3.1 – Exemple d'un réseau de processus.

Notation 3. *Soit* G *un graphe.* n *un nœud du graphe et* c *un canal du graphe.*

– $\bullet n$ *est l'ensemble des arcs entrants dans le nœud* n*.*
– $n\bullet$ *est l'ensemble des arcs sortants du nœud* n*.*
– $\bullet c$ *est le nœud à la base de* c*.*
– $c\bullet$ *est le nœud à la tête de* c*.*

$M(c)$ *donne le marquage du canal* c *(le nombre de jeton sur* c*).*

Remarque 5. *Le modèle présenté dans la définition 10 est équivalent au* Marked Graph*/*Event Graph *présenté par Even, Pnueli et al. dans [26]. Dans les* Marked graphs*, l'emplacement de stockage du canal est explicitement représenté comme une place. Les jetons sont dedans. La figure 3.2 représente un* Marked graph *pour un marquage quelconque.*

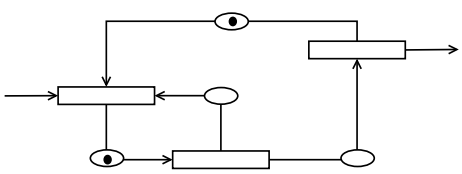


FIG. 3.2 – Exemple d'un *Marked Graph*.

Définition 11 (cycle critique et débit)**.** *Soit* G *un graphe, un cycle élémentaire dans* G *est un chemin d'un nœud vers lui même qui ne repasse pas deux fois par le même nœud.*

Au travers de toute exécution du graphe (c'est à dire l'exécution de ses nœuds de calculs), les marquages successifs préservent le nombre de jetons dans chaque cycle du graphe, qui est donc un invariant du système. On peut donc définir analytiquement le débit d'un cycle comme le ratio entre le nombre de jetons contenus dans le cycle et sa longueur. La longueur d'un cycle est définie par son nombre de nœuds ou de canaux.

Le débit du graphe G est défini comme égal au débit du cycle le plus lent, tous les cycles ayant ce débit sont appelés critiques.

On dit d'un nœud ou d'un canal qu'il est critique si un cycle critique passe par lui.

Définition 12 (Partie fortement connexe)**.** *Une partie fortement connexe d'un graphe est un sous ensemble d'un graphe tel qu'il existe un chemin de n'importe quel point vers n'importe quel autre point. Un graphe peut contenir plusieurs parties fortement connexes disjointes.*

Une partie fortement connexe critique est une partie fortement connexe telle que tous ses nœuds sont critiques.

La figure 3.3 présente un graphe contenant deux parties fortement connexes et une critique.

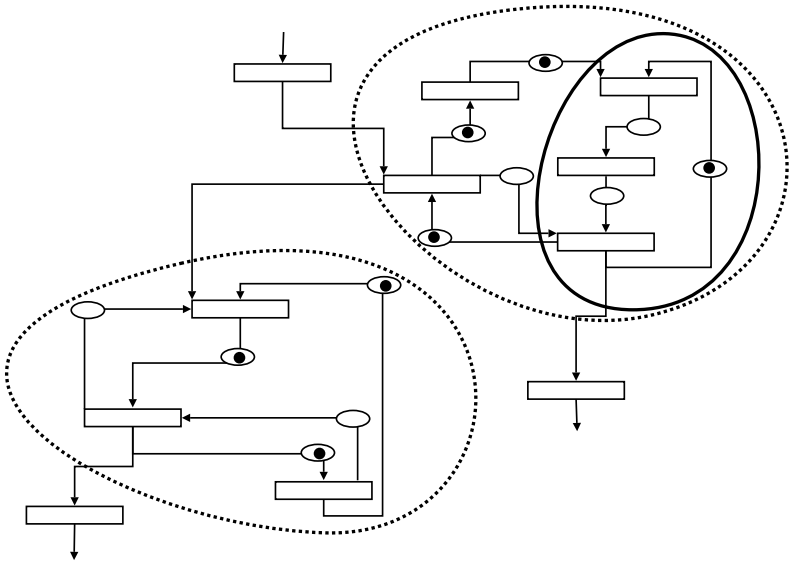


FIG. 3.3 – Les sous-graphes entourés de pointillés sont des parties fortement connexes. Le sous-graphe entouré d'une ligne pleine est une partie fortement connexe critique.

La dynamique des *Marked Graph* admet généralement des systèmes ouverts modélisés par des nœuds de calculs sans canaux entrant ou sortant. Nous nous restreindrons à des systèmes clos mais dans lesquels certains nœuds seront distingués comme nœuds de sortie ou d'entrée. L'hypothèse faite ici sera que les entrées sont exactement disponibles aux instants nécessaires à l'exécution. Ces instants sont définis par l'ordonnancement du système qui explicite donc les pré conditions sur les entrées et des post-conditions sur les sorties.

Hypothèse 1 (Graphe clos)**.** *On va supposer dans la suite du manuscrit que le graphe est considéré comme clos.*

Nos jetons abstraient des données. On considèrera qu'à l'initialisation une seule donnée au plus est présente dans chaque canal.

Hypothèse 2 (Marquage initial)**.** *Le marquage initial d'un graphe est tel que chaque canal contient soit* **0** *soit* **1** *jeton.*

Les définitions suivantes présentent les notions de graphe pré-égalisé et égalisé ou encore respectivement $\mathbb{N}$-égalisé et $\mathbb{Q}$-égalisé. Le lien avec les ensembles $\mathbb{N}$ et $\mathbb{Q}$ peut paraître flou pour le moment. Il apparaîtra plus clairement aux chapitres 5 puis 6 : on va d'abord ajouter des latences **entières** à un graphe jusqu'à obtenir de lui qu'il devienne $\mathbb{N}$-égalisé. Puis, on va lui rajouter des latences **rationnelles** (permettant de ne pas retarder tous les jetons mais seulement ceux qui en ont besoin) jusqu'à obtenir un graphe $\mathbb{Q}$-égalisé.

Définition 13 (Graphe $\mathbb{N}$-égalisé ou pré-égalisé)**.** *Soit G un graphe de débit k/p. On dit que G est $\mathbb{N}$-égalisé si pour tout canal appartenant à la partie fortement connexe de G, il existe au moins un cycle de débit j/l tel que $j/l \geq k/p > j/(l+1)$.*

Définition 14 (Graphe $\mathbb{Q}$-égalisé ou égalisé)**.** *Soit G un Graphe de débit k/p. On dit que G est $\mathbb{Q}$-égalisé si tous les cycles de G ont ce débit k/p.*

Dans le chapitre 6, nous allons construire le marquage d'un graphe afin que sa dynamique corresponde à des critères d'optimisation que nous expliciterons plus tard. Ensuite nous allons devoir comparer ce marquage construit analytiquement avec le marquage initial du graphe. Le but est de montrer que le marquage construit est accessible à partir du marquage initial. Pour cela nous devons définir la notion de marquages équipollents qui est une condition nécessaire à l'établisement de ce résultat.

Définition 15 (Marquages équipollents)**.** *Soit G un graphe. Soit M et M' deux marquages de G. Les marquages M et M' sont dit équipollents si quelque soit le cycle C considéré, la somme des jetons sur C est la même suivant M et M'.*

3.2 Sémantique dynamique

3.2.1 Règle d'exécution des nœuds de calculs

Nous présentons ici les variations possibles autour des différentes politiques d'exécutions qui peuvent être associées à notre modèle de graphe. On définit d'abord la notion d'exécution simultanée de plusieurs nœuds de calculs au cours d'un même pas global d'exécution (qui correspond à un instant global).

Définition 16 (Nœud exécutable)**.** *Un nœud n est dit exécutable si tous les canaux de $\bullet n$ contiennent au moins un jeton.*

Soit G un graphe et M son marquage. L'ensemble des nœuds exécutables du graphe G au marquage M est noté F_M (F pour firable node).

Définition 17 (Modèle d'exécution de graphe)**.** *Soit G un graphe et M son marquage initial. On définit un pas d'exécution ou instant d'exécution comme le passage du marquage M au marquage M_i noté $M \xrightarrow{N_i} M_i$ tel que $N_i \subseteq F_M$. Pour chaque nœud de N_i, on enlève un jeton dans chacun de ses arcs entrants et on ajoute un jeton dans chacun de ses arcs sortants. $\forall c$ un canal de G, $M_i(c) = M(c) + \delta(\bullet c) - \delta(c\bullet)$. ($\delta(n) = 1$ ssi $n \in N_i$, et 0 sinon).*

L'exécution d'un graphe noté $Exec$ est la succession finie ou infinie des pas/instants d'exécutions : $Exec = M \xrightarrow{N_1} M_1 \xrightarrow{N_2} M_2 \xrightarrow{N_3} ... \xrightarrow{N_i} M_i \xrightarrow{N_{i+1}} ...$ *où* $N_i \subseteq F_{M_{i-1}}$.

La définition du modèle d'exécution est telle que même si un nœud a plusieurs jetons dans chacun de ses canaux d'entrées, il ne peut s'exécuter qu'une seule fois dans un pas d'exécution.

Remarque 6. *Soit n un nœud exécutable. Tant que n n'est pas exécuté, $\forall c \in \bullet n$, $M(c) > 0$. Un nœud exécutable le reste au moins jusqu'à ce qu'il soit exécuté.*

Les différentes possibilités d'exécutions équitables consistent donc juste en des ordonnancements différents des mêmes calculs. Exemple : si les nœuds n_1 et n_2 sont exécutables, on a trois exécutions possibles équivalentes :

- $M \xrightarrow{n_1} M_1 \xrightarrow{n_2} M_{final}$
- $M \xrightarrow{n_2} M_2 \xrightarrow{n_1} M_{final}$
- $M \xrightarrow{\{n_1,n_2\}} M_{final}$

Remarque 7. *Soit G un graphe et M son marquage. A partir de ce modèle d'exécution, on peut retrouver différentes politiques d'exécutions existantes :*

- *Si pour chaque pas d'exécution, $|N| = 1$ (on exécute un seul nœud à la fois) on retrouve le modèle d'exécution asynchrone historique des réseaux de Petri.*
- *Si pour tout canal c, $M(c) = 1$. On a $F_M = T$ (tous les nœuds sont exécutables à chaque instant). De plus, si pour chaque pas d'exécution, $N = F_M = T$, le marquage M' résultant est égal à M. On retrouve ici le modèle d'exécution des circuits synchrones dans lequel tous les registres sont toujours pleins et tous les nœuds s'exécutent à tous les instants.*
- *Si pour chaque pas d'exécution, $N = F_m$ (tous les nœuds exécutables sont exécutés), on retrouve le modèle d'exécution dit "au plus tôt" (ASAP pour "As Soon As Possible").*

Hypothèse 3. *Sauf précision explicite dans la suite du manuscrit, la politique d'exécution sera le modèle "au plus tôt".*

Dans [25], les auteurs proposent de représenter l'ordonnancement des nœuds du graphe comme des mots binaires où **1** représente l'activité et **0** l'inactivité.

Définition 18 (Ordonnancement)**.** *Soit G un graphe de marquage M_0 suivant une exécution $Exec = M_0 \xrightarrow{N_1} M_1 \xrightarrow{N_2} M_2 \xrightarrow{N_3} ... \xrightarrow{N_i} M_i$.... L'ordonnancement d'un nœud n noté O_n est un mot binaire tel que $O_n(i) = 1$ si $n \in N_i$ et 0 sinon.*

L'ordonnancement d'un graphe est un ensemble de taille $|N|$ de mots binaires tel que chacun d'eux est l'ordonnancement d'un nœud du graphe.

Définition 19 (Ordonnancement balancé)**.** *Soit O l'ordonnancement d'un graphe. O est appelé ordonnancement balancé s'il existe un mot binaire périodique balancé tel que tous les ordonnancement des nœuds du graphe sont dans son orbite.*

Définition 20 (Marquage balancé)**.** *Soit G un graphe, O son ordonnancement, et M son marquage. Si O est un ordonnancement balancé, alors M est appelé marquage balancé.*

Définition 21 (Périodicité, k-périodicité)**.** *L'ordonnancement d'un nœud est dit k-périodique s'il existe un schéma d'exécution fini le décrivant. $(01101)^\omega$ est un ordonnancement valide pour un nœud 3-périodique de période 5.*

Le k qui est le nombre d'instants d'activation par période est appelé périodicité. Il est équivalent à la "cyclicité" d'une matrice dans l'algèbre (max, plus). Le p est la période : le nombre d'instants globales d'exécution entre deux états identiques.

Le comportement d'un graphe est dit périodique (k-périodique) si le comportement de chacun de ces nœuds l'est.

Formellement, Soit $Exec$ l'exécution d'un graphe. Le comportement d'un graphe est dit k-périodique de période p si l'exécution du graphe est de la forme : $Exec = M \xrightarrow{N_1} ...M_i \xrightarrow{N_{i+1}} M_{i+1}.... \xrightarrow{N_{i+p}} M_{i+p} = M_i$

Après une phase d'initialisation, de l'état M à l'état M_{i-1}, le graphe entre en régime permanent. Il va parcourir indéfiniment les états M_i à M_{i+p-1}. Parmi l'ensemble des nœuds exécuté pendant une période de longeur p du régime permanent ($\cup_{j=i}^{i+p-1} N_j$) chacun des nœuds du graphe apparait k fois.

Il existe des travaux sur l'ordonnancement périodique de réseau de processus [47] qui permettent de calculer le temps moyen entre deux exécutions d'un processus. Malheureusement, ce temps est souvent un nombre rationnel qui a un intérêt certain dans l'ordonnancement temps réel de processus sur des unités de calculs car la base de temps est un multiple de la seconde divisible par nature. Concernant notre domaine d'application : l'ordonnancement de système sur puce, la base de temps est le cycle d'horloge et est indivisible par définition. La solution d'ordonnancement périodique ne peut donc pas être retenue.

En revanche, les travaux sur l'ordonnancement k-périodique [14, 8] correspondent parfaitement à notre domaine d'application car ils prennent l'hypothèse que la base de temps est indivisible. Ainsi la période d'exécution ne contient plus un seul instant d'activité suivi d'une durée d'inactivité mais est composée d'une succession irrégulière d'instants d'activités et d'inactivités formant un motif qui, lui, se répète régulièrement et indéfiniment.

Propriété 2 (périodicité d'un graphe)**.** *La période p de fonctionnement d'un graphe et donc de tous ses nœuds est égale au ppcm (plus petit commun multiple) des périodes de chaque partie fortement connexe critique du graphe. La période de fonctionnement d'une partie fortement connexe critique est égale au pgcd (plus grand commun diviseur) des longueurs de chaque cycle (critique). La périodicité k de fonctionnement d'un graphe peut être calculée en appliquant la même formule sur les nombres de jetons de chaque cycle critique au lieu des longueurs. Elle peut aussi être obtenue en multipliant p par le débit du graphe. ($k = p * debit_graphe$)*

Dans [32] et dans [7], Les auteurs donnent une borne maximum à la longueur du régime transitoire de l'exécution "au plus tôt" d'un *Marked Graph*.

Remarque 8. *Nous avons pris l'hypothèse plus haut que pour tout canal c d'un graphe, $M(c) \leq 1$. L'implication de cette hypothèse est que le débit de n'importe quel cycle est inférieur à 1. On peut atteindre un rythme d'exécution égal au débit du graphe k/p. Si le débit du graphe était un nombre supérieur à 1 (le cycle le plus lent contient plus de jetons que de canaux), ce débit ne serait pas atteignable dynamiquement car un nœud ne peut s'exécuter qu'une seule fois dans un pas d'exécution. Le rythme d'exécution du graphe serait borné par 1.*

Théoréme 3. *Soit G un graphe, soit M et M' deux marquages équipollents (def. 15). M et M' sont mutuellement accessibles par exécution asynchrone.*

Démonstration. Ce résultat est du à J. Desel et J. Esparza dans [31] □

3.2.2 Propriétés

Définition 22 (Liveness)**.** *Un graphe est dit vivant si n'importe quel nœud peut être exécuté une infinité de fois.*

Définition 23 (Safety)**.** *Un graphe est dit sûr si chacune de ses canaux contient au maximum un nombre fini de jetons. Un canal est dit k-sûr si k est une borne supérieure au nombre de jetons maximum qui transite simultanément dans le canal. Un graphe est dit k-sûr si tous ses canaux le sont.*

Propriété 3 (Liveness)**.** *Pour qu'un graphe soit vivant, il faut que tous ses cycles contiennent au moins un jeton.*

Propriété 4 (Safety)**.** *Dans un graphe, le nombre maximum de jetons dans un canal est borné par le nombre de jetons dans le cycle passant par ce canal. Si plusieurs cycles passent par ce canal, il est borné par le minimum. (Le nombre de jetons par cycles est invariant.)*

3.2.3 Retard

Au cours de l'exécution "au plus tôt" d'un graphe, il peut arriver qu'un nœud ne soit pas exécutable parce que seulement une partie de ses canaux d'entrées contient un jeton. Lors de ce pas d'exécution, ses jetons vont rester là où ils sont. On dit que ces jetons ont été retardés. Une vision plus globale du système permet de voir ces retards comme le moyen pour le graphe de resynchroniser ses cycles. Un cycle non critique aura tendance à prendre de l'avance par rapport aux cycles critiques (il est plus rapide) mais au final, le débit du graphe reste limité par celui de ses cycles critiques ; il faut donc freiner les cycles non critiques. Le retard est le moyen naturel pour un cycle de se ralentir.

Définition 24 (Retarder)**.** *Soit G un graphe, M son marquage et c un canal de G. Dans l'état M, c est retardé ssi $M(c) > 0$ et $c\bullet \notin F_M$ (c contient un jeton mais le nœud en sortie de c n'est pas exécutable) ou que $M(c) > 1$ (Au mieux, un seul jeton franchira $c\bullet$, les autres seront retardés).*

Définition 25. *Soit G un graphe de marquage M. Soit c un canal de G On définit la fonction $Retard$ de $Chan * M \rightarrow \mathbb{N}$ avec $Retard(c, M) = M(c)$ si $c\bullet \notin F_M$ et $Retard(c, M) = M(c) - 1$ si $c\bullet \in F_M$. (Si $c\bullet \in F_M$, on a $M(c) \geq 1 \leftrightarrow M(c) - 1 \geq 0$).*

Soit C un cycle, la somme des retards subis par les l canaux de C dans l'état M est noté $Retard(C, M) = \Sigma_{i=1}^{l} Retard(c_i, M)$ et est appelé "retards du cycle" C (c_i est un canal de C).

Propriété 5 (Somme des retards sur un cycle durant une période)**.** *Soit G un graphe k-périodique de période p. Soit C un cycle appartenant à G de débit $j/l > k/p$. Soit $Exec$ l'exécution "au plus tôt" de G. $Exec$ est donc une exécution k-périodique de période p car G l'est.*

La somme des retards subie par les canaux appartenant au cycle C durant une période (p) d'exécution du régime permanent de l'exécution de G est noté r et est égale à $r = j * p - l * k$.

Démonstration. Si les nœuds du cycle C s'exécutent indépendamment du reste du graphe, à chaque instant j nœuds sont exécutables. Sur p instant, on a $p * j$ nœuds qui se sont exécutés.

En régime permanent, durant une période, chaque nœud de G s'exécute k fois. On a $l * k$ nœuds de C qui se sont exécutés.

La différence $j * p - l * k$ entre le nombre maximum d'exécution dans le cycle C ($p * j$) et le nombre d'exécution réelle ($l * k$) dans ce même cycle est le nombre de retard r subi par les canaux de C durant une période. □

3.3 Modèle avec latence

Dans le modèle de base, les jetons sont consommés au début de l'instant et produit à la fin du même instant. Ils sont disponibles pour de nouveaux calculs au prochain instant global. Le modèle se comporte donc comme si les calculs étaient instantanés et les transports dans les canaux avaient une durée unitaire. On introduit maintenant un modèle où les nœuds de calculs comme les canaux de communication peuvent avoir des latences entières distinctes.

Définition 26 (graphe pondéré)**.** *C'est un graphe dans lequel on associe aux nœuds et aux canaux une latence. Celle-ci représentant le temps que met un jeton à parcourir l'élément. Elle s'exprime en nombre d'instant d'exécution. Formellement, un graphe pondéré est une structure $< G, L_c, L_t >$ où*

- *G est un réseau de processus*
- *L_c est un vecteur qui associe à chaque nœud sa latence de calcul.*
- *L_t est un vecteur qui associe à chaque canal sa latence de transport.*

Dans un soucis de lisibilité, seulement les latences de communication différentes de 1 et les latences de calcul différentes de 0 apparaissent sur le graphe. 1 et 0 sont les valeurs par défaut resp. des latences de communication et de calculs.

La figure 3.4 représente un graphe pondéré.

Dans un premier temps, nous allons traiter le cas où les latences de canaux sont au moins de 1.

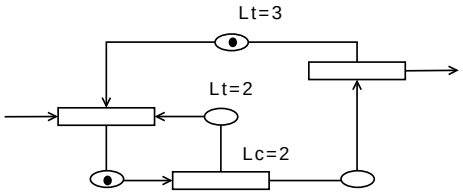


FIG. 3.4 – Exemple d'un graphe pondéré.

Définition 27 (Expansion d'un graphe pondéré en un non pondéré)**.** *Soit G un graphe pondéré. Pour tout canal ayant une latence de transport de n, on la remplace par une succession de n canaux "classiques" (ayant une latence 1) intercalés de $n-1$ nœuds instantanés (ayant une latence 0). pour tout nœud ayant une latence de calcul de m, on la remplace par $m+1$ nœuds instantanés intercalés de m canaux classiques. La latence de calcul se transforme, en un sens, en une latence de transport dans l'unité de calculs elle-même. Le graphe résultant est un graphe non pondéré ayant le même comportement temporel que le précédent.*

La figure 3.5 illustre les transformations permettant de passer d'un graphe avec latence à un graphe sans latence.

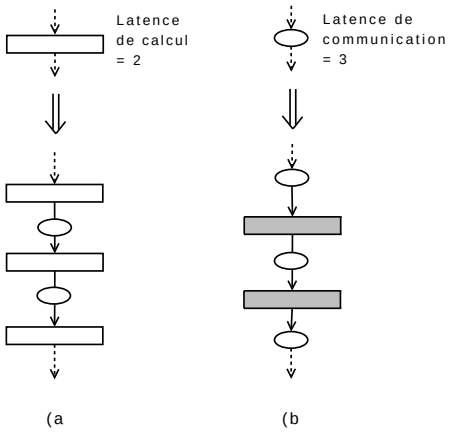


FIG. 3.5 – (a) Transformation d'un nœud avec une latence de calcul de 2 en une succession de nœuds instantanés. (b) Transformation d'un canal avec une latence de communication de 3 en une succession de canaux "classiques".

Remarque 9. *Le modèle d'exécution d'un graphe avec latence est difficilement représentable à cause justement des latences : quand un jeton arrive dans un canal ou est consommé par un nœud, il n'est pas utilisable ou disponible à l'instant suivant. De plus, la notion de latence ne prend du sens que pour une politique d'exécution "au plus tôt".*

L'expansion de la définition 27 permet de représenter une latence comme le franchissement d'un nœud explicite (appelé nœud de transport en opposition aux nœuds de calculs) et permet de visualiser la progression d'un jeton à l'intérieur d'un nœud ou d'un arc avec latence. Le modèle d'exécution d'un graphe avec latence est le modèle d'exécution de la définition 17 appliqué à l'expansion du graphe.

Considérons maintenant le cas plus complexe où on autorise des latences nulles pour les canaux. Nous considérons dans notre étude que tous les nœuds et canaux à latence respectivement supérieurs à 0 et 1 ont subi l'expansion de la def. 27. Si une donnée arrive sur un canal à latence nulle, elle peut potentiellement être utilisée par le nœud en sortie du canal dans le même instant. La politique d'exécution définie dans la def. 17 ne nous permet pas de prendre en compte cette possiblilité, car le nœud en sortie du canal à latence nulle ne sera exécuté, suivant cette politique, qu'à l'instant suivant.

Nous devons raffiner cette sémantique pour autoriser l'exécution en cascade de nœuds dépendants de canaux à latence instantanée avant ou après l'exécution d'autres nœuds dépendants de canaux à latence 1.

Définition 28 (Modèle d'exécution dans un graphe à canal à latence nulle)**.** *Soit G un graphe contenant des canaux à latence nulle et M_0 son marquage. F_M est l'ensemble des nœuds exécutables au marquage M. F_M^0 est l'ensemble des nœuds qui nécessite un jeton uniquement sur les canaux d'entrée à latence nulle pour devenir exécutable pour le marquage M. On définit le pas d'exécution dans un graphe contenant des canaux à latence nulle, du marquage M_0 au marquage M_n, comme une succession de pas d'exécution classique $M_0 \xrightarrow{N_1} M_1 ... \xrightarrow{N_i} M_i ... \xrightarrow{N_n} M_n$.*

Avec $N_i \subseteq F_{M_0} \cup (F^o_{M_0} \cap F_{M_i})$.

F_{M_0} est l'ensemble des nœuds exécutables au marquage M_0. $(F_{M_i} \cap F^o_{M_0})$ représente l'ensemble des nœuds qui dépendent d'un canal à latence nulle pour devenir exécutable et qui le sont dans l'état M_i.

De l'état M_0 ł'état M_n, chaque nœud ne peut s'exécuter au maximum qu'une fois.

A partir du moment où le graphe autorise des canaux à latence nulle, il se pose le problème des cycles à latence nulle (formé uniquement d'arc et de nœud à latence nulle). Un pas d'exécution tel que définit dans la def 28 est susceptible d'exécuter une infinité de fois les nœuds de ce cycle. On peut faire le parallèle entre ce problème et le problème des cycles combinatoires dans les circuits électroniques. On va supposer que les circuits que l'on abstrait en *Marked graph* n'ont pas de cycle combinatoire. On peut donc limiter notre étude au graphe n'ayant pas de cycle à latence nulle.

3.4 Modèle avec capacité de stockage borné

Dans le modèle de base, les canaux contiennent une place à capacité illimitée pour le stockage des jetons. Nous allons maintenant vouloir limiter la capacité de ces canaux. Cela est différent de la notion de k-sûreté. Ici, la capacité modifie le comportement du système afin de se faire respecter.

Définition 29 (graphe à capacité)**.** *C'est un graphe* **sans latence** *dans lequel on associe à chaque canal le nombre maximum de jetons qu'elle peut contenir. Formellement, un graphe à capacité est une structure $< G, C >$ où*

- *G est un réseau de processus.*
- *$Capa$ est un vecteur qui associe à chaque canal sa capacité : le nombre maximum de jetons qu'il peut contenir.*

Usuellement, seules les capacités finies apparaissent sur le graphe. Par défaut, une capacité est infinie.

La figure 3.6 représente un graphe à capacité.

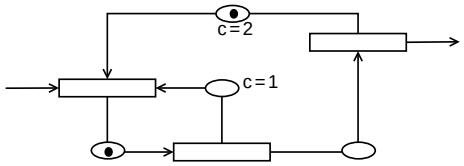


FIG. 3.6 – Exemple d'un graphe à capacité.

Définition 30 (Nœud exécutable dans un graphe avec capacité)**.** *Un nœud n est dit exécutable si tous les canaux de $\bullet n$ contiennent au moins un jeton et que pour tous les canaux $c \in n\bullet$: $M(c) < Capa(c)$.*

Soit G un graphe à capacité et M son marquage. L'ensemble des nœuds exécutables du graphe G selon le marquage M est noté F_M^{capa}.

Remarque 10. *Le modèle d'exécution d'un graphe à capacité est le même que dans la définition 17 à condition de substituer F_M par F_M^{capa} comme défini dans la définition 30.*

Définition 31 (Expansion d'un graphe avec capacité en un sans capacité)**.** *Soit G un graphe à capacité. Soit $c \in L$ un canal à capacité k contenant initialement n jetons. On ajoute au système un canal c' contenant $k - n$ jetons tel que $\bullet c' = c\bullet$ et $c'\bullet = \bullet c$. On crée un cycle formé par $c \rightarrow c\bullet \rightarrow c' \rightarrow c'\bullet$ dans lequel le nombre de jetons est de $n + n - k = k$. Le nombre de jetons par cycle étant invariant [26], le canal c ne contiendra jamais plus de k jetons.*

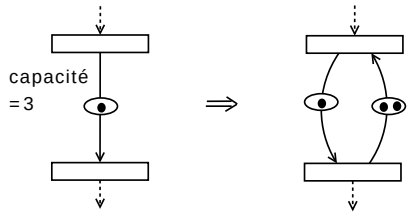


FIG. 3.7 – Transformation d'un canal de capacité $k = 3$ contenant $n = 1$ jeton initial en deux canaux sans capacité contenant $n = 1$ jeton et $k - n = 2$ jetons formant un cycle.

Remarque 11. *L'ensemble F_M des nœuds exécutables au marquage M dans l'expansion d'un graphe à capacité G est égal à l'ensemble F_M^{capa} de G. Donc le modèle d'exécution de l'expansion d'un graphe à capacité est celui défini dans la déf. 17*

3.4.1 Débit de graphe avec capacité

On peut montrer que quand tous les canaux ont une capacité au moins de 2, il est toujours possible d'alimenter les calculs vers l'aval avec des jetons au même rythme que sans capacités (les embouteillages ne causent pas de retard en aval). Dans les travaux sur le *Latency Insensitive Design*, et aussi dans nos travaux antérieurs, nous avons fait l'hypothèse implicite que les réseaux pouvaient opérer au débit du système

sans capacité. Cela se révèle faux comme le démontrent les trois exemples de cette section pour lesquels l'introduction de capacité crée des nouveaux cycles plus lents que le cyle critique du graphe sans capacité.

Dans la figure 3.8, le graphe dessiné en ligne pleine est composé de deux cycles : celui de gauche de débit $8/9$ et celui de droite de débit $7/9$. En absence de capacité sur les canaux, le débit du graphe est de$7/9$. Si on limite les capacités des canaux à 2, le débit du graphe tombe à $3/4$. Les arcs en pointillé représentent la transformation des canaux à capacité de gauche en canaux à capacité non borné comme défini dans la propriété 31. on voit apparaitre un cycle de débit $3/4$ qui passe par les canaux en pointillé puis par les canaux de droite. Ce cycle limite le débit.

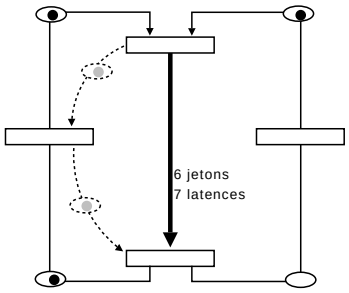


FIG. 3.8 – Le graphe à un débit inférieur à son débit maximum à cause des capacités.

Dans la figure 3.9, le même cas de figure se représente, à la différence que les canaux de gauche ne sont pas tous pleins. Cela montre que ce comportement ne concerne pas seulement les graphes dont le débit s'approche de 1. Les conditions sur le graphe qui mène à ce type de comportement sont ailleurs. Malheureusement, nous avons découvert ce problème trop tard pour que sa résolution apparaisse dans ce manuscrit. En revanche, les chapitres suivants seront consacrés à l'étude d'un problème proche de celui-là. Nous allons chercher à limiter au maximum la capacité des places sans que cela n'affecte le débit du graphe.

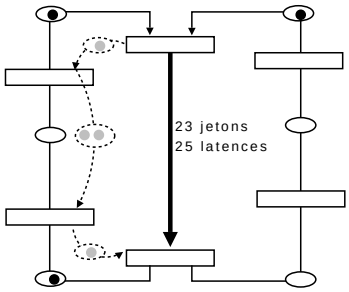


FIG. 3.9 – Le graphe a un débit inférieur à son débit maximum à cause des capacités.

Dans [18], les auteurs de la théorie du LID prétendent que le débit d'un système LID est égal au débit de son cycle le plus lent. Ce n'est pas toujours le cas. La limitation des capacités à deux peut créer des cycles plus lents que le critique qui réduisent encore le débit du graphe. De même, eux comme nous prétendions que le débit d'un DAG (graphe acyclique) est toujours de 1. La aussi, la limitation des capacités des places à deux peut créer des cycles non pleins comme c'est le cas dans la figure 3.10, il apparait un cycle de débit

$3/4$ réduisant le débit.

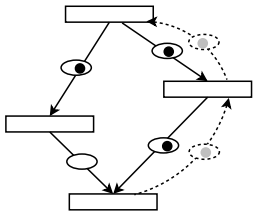


FIG. 3.10 – Limiter la capacité des places d'un DAG fait apparaitre un cycle de débit $3/4$. Ce cycle réduit le débit du DAG.

3.4.2 Capacité instantanée

Dans un canal à capacité, quand le nombre de jetons dans le canal atteint la capacité, on ne peut pas exécuter le nœud entrant. On doit d'abord exécuter le nœud sortant pour pouvoir, à l'instant suivant, le faire. Nous allons définir ici un nouveau type de capacité dit "instantané" qui se comporte comme une place à capacité classique tant que sa capacité n'est pas atteinte mais qui autorise, quand sa capacité est atteinte, à exécuter $\bullet c$ à condition d'exécuter aussi $c\bullet$. (Ce type de canal permet à $\bullet c$ de réagir instantanément à l'exécution de $c\bullet$, d'où le terme "instantané"). Ce type de comportement correspond exactement au comportement de l'implémentation d'un canal tel que nous allons le faire dans les chapitres 5 et 6.

Pour obtenir ce comportement, un canal à capacité instantanée est expandu de la même manière qu'un canal à capacité classique (def. 31) excepté que le canal de retour ajouté à une latence 0. La figure 3.11 montre un exemple de ce type de canal.

Définition 32 (Expansion d'un graphe à capacités instantanées)**.** *Soit G un graphe contenant des canaux ayant une capacité instantanée. Chacun des canaux c ayant une capacité instantanée $capa$ et contenant i jetons peut être remplacé par un canal sans capacité contenant i jetons auquel on ajoute un autre canal de $c\bullet$ vers $\bullet c$ de latence nulle et contenant $capa - i$ jetons.*

Remarque 12. *Un graphe à capacité instantanée peut donc être expandu en un graphe sans capacité contenant des canaux à latence instantanée. Il peut être ordonnancé en utilisant le modèle d'exécution "au plus tôt" de la définition 28.*

Remarque 13. *Quand on expand un canal c à capacité instantanée, on crée un cycle formé de c et du canal ajouté. Le débit de ce cycle est égal à la capacité de c divisée par la latence du cycle :* 1*. Comme la capacité de c est d'au moins* 1*, le cycle, qui a donc un débit*≥ 1 *ne peut jamais ralentir le système.*

Remarque 14. *Si tous les canaux d'un cycle ont une capacité instantanée, leur expansion crée un cycle de latence nulle. Ce nouveau cycle ne posera pas de problème d'exécution infinie car il est limité par les autres cycles du graphe.*

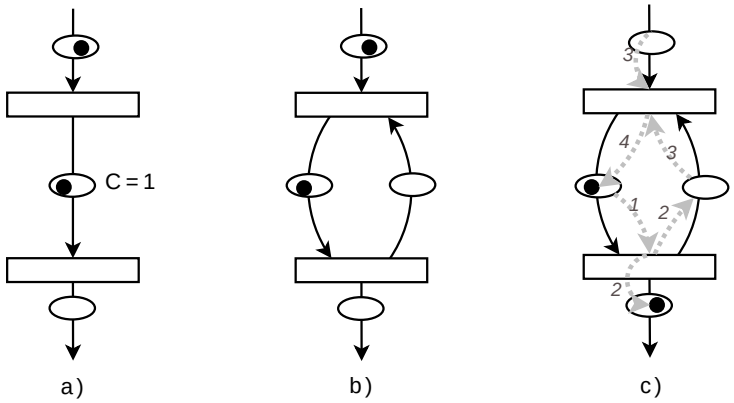


FIG. 3.11 – a) Un canal à capacité instantanée b) son équivalent par l'expansion de la def. 32, c) un pas d'exécution représenté par les flèches grises et décomposé en 4 étapes.

Remarque 15. *Le fait de fixer une capacité instantanée $capa$ sur un canal c risque de limiter le débit du graphe seulement si l'exécution "au plus tôt" du même graphe sans capacité sur c nécessite de stocker dans c un nombre de jetons j tel que $j > capa$.*

Si la capacité d'un canal est suffisante :$capa \geq j$, l'exécution de $\bullet c$ ne dépendra jamais de l'état de c mais seulement de la présence de jeton dans les canaux d'entrées de $\bullet c$ (dans le cas où $M(c) = capa$, soit $\bullet c$ n'est pas exécutable soit $\bullet c$ et $c\bullet$ sont exécutables).

3.5 Conclusion

Maintenant que les modèles et les notions de base autour des réseaux de processus sont présentés, nous allons pouvoir développer les théories de ce manuscrit. Le chapitre 5 sur l'égalisation repose exclusivement sur les définitions de cette section. Le chapitre 6 nécessite autant les résultats du chapitre 2 que ceux de ce chapitre. L'étude du Latency-Insensitive Design dans le chapitre 4 ne nécessite aucune des notions présentées dans ce chapitre car l'approche du sujet se fait par la description des éléments de micro électronique nécessaires à la théorie. Le fait qu'un système LID ne soit rien de plus qu'un graphe où tous les canaux ont une capacité de 2 et pour lequel la politique d'exécution est "au plus tôt" doit se lire entre les lignes de la description pratique de la théorie.

Chapitre 4

Conception insensible aux latences (LID)

Cette section contient une étude complète de la théorie de la conception insensible aux latences (LID : Latency Insensitive Design). Nous allons d'abord présenter la théorie originelle dans la section 4.1 puis nous nous intéresserons à notre contribution propre dans la section 4.2. Ensuite,nous passerons en revue les travaux de la communauté inspirés par cette théorie dans la section 4.3.

La théorie du LID répond aux problèmes des latences sur les fils d'interconnections et de la réutilisation des IPs (Intellectual Property : Bloc fonctionnel de micro-électronique propriétaire). Nous allons nous y intéresser comme le point de base de notre approche. A l'origine, il y avait les problèmes d'interconnection et de réutilisation. Puis il y a eu le LID et ensuite, nous avons cherché à améliorer le LID. Notre solution finale (chapitre 6) s'en éloigne sensiblement, mais le LID reste le point de base de notre réflexion.

4.1 La théorie du "Latency Insensitive Design" (LID)

Cette section présente la théorie de la conception insensible aux latences (*Latency Insensitive design*) telle qu'elle a été imaginée par ses auteurs : Luca P. Carloni, Kenneth L. McMillan, and Alberto L. Sangiovanni-Vincentelli. Cette théorie a été publiée pour la première fois en 1999 dans [17], elle a ensuite été étayée et complétée par ses auteurs grâce à plusieurs publications [15, 16, 18, 19]. Il est à noter que la première publication s'évertue à prouver l'équivalence fonctionnelle entre un circuit idéal synchrone et un circuit insensible aux latences. Par fonctionnelle on entend que pour les mêmes valeurs en entrée, on obtient bien les mêmes valeurs en sortie, dans le même ordre, seul le timing change. Alors que le circuit idéal synchrone consomme un jeu d'entrée et produit un jeu de sortie à tous les instants, dans le circuit insensible aux latences, il arrive que certains instants ne soient pas sanctionnés par une consommation d'entrée ou une production de sortie. Le circuit insensible aux latences est donc plus lent que son idéal synchrone qui lui peut être irréalisable physiquement, dû justement aux problèmes de latences.

4.1.1 Les problèmes de conception

La théorie de LID permet de résoudre deux problèmes de conception. Le premier est apparu avec les avancées technologiques en matière de miniaturisation de circuit intégré et de l'augmentation de la fréquence des horloges qui séquencent ces circuits. C'est le problème des latences sur les fils d'interconnexions déjà présenté précédemment. Un système sur puce est un ensemble de blocs fonctionnels qui communiquent entre eux. Un signal qui est émis par un bloc n'a pas le temps de se propager jusqu'au bloc destination avant que le coup d'horloge suivant n'arrive. La valeur qui est récupérée en entrée du bloc de réception n'est donc pas la bonne.

Le second problème est lui aussi apparu avec les avancées technologiques en matière de miniaturisation de circuit intégré mais ce n'en est pas la cause. Avec le progrès, on peut mettre de plus en plus de choses sur une puce, alors qu'il y a 20 ans le bloc de base était le transistor, il est devenu la porte logique, puis le registre. Aujourd'hui, on aimerait passer à l'étape suivante qui consisterait à considérer comme bloc de base des unités fonctionnelles formées d'un ensemble de registres et de logiques combinatoires et donc de pouvoir réutiliser ces unités fonctionnelles d'une conception de puce à une autre. Malheureusement, à chaque conception de puce, il faut vérifier que le pire délai de propagation entre deux registres d'un même bloc ne soit pas plus grand que la période de l'horloge. Il faut regarder à l'intérieur des unités fonctionnelles et donc casser leur atomicité.

4.1.2 Présentation générale

La théorie du LID consiste à envelopper chacun des blocs de base de notre système sur puce par une interface : le *Shell-Wrapper* capable de gérer ses entrées/sorties et son fonctionnement. Le bloc de base devient donc un ensemble de registre et de logique combinatoire tel que le temps de propagation d'un signal sur son chemin le plus long ne dépasse pas la période de l'horloge. Il est à noter que les structures pipelinées sont considérées comme bloc de base.

Etant donné que le bloc de base est entièrement contrôlé par son interface, le *Shell-Wrapper*, il est possible que celle-ci ne lui transmette pas le signal d'horloge. Dans ce cas, l'état interne du bloc de base doit rester le même. Si l'absence du signal d'horloge modifie l'état interne d'un bloc, il n'est pas compatible avec la théorie du LID. S'il respecte cette contrainte, on dit d'un bloc de base qu'il est patient (*patient process*).

Un bloc de base est tel qu'à chaque fois qu'il reçoit un coup d'horloge, il consomme une donnée sur chacune de ses entrées et produit une donnée sur chacune de ses sorties. Un fil d'interconnexion est une liaison point à point, il a une et une seule source et une seule destination.

La figure 4.1 nous montre les transformations qui sont apportées à (a) un système synchrone avec un problème de délai d'interconnexion afin de devenir (b) un système insensible aux latences.

4.1.3 Shell-Wrapper

Le *Shell-Wrapper* est une interface entre un bloc fonctionnel (bloc de base) et ses signaux d'entrée/sortie (y compris l'horloge du système). Il a pour mission de stocker les valeurs qui arrivent sur les entrées, puis,

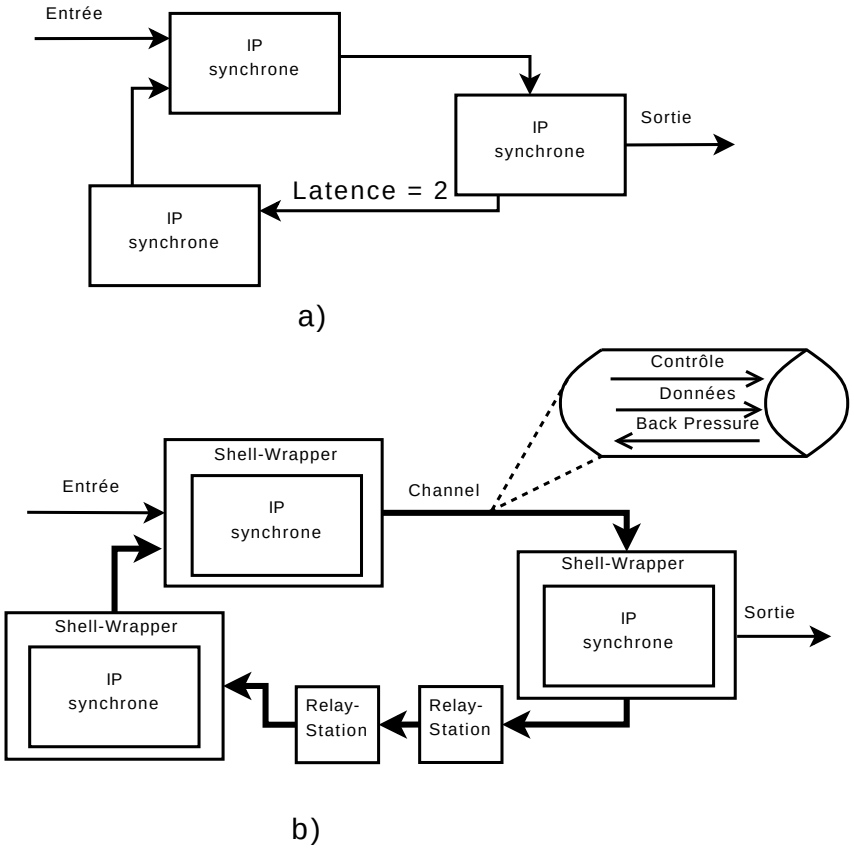


FIG. 4.1 – Transformation (a) d'un système synchrone avec des problèmes de délai d'interconnexion en (b) un système insensible aux latences.

une fois que chaque entrée a sa valeur, de les propager au bloc avec le signal d'horloge afin de le faire travailler. En l'absence d'entrée valide, le signal d'horloge n'est pas propagé. Le *Shell-Wrapper* doit ensuite stocker les valeurs produites sur les sorties puis les propager en aval. Le bloc fonctionnel est appelé perle (*Pearl*) car il est entouré d'une coquille, d'un coquillage. (le *Shell*).

Le *Shell-Wrapper* doit s'assurer qu'aucune valeur n'est perdue en entrée comme en sortie. Pour cela, une fois qu'il a reçu une valeur sur une de ses entrées, il envoit un signal à l'élément (*Shell-Wrapper* ou *Relay-Station*) qui lui a transmis la donnée pour bloquer sa production de donnée jusqu'à ce que toutes les autres entrées du *Shell-Wrapper* contiennent une donnée et qu'elles soient transmises à la perle. Une fois les buffers d'entrées du *Shell-Wrapper* vidés, il lève les signaux de blocage et recommence à collecter des données. Concernant ses sorties, une fois qu'elles ont été produites par la perle, il ne peut les transmettre que si le destinataire est prêt à les recevoir. Si le destinataire lui envoie un signal de blocage, il ne peut rien transmettre et comme ses buffer de sortie sont pleins, il est obligé d'empêcher sa perle de travailler sinon les données produites par sa perle seraient perdues ou viendraient écraser les données actuellement présentes dans le buffer de sortie. Par contre, si le destinataire ne lui envoie pas de signal de blocage, il peut transmettre ses données et attendre que la perle en produise des nouvelles.

4.1.4 Relay-Station

La *Relay-Station* permet de découper un fil d'interconnexion sur lequel le temps de propagation est supérieur à la période de l'horloge en un ensemble de sections successives sur lesquelles la propagation est correcte : Le temps de propagation d'un bout à l'autre de la section est inférieur ou égal à la période de l'horloge. Grâce à cela, l'hypothèse synchrone est conservée. Une donnée qui transite sur le fil d'interconnexion est donc transmise de *Relay-Station* en *Relay-Station*. Le rôle principal de la *Relay-Station* est donc de recevoir, stocker puis réémettre une donnée. De plus elle sert de borne à la propagation des signaux sur un instant, c'est à dire qu'un signal émis en amont de la *Relay-Station* ne peut pas être transmis en aval dans le même instant.

De même que le *Shell-Wrapper*, la *Relay-Station* doit s'assurer qu'aucune valeur n'est perdue. Elle émet un signal seulement si le destinataire ne lui transmet pas de signal de blocage. Dans le cas où effectivement, le destinataire envoie un signal de blocage, la *Relay-Station* doit être prête à accueillir une deuxième donnée qui lui aurait été transmise par son prédécesseur. Par contre, à partir de l'instant suivant, elle transmettra à son tour le signal de blocage à son prédécesseur et ce jusqu'à ce que la plus vieille des deux données qu'elle contient soit transmise.

4.1.5 Channel

En lisant la description du *Shell-Wrapper* et de la *Relay-Station*, il apparait que le trafic entre deux éléments du système est composé des signaux suivants :

- Les données vont de l'émetteur au récepteur.
- Un signal parallèle aux données indiquant au récepteur la présence de ces données. (la validité du signal reçu sur le fil de donnée)

– Le signal de blocage allant du récepteur vers l'émetteur pour lui signifier l'incapacité de recevoir une donnée.

Ce qui était un simple fil d'interconnexion dans le circuit idéal synchrone est maintenant un canal de communication bilatérale (deux fils dans le sens émetteur-récepteur, un dans l'autre) appelé : *channel*.

4.1.6 Protocole de rétroaction (*Back-Pressure Protocol*)

Le *Shell-Wrapper* et la *Relay-Station* assurent qu'aucune donnée n'est perdue. Pour cela, ils utilisent le signal de blocage. Chacun des éléments du système est soit une *Relay-Station* soit une perle entourée de son *Shell-Wrapper*, le tout relié par un *channel*. On a donc un protocole de communication répartie, géré par ces trois éléments, qui est régi par les deux règles suivantes :

– Un élément n'émet pas de donnée s'il reçoit un signal de blocage.

– Un élément plein émet un signal de blocage.

Ce protocole basé sur l'envoi d'un signal de rétroaction est appelé :*back-pressure Protocol*.

4.1.7 Propriétés de correction

Quand on analyse le fonctionnement d'un circuit insensible aux latences, on se rend compte que deux propriétés doivent être respectées pour garantir le bon fonctionnement du système.

Si le circuit contient des cycles, il faut qu'à l'initialisation du circuit au moins une donnée valide soit chargée dans un élément de chaque cycle. Si un cycle est vide, aucun des *Shell-Wrappers* qu'il contient ne pourra faire marcher sa perle car il manquera toujours une entrée. Cette propriété est à mettre en parallèle avec la propriété de vivacité (*liveness*) définie exactement de la même manière dans les *Marked Event Graph*.

Le protocole de rétroaction est implanté dans tout le circuit grâce à la présence de *Shell-Wrapper* et de *Relay-Station*. Mais qu'en est-il des extrémités du circuit ? ses entrées/sorties ? Si un circuit insensible aux latences est interconnecté à un autre type de circuit, il faut que le bloc en aval soit capable de détecter les instants où notre circuit produit un jeu de données valides. Plus important, pour notre système, s'il y a un ralentissement dans notre circuit et que le signal de blocage remonte jusqu'à l'entrée globale du système, il ne faut surtout pas que le circuit en amont envoie une donnée. Il doit comprendre la situation et réagir correctement. Il serait intéressant d'étudier quels mécanismes il faut mettre en place pour assurer le bon fonctionnement d'un circuit insensible aux latences dans un système plus grand. Une piste possible serait de placer un buffer d'interconnexion servant d'interface entre le circuit lid et son entourage. Puis de vérifier certaines conditions sur les débits de production et de consommation de données afin de s'assurer que le buffer d'entrée n'est jamais plein et que le buffer de sortie n'est jamais vide. Nous avons brièvement étudié ce problème dans [43]

4.2 Formal Latency Insensitive Design

Cette section présente le travail que nous avons réalisé sur la modélisation formelle des éléments de la théorie des systèmes insensibles aux latences. Il présente la modélisation formelle des deux éléments de la théorie : la *Relay-Station* et le *Shell-Wrapper*. Ces résultats sont détaillés dans [44] et [46].

Notre approche utilise les langages synchrones [9] comme moyen de définir formellement les éléments qui composent la théorie du *LID*. Ensuite, nous avons vérifié les hypothèses de fonctionnement de la théorie du *LID* sur notre implantation.

4.2.1 Description formelle de la *Relay-Station*(RS)

La figure 4.2 présente l'interface de la *Relay-Station* ; elle est composée de deux signaux d'entrées et de deux signaux de sorties. La terminologie des signaux est issue du travail du M. R. Casu dans [20].

- Le signal d'entrée *val_in* prévient la RS de la présence d'une donnée en entrée.
- Le signal de sortie *val_out* est émis par la RS en même temps qu'une donnée. Il prévient le successeur de l'arrivée de cette donnée.
- Le signal de sortie *stop_in* est émis vers le prédécesseur par la RS quand elle ne peut plus recevoir de donnée.
- Le signal d'entrée *stop_out* est reçu par la RS que son successeur ne peut plus recevoir de donnée.

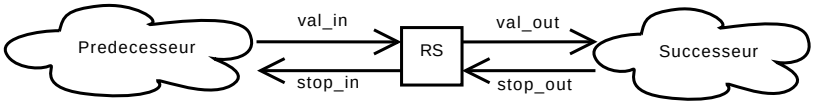


FIG. 4.2 – Signaux d'interface de la *Relay-Station*(RS).

La figure 4.3 décrit le fonctionnement d'une RS sous la forme d'un *Sync-Chart* [6]. Les Sync-Charts sont des représentations graphiques du langage réactif synchrone ESTEREL [11, 12]. Le *Sync-Chart* présenté dans la figure 4.3 a la même sémantique qu'une machine de Mealy.

Une RS contient deux registres, elle a trois états possibles. Soit elle est vide : état EMPTY, soit le buffer principal (*main*) contient une donnée et pas le buffer auxiliaire (*aux*) :(état HALF), soit les deux buffers sont pleins : état FULL. Le buffer auxiliaire ne peut pas être plein alors que le principal est vide. En fonction de ces trois états, le fonctionnement est le suivant :

- Etat EMPTY : Tant que l'on ne reçoit pas de donnée (tant que le signal *val_in* n'est pas présent), on reste dans cet état et on ne fait rien. Si on reçoit une donnée (le signal *val_in* est présent), on la conserve, on passe dans l'état HALF. Dans l'état EMPTY, le signal *stop_out* est ignoré car on n'a pas de donnée à émettre.
- Etat HALF : Si on reçoit une nouvelle donnée et pas de signal de blocage (*val_in et not(stop_out)*), on émet notre donnée courante (*val_out(main)*) et on conserve la nouvelle. On reste dans l'état HALF. Le même état est aussi conservé si on ne reçoit pas de nouvelle donnée mais qu'un signal de blocage nous parvient (*not(val_in et stop_out)*), on ne fait rien. Toujours à partir de l'état HALF, si on ne

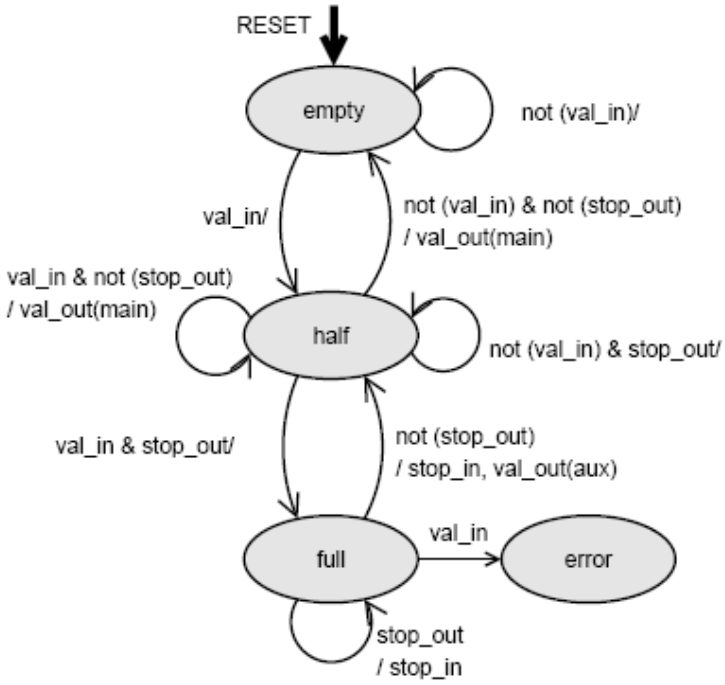


FIG. 4.3 – Sync-Chart de la *Relay-Station*(RS).

reçoit ni nouvelle donnée ni signal de blocage (*not(val_in) et not(stop_out)*), on émet notre donnée courante (*val_out(main)*), le registre principal est vidé, on retourne dans l'état EMPTY. Inversement, si on reçoit une nouvelle valeur et un signal de blocage (*val_in et stop_out*), on ne peut pas émettre notre donnée courante. De plus on doit conserver la nouvelle donnée. On conserve donc ces deux données, la plus vieille dans le buffer auxiliaire et la nouvelle dans le buffer principal. On passe dans l'état FULL.

– Etat FULL : Tout d'abord, tant que l'on est dans cet état, on émet le signal stop_in à tous les instants. Ceci implique que le prédécesseur ne nous enverra jamais de nouvelle donnée (le signal *val_in* n'est jamais reçu). Le comportement de la RS dans cette état est donc le suivant : tant que le signal de blocage est actif(*stop_out*), on conserve nos deux données. Quand il ne l'est plus (*not(stop_out)*), on émet notre plus ancienne valeur (*val_out(aux)*) et on retourne dans l'état HALF.

La figure 4.4(a) est la traduction en circuit de logique séquencielle de la machine de Mealy présentée dans la figure 4.3. On retrouve les deux registres, le principal et l'auxiliaire, les signaux d'entrée/sortie : *val_in*, *stop_out*, *val_out*, *stop_in*.

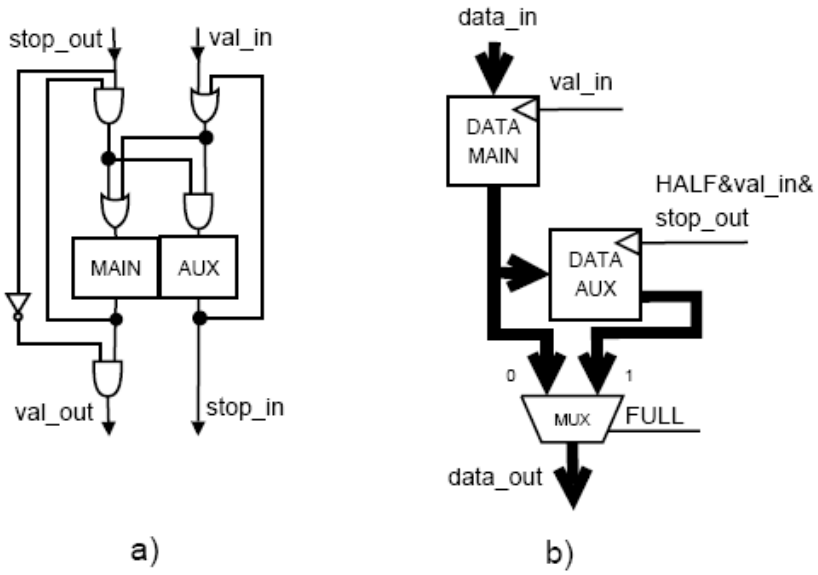


FIG. 4.4 – (a) Circuit logique de la *Relay-Station* et (b) son chemin de données.

La figure 4.4(b) représente le chemin des données à travers la *Relay-Station*.

4.2.2 Vérification formelle du bon comportement de la *Relay-Station*

La *Relay-Station* permet de découper un fil d'interconnexion en section synchrone (Le temps de propagation d'un signal d'un bout à l'autre de la section est inférieur à la période de l'horloge). Le premier

critère que doit respecter la *Relay-Station*, c'est de ne jamais propager combinatoirement un signal d'entrée sur sa sortie. Le circuit de la figure 4.4 a) montre que la propagation des signaux d'entrées est soit arrêtée par les registres "MAIN" et "AUX", soit limitée à la même section synchrone : le signal *val_out* envoyé au successeur dépend du signal *stop_out* reçu du même successeur. Le découpage en section synchrone est donc conservé.

Le but de la *Relay-Station* est de transmettre des données dans le même ordre qu'elle les a reçues. Tant qu'elle reste dans les états "EMPTY" ou "HALF", elle ne contient qu'une seule donnée à la fois. Par contre, le passage dans l'état "FULL" mérite une attention supérieure. Le chemin de donnée représenté par la figure 4.4 b) montre que lors du passage de l'état "HALF" à "FULL", la donnée courante est envoyée dans le buffer auxiliaire alors que la nouvelle est mise dans le buffer principal. Dès que le signal de blocage est levé, la première donnée à sortir est celle contenue dans le buffer auxiliaire (la plus ancienne) car le multiplexeur pointe sa branche de droite (on est dans l'état "FULL"). Suite à cela, on retourne dans létat "HALF". L'ordre est conservé.

Il résulte une condition assurant que la *Relay-Station* ne perd pas de données. Quand la *Relay-Station* est pleine, elle émet le signal *stop_in* à destination de son prédécesseur. Si le prédécesseur émet quand même une donnée, elle sera perdue. Il faut donc que ce dernier (qui est soit une *Relay-Station* soit un *Shell-Wrapper*) respecte l'assertion suivante : *stop_out* $\Rightarrow$ *not(val_out)*. La branche la plus à gauche du circuit de la RS présenté dans la figure 4.4 (a) vérifie cette assertion pour la RS.

4.2.3 Description formelle du *Shell-Wrapper*

Le comportement du *Shell-Wrapper* est plus difficilement représentable par un automate car son nombre d'état varie en fonction du nombre d'entrée de la perle associé. La figure 4.5(a) et (b) nous présente donc une description formelle sous forme d'un circuit de logique séquentielle. La figure 4.5 (c) représente le chemin de données à travers le *Shell-Wrapper*. Dans la figure 4.5 la perle associée à n entrées et m sorties.

Pour chaque entrée de la perle, le bloc d'entrée de la figure 4.5 (b) permet de recevoir une valeur et de la transmettre ou de la conserver si nécessaire. De plus, il permet aussi d'émettre un signal de blocage à destination du prédécesseur à partir du moment où une donnée a déjà été reçue. Si le bloc d'entrée numéro i reçoit une valeur ou en détient une dans son registre, il émet un signal *VAL_IN[i]*. La figure 4.5 a) montre que si tous les blocs d'entrées émettent un signal *VAL_IN*, qu'aucun des successeurs n'émette de signal de blocage (*stop_out[1...,i...,m]*) alors le *Shell-Wrapper* active la perle (*Fire_Pearl*), lui transmet les données et prévient les m successeurs de l'arrivée d'une donnée produite par la perle (*val_out[1,...i,..m]*).

Le signal d'activation de la perle (*Fire_Pearl*) est renvoyé à chacun des blocs d'entrée afin de mettre à jour l'état de leur registre.

4.2.4 Vérification formelle du bon comportement du *Shell-Wrapper*

Même si toutes les entrées de la perle sont capables de fournir une donnée (*ALL_VALL_IN*), le *Shell-Wrapper* ne fait travailler sa perle que si aucun des successeurs n'a èmis de signal de blocage (*STOP_OUT*), L'assertion *stop_out* $\Rightarrow$ *not(val_out)* est donc préservée par le *Shell-Wrapper*.

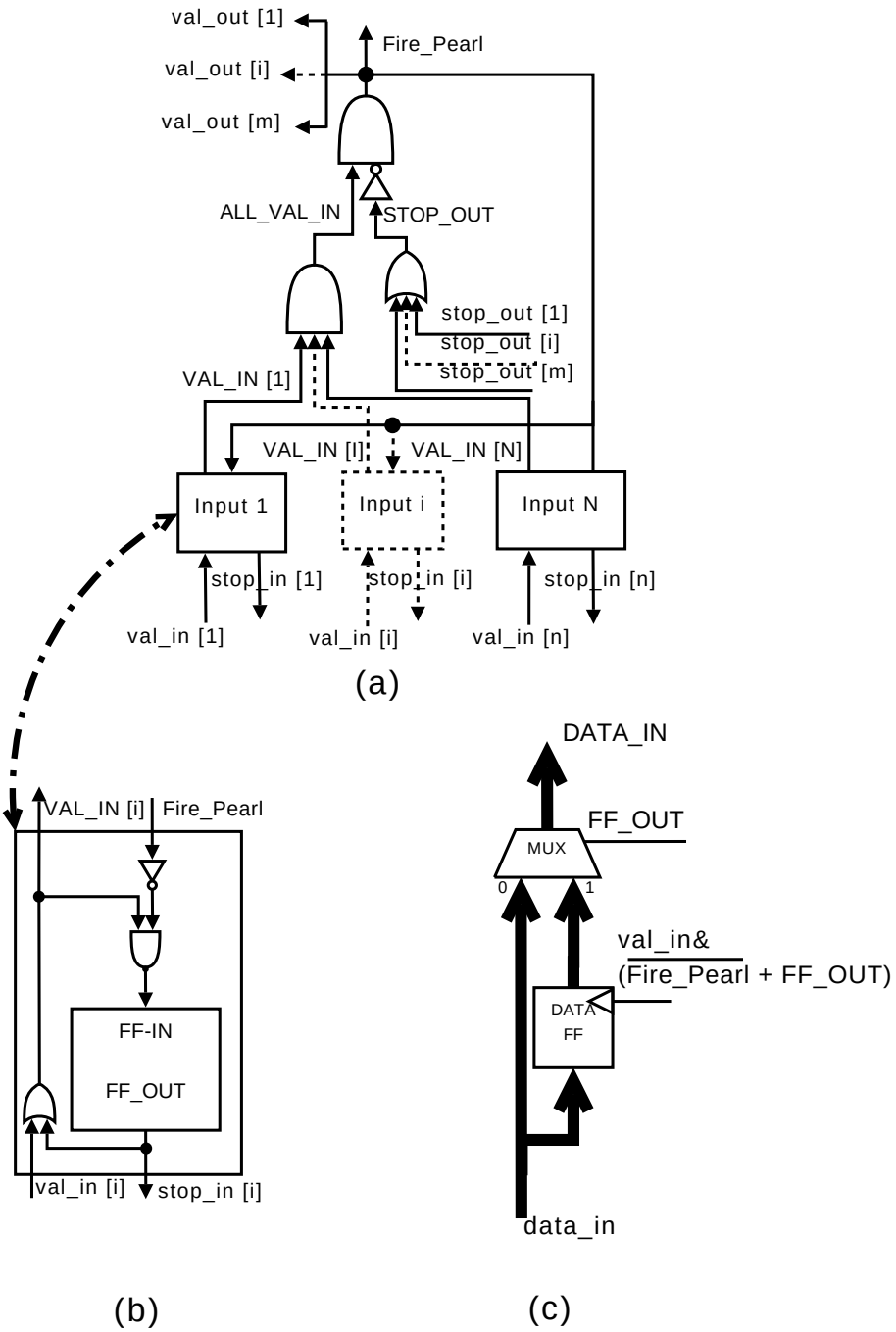


FIG. 4.5 – (a) Circuit logique du *Shell-Wrapper*, (b) détail du bloc associé à une entrée, et (c) chemin de données à travers le *Shell-Wrapper*.

4.3 Vue d'ensemble des contributions aux "Latency Insensitive Design"

Cette section est dédiée à la présentation des travaux des équipes concurrentes sur le thème du *Latency Insensitive Design*. Beaucoup de papiers ont été publiés sur le sujet, nous allons nous contenter de présenter les résultats les plus proches de nos activités. Tout d'abord ceux du Dr Montek Singh de l'université de Chapel Hillet, USA et du Dr Michael Theobald de l'université de Pittsburgh,USA([52, 51]) puis ceux de M. R. Casu et de L. Macchiarulo du Politechnico di torino ([20, 21, 22, 23]).

4.3.1 Systèmes insensibles aux latences généralisés

Il est ici question de rajouter de la souplesse au modèle LID à travers deux améliorations distinctes.

La première permet de nous affranchir d'une des hypothèses de base de la théorie du LID : un bloc de base, une perle, doit à chaque instant d'activation consommer sur toutes ses entrées et produire sur toutes ses sorties. Si la perle en question ne fonctionne pas de cette manière, qu'à certains de ses instants d'activation elle n'a besoin que d'un sous ensemble de ses entrées, on est obligé, dans la théorie originelle, de représenter l'absence de donnée sur les entrées inutiles à cet instant comme des valeurs vides. Le *Shell-Wrapper* doit donc attendre l'arrivée de ces valeurs vides avant d'activer la perle. Son activation peut être retardée à cause d'une donnée qui lui sera inutile. Il en est de même pour les sorties du système, si à un instant d'activation donnée, la perle ne produit des données que sur un sous ensemble de ses sorties, on doit représenter l'absence de donnée émise sur les sorties qui ne sont pas concernées à cet instant par une production de valeurs vides.

Pour éviter cela, il faut connaître pour chaque perle, à chaque instant, le sous ensemble de ses entrées concernées. Etant donné que l'on sait prévoir le comportement de la perle de manière statique en fonction des stimulations qu'elle reçoit (le comportement est déterministe), on peut calculer un automate à état fini qui fournit cette information au *S*hell-Wrapper.

La seconde permet de nous affranchir d'une autre hypothèse de base de la théorie du LID : l'exclusivité des liaisons point à point comme moyen d'interconnexion. Si la topologie de la spécification idéale synchrone contient des liaisons multiples comme par exemple une donnée produite et transmise en parallèle à deux blocs fonctionnels, la représentation de cette topologie dans un système insensible aux latences oblige à considérer chaque chemin comme indépendant ce qui provoque une multiplication du nombre de connexion. Plus grave, si par exemple la topologie de la spécification idéale synchrone est telle qu'une entrée reçoit des données alternativement de deux blocs fonctionnels, on ne va pas pouvoir la rendre insensible aux latences.

Les auteurs introduisent un jeu de trois éléments qui permettent de raffiner la gestion des communications dans la théorie des systèmes insensibles aux latences :

- Le *fork* qui duplique une entrée.
- Le *split* qui divise un flot d'entrées en plusieurs (2) flots de sortie.
- Le *Join* qui rassemble plusieurs (2) flots d'entrées en un seul flot de sortie.

Si le comportement du *fork* est trivial, il est à noter que le *split* et le *Join* doivent être dirigés par un "oracle" qui détermine sur quelle sortie le *split* doit envoyer la donnée courante ou sur quelle entrée le *Join* doit attendre et récupérer sa prochaine valeur. Etant donné que le comportement du système est déterministe par nature, on peut construire un programme qui dirige chacun de ces éléments, on peut construire "l'oracle".

4.3.2 Une nouvelle approche du LID

Le travail du Dr Mario R. Casu et du Dr Luca Macchiarulo sur les systèmes insensibles aux latences, présenté dans [20, 21, 22, 23], est similaire à certains de nos travaux relatés dans cette thèse en section 4.2 et dans le chapitre 5. Ils concernent premièrement l'implémentation formelle de la théorie du LID et deuxièmement une simplification du protocole de rétroaction des systèmes insensibles aux latences basée sur une analyse statique du comportement du système et une prédiction des instants d'activations de chaque perle.

Implémentation formelle de la théorie du LID L'implémentation formelle des éléments de la théorie du LID, la *Relay-Station* et le *Shell-Wrapper*, est détaillée dans [20]. La *Relay-Station* est présentée sous la forme d'un automate à états finis (présenté dans la figure 4.6) à 6 états stimulés par les signaux suivants :

- Le signal *valin* (pour value in input) arrive parallèlement à une donnée dans la *Relay-Station*. Il sert simplement à la prévenir de l'arrivée de cette donnée.
- Le signal *stopout* (pour l'injonction « Stop the output ! ») arrive à la *Relay-Station* en provenance du bloc suivant. Il permet à ce dernier de signaler son incapacité à recevoir une nouvelle donnée.

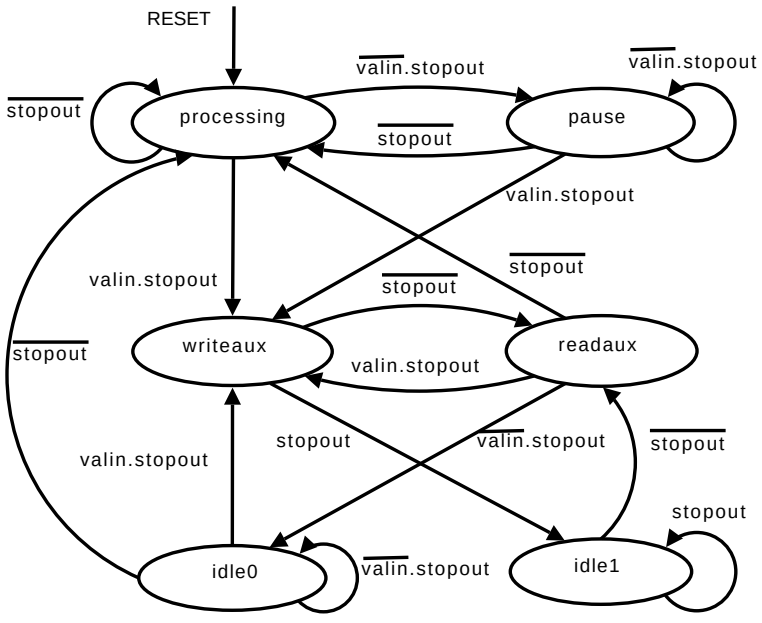


FIG. 4.6 – Automate à état fini décrivant le comportement d'une *Relay-Station*.

L'automate à état fini est tel que les conditions de changement d'état sont inscrites sur les canaux et les actions sont associées aux états. Malheureusement, les actions n'apparaissent pas formellement sur

l'automate.

L'implémentation formelle du *Shell-Wrapper* est donnée sous la forme de circuit de porte logique. La figure 4.7 présente ce circuit pour un *Shell-Wrapper* contrôlant une perle à deux entrées et trois sorties. Pour chaque entrée de la perle, son interface est composée de :

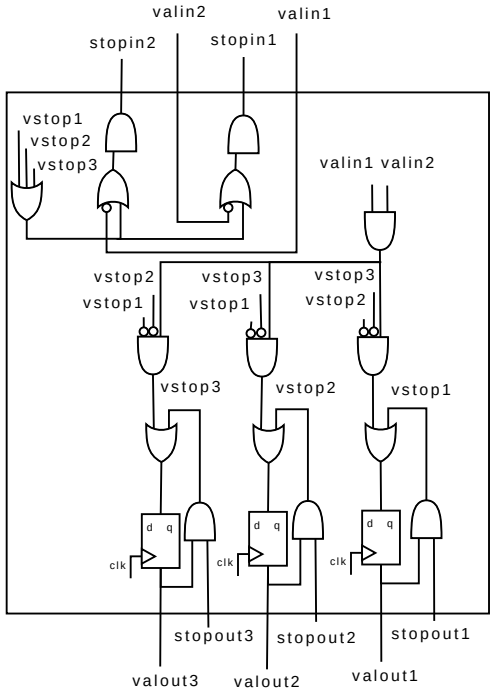


FIG. 4.7 – Circuit logique décrivant le comportement d'un *Shell-Wrapper*.

- Le signal d'entrée *valin* qui prévient de l'arrivée d'une donnée sur cette entrée à cet instant.
- Le signal de sortie *stopin* (« stop the input ! ») qui prévient le prédécesseur de l'incapacité du *Shell-Wrapper* de recevoir une nouvelle donnée sur cette entrée.

Pour chacune des sorties de la perle, son interface est aussi composée de :

- Le signal d'entrée *stopout*, envoyé par le bloc suivant, il permet à ce dernier de signaler son incapacité à recevoir une nouvelle donnée.
- Le signal de sortie *valout* (value in output) est envoyé au bloc suivant pour le prévenir de l'arrivée d'une donnée.

Simplification du protocole de rétroaction Etant donné le caractère déterministe des systèmes insensibles aux latences, la simplification du protocole de rétroaction peut se faire grâce à des informations extraites d'une analyse statique du comportement du système. On peut savoir à quels instants, quels *Shell-Wrapper* activent leur perle et lesquels ne le font pas. A partir de là, on peut implanter l'ordonnancement de chaque perle en utilisant la technique du *Clock gatting* comme l'illustre la figure 4.8.

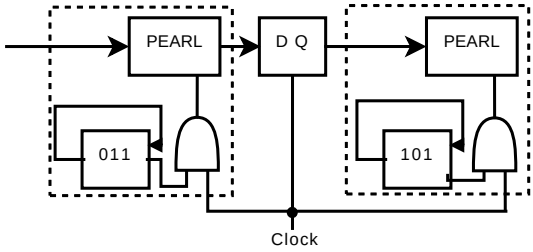


FIG. 4.8 – Ordonnancement statique d'une perle utilisant la technique de *Clock gatting*.

Ce nouvel élément remplace le *Shell-Wrappers* car il réalise la même fonction. La disparition du signal de rétroaction permet aux *Relay-Stations* de se passer de leur buffer auxiliaire et ainsi de redevenir de simple registre de transport.

Cette simplification induit des complications au niveau des points de reconvergence du système (au niveau des *Shell-Wrapper* avec plusieurs entrées). Les données n'arrivent pas forcément de manière synchrone en entrée du *Shell-Wrapper*, certaines arrivent avec de l'avance. Le protocole de rétroaction permettait de retenir ces données jusqu'à leur utilisation par la perle. Pour compenser l'absence de protocole de rétroaction, nous devons placer des éléments supplémentaires en entrée des *Shell-Wrappers* appelés *Fractional synchronizer*. L'analyse statique du système nous permet de savoir quelles données doivent être retenues. Le *Fractional synchronizer* s'occupe de retenir uniquement ces données. Un circuit comme celui présenté dans la figure 4.9 permet de réaliser cette fonction.

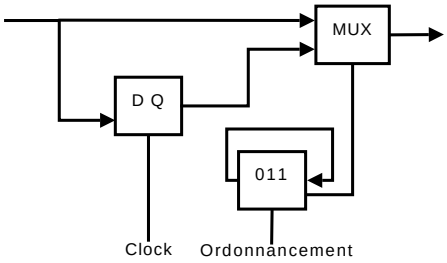


FIG. 4.9 – Un *Fractional synchronizer* et son ordonnancement (011).

Un système déterministe est ultimement périodique, avant d'avoir un comportement répétitif, il passe par une phase d'initialisation où le comportement de chaque élément est singulier. La figure 4.10 montre comment implanter cette phase d'initialisation dans l'ordonnancement des éléments du système.

4.3.3 Circuits synchrones élastiques

Par le passé, Jordi Cortadella de l'Universitat Politecnica de Catalunya, Barcelone, Espagne et Mike Kishinevsky d'Intel Corp., Hillsboro, USA ont travaillé sur les protocoles de désynchronisation de circuit permettant de se passer de l'hypothèse synchrone et donc de l'horloge globale afin d'améliorer les performances [37, 49, 27]. Suite à l'apparition de la théorie du LID, ils ont développé un protocole de resynchro-

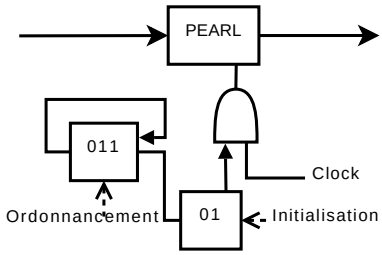


FIG. 4.10 – Ajout de la phase d'initialisation à l'ordonnancement de la perle.

nisation de circuit inspiré à la fois des systèmes LID et de leur protocole de désynchronisation [28, 29]. Leurs résultats ont donné le jour à une solution implantable permettant soit de réduire la consommation en énergie du circuit soit d'augmenter les performances.

Chapitre 5

égalisation

Nous allons maintenant chercher à ordonnancer statiquement les systèmes LID. Afin d'être cohérent avec la philosophie suivie pour la création des *Relay-Station* qui consiste à éviter d'importantes accumulations de données en un point, nous allons dans un premier temps égaliser les flots afin que les jetons arrivent le plus souvent possible simultanément au nœud de calculs. Pour faire cela, nous allons ajouter des latences virtuelles sur les chemins rapides de notre système afin de les ralentir aussi près que possible des chemins les plus lents du système.

Il existe des cas pour lesquels l'ajout de latence entière permet d'égaliser parfaitement le système. Les jetons arrivent toujours simultanément au nœud de calculs. Dans ce cas, le système peut être implémenter sans contôle de flux (*Back-Pressure*) et sans *Relay-Station*. Dans [19], les auteurs donnent une méthode permettant d'atteindre cette égalisation parfaite quelque soit la topologie du système mais qui oblige à pessimiser le débit du système. Par exemple, un système ayant un débit de $0,6$, peut s'égaliser parfaitement si on réduit son débit à $0,5$.

Si on choisit de ne pas réduire le débit global du système, on ne peut pas toujours atteindre cette solution. L'égalisation permet de réduire les différences de débit ; pour les gommer complètement, il faut y ajouter un *Registre Fractionnaire* similaire au *fractionnal synchronizer* de Casu (section 4.3.2) là où les différences de débit entre les chemins se concrétisent par des décalages dans les instants d'arrivées des jetons à un nœud.

Le *Registre Fractionnaire*, présenté dans la section 5.2, a pour fonction de retenir certains jetons un instant et de laisser passer les autres.

La section 5.1 présente le processus d'égalisation dans son ensemble. La section 5.2 est dédiée au *registre fractionnaire* et enfin, la section 5.3 nous permet de discuter de nos choix de conception du processus d'égalisation et des possibles optimisations qui peuvent être apportées.

Le but du processus d'égalisation est de $\mathbb{Q}$-égaliser un graphe et de l'ordonnancer.

Abstraction de circuit en *Marked Graph* Comme on a pu le voir dans l'exemple de la section 1.1, nous allons avoir besoin d'abstraire les circuits électroniques en *Marked Graph* afin de pouvoir les optimiser. L'abstraction de circuit en *Marked Graph* est assez direct :

– Les blocs fonctionnels deviennent des nœuds de calculs.

– Les fils d'interconnexion deviennent des canaux à latence insantannée.
– Les fils d'interconnexion contenant des registres deviennent des canaux à latence non nule.
– Les latences de calculs restent des latences de calculs
– et enfin les latences de communications restent aussi des latences de communications

Le passage de la figure 1.1 à 1.2 de la section 1.1 illustre cette abstraction.

5.1 Processus d'égalisation

Soit G un graphe issu de l'abstraction d'un circuit électronique. L'algorithme d'égalisation de G est le suivant :

1. Lister les cycles élémentaires de G.
2. Calculer le débit de chacun de ces cycles.
3. Calculer la k-périodicité du graphe.
4. Ajouter les latences entières.
5. Déterminer l'ordonnancement des nœuds et des *Registres fractionnaires.*
6. Placer les *Registres fractionnaires*.
7. Optimiser l'initialisation du système.

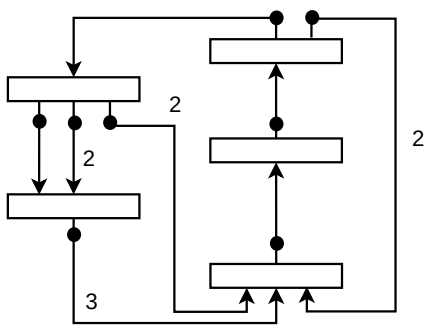


FIG. 5.1 – Example d'un système à égaliser.

La figure 5.1 représente un système à égaliser. Pour satisfaire un besoin de simplicité, comme dans le chapitre 1.1, cet exemple est volontairement abstrait. Les valeurs sur les canaux représentent les latences de communication. Nous allons utiliser cet exemple tout au long du chapitre pour illustrer le processus d'égalisation.

5.1.1 Lister les cycles :

Nous avons besoin d'énumérer tous les cycles élémentaires du système, puis de calculer leur débit afin de générer un système d'équation linéaire (étape suivante). Bien qu'en général, le nombre de cycles dans un graphe est exponentiel par rapport au nombre de nœuds, ce n'est que rarement le cas dans le

domaine de la modélisation de système sur puce, surtout pour des systèmes data-flow. La figure 5.1 nous montre les quatre cycles élémentaires de notre exemple. Le meilleur algorithm permettant de faire cela a une complexité exponentielle en fonction du nombre de nœuds. En revanche, la densité des graphes issu de circuit électronique est généralement assez faible. Les temps de calcul restent acceptables.

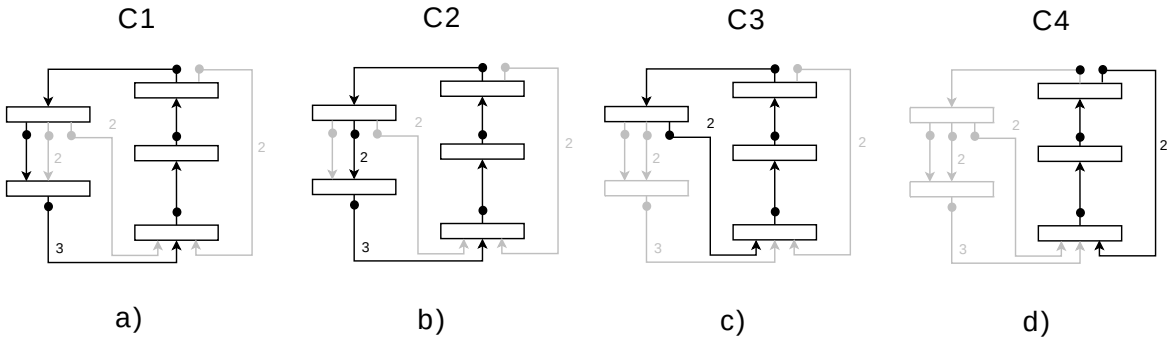


FIG. 5.2 – Liste des cycles élémentaires du système : a) C1, b) C2, c) C3, d) C4.

5.1.2 Calculer le débit :

Pour chacun des cycles élémentaires du système, on calcul son débit j/l (j est le nombre de jetons du cycle, l sa latence). Une fois le débit de chaque cycle connu, on cherche le(les) cycle(s) avec le pire débit. Ce débit est appelé débit critique. Tout cycle ayant ce débit est un cycle critique. Concernant notre exemple, on a :

- Le cycle C1 de débit $5/7$
- Le cycle C2 de débit $5/8$ est un cycle critique.
- Le cycle C3 de débit $4/5$
- Le cycle C4 de débit $3/4$

5.1.3 Calculer la k-périodicité du graphe.

Si on considère seulement les parties fortement connexes critiques du graphe, la formule de la propriété 2 peut déterminer la k-périodicité et la période de fonctionnement du graphe en régime permanent. Dans notre exemple, le graphe est 5-périodique de période 8 ($k = 5$,$p = 8$).

5.1.4 Ajouter les latences entières :

On utilise un solveur d'équation linéaire pour ajouter de la latence aux cycles rapides afin de réduire leur débit au plus proche du débit des cycles critiques. Dans un système k-périodique de période p, on peut ajouter sur un cycle C de débit j/l un nombre de latences entières égales à $\lfloor (p*j - k*l)/k \rfloor$. (Voir propriété 5) Dans le système d'équation, une variable est associée à un canal et représente le nombre de latences que

l'on doit lui ajouter pour égaliser le système (c'est un entier positif). On cherche à maximiser la somme des variables c'est à dire, à ajouter autant de latences que possible. Dans l'exemple, on a :

- Pour le cycle C1 : $a_1 + a_2 + a_4 + a_5 + a_8 \leq 1$ ($\lfloor(8 * 5 - 5 * 7)/5\rfloor$)
- Pour le cycle C2 : $a_1 + a_2 + a_4 + a_6 + a_8 \leq 0$ ($\lfloor(8 * 5 - 5 * 8)/5\rfloor$)
- Pour le cycle C3 : $a_1 + a_2 + a_4 + a_7 \leq 1$ ($\lfloor(8 * 4 - 5 * 5)/5\rfloor$)
- Pour le cycle C4 : $a_1 + a_2 + a_3 \leq 0$ ($\lfloor(8 * 3 - 5 * 4)/5\rfloor$)
- On maximise $\Sigma_{i=1}^{9} a_i$.

Comme on peut le voir sur notre exemple présenté dans la figure 5.3, la simplification des équations nous mène à une solution triviale : On ajoute une latence sur a_5 et a_7, mais dès que les canaux appartiennent à plusieurs cycles non critiques, le choix du canal pour le placement de la (les) latence(s) justifie l'utilisation de solveur d'équation linéaire. De plus, il existe des cas où les bornes ne peuvent pas être atteintes. La section 5.3.2 détaille ce cas de figure. La résolution de système linéaire à solution entière a une complexité exponentielle.

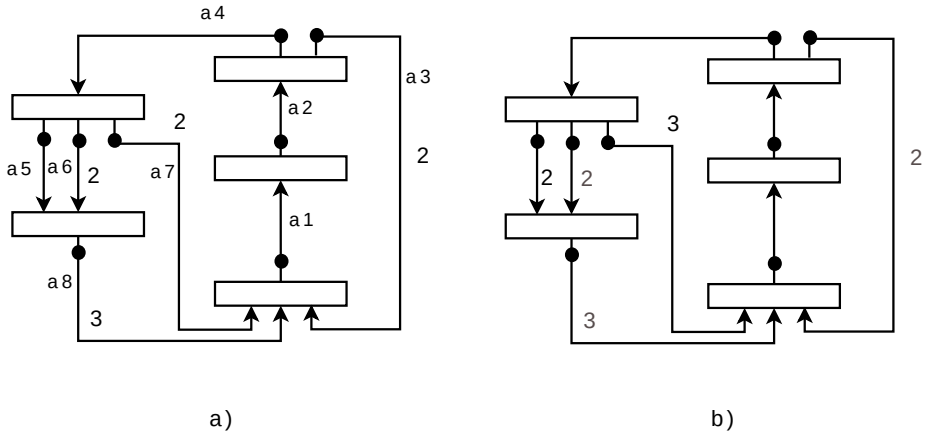


FIG. 5.3 – a) Système avant l'ajout de latence b) Système après l'ajout de latence.

Une fois les latences entières ajoutées, il faut recalculer la k-périodicité du graphe. L'ajout de latence peut transformer des cycles non critiques en cycles critiques. C'est le cas de $C1$ dans notre exemple. La topologie de la partie fortement connexe critique du graphe peut changer. Le rapport k/p restera le même ; la valeur de k et de p peuvent proportionnellement changer.

5.1.5 Déterminer les ordonnancements

Une fois les latences ajoutées, on simule le comportement du système suivant une politique d'ordonnancement au plus tôt pour obtenir l'ordonnancement du graphe, c'est à dire l'ordonnancement de chacun de ses nœuds. Dans la figure 5.4, le mot binaire inscrit dans un nœud indique ses instants d'activation (1) et de repos (0). Cette séquence décrit une période d'exécution. A la fin, la séquence reprend depuis le début. Comme tout système déterministe, après une phase initiale chaotique, le comportement du système devient régulier, k-périodique. Dans la figure 5.4, la phase d'initialisation du système n'apparait pas, elle a été rem-

placée par "...". Nous y reviendrons dans la figure 5.6. L'algorithm de simulation a une complexité égale au nombre de nœud multiplié par la longueur de la période du régime permanent.

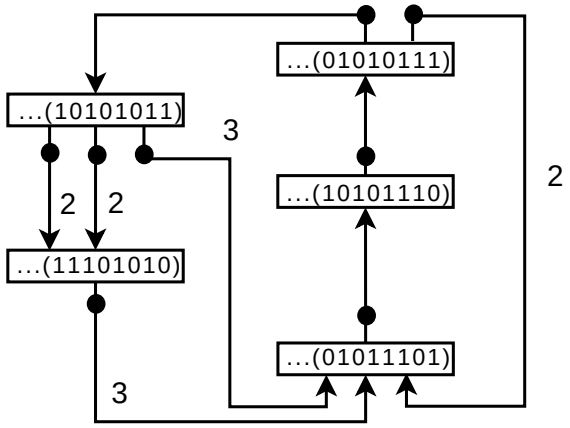


FIG. 5.4 – Ordonnancement des nœuds du système.

5.1.6 Placer les registres fractionnaires :

En cas d'égalisation parfaite, l'ordonnancement d'un nœud, représenté sous forme d'un mot binaire, est égal à l'ordonnancement du nœud précédent décalé d'un cran vers la droite. Si ce n'est pas le cas, un *registre fractionnaire* est nécessaire entre les deux nœuds. L'étude des différences entre les ordonnancements du précédent et du suivant permet de déterminer son taux d'utilisation ainsi que ses instants d'activation. On peut aussi déterminer cela par simulation en même temps que les ordonnancements des nœuds. A chaque fois qu'une donnée est retardée, elle est envoyée du registre principal du canal vers le *registre fractionnaire*. La simulation permet de déterminer les instants concernés. La figure 5.5 présente la répartition et les instants d'activations des *registres fractionnaires* nécessaires au fonctionnement de notre exemple. Le fonctionnement détaillé du *registre fractionnaire* est présenté dans la section 5.2 de ce chapitre.

Sur la figure 5.5, le *registre fractionnaire* est représenté comme un carré contenant le nombre de fois où il est actif par période. Le mot binaire associé est son ordonnancement.

Etant donné que la capacité du canal n'est pas limitée, le débit du graphe est maximum (k/p). La répartition des retards forme un vecteur de retards associé à G tel que le graphe est maintenant égalisé.

Il arrive dans certains cas qu'un canal ait besoin de retenir plus d'un jeton durant un instant. Pour cela il faut mettre un second *registre fractionnaire* en série sur le canal. Il sera actif seulement quand le canal doit stocker plus d'une donnée supplémentaire. A partir du moment où le graphe est pré-égalisé avant d'être égalisé, il existe toujours un autre ordonnancement pour le graphe qui garantit que tous les retards du canal peuvent être appliqués avec un seul *registre fractionnaire*. C'est le cas pour les ordonnancements balancés qui seront traités dans le chapitre 6. Etant donné que dans le processus d'égalisation, l'ordonnancement est déterminé par simulation, il est fixe et pas toujours optimal.

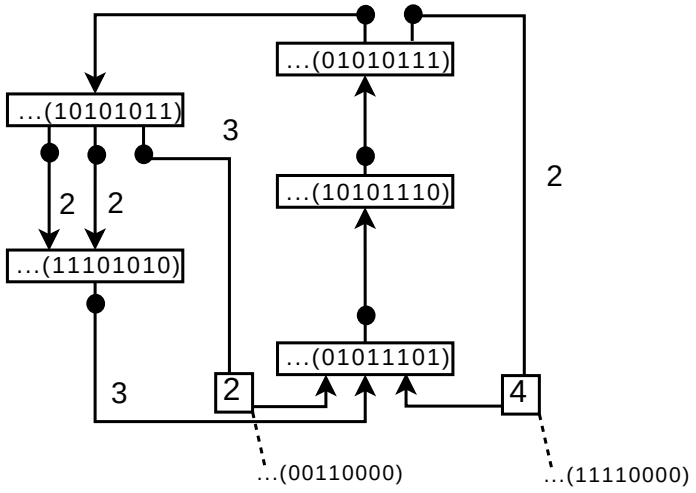


FIG. 5.5 – Système après la répartition des *registres fractionnaires*. Les valeurs numériques associées aux *registres fractionnaires* sont leur nombre d'utilisation par période. Le mot binaire associé donne leurs instants d'activation.

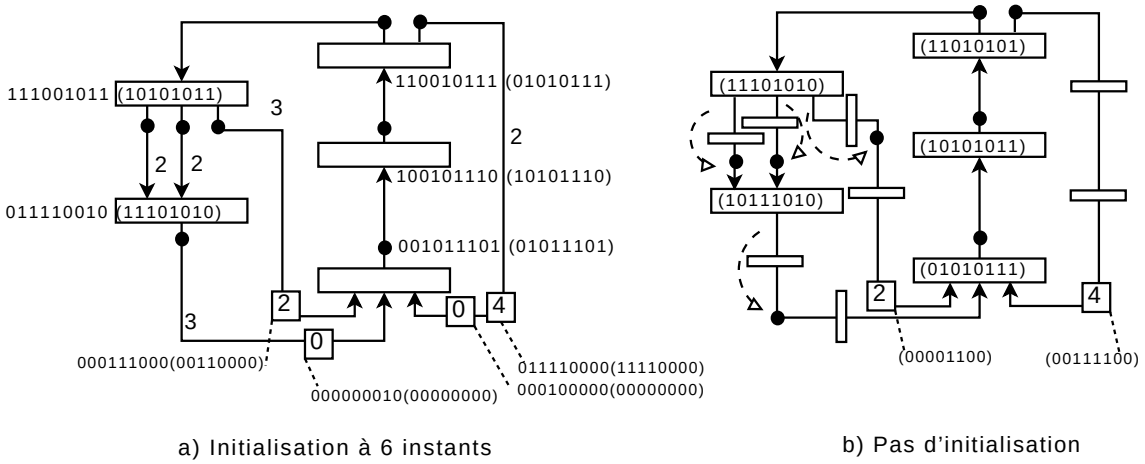
5.1.7 Optimiser l'initialisation :

La phase d'initialisation d'un *Marked Graph* peut être longue et nécessiter des ressources de stockage qui ne seront plus utiles durant le régime permanent. On peut voir sur la figure 5.6 a) qu'elle a une longueur 6 et nécessite deux *registres fractionnaires* supplémentaires qui ne seront pas utilisés durant le régime permanent (Ceux contenant la valeur "0"). Pour éviter cela, nous allons chercher à initialiser le système par une séquence d'exécution asynchrone. Le but est d'atteindre un des états du régime permanent à partir de l'état initial en utilisant seulement les ressources de stockage disponible le plus rapidement possible. La figure 5.6 b) montre qu'en un instant d'exécution, en déplacent seulement les quatre jetons mentionnés, on atteint le régime permanent. Malheureusement, il n'existe pas de méthode automatique qui permet de déterminer l'initialisation asynchrone la plus courte.

5.2 Le *Registre Fractionnaire*

La figure 5.7 présente le bloc diagramme d'un *registre fractionnaire*. Il s'insère entre un registre entier et un nœud de calculs. Il permet de stocker des données précédemment dans le registre entier et dont le nœud de calculs n'a pas encore besoin. Si par exemple le registre contient une donnée et en reçoit une nouvelle, il transmet la plus ancienne au *registre fractionnaire*. Son interface est constituée de trois signaux : deux entrants et un sortant.

- Le signal d'entrée val_in qui représente l'arrivée d'une donnée dans le *registre fractionnaire*.
- Le signal de sortie val_out qui représente l'émission d'une donnée par le *registre fractionnaire* vers le nœud de calculs.



a) Initialisation à 6 instants b) Pas d'initialisation

FIG. 5.6 – a) L'initialisation classique du système qui dure neuf instants peut être remplacé par b) le déplacement des quatre jetons de gauche afin d'atteindre directement le régime permanent.

– Le signal d'entrée $hold$, qui contrôle la rétention des données, vaut 1 quand la donnée arrivant dans le *registre fractionnaire* ou est déjà présente et doit être conservée à cet instant. Sinon elle vaut 0.

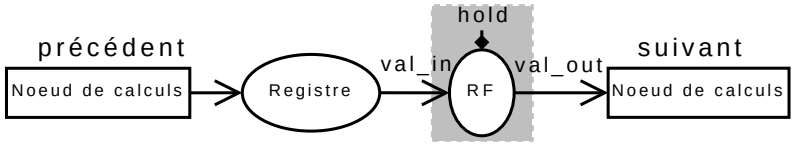


FIG. 5.7 – Bloc diagramme d'un *Registre Fractionnaire*.

La figure 5.8 a) présente le Synch-Chart [6] décrivant le comportement d'un *registre fractionnaire* et en voici la lecture :

Il contient un registre qui peut être plein ou vide. Son automate a donc deux états

– L'état $not(reg)$ quand son registre (reg) est vide. S'il reçoit une donnée (val_in) mais que $hold$ reste à 0, la donnée le traverse et est émise dans l'instant (val_out). S'il ne reçoit pas de donnée($not(val_in)$), il reste aussi dans le même état. En revanche, s'il reçoit une donnée(val_in) et que $hold$ est à 1, alors il la conserve et passe dans l'état reg.
– L'état reg quand son registre (reg) est plein. S'il ne reçoit pas de nouvelle donnée ($not(val_in)$) et que $hold$ reste actif, rien ne change. S'il reçoit une nouvelle donnée (val_in), il émet la plus ancienne (val_out) et conserve la nouvelle dans son registre. (dans ce cas $hold$ est forcément présent mais concerne la nouvelle donnée). Et enfin, s'il ne reçoit ni nouvelle donnée ($not(val_in)$) ni $hold$, il émet sa donnée (val_out) et retourne dans l'état $not(reg)$.

La figure 5.8 b) est la transcription de l'automate à état fini de la figure 5.8 a) sous forme de circuit. La figure 5.8 c) présente le flot des données du *Registre Fractionnaire*. Quand une donnée arrive, si $hold$ est présent, elle est stockée dans le registre ; sinon elle file directement en sortie. Concernant la commande du

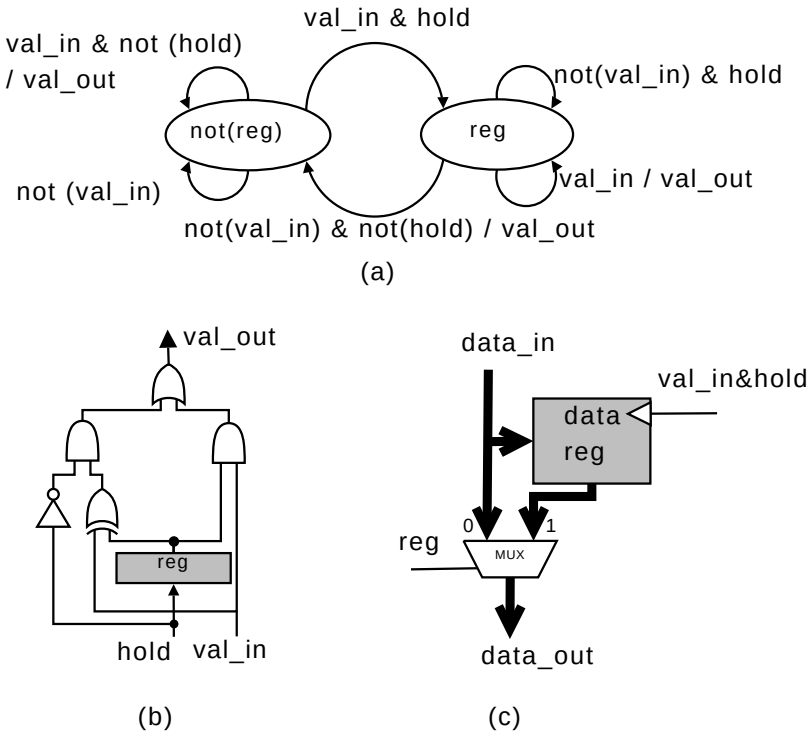


FIG. 5.8 – a) SynchChart, b) circuit et c) flot de données du *Registre Fractionnaire*.

multiplexeur, si une donnée est dans le registre, elle est prioritaire car plus ancienne.

5.2.1 Hold

La difficulté dans l'utilisation d'un *registre fractionnaire* est d'ordonnancer le signal $hold$. Un circuit de logique combinatoire peut être construit à partir de l'état du nœud précédent, du nœud suivant et du registre entier précédent le *Registre Fractionnaire*. Cela permet de gérer dynamiquement le système. D'un autre côté, la simulation qui nous a permis de placer les *Registres Fractionnaires* nous donne directement les instants d'activations du signal $hold$ comme sur la figure 5.5. On peut donc implanter statiquement l'ordonnancement du signal $hold$.

La figure 5.9 nous présente ces deux possibilités. Dans la figure 5.9 a), le signal $next$ vaut 1 quand le nœud de calculs suivant est actif dans l'instant. 0 sinon. Le signal $FRreg$ nous donne l'état du *registre fractionnaire* et le signal $pre(previous)$ vaut 1 quand le nœud de calculs précédent est actif à l'instant précédent. Ce dernier signal pourrait être remplacé par l'état du registre entier juste avant le *registre fractionnaire*. (Quand le nœud précédent travaille, il produit une donnée dans le registre)

La figure 5.9 b) est l'implantation statique de l'ordonnancement (signal $hold$) du *registre fractionnaire* de droite de la figure 5.5. Originellement, la période de fonctionnement est de longueur 8, mais il existe une sous période de longueur 4 qui nous permet de simplifier l'implantation. ($10001000^* = 1000^*$). Ce n'est pas le cas sur cet exemple, mais si l'ordonnancement du *registre fractionnaire* nécessite une phase

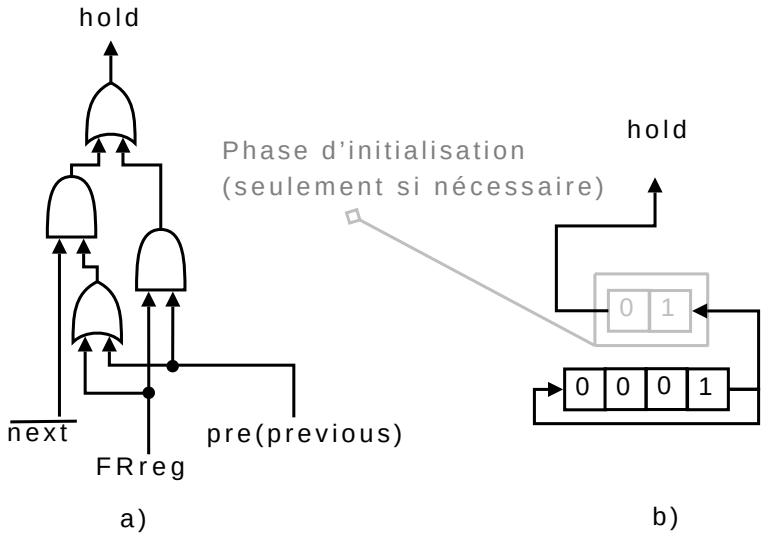


FIG. 5.9 – a) implantation dynamique du signal $hold$, b) implantation statique du signal $hold$.

d'initialisation, elle pourrait être implantée grâce aux registres à décalage qui apparaissent en gris sur la figure 5.9 b).

5.2.2 Vérifications formelles du comportement d'un *registre fractionnaire*.

Il y a deux conditions internes au fonctionnement du *registre fractionnaire* à vérifier :

- $hold \Rightarrow val_in + reg$: Pour retenir une donnée, il faut qu'elle soit présente.
- $val_in * reg \Rightarrow hold$: Si le *registre fractionnaire* contient une donnée et en reçoit une nouvelle, la nouvelle doit être retenue.

Dans le cas d'une implémentation statique du signal $hold$, la simulation nous assure que le signal est utilisé exactement et uniquement quand on en a besoin. Concernant l'implémentation dynamique, la figure 5.9 a) permet de vérifier ces conditions. Le signal val_in est équivalent au signal $pre(previous)$. Si $FRreg$ et $pre(previous)$ sont absents tous les deux, $hold$ vaut 0 et s'ils sont tous les deux présents, $hold$ vaut 1.

Il y a une condition externe au fonctionnement du *registre fractionnaire* à vérifier :

$$val_in \& reg \Rightarrow next \tag{5.1}$$

Il peut arriver que le nœud précédent prenne suffisamment d'avance par rapport au nœud suivant pour que deux données ou plus doivent être stockées simultanément dans le *registre fractionnaire*. Dans ce cas, on est obligé de mettre deux *registres fractionnaires* d'affilée. En revanche, étant donné que le système est égalisé, ce problème peut être évité si on atteint (par initialisation asynchrone) un autre régime permanent dans lequel l'ordonnancement des nœuds de calculs du graphe est plus balancé. C'est-à-dire que dans le mot binaire qui définit l'ordonnancement d'un nœud, les 1 et les 0 sont les plus étalés possible. Les mots balancés présentés dans le chapitre 2 sont des candidats idéaux à l'obtention d'un régime permanent où cette condition est respectée. La figure 5.10 illustre cette affirmation. Dans le graphe de gauche, on a besoin

de deux *registres fractionnaires* avant le nœud de reconvergence. Dans celui de droite, un seul suffit. On peut passer d'un graphe à l'autre seulement en déplaçant des jetons. Le chapitre 6 présente la méthodologie qui permet d'obtenir des graphes ayant un régime permanent balancé. On peut noter qu'obtenir un régime permanent où l'ordonnancement des nœuds du graphe est balancé est une condition suffisante au respect de cette hypothèse mais elle n'est pas toujours nécessaire.

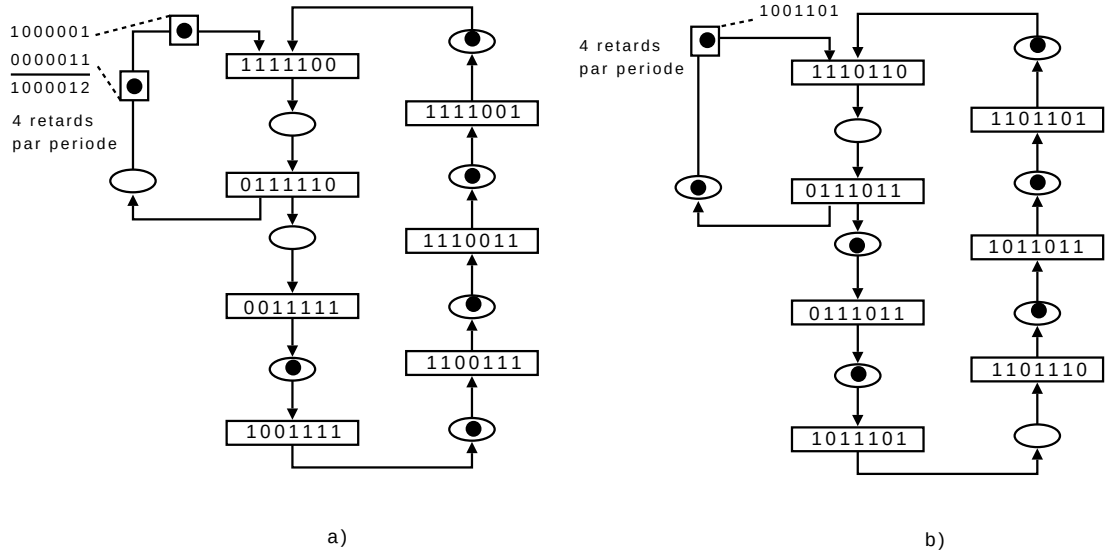


FIG. 5.10 – Le même système est ordonnancé différemment. a) On a besoin de 2 *registre fractionnaire*, b) un seul suffit.

5.3 Problème d'optimisation

Nous allons discuter dans cette section de nos choix de conception du processus d'égalisation et nous allons les confronter à des solutions alternatives.

5.3.1 Calcul du débit

Afin de trouver le débit du cycle critique, on commence par chercher tous les cycles puis par calculer leur débit. Une alternative à cet algorithme plus que coûteux, est l'algorithme "minimum cycle mean ratio problem" ([30] pour voir ces tests de performances) qui permet de trouver le débit du graphe et le(les) cycle(s) critique(s) et donc de déduire la k-périodicité du graphe ; et tout cela en un temps polynomial.

En revanche, on a besoin de tous les cycles pour construire le système d'équation linéaire qui nous permet d'ajouter les latences entières. Tant qu'une solution alternative à ce dernier problème ne sera pas trouvée, la recherche de tous les cycles sera nécessaire.

5.3.2 Ajout de latences

Atteignabilité de la solution au système d'équation linéaire Il existe des topologies, comme dans la figure 5.11 où certains cycles sont suffisamment rapides pour que nous pourrions théoriquement ajouter au moins une latence entière, mais pour lesquels la topologie du système est telle qu'il n'existe aucun canal appartenant à ce cycle pouvant accueillir cette latence. La raison est que tous ses canaux appartiennent aussi à d'autres cycles plus lents qui ne supporteraient pas l'ajout de latences entières. Dans la figure 5.11, le graphe a un débit de $3/16$. le cycle extérieur de débit $1/4$ pourrait être ralenti mais chacun de ses quatre canaux est partagés avec un autre cyles de débit $1/5$, qui ne peut pas être ralenti. En conséquence, la latence entière, que l'on a pas pu placer, est explosée en plusieurs latences rationnelles (*registres fractionnaires*) réparties sur le cycle extérieur. Le X dans les mots d'ordonnancement signifie un moment d'inactivité (0) pendant lequel le *registre fractionnaire* du canal entrant représenté en gras sur la figure est utilisé.

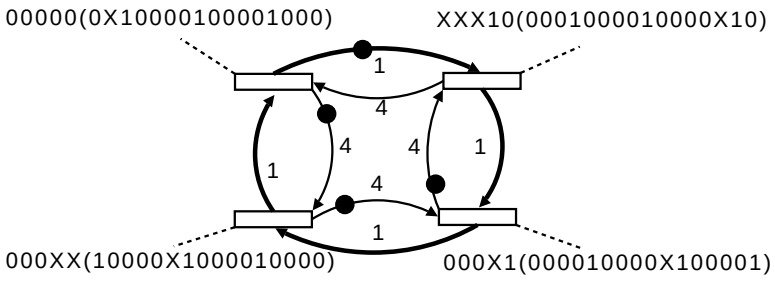


FIG. 5.11 – Système pour lequel l'ajout de latences entières ne peut pas amener tous les cycles le plus près possible du débit critique.

Le corollaire 7 montre que dans le graphe pré égalisé, même si tous les cycles n'ont pas un débit aussi proche que possible du débit du graphe, aucune latence entière ne peut être ajoutée.

Critère d'optimisation additionnelle Il existe plusieurs solutions équivalentes au système d'équations linéaires. Par exemple, si un nœud a une entrée et deux sorties, une latence ajoutée sur son canal entrant peut être remplacée par une latence sur chacun des canaux sortants et inversement. On pourrait chercher parmi l'ensemble des solutions équivalentes celle qui satisfasse un critère d'optimisation pré décidé. On peut, par exemple, souhaiter que l'égalisation se fasse en ajoutant un minimum de latence entière. Nous concernant, notre critère d'optimisation est double. Dans un premier temps, on voudrait chercher une répartition des latences entières qui minimisera le nombre des *registres fractionnaires*. Ensuite, parmi l'ensemble des solutions optimales, on choisira celle qui minimise aussi le nombre de latences entières. Etant donnée l'interdépendance "supposée" de ces deux critères d'optimisation, nous avons décidé de laisser ce problème ouvert afin de nous consacrer aux problématiques initiales de notre sujet.

Répartition des *registres fractionnaires* en utilisant le solveur d'équation linéaire Jusqu'à maintenant, on utilisait les résultats de la simulation pour placer les *registres fractionnaires*. Mais comme l'ajout de latences entières, l'ajout de latences rationnelles (représenté par les *registres fractionnaires*) peut être fait grâce au solveur d'équation linéaire. Soit (1) en même temps que les latences entières, soit (2) après les

latences entières.

(1)Dans ce cas, un seul système d'équation nous retourne des valeurs dont la partie entière représente le nombre de registre entier à ajouter au canal et le reste le nombre de retards associés au canal. Malheureusement, cette méthode de résolution a tendance à démultiplier les recours au *registre fractionnaire* ; un cycle peut se voir attribuer plusieurs *registres fractionnaires* qui pourraient être recombinés en un registre entier.

(2) Le problème de cette approche, qui existe aussi dans l'approche (1), est qu'il existe plusieurs solutions équivalentes. A chacune, on peut associer un régime permanent particulier mais il se pose le problème de l'atteignablité de ce régime. Partant de l'état initial du système, l'initialisation ASAP nous mène au régime permanent "primaire" ou "naturel". Pour atteindre les autres régimes permanents possibles, il faut faire une initialisation asynchrone vers un des états du régime permanent souhaité.

5.4 Propriété de correction

5.4.1 Validation du processus d'égalisation

Les propriétés de correction suivantes servent à montrer que notre processus d'égalisation permet bien d'égaliser le graphe. Le théorème 4 montre que le solveur d'équation linéaire utilisé pour ajouter un maximum de latence entière permet bien de construire un graphe pré-égalisé. Puis que la simulation permet d'obtenir un vecteur de répartition des retards correct et des ordonnancements pour tous les nœuds du graphe et qu'en conséquence, le graphe est égalisé.

Théoréme 4. *Soit G un graphe k-périodique de période p auquel on a ajouté des latences entières en suivant la méthode de la section 5.1.4. G est pré-égalisé ou $\mathbb{N}$-égalisé.*

De plus, G contient des registres fractionnaires *placés et ordonnancés selon la méthode de simulation des sections 5.1.5) et 5.1.6). G est égalisé ($\mathbb{Q}$-égalisé).*

Démonstration. $\mathbb{N}$-égalisé :
Pour tout canal, il existe au moins un cycle de débit (j/l) tel que $0 \leq p * j - k * l < k$. Ce qui peut être réécrit en $j/l \geq k/p > j/(l+1)$.

$\mathbb{Q}$-égalisé :
La capacité de chaque canal est suffisante pour ne pas restreindre l'exécution des nœuds du graphe à cause d'une incapacité de stockage en sortie du nœud. La capacité des canaux est supposée infinie durant la simulation puis est réduite au strict nécessaire après (La propriété 4 de safety nous garantit un fonctionnement à débit optimal si la capacité de tous les canaux est au moins égale au plus petit nombre de jetons dans les cycles passant par lui). Donc le comportement du graphe est k-périodique de période p. Tous les nœuds du graphe fonctionnent à un débit de k/p : tous les cycles passant pas un nœud de débit k/p ont un débit de k/p. G est $\mathbb{Q}$-égalisé. □

Pendant la pré-égalisation d'un graphe, il peut arriver qu'un cycle rapide ne soit pas ralenti alors que son débit l'aurait permis. Cela arrive quand aucun de ses canaux ne peut être allongé car chacun d'eux

appartient aussi à un cycle plus lent. Malgré cela, on est sûr qu'aucune latence entière ne pourrait être ajoutée au graphe. Le corrolaire 7 formalise ce résultat.

Corollaire 7. *Soit G un graphe k-périodique de période p $\mathbb{N}$-égalisé pour tous les* registres fractionnaires *appartenant à la partie fortement connexe de G, durant le régime permanent, il existe au moins un jeton par période qui n'est pas retenu par le* registre fractionnaire.

Démonstration. Soit c un cycle de G de débit j/l. Si $j * p - l * k < k$ au moins un jeton par période n'est par retenu. Si $j * p - l * k \geq k$, pour tout les canaux de c, il existe un cycle c' tel que $|c'|_1 * p - |c'| * k < k$. ($0 \leq j * p - l * k < k$ est une réécriture de la double inéquation de la définition 13) □

Au cours de l'exécution, un canal de communication doit à certains moments retenir des jetons jusqu'à ce que le nœud en sortie du canal en ait besoin. Les *registres fractionnaires* sont là pour ça. On a vu dans la section 5.1.6 que plusieurs *registres fractionnaires* peuvent être nécessaires sur le même canal pour retenir plusieurs jetons au même instant. Le lemme suivant borne le nombre de *registres fractionnaires* en fonction du nombre de retards que doivent subir les jetons sur ce canal.

Lemme 16 (Quantité maximum de *registres fractionnaires* par canal)**.** *Soit G un graphe k-périodique de période p et ($\mathbb{Q}$-égalisé).*

Soit c un canal de G qui applique r retards (au total) aux jetons passant par lui.

le nombre de registres fractionnaires *nécessaires sur ce canal est borné par $\lfloor\sqrt{r}\rfloor$.*

Démonstration. Supposons que le canal c veuille retenir n valeurs. Il doit en retenir 1, puis 2, puis ...n puis $n-1$ ... puis 2 puis 1. Au total, il doit en retenir $\Sigma_{i=1}^{n-1} 2 * i + n = n(n-1) + n = n^2 - n + n = n^2$. Pour tout $n^2 \leq c < (n+1)^2$, le nombre maximum de *registres fractionnaires* nécessaire sur ce canal est $\lfloor\sqrt{n}\rfloor$. □

5.4.2 Modélisation des canaux de communication

La simulation du système nous a permis de déterminer, pour chaque canal, le nombre de jetons maximum qui y sont stockés. Pour un canal c, on appelle cette quantité $J_{max}(c)$. A partir de cette information, on a pu trouver une modélisation finie d'un canal comme un registre simple suivi de $J_{max}(c) - 1$ *registre(s) fractionnaire(s)*. D'un point de vue modèle, on a remplacé nos canaux à capacité de stockage infinie par des canaux à capacité finie.

Le comportement du canal a la particularité que s'il est plein, $\bullet c$ peut s'exécuter à condition que $c\bullet$ s'exécute aussi. Ce comportement est celui d'un canal à capacité instantanée tel que défini dans la section 3.4.2. Comme la capacité du canal c est égale à $J_{max}(c)$, on est sûr que si $\bullet c$ est exécutable, alors $c\bullet$ l'est aussi.

On a donc l'assurance que, contrairement aux exemples de la section 3.4.1, la capacité ne limitera pas le débit. Le comportement du système à capacité bornée est donc conforme à la simulation.

5.5 Conclusion

A partir d'un réseau de processus modélisant le fonctionnement d'un système sur puce formé de composants IP divers et ayant des problèmes de latence sur ses fils d'interconnexions, on a élaboré un processus permettant d'obtenir un ordonnancement statique et de borner la taille des ressources de communications du système.

Pour cela, nous avons commencé par analyser la topologie du graphe (cycles, débits) puis par lui ajouter autant de latences entières que possible. Ensuite, nous avons simulé le comportement du système afin d'obtenir l'ordonnancement de chacun de ses nœuds et les capacités de chacun de ses canaux.

Comme nous le montre le dernier lemme de ce chapitre (lemme 16) il nous arrive d'avoir besoin de plusieurs registres fractionnaires sur un même canal pour pouvoir retenir plusieurs jetons au même instant. Dans le chapitre suivant, nous allons chercher à obtenir un régime permanent où il suffit au maximum d'un *registre fractionnaire* pour appliquer tous les retards d'un canal. Pour cela nous allons construire un ordonnancement conforme à nos spécifications formé exclusivement de mots balancés.

Chapitre 6

Régimes stationnaires balancés

Dans ce chapitre, on va chercher un ordonnancement calculé analytiquement (sans simulation). On a vu que, intuitivement, si les ordonnancements des nœuds d'un graphe sont des mots binaires équitablement répartis (entre les **0** et les **1**), on minimise la capacité de stockage des canaux du graphe. On va donc considérer la construction d'un ordonnancement balancé (def. 5)

Etant donné G, on connait sa période k et sa périodicité p, donc on connait l'ensemble $\mathbb{S}_p^k$ des mots balancés candidats à l'ordonnancement des nœuds de G. L'objectif sera d'attribuer à chaque nœud un ordonnancement balancé en accord avec le modèle d'exécution "au plus tôt". Ceci va demander l'introduction de latences entières et de *registres fractionnaires* comme auparavant (processus d'égalisation), mais à des endroits calculés analytiquement. C'était déjà le cas pour les latences entières, pas pour les fractionnaires.

Nous allons prouver un certain nombre de résultats :

Théoréme 5 (Existence du régime permanent balancé)**.** *Pour tout graphe G de marquage M, il existe un ordonnancement balancé de G à partir du marquage M' équipollent à M.*

Démonstration. Le chapitre 6 décrit le processus visant à construire l'ordonnancement balancé d'un graphe donné. Au fur et à mesure des étapes du processus, les preuves de validation et de vérification de la solution seront apportées. La section 6.3 contient la preuve de ce théorème. □

Théoréme 6 (Atteignabilité du régime permanent balancé)**.** *Pour tout graphe G de marquage M, il existe une exécution asynchrone de M vers M' où M' est un marquage balancé. (Définition 20)*

Démonstration. La preuve de ce théorème est repoussée à la section 6.3.6 □

Théoréme 7 (Capacité des canaux d'un graphe en régime permanent balancé)**.** *Soit G un graphe $\mathbb{Q}$-égalisé ayant un ordonnancement balancé. Pour tout canal nécessitant des retards (définition 25), un et un seul* registre fractionnaire *suffit à les appliquer.*

Démonstration. La preuve de ce théorème est repoussée à la section 6.3.6 □

Nous allons dans un premier temps (section 6.2) détailler le processus qui permet d'ordonnancer un graphe avec un régime permanent balancé. Pour chacune des étapes, nous justifierons leur bon fonctionnement en nous référant à des résultats préétablis dans les chapitres précédents et à ceux présentés dans la section 6.3.

6.1 Bibliographie

Si on regarde la littérature, on peut y voir que nous ne sommes pas les premiers à chercher à optimiser certaines caractéristiques d'un graphe en utilisant les mots balancés. Ici, nous cherchons à minimiser la taille des ressources de stockage en concervant un débit maximum. Les mots balancés servent d'ordonnancement. En 1995, Bruce Hajek est le premier à y avoir pensé dans [1, 2, 35].

Ensuite, dans [4, 33], B. Gaujal, E. Altman et S. Bhulai essayent de minimiser les pertes de données dans un graphe à capacité de stockage fixe en routant ces données grâces à des mots balancés

Dans [41], J. Mairesse et L. Vuillon traitent un problème similaire.

6.2 Méthodologie générale

A partir d'un graphe, on peut calculer sa période (p) et sa périodicité (k). De là, on peut construire l'ensemble $\mathbb{S}_p^k$ des mots balancés de longueur p contenant k **1** . L'essence de la méthode consiste à fixer l'ordonnancement d'un des nœuds du graphe et d'en déduire l'ordonnancement de tous les autres nœuds. Puis de déduire l'ordonnancement des *registres fractionnaires* du graphe et enfin de calculer un marquage correspondant à ces ordonnancements. L'étape finale consiste à trouver une phase d'initialisation asynchrone du marquage initial du graphe vers son régime permanent calculé.

6.2.1 Ajouter les registres fractionnaires :

Voici la première étape particulière à ce processus. Dans le chapitre 5, la simulation du comportement du graphe suivant une politique au plus tôt nous menait à un régime permanent avec une répartition particulière fixe et définitive des *registres fractionnaires*. Ici, nous allons construire un système d'équation linéaire à solution entière pour obtenir une répartition possible des *registres fractionnaires*. A partir de cette solution primaire, on va pouvoir déplacer, regrouper ou diviser les *registres fractionnaires* en fonction de contraintes ou de critères d'optimisation.

Obtenir la solution primaire Soit G un graphe k périodique de période p et pré-égalisé. On associe à G un vecteur d'entier L qui à chaque canal de G lui associe un nombre de retard par période d'exécution. L est tel que pour tout cycle C de G contenant j jetons et de latence l, la somme des retards associés aux canaux de C est de $p * j - k * l$.(propriété 5). On peut donc construire un système d'équation linéaire à solution entière où chaque variable représente un élément de L. et où chaque équation donne le nombre de délai à appliquer à un cycle. Le système permet de répartir ces délais sur les différents canaux des cycles en respectant les contraintes.

Si on reprend l'exemple du chapitre 5 représenté dans la figure 6.1, on a :

- Pour le cycle C1 : $a_1 + a_2 + a_4 + a_5 + a_8 \leq 0$ $(8 * 5 - 5 * 8))$
- Pour le cycle C2 : $a_1 + a_2 + a_4 + a_6 + a_8 \leq 0$ $(8 * 5 - 5 * 8))$
- Pour le cycle C3 : $a_1 + a_2 + a_4 + a_7 \leq 2$ $(8 * 4 - 5 * 6))$
- Pour le cycle C4 : $a_1 + a_2 + a_3 \leq 4$ $(8 * 3 - 5 * 4))$
- On maximise $\Sigma_{i=1}^{9} a_i$.

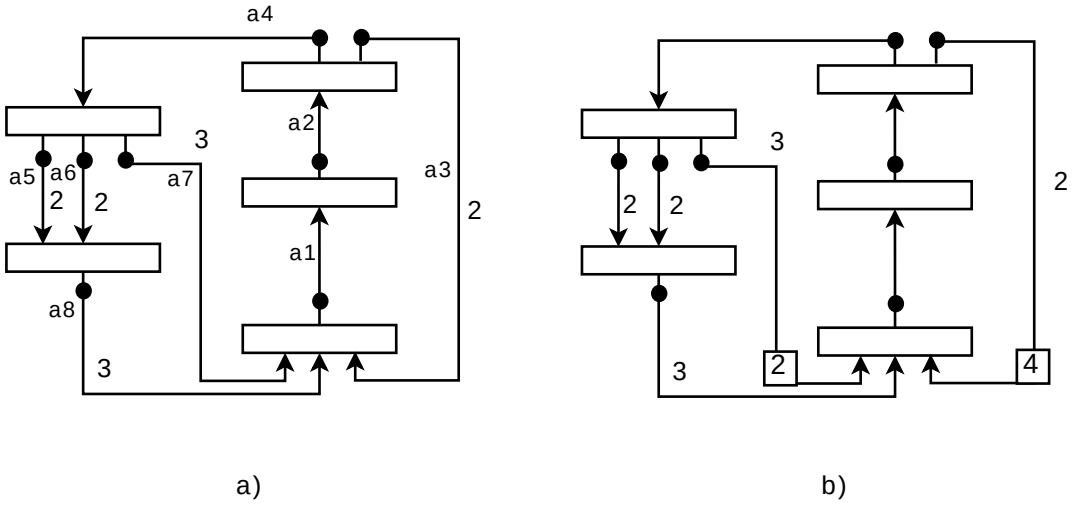


FIG. 6.1 – Ajout des *registres fractionnaires* grâce au solveur d'équation linéaire a) Avant l'ajout, b)Il faut 6 délais répartis en deux *registres fractionnaires* sur les canaux a3 et a7.

Ce système a toujours une solution exacte (les bornes des contraintes sont atteintes) car l'existence d'une solution est une condition nécessaire à l'existence d'un régime permanent. Étant donné que le graphe à partir duquel on construit le système est clos, son exécution est déterministe et donc ultimement périodique : il existe toujours une solution. Le lemme 18 formalise ce résultat. Placer les *registres fractionnaires* nécessite de résoudre un système linéaire à solutions entières. C'est un problème de complexité exponentielle.

Vecteur de retards associés à G Le vecteur d'entiers qui à chaque canal associe un nombre de retards calculés par le systéme d'inéquation linéaire est appelé vecteur de retards associé à G.

Définition 33 (Vecteur de retards associés à G)**.** *Soit G un graphe k-périodique de période p. On associe à G un vecteur d'entier L qui à chaque canal de G lui associe un nombre de retard par période d'exécution. L est appelé vecteur de retards associé à G. $\forall c$, un canal de G, $L(c)$ donne le nombre de retard associé au canal c.*
$\forall C$, un cycle de G, $L(C)$ donne la somme des retards sur les canaux de C. $\forall path$, un chemin de G, $L(path)$ donne la somme des retards sur les canaux de $path$.

Lemme 17. *Soit G un graphe k-périodique de période p. Soit L le vecteur de retards associé à G. L est tel que pour tout cycle C de G contenant j jetons et de latence l, la somme des retards associés aux canaux de C est $L(c) = p * j - k * l$. G est $\mathbb{Q}$-égalisé*

Démonstration. Pour tout cycle C de G contenant j jetons et de latence l, le débit de C en prenant en compte les retards est de

$$\frac{j}{l+(p*j-k*l)/k} = \frac{j*k}{(k*l+p*j-k*l)} = \frac{j*k}{(p*j)} = k/p$$

□

Remarque 16. *Un graphe, k-périodique de période p, est considéré comme égalisé soit parce que chacun de ses cycles a un nombre de jetons qui est un multiple de k, et une latence qui est un multiple de p, tel que le débit du cycle est k/p, soit parce qu'on a associé un vecteur de répartition de retards au graphe tel que, pour tout cycle C de G contenant j jetons et de latence l, la somme des retards associés aux canaux de C est $p*j-k*l$. Le premier cas étant un cas particulier du second, dans la suite du manuscrit, on va considérer que si un graphe est égalisé ($\mathbb{Q}$-égalisé), c'est qu'il existe un vecteur de retards associé au graphe.*

Repositionnement Comme pour les latences entières, si toutes les entrées d'un nœud ont un *registre fractionnaire* et que chacun est utilisé au moins n fois, n délais peuvent être supprimés de chacun de ses *registres fractionnaires* pour être placés dans des *registres fractionnaires* sur chacun des canaux en sortie du nœud. La figure 6.2 illustre cela. De a) à b), un retard est pris sur chacune des entrées du nœud le plus haut, et est propagé en sortie. Puis, de b) à c), les deux retards sur la boucle, en haut à droite, sont regroupés ; les deux retards en entrée du nœud du milieu sont déplacés sur son unique sortie.

Pour chacun des chemins de la figure 6.2, le nombre de retards est le même dans a), b) et c). En revanche, la quantité de *registres fractionnaires* varie. La possibilité de repositionner les *registres fractionnaires* permet d'optimiser certains critères comme le nombre total de *registres fractionnaires*.

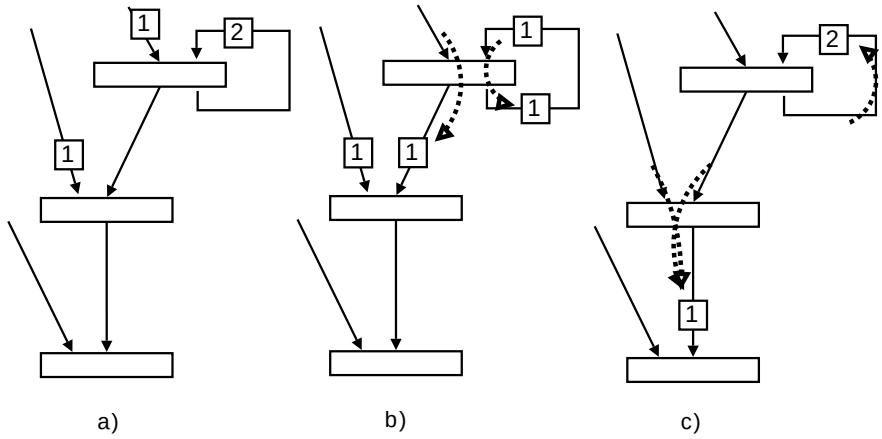


FIG. 6.2 – Repositionnement des délais des *registres fractionnaires*.

Répartition "au plus tard"

Définition 34 (Répartition "au plus tard")**.** *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G.*

On appel répartition "au plus tard" la répartition des retards sur les canaux de G décrit par le vecteur L s'il est tel que pour tout nœud n de G, au moins un des canaux d'entrée de n a un nombre de retards par période de 0.

Formellement, $\forall n$ un nœud du graphe, $\exists c \in \bullet n$ tel que $L(c) = 0$.

Le lemme 21 montre que pour un graphe pré-égalisé, il existe une et une seule répartition des jetons dite au plus tard (definition 34). Le lemme 20 montre qu'à partir de n'importe quelle répartition des retards, on peut obtenir la répartition au plus tard grâce à l'opération de repositionnement décrite dans ce paragraphe et formalisée dans la définition 39. Il sera intéressant, dans les sections 6.2.5 et 6.3.6, de considérer que la répartition choisie pour les retards est celle dite "au plus tard".

6.2.2 Ordonnancer les nœuds du graphe :

Maintenant que le graphe est égalisé, on peut l'ordonnancer en fixant l'ordonnancement et en déduisant celui de ses voisins. Puis, de proche en proche, de tous les nœuds. Dans un ordonnancement au plus tôt, l'ordonnancement d'un nœud est le même que celui de son prédécesseur décalé d'un instant. On a vu dans le chapitre 2 que l'utilisation d'un retard sur un canal revient à une transposition élémentaire sur l'ordonnancement du nœud en tête du canal. On rappelle que la transposition élémentaire est équivalente à une rotation de $-\alpha$ (Théorème 2).

Définition 35 (Règle de calcul des ordonnancements balancés d'un graphe)**.** *G est un graphe $\mathbb{Q}$-égalisé k-périodique de période p. Soit L le vecteur de répartition de retards associé à G. Soit c un canal de G. Soit $O_{\bullet c}$,$O_{c\bullet}$ l'ordonnancement de resp. $\bullet c$ et $c\bullet$.*

- $O_{c\bullet} = O_{\bullet c} \circlearrowleft 1 - L(c) * \alpha$.
- $O_{\bullet c} = O_{c\bullet} \circlearrowleft L(c) * \alpha - 1$.

Quand un nœud a plusieurs entrées (sorties), son ordonnancement peut se calculer à partir de l'ordonnancement de n'importe quel nœud qui le précède (succède) directement à condition que l'ordonnancement de ce dernier soit déjà connu (quelque soit le nœud prédécesseur (successeur) considéré, la solution sera la même). L'ordonnancement d'un nœud à plusieurs entrées (sorties) est égal à l'ordonnancement d'un de ses prédécesseurs (successeurs) retardés (avancés) d'un instant et auxquels on applique une rotation vers la gauche (droite) de taille α (pour un k et un p donné) par retard associé au canal. (voir def. 9 pour la définition de α)

Générer les ordonancements des nœuds du graphe se fait en temps linéaire par rapport au nombre de nœuds.

A partir de cette règle de propagation de l'ordonnancement, il nous suffit de fixer l'ordonnancement d'un seul nœud du graphe pour déterminer l'ordonnancement de tous les autres. Connaissant k et p, on peut générer l'ensemble $\mathbb{S}_p^k$, choisir arbitrairement un mot de l'ensemble et l'associer là aussi arbitrairement à un nœud du graphe. Puis d'utiliser la règle ci-dessus pour générer l'ordonnancement complet du graphe.

La figure 6.3 illustre notre règle de génération de l'ordonnancement du graphe. L'ordonnancement du nœud à deux entrées est le même qu'on le calcule à partir de son entrée de droite ou de gauche. Par rapport à son entrée de droite, l'ordonnancement a été retardé d'un instant (une rotation vers la droite : $1110 \circlearrowright 1 = 0111$). Par rapport à son entrée de gauche, l'ordonnancement à été retardé d'un instant puis rotaté de deux fois α vers la gauche (2*1=2). ($(1011 \circlearrowright 1) \circlearrowleft 2 = 1011 \circlearrowleft 1 = 0111$).

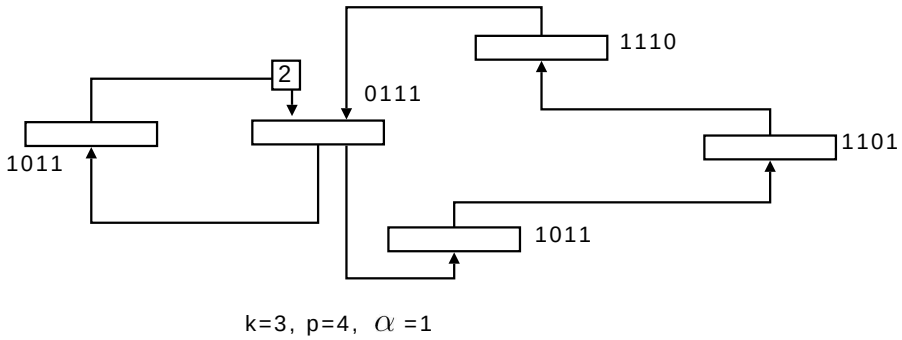


FIG. 6.3 – Propagation de l'ordonnancement des nœuds de proche en proche.

Le lemme 22 montre formellement que l'ordonnancement du graphe obtenu est cohérent. C'est à dire que l'ordonnancement d'un nœud à plusieurs entrées est effectivement le même quelque soit le nœud précédent considéré pour le calcul. De plus, le lemme 23 montre que le premier ordonnancement fixé peut être retrouvé par le calcul à partir des autres ordonnancements. Enfin, le lemme 24 montre que quelque soit le premier nœud et au premier ordonnancement choisi, on trouve des moments différents du même régime permanent. On peut donc retrouver tous les autres moments par simple rotation de tous les ordonnancements.

6.2.3 Ordonnancer les *registres fractionnaires* :

Une étape précédente nous permettait de placer les *registres fractionnaires* et de savoir combien de fois ils sont utilisés par période. Cette étape permet de déterminer à quel instant exactement les retards seront appliqués en fonction de l'ordonnancement des nœuds.

Le lemme 25 nous montre que dans un graphe d'ordonnancement balancé, si un jeton est retardé, ce ne sera que durant un seul instant. Etant donné que le graphe (sans vecteur de retards associé) est $\mathbb{N}$-égalisé, on sait que le nombre de retard par canal est inférieur à k. A partir de ce résultat, on peut définir l'ordonnancement d'un *registre fractionnaire* sur un canal c (ayant r retards à appliquer) comme un mot binaire de longueur p contenant r **1** . Ces r **1** représentent les r instants d'activations (les r retards). On ne retient jamais deux instants d'affilées le même jeton, si deux **1** se suivent dans l'ordonnancement du *registre fractionnaire*, cela concerne deux jetons différents successifs.

Définition 36 (Règle de calcul des ordonnancements des *registres fractionnaires* des canaux du graphe.)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p. Soit L le vecteur de répartition de retards associé à G. Soit v l'ordonnancement d'un* registre fractionnaire *actif r fois par période placé en entrée d'un nœud*

$n \in G$ d'ordonnancement O_n. $v(1)$ (resp $v(p)$) désigne le premier (resp. dernier) bit du mot v. $\exists! i \in [0, p[$ *tel que* $O_n \circlearrowleft i = sup(\mathbb{S}_p^k)$.
$\forall j \in [1, p]$, $v(j) = 1$ *ssi* $(j \mod p) \in \{p - i\,;\, p - i + \alpha\,;p - i + 2 * \alpha\,;\, ...\,;p - i + (r - 1) * \alpha\}$.

Générer les ordonancements des *registres fractionnaires* du graphe se fait en temps linéaire par rapport au nombre de canaux. Le nombre de canaux est une borne strictement supérieure au nombre de *registre fractionnaire*.

Si la répartition des *registres fractionnaires* est la solution particulière dite "au plus tard", pour tout nœud, il existe au moins une entrée qui n'utilise pas de *registres fractionnaires*. Dans ce cas, toutes les utilisations de *registres fractionnaires* par les autres entrées sont "légitimes" au sens où il y a effectivement un jeton qui arrive après les autres. Pour les autres répartitions de retard, il existe au moins un nœud tel que toutes ses entrées ont un *registre fractionnaire*. Suivant la définition 36, il seront tous actifs au(x) même(s) instant(s).

Le lemme 26 montre que l'ordonnancement des *registres fractionnaires* tel que défini dans la def. 36 coïncide avec celui des nœuds. C'est à dire que les *registres fractionnaires* sont actifs quand l'état courant nécessite d'appliquer un retard sur le canal.

6.2.4 Définir les états du régime permanent :

Maintenant que l'on a déterminé l'ordonnancement complet du graphe et de ses *registres fractionnaires*, nous devons déterminer à quels placements des jetons ces ordonnancements correspondent. Il existe p états différents du régime permanent et à chacun correspond un indice i tel que le graphe est dans l'état i après avoir exécuté les i premières instructions de l'ordonnancement de chaque nœud.

Définition 37 (Règle de calcul du marquage du graphe)**.** *Soit G un graphe k-périodique de période p et égalisé. Soit O son ordonnancement. Soit c un canal de G contenant un* registre fractionnaire*. Soit O_n l'ordonnancement de $n = \bullet c$ et O_{fr} l'ordonnancement du* registre fractionnaire *de c. Soit M_0, M_1,...,M_i,...,$M_p = M_0$ les états successifs du régime permanent de l'exécution de G selon l'ordonnancement O.*

$$M_i(c) = O_n(p + i) + O_{fr}(p + i) \tag{6.1}$$

(si c ne contient pas de registre fractionnaire*, $O_{fr} = 0^p$).*

- *le bit d'indice i de l'ordonnancement de chaque nœud donne l'état de ses canaux en sortie au marquage M_i. S'il est à* **1** *, le nœud vient de s'exécuter, les canaux de sorties contiennent un jeton. Sinon, il est vide.*
- *De même, le bit d'indice i de l'ordonnancement de chaque* registre fractionnaire *donne son état au marquage M_i. S'il est à* **1** *, le* registre fractionnaire *vient de retenir une donnée : il est plein. Sinon, il est vide.*

Générer le marquage des places du graphe se fait en temps linéaire par rapport au nombre de nœuds. Générer le marquage des *registres fractionnaires* du graphe se fait en temps linéaire par rapport au nombre de canaux.

On peut maintenant construire la séquence d'exécution $Exec$ du graphe G tel que :

Définition 38. $Exec = M_0 \xrightarrow{F_1} M_1 ... M_{i-1} \xrightarrow{F_i} M_i ... \xrightarrow{F_p} M_p = M_0$.

Avec M_i le i^{eme} marquage calculé grâce aux règles de la def. 37. F_i est l'ensemble des nœuds n tel que $O_n(i) = 1$.

Le lemme 27 montre que le nombre de jetons instanciés par cycle de cette manière pour chacun des marquages est conforme aux spécifications initiales du graphe. Il montre que le marquage initial et le marquage calculé sont équipollents.

Le lemme 31 montre qu'à partir du marquage calculé, le jeu des jetons est cohérent avec les ordonnancements calculés.

6.2.5 Trouver une initialisation asynchrone :

Comme dans la section **??** du processus d'égalisation, nous allons terminer notre construction par la recherche d'une séquence d'initialisation asynchrone la plus courte possible et surtout qui nécessite seulement les ressources déjà disponibles. Là encore, aucune méthode automatique ne permet d'obtenir cette initialisation. Le théorème 6 nous garanti seulement qu'il existe une séquence d'exécution entre l'état initial du système et l'un des états du régime permanent.

6.3 Propriétés de correction

6.3.1 Ajout des *registres fractionnaires*

Dans le processus d'égalisation, la répartition des retards est donnée par la simulation. Ici nous utilisons le solveur d'équation linéaire pour déterminer cette répartition. La fonction F(définition 39) et les lemmes 20 et 21 nous garantissent la possibilité d'atteindre la répartition "au plus tard" pour la répartition des retards à partir de la solution obtenue par équation linéaire.

Lemme 18 (égalisation par solveur linéaire)**.** *Soit G un graphe k périodique de période p et pré-égalisé. Soit L le vecteur de répartition des retards associé à G. L est obtenu en utilisant le système d'équation de la section 6.2.1. G est $\mathbb{Q}$-égalisé.*

Démonstration. Le système d'équation de la section 6.2.1 a toujours une solution maximale donc L est tel que pour tout cycle C de G contenant j jetons et de latence l, la somme des retards associés aux canaux de C est de $p * j - k * l$.. Le lemme 17 montre que si on associe L à G alors G est égalisé. □

Nous allons maintenant prouver qu'à partir de la solution du solveur d'inéquation, on peut la modifier pour obtenir une répartition "au plus tard" des retards (def 34). Cette répartition a la particularité d'être naturellement obtenue par simulation "au plus tôt" à partir d'un marquage balancé. Nous allons nous poser la question de l'atteignabilité de la répartition "au plus tard" à partir de la solution obtenue par le solveur d'inéquation linéaire.

Nous allons commencer par définir formellement la fonction de déplacement de retards (définition 39) que nous avons intuitivement introduite dans la section 6.2.1 grâce à la figure 6.2. Nous vérifierons que

cette fonction conserve la propriété de $\mathbb{Q}$-égalisation sur le graphe (lemme 19). Ensuite, nous allons montrer que la répartition "au plus tard" est accessible à partir de n'importe quelle répartition (lemme 20) grâce à l'utilisation répétée de la fonction de déplacement de retards. Enfin, nous prouverons que pour un graphe donné, la répartition "au plus tard" est unique (lemme 21).

Définition 39 (Fonction de déplacement de retards)**.** *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G.*

Soit F une fonction qui à L associe L' un autre vecteur de retards tel que : soit n un nœud du graphe, le nombre de retards minimum commun à tous les canaux d'entrées de n sont transmis sur chacun des canaux de sortie de n. Le nombre de retards sur n'importe quel autre canal de G reste inchangé.

Formellement, On note $L' = F(L, n)$ ou $L' = F_n(L)$ si soit $\{c_1, c_2...c_n\}$ les canaux d'entrés de n et $\{c'_1, c'_2...c'_m\}$. les canaux de sorties de n. $\forall i \in [1, n]$, $L(c_i) \geq r$.
$\forall i \in [1, n]$, $L'(c_i) = L(c_i) - r$.
$\forall j \in [1, m]$, $L'(c'_j) = L(c_j) + r$.
$\forall c$ un canal de G tel que $c \notin \{\bullet n \cup n\bullet\}$, $L'(c) = L(c)$.

Lemme 19 (Fiabilité de la fonction de déplacement des retards)**.** *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G. Soit n un nœud du graphe. Soit $L' = F(L, n)$. (F est la fonction de déplacement de retards) G associé à L est $\mathbb{Q}$-égalisé, alors G associé à L' l'est aussi.*

Démonstration. Soit n un nœud du graphe. $\{c_1, c_2...c_n\}$ ses canaux d'entrés et $\{c'_1, c'_2...c'_m\}$. ses canaux de sorties. $\forall i \in [1, n]$, $L(c_i) \geq r$. On a L' un autre vecteur de retards tel que $L' = F(L, n)$ Soit C un cycle de G passant par c_i puis c'_j ($i \in [1, n]$, $j \in [1, m]$).

$L'(C) = L(C) - r + r = L(C)$. □

Il existe une répartition dite "au plus tard" (def. 34) pour laquelle un nœud a toujours au moins une entrée qui n'utilise pas de retards. Cette répartition est intéressante car premièrement elle est accessible à partir de n'importe quelle autre répartition (lemme 20). Deuxièmement, elle est unique (lemme 21) et enfin parce que dans le cas où la simulation d'un graphe retourne un ordonnancement balancé, le vecteur de retards associé l'est aussi.

Notation 4. *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G.*

$L' = F(L)$ signifie qu'il existe un nœud n appartenant à G tel que $L' = F(L, n)$.

Le lemme 20 montre qu'à partir de n'importe quelle répartition de retards, on peut attendre la répartition "au plus tard". Pour cela, on va montrer que pour tout nœud du graphe, il existe un chemin provenant d'un cycle critique sur lequel le nombre de retards ne peut que décroitre par l'opération F (car un cycle critique ne contient aucun retard).

On conclut qu'au plus tard quand le nombre de retards sur le chemin est de 0, le nœud a forcément un canal d'entrée avec 0 retard et que cela ne changera plus par la fonction F.

Bien que nous n'ayons pas borné le nombre de fois où il faut appliquer la fonction F sur les nœuds d'un graphe pour obtenir la répartition "au plus tard", on sait que ce nombre est fini et que quelque soit le nœud, l'utilisation de la fonction F nous rapproche de la solution.

Lemme 20 (accessibilité de la répartition "au plus tard")**.** *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G.*

$\forall L$, $\exists L'$ la répartition dite "au plus tard" tel que $L' = FoFo...oF(L)$.

Démonstration. Soit n_c un nœud critique du graphe. Les cycles critiques n'ont par définition aucun retard sur leurs canaux. Donc $\exists c$ un canal d'entrée de n_c tel que $L(c) = 0$. De plus, $F(L, n_c) = L$ car le minimum retard commun aux canaux d'entrées de n_c est égal à $L(c) = 0$.

Soit n un nœud du graphe quelconque. Soit p_n le chemin le plus court allant d'un nœud critique de G vers n.

Soit $\{c_1, c_2, ...c_m\}$ l'ensemble des canaux d'entrée de n. r est le nombre minimum commun de retards à l'ensemble des canaux d'entrée de n.

$\forall n' \neq n$ un nœud du chemin p_n. On a $L_2 = F(L, n')$ et $L_2(p_n) = L(p_n)$ car les retards enlevés sur le canal de p_n en entrée de n' sont rajoutés sur le canal de p_n en sortie de n'. Etant donné que p_n commence par un nœud critique, le nombre de retards sur p_n ne peut pas augmenter par F, il est borné par $L(p_n)$. En revanche, on a $L_n = F(L, n)$ et $L_n(p_n) = L(p_n) - r$. L'utilisation de la fonction F sur le nœud n diminue le nombre de retards sur p_n.

L'utilisation de la fonction F sur d'autres nœuds de G peut amener des retards sur tous les canaux d'entrée de n. Dans ce cas, l'utilisation de la fonction F sur le nœud n en enlèvera le nombre minimum commun. Il réduira aussi le nombre de retards sur p_n.

Au plus tard, quand le nombre de retards sur p_n est 0, la répartition des retards dans G noté L_3 est telle que $\exists j \in [1, m]$ et $L_3(c_j)$ devient invariant par F. C'est à dire que $\forall n''$ un nœud du graphe, le nombre de retards sur ci_j pour la répartition $F(L_3, n'')$ est toujours de $L_3(c_j)$. □

Les définitions 40 et 41 sont nécessaires à la preuve du lemme 21. Pour montrer que la répartition "au plus tard" est unique, on va en supposer deux différentes, montrer que pour l'une d'elles, il existe un ensemble de chemins qui contiennent au moins i retards ($i \in \mathbb{N}$). On va conclure par l'absurde en montrant que parmi les chemins contenant au moins i retards, il y en a forcément au moins un qui ne contient aucun retard.

Définition 40. *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G tel que la répartition des retards est "au plus tard".*

On définit la relation EPN(empty previous node) qui à un nœud n de G associe un autre nœud n' de G. On dit que $n' = EPN(n)$ si $\exists path$ un chemin de n' vers n avec $L(path) = 0$.

Définition 41. *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retards associé à G tel que la répartition des retards est "au plus tard". Soit n un nœud du graphe.*

soit $c_0(n)$ la fonction qui retourne un des canaux entrant de n ne contenant aucun retard. Pour un nœud n, plusieurs canaux entrant peuvent être dénués de retard. Pour nos besoins, il nous suffit de n'en considérer

qu'un seul. Donc $c_0(n)$ retourne un de ses canaux arbitraitement.

Lemme 21 (Unicité de la répartition "au plus tard")**.** *Soit G un graphe k-périodique de période p $\mathbb{Q}$-égalisé. Soit L le vecteur de retard associé à G.*

Il existe un et un seul vecteur L associé à G tel que la répartition des retards est "au plus tard".

Démonstration. On suppose que L et L' sont deux vecteurs tels que G associé à L est égalisé, que G associé à L' est aussi égalisé et que la répartition des retards sur G suivant L ou L' est "au plus tard", .

Soit n_0 un nœud du graphe.

Si L et L' sont différents, il existe au moins un nœud n_o de G tel que pour un canal d'entrée de n_0 appelé c, $L(c) \neq L'(c)$. Disons que $L(c) = L'(c) + i$. De plus soit n' le nœud à la base de c, pour tout chemin allant d'un nœud critique vers n', le nombre de retards est le même suivant L et L'. Sinon, il existe un autre canal avec une différence dans le nombre de retards et pour lequel les chemins venant d'un nœud critique qui le précède ont le même nombre de retards dans L et L'.

On en déduit que pour tout chemin $path$ allant d'un nœud critique vers n_0, $L(path) = L'(path) + i$. Et pour tout chemin $path'$ allant de n_0 vers un nœud critique, $L(path') = L'(path') - i$. (Sinon, il existe au moins un cycle c tel que $L(c) \neq L'(c)$ dans ce cas, soit L soit L' n'est pas valide car il n'égalise pas G)

Dans la répartition associée à L, on note n_1 le nœud à la base de $c_0(n_0)$. On a $n_1 = EPN(n_0)$. Il existe n_2 tel que $n_2 = EPN(n_1)$ car $c_0(n_1)$ existe (on est dans la répartition au plus tard). De plus $n_2 = EPN(n_0)$.

Si on construit la suite des nœuds n_i tel que $n_{i+1} = EPN(n_i)$, étant donné que la répartition des retards est "au plus tard", cette suite est infinie. Mais comme le nombre de nœuds dans le graphe est fini, $\exists i, j$ avec $i > j \geq 0$ tel que $n_i = EPN(n_j)$ et $n_i = n_j$.

Le chemin de n_i vers n_j est un cycle critique et le chemin de n_j vers n_0 est un chemin d'un nœud du cycle critique (n_j) vers n_0 qui ne contient aucun retard. Or selon le vecteur de répartition des retards L, le nombre de retards doit au moins être de i pour tout chemin d'un nœud critique vers n_0. L n'est pas une répartition "au plus tard". □

6.3.2 Ordonnancements des nœuds de calculs

Le lemme 22 montre que la méthode de calcul des ordonnancements de la définition 35 permet d'obtenir une solution unique à partir d'un point fixe. Pour cela, on suppose l'ordonnancement d'un nœud connu, et on montre que l'ordonnancement d'un autre nœud accessible à partir du premier par plusieurs chemins est le même quel que soit le chemin considéré.

Lemme 22 (Convergence de la méthode de calcul des ordonnancements : Chemins parallèles)**.** *Soit $n1$ et $n2$ deux nœuds d'un graphe G k périodique de période p et $\mathbb{Q}$-égalisé. tel qu'il existe plusieurs chemins (p_1,p_2,...,p_m) allant de $n1$ à $n2$. Soit O_{n1} l'ordonnancement de $n1$.*

Soit $O_{n2}^{p_1}, O_{n2}^{p_2}, ..., O_{n2}^{p_m}$, les m ordonnancements possibles de $n2$ calculés à partir de O_1 par le chemin (p_1,p_2,...,p_m) en suivant les règles de la section 6.2.2.

On a $\forall i, j \in [1, m]$, $O_{n2}^{p_i} = O_{n2}^{p_j}$.

Démonstration. G étant fortement connexe, il existe au moins un chemin p de longueur L allant de $n2$ vers $n1$. On appelle $C1$ et $C2$ les cycles formés resp. par les chemins p suivi de p_1 et p suivi de p_2. La figure 6.4 montre ces deux cycles extraits du graphe G. $C1$ (resp. $C2$) contient $k1$ (resp. $k2$) jetons pour une longueur $L + L_1$ (resp. $L + L_2$). Le débit de $C1$ (resp. $C2$) est tel que $k1 * p - k * (L + L_1) = c + x$ (resp. $k2 * p - k * (L + L_2) = c + y$). Les c retards communs à $c1$ et $c2$ interviennent sur le chemin p. Les x retards de $C1$ (resp. les y retards de $C2$) interviennent sur le chemin p_1 (resp. p_2).

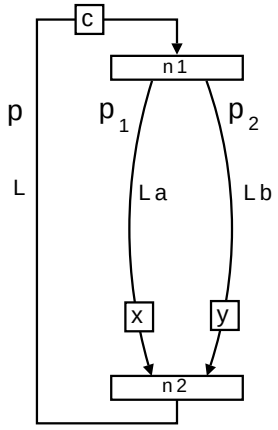


FIG. 6.4 – Deux cycles C_1 et C_2 de longueur resp. $L + L_1$ et $L + L_2$.

Montrer que $O_{n2}^{p_1} = O_{n2}^{p_2}$ équivaut à montrer que $O_{n1} \circlearrowleft L_1 - \alpha * x = O_{n1} \circlearrowleft L_2 - \alpha * y$
$\Leftrightarrow L_1 - \alpha * x \equiv L_2 - \alpha * y \mod p$.

Partant du système d'équation :

$$\left\{\begin{array}{l} k1 * p - k * (L + L_1) = c + x \\ k2 * p - k * (L + L_2) = c + y \end{array}\right\}$$

on obtient

$$\left\{\begin{array}{l} L_2 = L_1 + (x - y + p(k2 - k1))/k \\ y = x + p(k2 - k1) + k(L_1 - L_2) \end{array}\right\} \tag{6.2}$$

La quantité $L_2 - \alpha * y$ est donc égale à :

$$L_2 - \alpha * y = L_1 + (x - y + p(k2 - k1))/k - \alpha * (x + p(k2 - k1) + k(L_1 - L_2))$$

Ce qui peut être réécrit en

$$L_2 - \alpha * y = L_1 - \alpha * x + (x - y + p(k2 - k1))/k - \alpha * (p(k2 - k1) + k(L_1 - L_2))$$

Il nous reste à montrer que la quantité $A = (x - y + p(k2 - k1))/k - \alpha * (p(k2 - k1) + k(L_1 - L_2)) \equiv 0 \mod p$. Si dans A, on remplace $x - y$ par sa valeur dans l'équation 6.2 : $-p(k2 - k1) - k(L_1 - L_2)$ et qu'on simplifie, on obtient :

$$A = (L_2 - L_1) - \alpha * p(k2 - k1) - \alpha * k(L_1 - L_2)$$

Si on remplace α par sa valeur $(p^{-1} * p - 1)/k$, on obtient

$$\begin{aligned} A = L_2 - L_1 - \alpha p(k2 - k1) - (p^{-1}p - 1)(L_1 - L_2)k/k = \\ L_2 - L_1 - \alpha p(k2 - k1) - p^{-1}p(L_1 - L_2) + L_1 - L_2 = \\ -\alpha p(k2 - k1) - p^{-1}p(L_1 - L_2) = \\ -p\Big(\alpha(k2 - k1) + p^{-1}(L_1 - L_2)\Big) \end{aligned}$$

La quantité A est congrue à $-p \equiv 0 \mod p$. □

Lemme 23 (Convergence de la méthode de calcul des ordonnancements : Cycles)**.** *Soit O_n l'ordonnancement d'un nœud n dans un graphe G k périodique de période p $\mathbb{Q}$-égalisé. On suppose que n est le nœud auquel on a attribué un mot d'ordonnancement arbitraire $\in \mathbb{S}_p^k$, en suivant les règles de la section 6.2.2. On peut calculer l'ordonnancement de tous les nœuds du graphe. O_n peut être recalculé à partir de l'ordonnancement des nœuds qu'il a lui même générés.*

Démonstration. G est fortement connexe, il existe au moins un cycle C passant par n de longueur l contenant j jetons nécessitant r retards. Soit O'_n l'ordonnancement du nœud n calculé à partir de O_n.
On a $O'_n = O_n \circlearrowleft l - r * \alpha$. Intéressons nous à la quantité $l - r * \alpha$.
On sait que $j * p - k * l = r$ et que $\alpha * k \equiv -1 \mod p$. Si on remplace r par leur valeur, on obtient :

$$\begin{aligned} l - r * \alpha = \\ l - \alpha * (j * p - k * l) = l - \alpha * j * p + \alpha * k * l = \\ l - \alpha * j * p - l = -\alpha * j * p \end{aligned}$$

On a $l - r * \alpha \equiv p \equiv 0 \mod p$. donc $O'_n = O_n$. □

Dans notre méthode de calcul des ordonnancements du graphe, notre seul aléa réside dans les choix arbitraires d'un nœud et d'un élément de l'ensemble $\mathbb{S}_p^k$ comme point de base du développement. Le lemme 24 montre que quels que soient ces choix, les solutions sont équivalentes modulo rotation.

Lemme 24 (Convergence de la méthode de calcul des ordonnancements : équivalence entre les différentes solutions)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p. Soit L le vecteur de retards associé à G.*

Soit O_1 et O_2 deux ordonnancements balancés de G.

$\exists! i$ tel que $\forall n$ un nœud de G, $O_1(n) = O_2(n) \circlearrowleft i$.

Démonstration. Les lemmes 22 et 23 nous montrent qu'à partir de l'ordonnancement d'un nœud n du graphe, on peut calculer l'ordonnancement unique de tous les nœuds du graphe.

Donc si $\exists i$ tel que $O_1(n) = O_2(n) \circlearrowleft i$ alors $\forall n'$ un autre nœud de G tel que $O_1(n') = O_1(n) \circlearrowleft l$ mais aussi $O_2(n') = O_2(n) \circlearrowleft l$ d'où $O_2(n') = O_1(n') \circlearrowleft l - i - l$
$O_2(n') = O_1(n') \circlearrowleft -i$. □

6.3.3 Ordonnancements des *registres fractionaires*

Lemme 25. *Soit G un graphe $\mathbb{N}$-égalisé. Soit L un vecteur de répartition de retards tel que G associé à L est $\mathbb{Q}$-égalisé. Soit O l'ordonnancement balancé de G. Soit c un canal de G tel que $L(c) = r$.*

c retarde r jetons distincts (Chaque jeton est retardé un seul instant).

Démonstration. Dans le chapitre 2, nous avons étudié les mots balancés. Pour un k et un p donnés, l'ensemble $\mathbb{S}_p^k$ peut être ordonné grâce à un ordre de précédence (lemme 11). Le passage d'un élément à son inférieur direct se fait par une transposition élémentaire (lemme 12). De plus, à chaque élément de l'ensemble est associée une position unique et différente pour chaque mot telle que la transposition de l'élément à cette position retourne son inférieur direct dans l'ordre de précédence (corollaire 5).

Soit O_1 et O_2 les ordonnancements de resp. $\bullet c$ et $c\bullet$. $O_1 \circlearrowleft 1$ représente l'ordonnancement de $c\bullet$ si $L(c) = 0$. Mais $L(c) = r$ donc, $O_2 = \tau^r(O_1 \circlearrowleft 1)$. Chaque retard dans le canal c a l'effet d'une transposition élémentaire sur O_2.

On sait que $-k * \alpha \equiv 1$ (lemme 13). Donc k transpositions reviennent à une rotation : chaque transposition concerne un **1** différent du mot. □

Le lemme 26 vérifie que les règles de calcul de l'ordonnancement d'un *registre fractionnaire* de la définition 36 permettent de l'utiliser effectivement quand un jeton doit être retardé.

Lemme 26. *Soit G un graphe k-périodique de période p et $\mathbb{Q}$-égalisé, M son marquage et L le vecteur de répartition des retards associé à G. Soit O l'ordonnancement balancé de G. On va supposer qu'il existe un marquage M_0 tel que l'on peut définir $Exec$ l'exécution en régime permanent de G tel que $Exec = M_0 \xrightarrow{F_1} M_1 ... M_{i-1} \xrightarrow{F_i} M_i ... \xrightarrow{F_p} M_p = M_0$. F_i est l'ensemble des nœuds n tel que $O_n(i) = 1$ (le lemme 27 nous montre que ce marquage existe).*

Soit c un canal du graphe tel que $L(c) = r$. On appel n_1 et n_2 les nœuds resp. $\bullet c$ et $c\bullet$. On a O_{n1} (resp. O_{n2}) l'ordonnancement de n_1 (resp. n_2). On définit l'ordonnancement du registre fractionnaire *du canal c comme un mot binaire O_{fr} tel que : $\exists! i \in [0, p[$ tel que $O_{n2} \circlearrowleft i = sup(\mathbb{S}_p^k)$.*
$\forall j \in [1, p]$, $O_{fr}(j) = 1$ ssi $(j \mod p) \in \{p - i\ ;\ p - i + \alpha\ ; p - i + 2 * \alpha\ ;\ ...\ ; p - i + (r-1) * \alpha\}$ (def 36).

$forall j$ tel que $O_{fr}(j) = 1$, on a $Retard(c, M_{j-1}) = 1$.

Démonstration. On a $O_{n1}(p)$ qui détermine l'action de n_1 à cet instant. $O_{n1}(p)$ donne aussi l'état du canal c à cet instant. (Si $O_{n1}(p) = 1$, n_1 à été actif, le canal c contient un jeton. si $O_{n1}(p) = 0$, n_1 n'a pas travaillé, le canal c est vide). De plus, $O_{n1}(p-1) = (O_{n1} \circlearrowleft 1)(p)$ donne l'état du canal c à l'instant précédent.

Dans l'ordre lexicographique, $O_{n1} \circlearrowleft 1$ et O_{n2} sont distant de r car $O_{n2} = \tau^r(O_{n1} \circlearrowleft 1) = O_{n1} \circlearrowleft 1 - r\alpha$.

Il existe r indices x différents tel que $(O_{n1} \circlearrowleft 1) \uparrow x \neq O_{n2} \uparrow x$. Etant donné que l'ordre de précédence est total sur $\mathbb{S}_p^k$,

Si $(O_{n1} \circlearrowleft 1 \prec O_{n2}$, alors il existe r **1** dans O_{n1} dont la position précède le **1** de même indice dans O_{n2}. Le lemme 25 nous dit que, dans ce cas, l'écart de position entre deux **1** de même indice est au maximum de

un : $((O_{n1} \circlearrowleft 1) \uparrow x) + 1 = O_{n2} \uparrow x$.

En revanche, si $(O_{n1} \circlearrowleft 1 \succ O_{n2}$, on a
$(O_{n1} \circlearrowleft 1) \uparrow x > O_{n2} \uparrow x$. Par contre,
$(O_{n1} \circlearrowleft 1) \uparrow x < O_{n2} \uparrow (x+1)$.
Etant donné que $O_{n2} = \tau^r(O_{n1} \circlearrowleft 1)$ est toujours vrai,
on a $((O_{n1} \circlearrowleft 1) \uparrow x) + 1 = O_{n2} \uparrow (x+1)$.

L'interprétation de ce résultat est que le jeton produit par n_1 à l'instant $O_{n1} \uparrow x = ((O_{n1} \circlearrowleft 1) \uparrow x) - 1$ est utilisé à l'instant $O_{n2} \uparrow x$. Or $O_{n1} \uparrow x + 2 = O_{n2} \uparrow x$. C'est à dire que le jeton est retenu à l'instant $(O_{n1} \uparrow x) + 1$. Donc $Retard(c, M_j) = 1$ avec $j = O_{n1} \uparrow x$.

Exprimons maintenant explicitement les instants $((O_{n1} \circlearrowleft 1) \uparrow x)$. Si $O_{n2} = inf(\mathbb{S}_p^k)$, On a $((O_{n1} \circlearrowleft 1) \uparrow x) + 1 = O_{n2} \uparrow x$ (c'est O_{n1} le plus grand des deux).

De plus, $\Delta(inf(\mathbb{S}_p^k)) = p = 0$ donc
$\Delta(inf(\mathbb{S}_p^k) \circlearrowleft \alpha) = \Delta(O_1 \circlearrowleft 1 - (r-1) * \alpha) = \alpha$ (corollaire 4) et
$\Delta(inf(\mathbb{S}_p^k) \circlearrowleft 2 * \alpha) = \Delta(O_1 \circlearrowleft 1 - (r-2) * \alpha) = 2 * \alpha$
on peut en déduire que $x = j * \alpha \mod p$ avec $j \in [1, r]$.

Si $O_{n2} = sup(\mathbb{S}_p^k) = inf(\mathbb{S}_p^k) \circlearrowleft \alpha$, $x = j * \alpha - \alpha \mod p = (j-1) * \alpha \mod p$ avec $j \in [1, r]$. Ou encore $x = J * \alpha \mod p$ avec $J \in [0, r[$.

Supposons qu'il existe i tel que $O_{n2} = sup(\mathbb{S}_p^k) \circlearrowleft i$. On a $((O_{n1} \circlearrowleft 1) \uparrow x) + 1 = O_{n2} \uparrow x$ ssi $x = p - i + J * \alpha \mod p$ avec $j \in [0, r[$.

Les instants d'activation du *registre fractionnaire* sont $\{p - i; p - i + \alpha; ...; p - i + (r-1)\alpha\}$. □

Le théorème 7 présente un des résultats les plus importants du manuscrit. Dans le processus d'égalisation, il est possible qu'un canal ait à retenir plusieurs valeurs au même instant. Pour cela, il faut que le canal possède plusieurs *registres fractionnaires*. Un des intérêts des mots balancés est que ceux-ci répartissent la charge des *registres fractionnaires* régulièrement sur la période. On obtient comme résultat que si un canal a des retards à appliquer, un seul *registre fractionnaire* suffit à tous les appliquer.

Théorème 7 (Capacité des canaux d'un graphe en régime permanent balancé).
Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé. Pour tout canal nécessitant des retards, un et un seul registre fractionnaire *suffit à appliquer ces retards.*

Démonstration. Le graphe est $\mathbb{N}$-égalisé, pour tout canal, le nombre de retards à appliquer est inférieur à k. Conformément au lemme 25, chaque retard concerne un jeton différent. La définition 36 nous dit que chaque retard est distant de $\alpha \mod p$ instants. Etant donné que α et p sont premiers entre eux, $\forall i, j \in [1, p]$, $i \neq j$, $i * \alpha \mod p \neq j * \alpha \mod p$. On peut appliquer jusqu'à p retards dans la même période avec le même *registre fractionnaire* or le nombre de retards par arc dans un graphe $\mathbb{N}$-égalisé est de $k < p$. □

6.3.4 Les jetons

Le lemme 27 montre qu'à partir de l'ordonnancement des nœuds et des *registres fractionnaires* du graphe calculés suivant les méthodes de ce chapitre, le placement des jetons résultant est équipollent au placement initial des jetons du graphe.

Lemme 27 (Validité de la méthode de calcul du marquage balancé)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O. Les règles de calcul, de la définition 37, des états du régime permanent de l'exécution de G à partir de son ordonnancement permettent de calculer des marquages équipollents (définition 15) au marquage initial du système.*

Démonstration. La preuve de ce lemme nécessite de définir la notion de mots de marquage des places d'un cycle et de mots de marquage des *registres fractionnaires* d'un cycle, puis de montrer quelques résultats préliminaires sur ces mots. □

Dans un graphe ordonnancé, si un nœud n travaille, tous ses canaux en sortie sont pleins. S'il ne travaille pas, ils seront vides. L'état courant de chaque canal du système peut être donné par le dernier élément de l'ordonnancement du nœud précédent le canal. Si $O_n(p) = 1$, le canal en sortie de n est plein. Si $O_n(p) = 0$, il est vide. Pour les *registres fractionnaires*, le lien est encore plus rapide, si la dernière valeur de l'ordonnancement d'un *registre fractionnaire* $O_{fr}(p)$ vaut 1 le registre est plein, sinon, il est vide. La figure 6.5 montre cette correspondance.

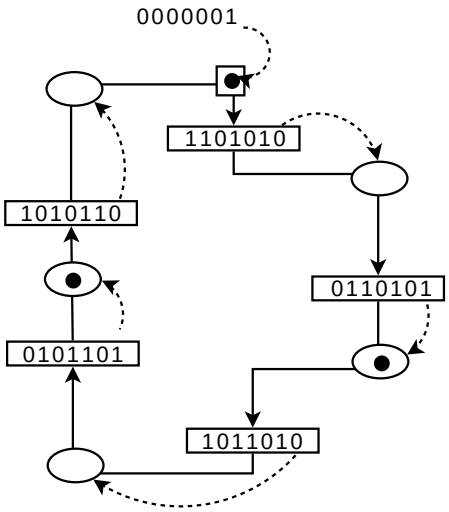


FIG. 6.5 – L'état du cycle est calculé à partir des ordonnancements de ses nœuds et de ses *registres fractionnaires*.

Définition 42 (Mots de marquage des places (*registres fractionnaires*) du cycle C)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O. Soit C un cycle de G contenant l nœuds (n_1,n_2,...n_l) et l canaux (c_1, c_2, ...,c_l) et utilisant r retards répartis sur les l canaux de C ($j * p - l * k = r$). c_i est un canal en sortie de n_i ($i \in [1, l]$). On note O_{ni} l'ordonnancement associé au*

nœud n_i et O_{fri} l'ordonnancement associé au registre fractionnaire *sur canal c_i ($i \in [1,l]$). Si le canal c_i ne contient pas de* registre fractionnaire, $O_{fri} = 0^p$.

On appelle mot de marquage des places du cycle C (à partir du nœud n_1) le mot M_{place} de longueur l tel que $M_{place}(i) = O_{ni}(p)$.

On appelle mot de marquage des registres fractionnaires *du cycle C(à partir du nœud n_1) le mot M_{fr} de longueur l tel que $M_{fr}(i) = O_{fri}(p)$.*

Définition 43 (Vecteur de retards cumulés)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O et un vecteur de répartition des retards noté L. Soit C un cycle de G contenant l nœuds (n_1,n_2,...n_l) et l canaux (c_1, c_2, ...,c_l) et utilisant r retards répartis sur les l canaux de C ($r = j*p - l*k$). c_i est un canal en sortie de n_i ($i \in [1,l]$). Soit I le vecteur de retards cumulés du cycle C à partir du nœud n_1 associé à L tel que $I(m) = \Sigma_{i=1}^{m} L(c_i)$.*

Lemme 28 (Expression des mots de marquage à partir d'un seul mot binaire)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O et un vecteur de répartition des retards noté L. Soit C un cycle de G contenant l nœuds (n_1,n_2,...n_l) et l canaux (c_1, c_2, ...,c_l) et utilisant r retards répartis sur les l canaux de C ($j*p - l*k = r$). c_i est un canal en sortie de n_i ($i \in [1,l]$). Soit I le vecteur de retards cumulés du cycle C à partir du nœud n_1. On note O_{ni} l'ordonnancement associé au nœud n_i et O_{fri} l'ordonnancement associé au* registre fractionnaire *sur canal c_i ($i \in [1,l]$). Si le canal c_i ne contient pas de* registre fractionnaire, $O_{fri} = 0^p$.

Soit $O' \in \mathbb{B}^*$ un mot binaire tel que $O'(i) = O_{n1}(p+1-i)$. (O' est l'inverse de O_{n1})

Le mot de marquage des places du cycle C (à partir du nœud n_1) noté M_{place} peut s'écrire :

$$M_{place}(n) = O'(n - I(n-1) * \alpha)$$

Le mot de marquage des registres fractionnaires *du cycle C (à partir du nœud n_1) noté M_{fr} peut être défini comme ceci :*

- $M_{fr}(n) = 1$ *ssi* $O'(n - I(n-1) * \alpha + 1) = 1$ *et* $\exists m \in [I(n-1), I(n)]$ *tel que* $O'(n+1-m*\alpha) = 0$.
- $M_{fr}(n) = 0$ *sinon.*

Démonstration. **mot de marquage des places du cycle C**

L'état des canaux est décrit par les bits $O_{n1}(p)$, $O_{n2}(p)$, ...,$O_{nl}(p)$. Or $O_{n2}(p) = (O_{n1} \circlearrowleft 1 - I(1)\alpha)(p) = O_{n1}(p - 1 + I(1) * \alpha)$. Plus généralement, $\forall j \in [1,p]$,

$$O_{nj}(p) = (O_{n1} \circlearrowleft j - I(j-1) * \alpha)(p) = O_{n1}(1 - j + I(j-1) * \alpha).$$

Soit $O' \in \mathbb{S}_p^k$ tel que $O'(i) = O_{n1}(p + 1 - i)$. On peut exprimer $O_{nj}(p)$ à partir de O' comme $O_{nj}(p) = O'(j - I(j-1) * \alpha)$. Donc $O'(j - I(j-1) * \alpha)$ décrit bien l'état du canal c_j.

mot de marquage des *registres fractionnaires* du cycle C

Un *registre fractionnaire* à r' retards en entrée d'un nœud n est plein aux instants $O_n(j)$ ssi $\exists i$ tel que $O_n \circlearrowleft i = sup(\mathbb{S}_p^k)$ et $j \in \{p - i\,;\, p - i + \alpha\,;p - i + 2 * \alpha\,;\, ...\,;p - i + (r'-1) * \alpha\}$. (lemme 26)

Cette condition peut être réécrite de la manière suivante :

Un *registre fractionnaire* à r' retards en entrée d'un nœud n est plein à l'instant courant, ssi

$O_n \in \{sup(\mathbb{S}_p^k), ..., sup(\mathbb{S}_p^k) \circlearrowleft -\alpha * (r'-1)\}$. On peut en déduire l'ordonnancement du nœud n' en entrée du canal contenant le *registre fractionnaire*. Pour $i \in [1, r']$, si $O_n = sup(\mathbb{S}_p^k) \circlearrowleft -\alpha * (i-1) \leftrightarrow O_{n'} \circlearrowleft 1 = sup(\mathbb{S}_p^k) \circlearrowleft -\alpha * (i-1-r)$. O_n appartient à l'ensemble des r' plus grand mot de $\mathbb{S}_p^k$ suivant l'ordre lexicographique. $O_{n'} \circlearrowleft 1$ appartient lui à l'ensemble des r' plus petit mot de $\mathbb{S}_p^k$ suivant l'ordre lexicographique. On sait que $O_{n'} \circlearrowleft 1(p) = O_{n'}(p-1) = 0$ car $r' < k$. De plus, $\exists j \in [1, r']$ tel que $O_{n'} \circlearrowleft 1 - j * alpha = sup(\mathbb{S}_p^k)$ car pour passer de l'ensemble des r' plus petit à l'ensemble des r' plus grand par rotation de $-\alpha$, on passe obligatoirement par $sup(\mathbb{S}_p^k)$. Or $sup(\mathbb{S}_p^k)(p) = 0$.

Donc un *registre fractionnaire* est plein à l'instant courant ssi $(O_{n'} \circlearrowleft 1)(p) = 1$ et $\exists j \in [1, r']$ tel que $(O_{n'} \circlearrowleft 1 - j * alpha)(p) = 0$.

Si le *registre fractionnaire* en question est sur le canal c_{i+1}, il est plein ssi $(O_{ni} \circlearrowleft 1)(p) = 1$ et $\exists j \in [I(i-1), I(i)]$ tel que $(O_{ni} \circlearrowleft 1 - j * alpha)(p) = 0$. Or $(O_{ni} \circlearrowleft 1)(p) = O'(n - I(n-1) * \alpha + 1)$ et $(O_{ni} \circlearrowleft 1 - j * alpha)(p) = O'(n + 1 - j * \alpha)$. □

Exemple 7. *La figure 6.6 montre une représentation de M_{place} pour $(k, p) = (4, 9)$, pour $(j, l) = (3, 6)$ ($p * j - k * l = 3$) Le vecteur de retards cumulés $I = \{0, 0, 1, 1, 1, 3\}$.*

On a $M_{place} = 100001$ et $M_{fr} = 001000$.

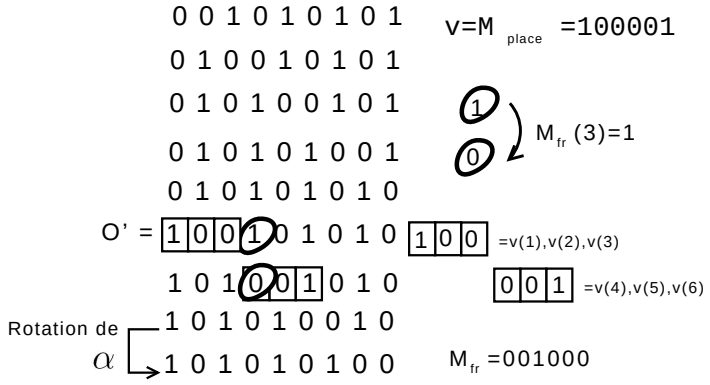


FIG. 6.6 – pour $(k, p) = (4, 9)$, pour $(j, l) = (3, 6)$ ($p * j - k * l = 3$. Le vecteur de retards cumulés $I = \{0, 0, 1, 1, 1, 3\}$, on a $M_{place} = 100001$ et $M_{fr} = 001000$.

Nous allons maintenant montrer que dans un graphe G, pour tout cycle C, $|M_{place}|_1 + |M_{fr}|_1$ d'un mot balancé est égal à $|C|_1$. (quel que soit la répartition des retards sur C) Pour cela nous allons commencer par montrer que c'est vrai quand tous les retards du cycle sont concentrés sur le même canal (lemme 29) puis nous généraliserons ce résultat à n'importe quelle répartition des retards (lemme 30).

Lemme 29 (Cardinalité des mots de marquage dans le cas simple)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O et un vecteur de répartition des retards notée L. Soit C un cycle de G contenant l nœuds (n_1,n_2,...n_l) et l canaux (c_1, c_2, ...,c_l) et utilisant r retards répartis sur les l canaux de C ($j * p - l * k = r$). c_i est un canal en sortie de n_i ($i \in [1, l]$). Soit I le vecteur de retards cumulés du cycle C à partir du nœud n_1. On note O_{ni} l'ordonnancement associé au nœud n_i et*

O_{fri} *l'ordonnancement asocié au* registre fractionnaire *sur canal* c_i *(*$i \in [1, l]$*). Si le canal* c_i *ne contient pas de* registre fractionnaire, $O_{fri} = 0^p$.

Soit $O' \in \mathbb{B}^*$ *un mot binaire tel que* $O'(i) = O_{n1}(p + 1 - i)$. *(*$O'$ *est l'inverse de* O_{n1}*)*

L *est tel que* $\forall i \in [1, l-1]$, $L(c_i) = 0$ *et* $L(c_l) = r$.

$|M_{place}|_1 + |M_{fr}|_1 = j$

Démonstration. Etant donné que l'on a un seul *registre fractionnaire*, $|M_{fr}|_1 \leq 1$. (théorème 7). M_{fr} est de la forme $0^{l-1}.X$ avec $X = 0|1$.

$M_{place} = O'(1).O'(2)...O'(l)$. Donc M_{place} est un sous-mot de O' de longueur l commençant à l'indice 1.

Dans $inf(\mathbb{S}_p^k)$, $inf(\mathbb{S}_p^k)(1) = 0$ (lemme 9) et $inf(\mathbb{S}_p^k)(1+\alpha) = 1$ car $inf(\mathbb{S}_p^k)(1+\alpha) = \lfloor(1+\alpha)k/p\rfloor - \lfloor(\alpha)k/p\rfloor$ Or $\alpha * k \equiv -1 mod p$ (lemme 13). donc $\alpha * k + k \equiv k - 1 mod p$. On peut en déduire qu'il existe n tel que $= \lfloor(1+\alpha)k/p\rfloor - \lfloor(\alpha)k/p\rfloor = (n+1) - n = 1$.

On identifie le bit $O'(1)$ dans $inf(\mathbb{S}_p^k)$ et on note son indice m. ($O' \circlearrowleft m = inf(\mathbb{S}_p^k)$).

Si $m \in \{\alpha, 2 * \alpha, ..., r * \alpha\}$, le décalage vers la droite suivi des r sauts de α réalisés entre $M_{place}(l)$ et $M_{place}(1)$ passent par les indices $1 + \alpha$ puis 1. on aura $M_{fr}(l) = 1$.

Le nombre de **1** dans M_{place} est :

$\forall r_o \in [1, p]$, le nombre de **1** dans un sous mot de $sup(\mathbb{S}_p^k)$ de longueur l et finissant à l'indice $r_o * \alpha$ est :
$|M_{place}|_1 = \lfloor(r_o * \alpha) * k/p\rfloor - \lfloor(r_o * \alpha - l) * k/p\rfloor$
On a $k * \alpha \equiv -1 \mod p$ donc
$|M_{place}|_1 = \lfloor -r_o/p\rfloor - \lfloor -(r_o + l * k)/p\rfloor$
$|M_{place}|_1 = \lceil(r_o + l * k)/p\rceil + \lfloor -r_o/p\rfloor$
$|M_{place}|_1 = \lceil(r_o + l * k)/p\rceil - 1$
On sait que $l * k \equiv -r \mod p$ car (p*j-l*k=r). donc $l * k + r_0 \equiv -r + r_0 \mod p$

On a deux cas :
$m \in \{\alpha, 2 * \alpha, ..., r * \alpha\}$, c'est à dire que $r_o \in [1, r]$.
La valeur de r_o n'est pas suffisante pour passer à l'entier supérieur : $|M_{place}|_1 = \lceil(l*k)/p\rceil - 1 = \lfloor l*k/p\rfloor$
Si on ajoute $M_{fr}(l) = 1$ induite par ce cas au nombre de **1** dans M_{place}, on obtient : $|M_{place}|_1 + |M_{fr}|_1 = \lfloor l * k/p\rfloor + 1 = \lceil(l * k)/p\rceil$

$m \in \{\alpha, 2 * \alpha, ..., r * \alpha\}$, c'est à dire que $r_o \in [r + 1, p]$.
La valeur de r_o est suffisante pour passer à l'entier supérieur : $|M_{place}|_1 = \lceil(l*k)/p\rceil + 1 - 1 = \lceil(l*k)/p\rceil$.
Dans ce cas, $M_{fr}(l) = 0$.
$|M_{place}|_1 + |M_{fr}|_1 = \lceil(l * k)/p\rceil$

$|M_{place}|_1 + |M_{fr}|_1 = \lceil(l * k)/p\rceil$. On sait que $p * j - k * l = r$. $j = (k * l + r)/p$. $k * l - r \equiv 0 \mod p$ d'où $j = \lceil(l * k)/p\rceil$. (pour $r < p$) □

La figure 6.7 montre deux couples (M_{place},M_{fr}) et (M'_{place},M'_{fr}) de mots de marquage des places

(resp. *registre fractionnaire*) du même cycle d'un même graphe à partir du même nœud n. La différence entre les deux réside dans la répartition des retards sur le cycle. Pour le premier, tous les retards du cycle sont regroupés sur le même canal. Pour le second, les retards sont répartis sur plusieurs canaux. (O' est le mot l'inverse du mot d'ordonnancement de n.) On voit que $|M_{place}|_1 + |M_{fr}|_1 = |M'_{place}|_1 + |M'_{fr}|_1$.

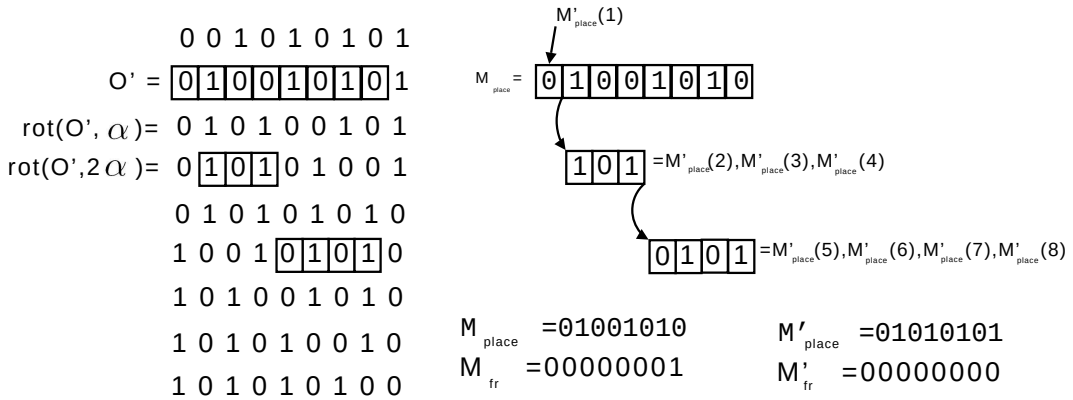


FIG. 6.7 – (M_{place},M_{fr}) avec $I = \{0, 0, 0, 0, 0, 0, 0, 4\}$ comme vecteur de retards cumulés et (M'_{place},M'_{fr}) avec $I = \{2, 2, 2, 4, 4, 4, 4, 4\}$ comme vecteur de retards cumulés.

Le lemme 30 généralise le résultat précédent. Pour prouver ce résultat, on va montrer que $|M_{place}|_1 + |M_{fr}|_1$ est constant quelle que soit la répartition des retards sur le cycle considéré. Si on se base sur la représentation des mots de marquage des places et des *registres fractionnaires* faite dans la figure 6.7, si les quatre derniers bits de M'_{place} avaient été pris dans $\rho(O', 2 * \alpha)$ au lieu de $\rho(O', 4 * \alpha)$, $|M'_{place}|_1$ aurait diminué de 1 mais $|M'_{fr}|$ aurait, lui, augmenté de 1. Maintenant, si les sept derniers bits de M'_{place} avaient été pris sur O' au lieu de $\rho(O', 2 * \alpha)$ c'est à dire $M'_{place} = M'_{place}$la position des **1** aurait changé, mais $|M'_{place}|_1$ serait resté le même et $|M'_{fr}|_1$ aussi. Le lemme 30 formalise ce résultat.

Lemme 30 (Cardinalité des mots de marquages dans le cas général)**.** *Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O et un vecteur de répartition des retards noté L. Soit C un cycle de G contenant l nœuds (n_1,n_2,...n_l), l canaux (c_1, c_2, ...,c_l) et utilisant r retards répartis sur les l canaux de C ($j * p - l * k = r$). c_i est un canal en sortie de n_i ($i \in [1, l]$). Soit I le vecteur de retards cumulés du cycle C à partir du nœud n_1. On note O_{ni} l'ordonnancement associé au nœud n_i et O_{fri} l'ordonnancement associé au* registre fractionnaire *sur canal c_i ($i \in [1, l]$). Si le canal c_i ne contient pas de* registre fractionnaire, *$O_{fri} = 0^p$.*

Soit $O' \in \mathbb{B}^*$ un mot binaire tel que $O'(i) = O_{n1}(p + 1 - i)$. ($O'$ est l'inverse de O_{n1})

Soit M_{place} (resp. M_{fr}) les mots de marquages des places (resp. registre fractionnaire*) du cycle C à partir du nœud n_1. $|M_{place}|_1 + |M_{fr}|_1 = j = |C|_1$.*

Démonstration. Soit N_{place} et N_{fr} les mots de marquage des places (resp. *registre fractionnaire*) du cycle C à partir du nœud n_1 pour J le vecteur de répartition des retards sur le cycle C à partir du nœud n_1 de la forme $\forall i < l$, $J(i) = 0$ et $J(l) = r$.

Nous allons montrer que $|M_{place}|_1 + |M_{fr}|_1 = |N_{place}|_1 + |N_{fr}|_1$.

Soit $I(n)$ la première occurrence de la valeur max de I excepté $I(l)$. Les $l-n$ dernières valeurs de M_{place} sont successives dans O' (Il n'y a pas de saut de α entre elles). Si on construit (M'_{place},M'_{fr}) les mots de marquage des places (resp. *registre fractionnaire*) du cycle C à partir du nœud n_1 pour I' le vecteur de répartition des retards sur le cycle C à partir du nœud n_1 de la forme :

- $\forall i < n$, $I'(i) = I(i)$
- $\forall i \in [n, l[$, $I'(i) = I(i) - 1$
- $I'(l) = I(l) + 1$: on regroupe les retards à la fin.

les $l-n$ dernières valeurs de M'_{place} sont aussi successives dans O' mais à une position décalée de $-\alpha$ par rapport aux $l-n$ derniers bits de M_{place}. On sait qu'un saut de $-\alpha$ équivaut à une transposition élémentaire sur le mot O'.

Si on compare les $l-n$ derniers bits de M'_{place} et les $l-n$ derniers bits de M_{place}, il n'y a que quatre possibilités exclusives en fonction de la position de $\Delta(O')$:

- Les $l-n$ derniers bits de M'_{place} sont égales aux $l-n$ derniers bits de M_{place}.
- $M_{place}(l) = 0$ et $M'_{place}(l) = 1$. Cela implique que le bit suivant $M_{place}(l)$ dans O' est un **1** et que le bit suivant $M'_{place}(l)$ dans O' est un **0** : $|M'_{place}|_1 = |M_{place}|_1 - 1$ et $|M'_{fr}|_1 = |M_{fr}|_1 + 1$. On a $|M'_{place}|_1 + |M'_{fr}|_1 = |M_{place}|_1 - 1 + |M_{fr}|_1 + 1$.
- $\exists j \in [l-n; l]$ tel que $M_{place}(j) = 1$, $M_{place}(j+1) = 0$ et $M'_{place}(j) = 0$, $M'_{place}(j+1) = 1$. $|M'_{place}|_1 = |M_{place}|_1$ seules les positions ont changé.

Dans tous les cas, on a $|M'_{place}|_1 + |M'_{fr}|_1 = |M_{place}|_1 - 1 + |M_{fr}|_1 + 1$.

On peut ainsi continuer de "regrouper" les retards du cycle avant le nœud $n-1$ jusqu'à obtenir le vecteur de répartition de retards J et N_{place} et N_{fr} les mots de marquage des places (resp. *registre fractionnaire*) du cycle C.

De proche en proche, on a $|M_{place}|_1 + |M_{fr}|_1 = |N_{place}|_1 + |N_{fr}|_1 = |C|_1$. □

Lemme 27 (Validité de la méthode de calcul du marquage balancé).

Soit G un graphe $\mathbb{Q}$-égalisé k-périodique de période p ayant un ordonnancement balancé noté O. Les règles de calcul, de la définition 37, des états du régime permanent de l'exécution de G à partir de son ordonnancement permettent de calculer des marquages équipollents (définition 15) au marquage initial du système.

Démonstration. Soit C un cycle de G, Soit M le marquage calculé grâce aux règles de la définition 37 pour $i = 0$. on a $|M_{place}|_1 + |M_{fr}|_1 = |C|_1$ (lemme 30). Le marquage $M = M_0 = M_p$ ainsi obtenu est équipollent au marquage initial.

Le lemme 24 nous montre qu'il existe un ordonnancement O_i pour G où $\exists! i$ tel que $\forall n$ un nœud de G, $O(n) = O_2(n) \circlearrowleft i$.

Si pour tous les nœuds n de G, on remplace O_n par $O_n \circlearrowleft 1$ et pour tous les *registres fractionnaires* fr de G, on remplace O_{fr} par $O_{fr} \circlearrowleft 1$, le marquage obtenu par les règles de la définition 37 est M_1 : il est aussi équipollent. De même pour M_2....$M_p = M_0$. □

6.3.5 Le régime permanent

Le lemme 31 montre que le jeu des jetons à partir de l'état construit est correct par rapport aux ordonnancements des nœuds qui ont été calculés. On va vérifier que quand l'ordonnancement d'un nœud lui ordonne de travailler, il a bien un jeton disponible dans chacune de ses entrées. Et inversement, s'il ne travaille pas, c'est qu'au moins une de ses entrées n'est pas disponible. Ensuite, nous allons nous assurer, dans le lemme 32, que si un *registre fractionnaire* doit retenir un jeton à un instant, c'est qu'un jeton est présent et qu'il ne peut pas être utilisé par le nœud de calculs en sortie. Pour prouver celà, on va s'assurer que les conditions d'exécutions d'un nœud, suivant une politique "au plus tôt", sont respectées par l'ordonnancement que l'on a construit.

Lemme 31 (Concordance entre ordonnancement des nœuds et marquages calculés)**.** *Soit G un graphe $\mathbb{Q}$-égalisé, k-périodique de période p ayant un vecteur de répartition des retards noté L, un ordonnancement balancé noté O et un ensemble d'état $\{M_0, ...M_i, ..., M_p = M_0\}$ du régime permanent de son exécution suivant O. calculé à partir de O comme décrit dans la définition 37.*

La définition 38 définit la séquence d'exécution du régime permanent de G suivant l'ordonnancement O : $Exec = M_0 \xrightarrow{F_1} M_1...M_{i-1} \xrightarrow{F_i} M_i... \xrightarrow{F_p} M_p = M_0$. Avec M_i le i^{eme} marquage calculé grâce aux règles de la def. 37. F_i est l'ensemble des nœuds n tel que $O_n(i) = 1$.

On a $F_i \subseteq F_{M_{i-1}}$. L'ensemble des nœuds n tel que $O_n(i) = 1$ est un sous ensemble des nœuds exécutable au marquage M_{i-1} (definition 16).

De plus, $\forall c$ un canal de G, $M_i(c) = M_{i-1}(c) + \delta(\bullet c) - \delta(c\bullet)$. ($\delta(n) = 1$ si $O_n(i) = 1$ et 0 sinon).

Démonstration. Soit n un nœud de G. $O_n(p)$(l'ordonnancement du nœud n à l'instant courant) donne l'état des places sur les canaux en sortie de n dans l'état M_0. Soit $\{c_1, c_2, ..., c_m\}$ les m canaux d'entrées de n.$\forall j \in [1, m]$. O_{fr_j} est l'ordonnancement du *registre fractionnaire* du canal c_j actif r_j instants.

Un nœud est exécutable ssi tous ses canaux d'entrées contiennent un jeton dans leur registre simple ou dans leur *registre fractionnaire* dans l'état courant. On doit prouver que $O_n(p) = 1 \Rightarrow \forall i \in [1, m]$, $O_{\bullet c_j}(p-1) = 1 \vee O_{fr_j}(p-1) = 1$.

On a $O_{\bullet c_j}(p-1) = O_{\bullet c_j} \circlearrowleft 1(p)$ et $O_n = O_{\bullet c_j} \circlearrowleft 1 - r_j * \alpha$.

Partant du dernier bit dans le mot binaire balancé O_n, on fait r_j rotation de taille α pour obtenir le dernier bit de $O_{\bullet c_j} \circlearrowleft 1$.

Si $O_n(p) = 1$, soit $O_{\bullet c_j} \circlearrowleft 1(p) = 1$ aussi soit $O_{\bullet c_j} \circlearrowleft 1(p) = 0$. Dans ce second cas, la définition 25 nous dit que $Retard(c_j, M_i) = 1$ et le lemme 26 en déduit que $O_{fr_j}(p-1) = 1$.

Maintenant, nous allons montrer que $\forall c$ un canal de G, $M_i(c_j) = M_{i-1}(c_j) + \delta(\bullet c_j) - \delta(n)$. ($\delta(n) = 1$ si $O_n(i) = 1$ et 0 sinon. Pareil pour $\bullet c_j$). Cette équation est équivalente à : $M_i(c_j) = M_{i-1}(c_j) + O_{\bullet c_j}(i) - O_n(i)$.
On a $M_i(c_j) = O_{\bullet c_j}(i) + O_{fr_j}(i)$ et $M_{i-1}(c_j) = O_{\bullet c_j}(i-1) + O_{fr_j}(i-1)$.

On sait qu'un *registre fractionnaire* est plein à l'instant i si le nœud $\bullet c_j$ a travaillé à l'instant précédent mais que le nœud n ne travaille pas à cet instant ($O_{\bullet c_j}(i-1) * (1 - O_n(i))$) ou que ce même *registre*

fractionnaire était plein à l'instant précédent et que le nœud $\bullet c_j$ à travailler lui aussi à l'instant précédent ($O_{fr_j}(i-1)*O_{\bullet c_j}(i-1)$). Formellement, on a : $O_{fr_j} = O_{\bullet c_j}(i-1)*(1-O_n(i))+O_{fr_j}(i-1)*O_{\bullet c_j}(i-1)$.
$O_{fr_j} = O_{\bullet c_j}(i-1) - O_{\bullet c_j}(i-1) * O_n(i) + O_{fr_j}(i-1) * O_{\bullet c_j}(i-1)$.
Nous allons simplifier cette équation. Intéressons nous aux deux derniers termes : $-O_{\bullet c_j}(i-1) * O_n(i) + O_{fr_j}(i-1) * O_{\bullet c_j}(i-1)$. Quand $O_{\bullet c_j}(i-1) = 1$, les deux derniers termes valent : $-O_n(i) + O_{fr_j}(i-1)$.
Quand $O_{\bullet c_j}(i-1) = 0$, les deux derniers termes valent : 0.
Mais quand $O_{\bullet c_j}(i-1) = 0$, $-O_n(i) + O_{fr_j}(i-1) = 0$ car $O_{fr_j}(i-1) \Rightarrow O_n(i)$. On peut donc simplifer l'équation en lui retirant $O_{\bullet c_j}(i-1)$ dans ses deux derniers termes :
$O_{fr_j} = O_{\bullet c_j}(i-1) - O_n(i) + O_{fr_j}(i-1)$.
Maintenant, si on remplace $O_{fr}(i)$ par sa valeur dans $M_i(C_j)$, on obtient :
$M_i(c_j) = O_{\bullet c_j}(i) + O_{\bullet c_j}(i-1) - O_n(i) + O_{fr_j}(i-1)$. Si on repositionne les termes, on obtient :
$M_i(c_j) = O_{\bullet c_j}(i-1) + O_{fr_j}(i-1) + O_{\bullet c_j}(i) - O_n(i)$. On identifie $M_{i-1}(c_j)$
$M_i(c_j) = M_{i-1}(c_j) + O_{\bullet c_j}(i) - O_n(i)$.
$M_i(c_j) = M_{i-1}(c_j) + \delta(\bullet c_j) - \delta(n)$. □

Lemme 32 (Concordance entre ordonnancement des *registres fractionnaires* et marquages calculés)**.** *Soit G un graphe $\mathbb{Q}$-égalisé, k-périodique de période p ayant un vecteur de répartition des retards "au plus tard" notée L, un ordonnancement balancé noté O et un ensemble d'état M_0,...M_i,...$M_p = M_0$ calculé à partir de O comme décrit dans la définition 37.*

Soit c un canal de G contenant un registre fractionnaire *et O_{fr}, son ordonnancement.*
$O_{fr}(i) = 1$ ssi ($M_{i-1}(c) = 1$ et $O_{c\bullet}(i) = 0$) ou ($M_{i-1}(c) = 2$ [et $O_{c\bullet}(i) = 1$])

Démonstration. La condition d'activation du *registre fractionnaire* est une réécriture de la notion de retard 25 : $O_{fr}(i) = 1$ ssi $Retard(c, M_{i-1}) = 1$.

Le lemme 26 montre la validité de cette hypothèse. □

Théorème 5 (Existence du régime permanent balancé).
Soit G un graphe. $\forall G$, il existe un ordonnancement balancé associé.

Démonstration. Les lemmes 22 et 26 montrent que l'on peut toujours construire un ordonnancement balancé associé à un graphe G. Le lemme 27 montre qu'a cet ordonnancement est associé un placement équipollent au marquage initial. C'est à dire qu'à partir du marquage calculé, l'ordonnancement "au plus tôt" du graphe est égale à l'ordonnancement balancé calculé. Le lemmes 31 et 32 montre ce fait.

Pour tout graphe, il existe donc un régime permanent balancé. □

6.3.6 Initialisation du système

Théorème 6 (Atteignabilité du régime permanent balancé).
Soit G un graphe. il existe un ordonnancement balancé associé à G accessible à partir de son état initial.

Démonstration. Le théorème 5 montre que pour tout graphe, il existe un régime permanent dont les états sont des marquages équipollents au marquage initiale du système. Le théorème 3 montre que deux marquages équipollents sont mutuellement accessibles par opération asynchrone. Il existe donc une séquence d'initialisation par opération asynchrone permettant d'atteindre le régime permanent balancé de tout graphe à partir de son état initial.

□

6.3.7 Modélisation des canaux de communication

La section 5.4.2 nous présentait la modélisation d'un canal comme un registre simple suivi de *registre(s) fractionnaire(s)*. Le théorème 7 montre, qu'au maximum, un seul *registre fractionnaire* suffit dans le cas de régime permanent balancé. On peut donc borner la capacité des canaux de communication à "1" pour les canaux sans retard et "2" pour les canaux avec retards.

Comme dans le processus d'égalisation, un canal formé d'un registre (suivi d'un *registre fractionnaire*) a le même comportement qu'un canal à capacité instantanée comme défini dans la section 3.4.2. Etant donné que la capacité de chacun des canaux du système est suffisante, le comportement du graphe à capacité est le même que celui du graphe sans capacité.

6.4 Conclusion

Nous avons ordonnancé statiquement un réseau de processus en minimisant la taille des ressources de communication. Pour ce faire, nous avons suivi les étapes suivantes :

- Ajouter au graphe autant de latences entières que possible afin de le pré-égaliser.
- Ajouter au graphe des latences rationnelles (retards) afin de l'égaliser.
- En déduire la répartition des *registres fractionnaires*.
- Fixer arbitrairement l'ordonnancement d'un de ses nœuds comme étant un mot balancé de l'ensemble $\$_p^k$ (où k et p sont la périodicité et la période du graphe).
- En déduire l'ordonnancement balancé de chacun des nœuds du graphe...
- puis celui des *registres fractionnaires* du graphe.
- Calculer les p états (marquage du graphe) du régime permanent associés à l'ordonnancement du graphe.
- Calculer une initialisation asynchrone pour le graphe du marquage initial vers l'un des états du régime permanent.

Dans la deuxième partie du chapitre, nous avons vérifié formellement le bon fonctionnement de notre processus.

Chapitre 7

Mise en oeuvre

Afin de pouvoir mener nos recherches sur le processus d'égalisation et sur les mots balancés, nous avions besoin de simuler l'exécution d'un graphe, de tester nos algorithmes d'égalisation et de pouvoir visualiser les ordonnancements des nœuds comme des mots binaires.

Notre besoin étant assez particulier, nous n'avons pas trouvé d'outil existant pouvant y répondre. Nous nous sommes lancés (avec Julien Boucaron) dans l'implantation d'un logiciel de création, visualisation et simulation de graphe puis nous y avons ajouté nos algorithmes d'égalisation. Cet outil a été baptisé *K-Passa* pour *K-Periodic Asap System, Simulation and Analisys* : analyse et simulation de système asap k-periodic. Nous n'avons encore pleinement traité que des mises en œuvre des travaux des chapitres 4 et 5 de ce document (et pas de modèles à ordonnancements balancés).

7.1 Présentation

K-Passa implémente les fonctionnalités suivantes :

- Représentation d'un graphe.
- Visualisation graphique d'un graphe.
- Modification de l'état du graphe manuellement (nombre de jetons, placement des jetons, longueur des canaux).
- Stockage et comparaison des différents états du graphe.
- Simulation de l'exécution "au plus tôt" du graphe jusqu'à obtention d'un régime permanent ; automatiquement ou pas à pas.
- Visualisation de l'ordonnancement des nœuds du graphe comme des mots binaires.
- Rappel de l'état initial du graphe.
- Liste des cycles du graphe
- Calcul du débit de chacun de ces cycles.
- Calcul de la périodicité du graphe.
- Comblement des cycles rapides par l'ajout de latences entières.
- Placement des *Registres Fractionnaires*.

La figure 7.1 montre l'interface graphique de *K-Passa*. Le graphe représenté est un module de décodage MPEG-2. Les Labels associés à chaque nœud permettent de nous donner le nom du nœud et son ordonnancement. Grâce au menu "Equalization", on a pu égaliser le graphe pour obtenir la liste de ces cycles et leur débit (à droite), sa périodicité (en bas) et le placement des *registres fractionnaires*. La simulation accessible par le menu "Run" nous a permis d'obtenir les ordonnancements des nœuds.

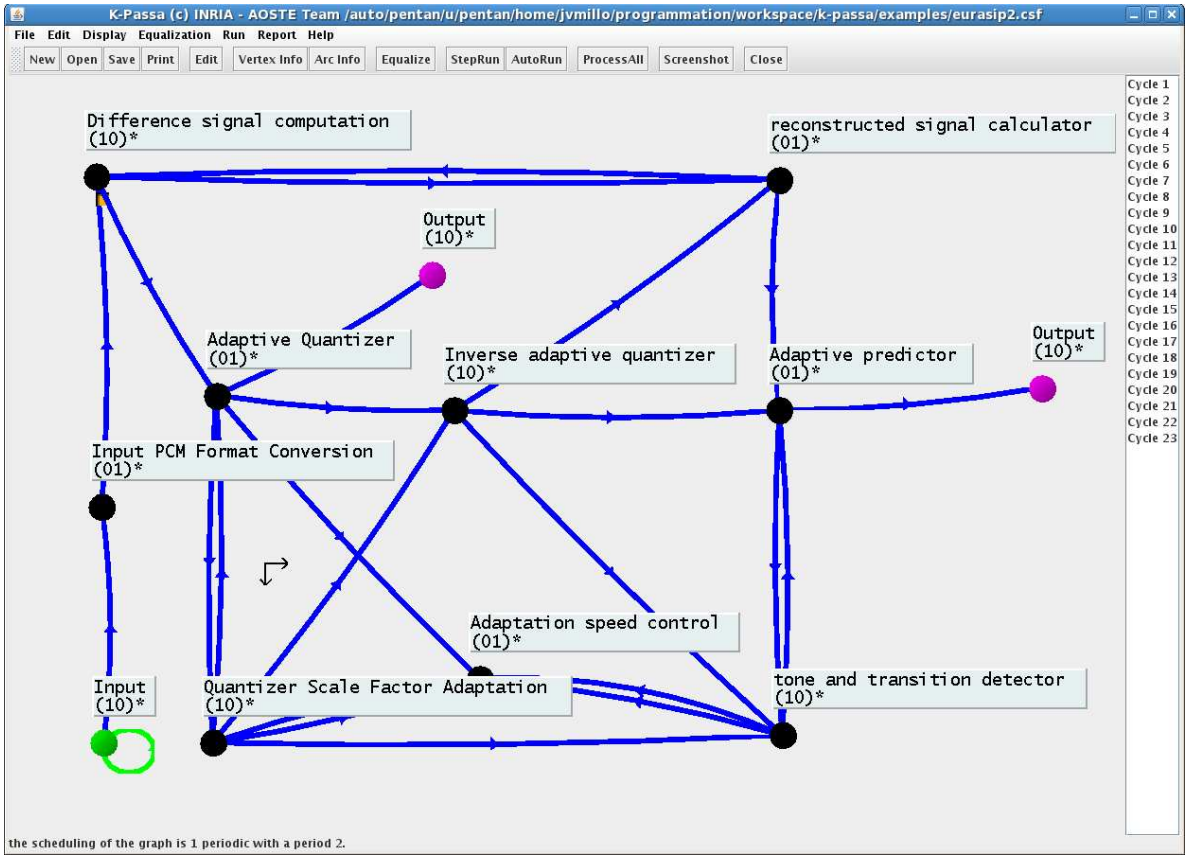


FIG. 7.1 – K-Passa.

7.2 Composants du logiciel

Le langage de programmation utilisé pour programmer K-Passa est le Java.

7.2.1 Structure de données

Pour la représentation interne et la visualisation du graphe, nous avons choisi d'utiliser la *Mascopt-Lib*. Cette bibliothèque de gestion de graphe développé par l'équipe Mascotte de l'INRIA Sophia-Antipolis permet de gérer un graphe composé de nœuds et de canaux. A ces trois éléments de base, on peut ajouter n'importe quel paramètre grâce à un système d'indexation ayant pour clé : un objet du graphe et un nom de paramètre ; pour valeur : une chaîne de caractères, un entier ou un flottant. Grâce à cela, on a pu définir

des attributs comme l'ordonnancement d'un nœud, la périodicité associée au graphe, la latence de communication associée aux canaux ... Le choix de la *MascoptLib* a permis d'obtenir un logiciel graphiquement accompli trés rapidement. Grâce à cela, l'essentiel de notre effort s'est porté sur les algorithmes de traitement. En revanche, l'absence du type bit comme type de paramètre nous a poussé à ne pas optimiser les ressources mémoires nécessaires à l'exécution de l'outil car cela aurait demandé un certain temps que l'on a préféré consacrer à l'accomplissement de nos objectifs principaux. La version 2 de *K-Passa* est en cours de développement par les ingénieurs de notre projet.

Exhaustivement, voici l'ensemble des paramètres classés par objet d'appartenance :

Les paramètres du graphe

- La période et la périodicité : ce sont deux entiers qui sont calculés pendant le processus d'égalisation. Leur ratio donne le débit du graphe.
- La période et la périodicité originales : ce sont les mêmes paramètres, Les précédent sont recalculés après l'ajout de latence. Ces valeurs servent à conserver la périodicité du graphe original.
- L'état d'avancement dans le processus d'égalisation : c'est un entier qui augmente avec l'accomplissement les étapes du processus d'égalisation. Il permet de ramener un graphe enregistré dans le même état d'avancement qu'à sa fermeture. *A ne pas confondre avec l'état du graphe qui concerne la répartition des jetons sur le graphe.*

Les paramètres d'un nœud

- Son nom : c'est le nom du nœud
- Nœud d'entrée : Si le nœud n'a aucun canal entrant, il est considéré comme un point d'entrée du système. A la simulation, il deviendra une source de jetons ; sans cela l'exécution s'arrêterait par famine.
- Nœud de sortie : si un canal n'a aucun canal sortant, il est considéré comme un point de sortie du système.
- La phase initiale de son ordonnancement : c'est un mot binaire. Comme le paramètre suivant, il est calculé pendant la simulation.
- La phase périodique de son ordonnancement : c'est aussi un mot binaire de longueur égale à la période du graphe.

Les paramètres d'un canal

- Son nom : c'est le nom du canal.
- Sa latence initiale : c'est la latence de communication du canal. Quand la latence de communication est trop importante, la représentation graphique de son expansion en latence unitaire intercalée de nœud de communication serait lourde. On a donc choisi de représenter les canaux sous leur forme contractée.
- Sa latence courante : Le processus d'égalisation peut augmenter les latences de communication, la nouvelle latence est stocké i ci. Ces deux paramètres sont des entiers positifs non nuls.

- Le marquage initial : un canal de latence n contient n places ayant chacune soit 1 soit 0 jeton. L'état de chacune des places successives du canal est conservé i ci.
- Le marquage courant : Le processus d'égalisation et la simulation modifient le marquage d'un canal. Ces deux derniers paramètres sont des mots binaires de longueur égale à la latence resp. initiale/courante.
- *Registre Fractionnaire* : si le canal en contient un, ce paramètre vaut 1, sinon 0.
- Jeton RF : c'est un entier qui compte le nombre de jeton contenu dans le *registre fractionnaire*. Théoriquement, un *registre fractionnaire* peut contenir une et une seule donnée. Si les flots de jetons à une jonction sont suffisamment déséquilibrés pour nécessiter le stockage de plusieurs données, il faut mettre plusieurs *Registres fractionnaires*. Mais comme pour l'expansion des latences de communications, la représentation graphique peut devenir lourde. On a choisi de représenter une suite de *Registres fractionnaires* comme un seul. La figure 7.2 montre comment décomposer un *registre fractionnaire* à la *K-Passa* en plusieurs *registres fractionnaire* classiques.
- Image : en fonction du nombre de valeur contenu dans le *registre fractionnaire*, le carré le représentant peut prendre différente teinte : il est vert quand le registre est vide, orange quand il contient une valeur et rouge quand il en contient plus.
- RF initial ordonnancement : C'est un mot d'entier positif ou nul. Il indique pour chaque instant de la phase d'initialisation le nombre de jetons contenu dans le *registre fractionnaire*.
- RF periodic ordonnancement : C'est un mot d'entier positif ou nul. Il indique pour chaque instant du régime permanent le nombre de jetons contenu dans le *registre fractionnaire*. La longueur du mot est égale à la période du graphe.

A partir de l'ordonnancement du *registre fractionnaire* dans *K-Passa*, on peut facilement retrouver le nombre de *registres fractionnaires* successifs qu'il faut à l'implantation (s'il y a, à un instant, plus d'une valeur à stocker). Le premier est actif quand l'ordonnancement indique autre chose que '0' et inactif pour '0'. Ensuite, on retire 1 à toutes les valeurs non nulles de l'ordonnancement et on recommence pour le deuxième. Ainsi de suite jusqu'à ce que l'ordonnancement n'indique que des '0'. La figure 7.2 montre la correspondance entre a) un *registre fractionnaire* dans *K-Passa* qui stocke deux jetons durant les mêmes instants et b) deux *Registres fractionnaires* consécutifs dans notre modèle de réseau de processus.

L'état du graphe est défini par l'état de ses canaux : latence, présence de *registre fractionnaire* répartitions des jetons. Deux états d'un graphe sont identiques si pour chaque canal, la latence courante est la même, le marquage est le même, la présence de *registre fractionnaire* est la même et enfin si le nombre de valeur dans les *registres fractionnaires* est le même.

Le stockage de l'état du graphe comme un objet indépendant nécessite ces données. Etant donné que la rapidité d'exécution et l'espace mémoire utilisé n'étaient pas des priorités de conception, nous n'avons pas cherché à optimiser le stockage d'un état

7.2.2 Algorithmique

Nous allons présenter ici quelques algorithmes qui sont utilisés dans *K-Passa*.

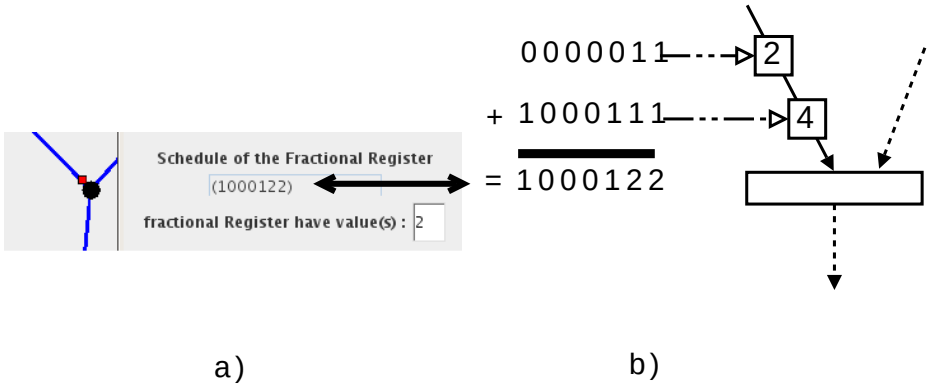


FIG. 7.2 – a) Un *registre fractionnaire* dans *K-Passa* stockant deux jetons aux mêmes instants correspond à b) deux *registres fractionnaires* consécutifs.

Sauvegarde Sauvegarder un graphe égalisé et simulé nécessite de conserver la liste des cycles qu'il contient, sa périodicité, l'ordonnancement de tous ses nœuds et de tous ses *Registres fractionnaires*. Ces informations peuvent être recalculées à partir du graphe initial. Nous avons choisi dans *K-Passa* de conserver seulement une indication sur l'état d'avancement dans le processus d'égalisation du graphe et de le regénérer tel qu'il était avant la sauvegarde en recalculant ses paramètres.

L'algorithme d'égalisation se découpe en six instructions séquentielles :

Recherche des cycles élémentaires du graphe : Nous avons utilisé l'algorithme de Wayne D. Grover [34] implémenté directement dans la *MascoptLib*. Cet algorithme optimal recherche tous les cycles élémentaires qui passent par un nœud puis une fois fini, il supprime le nœud du graphe et recommence pour un autre nœud.

Calcul du débit des cycles : Pour chaque cycle, on garde son nombre de jetons et sa latence. La division (qui donne le débit) n'est faite que pour l'affichage afin d'éviter toute perte de précision. La comparaison de débit se fait en entier à l'aide de produit en croix.

Calcul de la k-périodicité originelle : Nous avons réalisé nous-mêmes cet algorithme à partir de la formule de Baccelli et al. [8]. On restreint le graphe à ses parties fortement connexes critiques en utilisant la liste des cycles et leur débit précédemment calculé. Pour chaque partie fortement connexe critique, on calcule la périodicité et la période. Et enfin, à partir de la périodicité de chaque partie fortement connexe critique, on calcule la périodicité et la période du graphe.

Ajout de latence entière : Etant donné que l'INRIA jouit d'une relation privilégiée avec la société Ilog, nous avons pu utiliser CPLEX pour résoudre notre système d'équation linéaire à solution entière. Mais nous avons aussi utilisé une bibliothèque libre : GLPK afin que *K-Passa* reste fonctionnel même sans licence

CPLEX. Le choix entre les deux bibliothèques se fait automatiquement en fonction de la présence des variables d'environnement de CPLEX.

Comme précisé plus haut, les nouvelles latences sont inscrites dans le paramètre latence courante associé à un canal afin de conserver la latence initiale dans le paramètre latence initiale.

La cinquième instruction de la série est la recherche de la nouvelle k-périodicité qui a pu être modifiée par l'ajout des latences.

Simulation pour le placement des *registres fractionnaires* : Nous réalisons une simulation de l'exécution qui place dynamiquement les *registres fractionnaires* où ils sont nécessaires. Une fois en place, on recharge le placement initial des jetons en conservant les *registres fractionnaires*.

7.2.3 Simulation

La fonction de simulation permet d'ajouter dynamiquement les *registres fractionnaires* où ils sont nécessaires.

Avant de lancer la simulation, on doit ajouter une boucle sur chacun des nœuds d'entrée du graphe. Cette boucle est un canal de latence égal à la période du graphe et contenant autant de "1" que la périodicité du graphe. Ce cycle simule le flux de données en entrée du système. Pour que le comportement soit stable, il faut que le débit du flux soit égal au débit du graphe.

Pendant une étape de simulation, on fait avancer les jetons dans les canaux, on exécute les nœuds qui sont exécutables, on remplit l'ordonnancement des nœuds, et on met à jour l'état des *registres fractionnaires*. L'avantage par rapport au *latency-insensitive design* est que l'exécution d'un nœud dépend uniquement de ses entrées et pas de son suivant ; le canal en sortie est toujours prêt à recevoir une donnée.

On calcule les états successifs jusqu'à ce que l'état courant i soit égal à l'état $i-p$ (où p est la périodicité du graphe). Il nous suffit donc de stocker seulement p états.

Pour chaque nœud et pour chaque *registre fractionnaire*, les p derniers bits de l'ordonnancement sont leur régime permanent, les prédécesseurs leur phase initiale. Les boucles sur les nœuds d'entrée du système sont conservées.

Chapitre 8

Conclusion

Le point de départ de nos travaux de thèse a été la conception insensible aux latences des circuits électroniques digitaux, et plus précisément des systèmes sur puce (SoCs) assemblés à partir de composants IPs préexistants, et devant opérer dans des cas où les connexions globales entre les ports de ces composants réclament plus d'un cycle d'horloge pour acheminer les données. La proposition initiale de Carloni était de diviser ces longues connexions en sections de latence unitaire, puis d'installer des répéteurs sophistiqués en bout de chaque section, pour assurer la bonne mémorisation ainsi que la régulation du trafic, avec des composants simples en nombre borné (les *Relay-stations*). Les composants sont alors activés exactement dès que possible en fonction des médias de communication (par "clock-gating" depuis les *Shell-wrappers*).

Nous avons proposé tout d'abord une modélisation formelle de ces composants, étudié leurs propriétés fonctionnelles et extra-fonctionnelles, et ramené le problème à une spécification à la base d'Event/Marked Graphs avec des places de capacité 2 (pour les *Relay-stations*) et une sémantique d'exécution "au plus tôt" (pour les *Shell-wrappers*). On a montré que la limitation de capacité engendrait naturellement le protocole de contrôle de flux respectant les tailles de mémorisation, et l'impact que cela avait sur le débit des calculs. Au niveau théorique, ce travail peut être vu comme une étude des relations entre modèles GALS et implémentation multi horloges/polychrones par le biais d'ordonnancements.

La modélisation par "Marked/Event graphs" a naturellement fait resurgir des résultats classiques sur l'ordonnancement statique des systèmes clos (ou de manière équivalente, dans des systèmes non clos mais pour lesquels les entrées surviennent aux instants attendus) et contenant des boucles dans le graphe sur les trajets de données. Nous avons alors étudié la question d'introduire exactement les éléments de mémorisations (et les latences additionnelles) pour égaliser autant que possible les longueurs des trajets (et donc les latences) afin de tenter de se passer des protocoles dynamiques de contrôle de flux, ainsi que de simplifier l'expression des *Relay-stations*. Ce travail a été inspiré par la théorie des processus N-synchrones. Mais là où les auteurs de cette théorie demandent qu'un système de typage adéquat impose des débits égaux pour les différents composants parallèles (et règlent alors le problème des offsets et décalages entre calculs), nous nous attachons à égaliser ces débits entre cycles et trajets d'un même composant.

Ce second travail sur l'ordonnancement statique nous a, à son tour, démontré que les distributions de données qui minimisaient les tailles de mémorisation dans les lignes de communication étaient celles où ces

données étaient le plus régulièrement réparties et espacées. Nous avons donc ensuite étudié la théorie des mots infinis binaires périodiques balancés, aux fins de constater qu'ils étaient préservés par une sémantique d'exécution "au plus tôt" (ASAP) du système, et qu'on pouvait ainsi proposer analytiquement un ordonnancement valide, sans avoir besoin de simulation. Ce travail a également été inspiré par des travaux antérieurs, sur les mots mécaniques de B. Gaujal et al.

Il existe plusieurs perspectives de prolongation de ce travail. La question de la détermination efficace de séquence d'initialisation simple menant à des régimes stationnaires naturels est en particulier à envisager.Une autre piste, certainement plus large et plus innovante, concerne la généralisation du modèle des Marked/Event Graphs en introduisant des nœuds spécifiques de routage (mux/demux), comme on avait jusqu'à présent des nœuds de calculs et des nœuds de transport (expandus depuis les latences de communication). Les nœuds de routages permettent ensuite d'étudier les capacités de routage statique k-périodique, comme on l'a fait auparavant pour l'ordonnancement. Un partie de ce travail a déjà été commencé, y compris la relation avec les modèles formels étendant les Marked/Event graphs dans une philosophie inspirée des Réseaux de Kahn (Pas de conflit, que des choix internes aux nœuds). Les résultats partiels ont fait l'objet d'une partie de la thèse de J. Boucaron, et devrait se poursuivre dans la thèse d'A. Coadou (doctorant dans le projet AOSTE/ INRIA Sophia-Antipolis).

Bibliographie

[1] M. Alanyali and B. Hajek. On load balancing in erlang networks. *Stochastic Networks : Theory and Applications Oxford University Press*, pages 215–230, 1996.

[2] Murat Alanyali and Bruce Hajek. Analysis of simple algorithms for dynamic load balancing. In *Mathematics of Operations Research*, pages 230–238, 1995.

[3] Cyril Allauzen. Une caracterisation simple des nombres de sturm. *Journal de la theorie des nombres de Bordeaux*, 10.2 :237–241, 1998.

[4] Eitan Altman, Sandjai Bhulai, Bruno Gaujal, and Arie Hordijk. Rr3727 : Optimal routing problems and multimodularity.

[5] Eitan Altman, Bruno Gaujal, and Arie Hordijk. Balanced sequences and optimal routing. *J. ACM*, 47(4) :752–775, 2000.

[6] Charles André. Representation and analysis of reactive behaviors : A synchronous approach. In *Computational Engineering in Systems Applications, Lille, France*, pages 19–29, July 1996. Also available as I3S technical report RR 95-52.

[7] Cristina Arguelles and Mark Hartmann. Transcience bounds for long walks.

[8] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity : an algebra for discrete event systems*. John Wiley & Sons, 1992.

[9] Albert Benveniste, Paul Caspi, Stephen Edwards, Nicolas Hallbwachs, Paul Le Guernic, and Robert de Simone. The synchronous languages 12 years later. In *IEEE Proceedings*, pages 64–83. INRIA and IRISA/INRIA and Columbia University and Verimag/CNRS, January 2003.

[10] Jean Bernoulli. Recueil pour les astronomes. *A Berlin*, 1 :255–284, 1772.

[11] G. Berry. The constructive semantics of pure esterel.

[12] G. Berry. *The Foundations of Esterel*. MIT Press, 2000.

[13] Emmanuel Hyon Bruno Gaujal. A new factorization of mechanical words. *INRIA/RR 5175*, 2004.

[14] Jacques Carlier and Philippe Chrétienne. *Problème d'ordonnancement : modélisation, complexité, algorithmes*. Masson, Paris, 1988.

[15] Luca Carloni, Kenneth McMillan, and Alberto Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20(no. 9) :pp. 1059–1076, 2001.

[16] Luca .P Carloni, Keneth L. McMillan, and Alberto .L. Sangiovanni-Vincentelli. Latency insensitive protocols. In N. Halbwachs and LNCS 1633 D. Peled, editors, *Proc. of the 11th Intl. Conf. on Computer-Aided Verification (CAV)*, pages 123–133. UC Berkeley, Cadence Design Laboratories, July 1999.

[17] Luca P. Carloni, Kenneth L. McMillan, Alexander Saldanha, and Alberto L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *ICCAD '99 : Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 309–315, Piscataway, NJ, USA, 1999. IEEE Press.

[18] Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency-insensitive systems. In *The Proceedings of the Design Automation Conference*, pages 361–367. UC Berkeley, June 2000.

[19] Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli. Combining retiming and recycling to optimize the performance of synchronous circuits. In *SBCCI '03 : Proceedings of the 16th symposium on Integrated circuits and systems design*, page 47, Washington, DC, USA, 2003. IEEE Computer Society.

[20] Mario R. Casu and Luca Macchiarulo. A detailed implementation of latency insensitive protocols. In *Proceedings of Formal Methods for Globally Asyncronous Locally Syncronous Architectures*, pages 94 – 103, 2003.

[21] Mario R. Casu and Luca Macchiarulo. Issues in implementing latency insensitive protocols. In *DATE'04 : Proceedings of the Design automation and Test in Europe Conference and Exhibition*, Washington, DC, USA, 2004. IEEE Computer Society.

[22] Mario R. Casu and Luca Macchiarulo. A new approach to latency insensitive design. In *DAC '04 : Proceedings of the 41st annual conference on Design automation*, pages 576–581, New York, NY, USA, 2004. ACM Press.

[23] Mario R. Casu and Luca Macchiarulo. On-chip transparent wire pipelining. In *ICCD '04 : Proceedings of the IEEE International Conference on Computer Design*, pages 160–167, Washington, DC, USA, 2004. IEEE Computer Society.

[24] E. B. Christoffel. Observatio arithmetica. *Ann. Mat. Pura Appl*, 6 :148–152, 1875.

[25] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-synchronous kahn networks : a relaxed model of synchrony for real-time systems. In *POPL '06 : Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 180–193, New York, NY, USA, 2006. ACM Press.

[26] F. Commoner, Anatol W.Holt, Shimon Even, and Amir Pnueli. Marked directed graph. *Journal of Computer and System Sciences*, 5 :511–523, October 1971.

[27] J. Cortadella, A. Kontratyev, L. Lavagno, and C. P. Sotiriou. A concurrent model for desynchronization. In *12th International Workshop on Logic and Synthesis*, 2003.

[28] Jordi Cortadella and Mike Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. In *DAC '07 : Proceedings of the 44th annual conference on Design automation*, pages 416–419, New York, NY, USA, 2007. ACM.

[29] Jordi Cortadella, Mike Kishinevsky, and Bill Grundmann. Synthesis of synchronous elastic architectures. In *DAC '06 : Proceedings of the 43rd annual conference on Design automation*, pages 657–662, New York, NY, USA, 2006. ACM.

[30] Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Transactions on Design Automation of Electronic Systems*, 9(4) :385–418, October 2004.

[31] Jörg Desel and Javier Esparza. *Free choice Petri nets*. Cambridge University Press, New York, NY, USA, 1995.

[32] B. Gaujal and A. Bouillard. Coupling time of a (max, plus) matrix. In *Rapport de recherche INRIA : RR4068*, November 2000.

[33] Bruno Gaujal. Optimal allocation sequences of two processes sharing a resource. discrete event dynamic systems. *IEEE Transactions on Robotics and Automation*, 11 :327–354, 1997.

[34] Wayne D. Grover. *Mesh-based Survivable Transport Networks : Options and Strategies for Optical, MPLS, SONET and ATM Networking*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[35] Bruce Hajek. Balanced loads in infinite networks.

[36] Laurent Hardouin. *Habilitation à diriger des recherches : sur la commande linéaire de systèmes à événements discrets dans l'algèbre (max,+)*. PhD thesis, Université d'Angers ISTIA, Juin 2004.

[37] I.Blunno, J.Cortadella, A.Kondratyev, L.Lavagno, K.Lwin, and C.Sotiriou. Handshake protocols for de-synchronization. In *ASYNC'04*, 2004.

[38] A. Luca J. Berstel. Sturmian words, lyndon words and trees. *Theoretical Computer Science*, 178 :171–203, 1997.

[39] Eric Laurier. Operation sur les mots de christoffel. *Journal de la theorie des nombres de Bordeaux*, 11.1 :111–132, 1999.

[40] M. Lothaire. *Applied Combinatorics on Words*, volume 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005. (610 pp.).

[41] Jean Mairesse and Laurent Vuillon. Asymptotic behavior in a heap model with two pieces. *Comput. Sci*, 270 :525–560, 1998.

[42] Olivier Marchetti. *Dimensionnement des mémoires pour systèmes embarqués*. PhD thesis, Université Pierre et Marie Curie, Decembre 2006.

[43] Jean-Vivien Millo and Julien Boucaron. Compositionality of statically scheduled ip. In *Proceedings Third International Workshop on Formal Methods for Globally Asynchronous Locally Synchronous Design (FMGALS'2007)*. EATCS electronic publishing, 2007.

[44] Jean-Vivien Millo, Julien Boucaron, and Robert De Simone. Another glance at relay stations in latency-insensitive design. *Electr. Notes Theor. Comput. Sci.*, vol. 146(no. 2) :pages 41–59, 2006.

[45] Jean-Vivien Millo, Julien Boucaron, and Robert De Simone. Latency-insensitive design and central repetitive scheduling. In *Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on*, pages 175– 183, Piscataway, NJ, USA, 2006. IEEE Press.

[46] Jean-Vivien Millo, Julien Boucaron, and Robert De Simone. Formal methods of scheduling for latency-insensitive designs. *EURASIP journal on embedded system*, 2007 (not yet published).

[47] Alix Munier. The basic cyclic scheduling problem with linear precedence constraints. *Discrete Applied Mathematics*, 64(3) :219–238, 1996.

[48] Carl Adam Petri. Kommunikation mit automaten. In *PhD thesis, Technische Universitat Darmstadt, Germany*, 1962.

[49] Christos P.Sotiriou and Luciano Lavagno. De-synchronization : Asynchronous circuits from synchronous specifications. In *IEEE SOC 2003*, 2003.

[50] Gwénaël Richomme. A local balance property of episturmian words. *CoRR*, abs/cs/0702060, 2007.

[51] M. Singh and M. Theobald. Generalized latency-insensitive systems for gals architectures. In *Proc. of the Workshop on Formal Methods for Globally Asynchronous Locally Synchronous (GALS) Architecture (FMGALS-03)*, 2003.

[52] Montek Singh and Michael Theobald. Generalized latency-insensitive systems for single-clock and multi-clock architectures. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1008–1013, February 2004.

Table des figures

1.1 Système sur puce ayant des problèmes de délais de communication. 10
1.2 Transformation du système de la figure 1.1 en graphe. 11
1.3 Découpage du pipeline du bloc de traitement en plusieurs blocs et représentation explicite du délai de communication. 11
1.4 Représentation du système comme un *Marked Graph*. Les mots binaires inscrits dans les nœuds du graphe représentent leur ordonnancement. (Cette manière de représenter les ordonnancements ne fait pas partie des *Marked graph*, elle a été introduite dans [25]). 12
1.5 Simulation du fonctionnement du système insensible aux latences. 13
1.6 Simulation du fonctionnement du système pré-égalisé 14
1.7 Simulation du fonctionnement du système pré-égalisé avec un ordonnancement balancé. . 15
1.8 Une analyse statique du système nous permet de gérer localement la resynchronisation de la branche de droite avec celle de gauche grâce à notre *registre fractionnaire*. 16
1.9 Le système égalisé et réordonnancé est muni d'un registre fractionnaire afin d'être implanté. Il n'a plus de problème de latence. 16

2.1 L'orbite d'un mot balancé u pour (a) $u \in \mathbb{S}_7^3$ and (b) $u \in \mathbb{S}_6^2$ 26
2.2 Algorithme de création de mot balancé utilisant α pour $(k, p) = (3, 5) \Rightarrow \alpha = 3$. 30

3.1 Exemple d'un réseau de processus. 34
3.2 Exemple d'un *Marked Graph*. 34
3.3 Les sous-graphes entourés de pointillés sont des parties fortement connexes. Le sous-graphe entouré d'une ligne pleine est une partie fortement connexe critique. 35
3.4 Exemple d'un graphe pondéré. 41
3.5 (a) Transformation d'un nœud avec une latence de calcul de 2 en une succession de nœuds instantanés. (b) Transformation d'un canal avec une latence de communication de 3 en une succession de canaux "classiques". 41
3.6 Exemple d'un graphe à capacité. 43
3.7 Transformation d'un canal de capacité $k = 3$ contenant $n = 1$ jeton initial en deux canaux sans capacité contenant $n = 1$ jeton et $k - n = 2$ jetons formant un cycle. 43
3.8 Le graphe à un débit inférieur à son débit maximum à cause des capacités. 44

3.9 Le graphe a un débit inférieur à son débit maximum à cause des capacités. 44
3.10 Limiter la capacité des places d'un DAG fait apparaitre un cycle de débit $3/4$. Ce cycle réduit le débit du DAG. 45
3.11 a) Un canal à capacité instantanée b) son équivalent par l'expansion de la def. 32, c) un pas d'exécution représenté par les flèches grises et décomposé en 4 étapes. 46

4.1 Transformation (a) d'un système synchrone avec des problèmes de délai d'interconnexion en (b) un système insensible aux latences. 49
4.2 Signaux d'interface de la *Relay-Station*(RS). 52
4.3 Sync-Chart de la *Relay-Station*(RS). 53
4.4 (a) Circuit logique de la *Relay-Station* et (b) son chemin de données. 54
4.5 (a) Circuit logique du *Shell-Wrapper*, (b) détail du bloc associé à une entrée, et (c) chemin de données à travers le *Shell-Wrapper*. 56
4.6 Automate à état fini décrivant le comportement d'une *Relay-Station*. 58
4.7 Circuit logique décrivant le comportement d'un *Shell-Wrapper*. 59
4.8 Ordonnancement statique d'une perle utilisant la technique de *Clock gatting*. 60
4.9 Un *Fractional synchronizer* et son ordonnancement (011). 60
4.10 Ajout de la phase d'initialisation à l'ordonnancement de la perle. 61

5.1 Example d'un système à égaliser. 63
5.2 Liste des cycles élémentaires du système : a) C1, b) C2, c) C3, d) C4. 64
5.3 a) Système avant l'ajout de latence b) Système après l'ajout de latence. 65
5.4 Ordonnancement des nœuds du système. 66
5.5 Système après la répartition des *registres fractionnaires*. Les valeurs numériques associées aux *registres fractionnaires* sont leur nombre d'utilisation par période. Le mot binaire associé donne leurs instants d'activation. 67
5.6 a) L'initialisation classique du système qui dure neuf instants peut être remplacé par b) le déplacement des quatre jetons de gauche afin d'atteindre directement le régime permanent. 68
5.7 Bloc diagramme d'un *Registre Fractionnaire*. 68
5.8 a) SynchChart, b) circuit et c) flot de données du *Registre Fractionnaire*. 69
5.9 a) implantation dynamique du signal $hold$, b) implantation statique du signal $hold$. 70
5.10 Le même système est ordonnancé différemment. a) On a besoin de 2 *registre fractionnaire*, b) un seul suffit. 71
5.11 Système pour lequel l'ajout de latences entières ne peut pas amener tous les cycles le plus près possible du débit critique. 72

6.1 Ajout des *registres fractionnaires* grâce au solveur d'équation linéaire a) Avant l'ajout, b)Il faut 6 délais répartis en deux *registres fractionnaires* sur les canaux a3 et a7. 78
6.2 Repositionnement des délais des *registres fractionnaires*. 79
6.3 Propagation de l'ordonnancement des nœuds de proche en proche. 81

6.4 Deux cycles C_1 et C_2 de longueur resp. $L + L_1$ et $L + L_2$. 87
6.5 L'état du cycle est calculé à partir des ordonnancements de ses nœuds et de ses *registres fractionnaires*. 91
6.6 pour $(k, p) = (4, 9)$, pour $(j, l) = (3, 6)$ ($p * j - k * l = 3$. Le vecteur de retards cumulés $I = \{0, 0, 1, 1, 1, 3\}$, on a $M_{place} = 100001$ et $M_{fr} = 001000$. 93
6.7 (M_{place},M_{fr}) avec $I = \{0, 0, 0, 0, 0, 0, 0, 4\}$ comme vecteur de retards cumulés et (M'_{place},M'_{fr}) avec $I = \{2, 2, 2, 4, 4, 4, 4, 4\}$ comme vecteur de retards cumulés. 95

7.1 K-Passa. 101
7.2 a) Un *registre fractionnaire* dans *K-Passa* stockant deux jetons aux mêmes instants correspond à b) deux *registres fractionnaires* consécutifs. 104

A.1 Histogramme du déroulement de mon projet 126
A.2 Tableau récapitulatif des charges et ressources de mon projet 128

Index

A

Abstract 1
Abstraction 10
Ajout 64, 77
Algorithme 30
Alpha 28, 30
Approche 9
Asynchrone 37
Atteignabilité 76
Au plus tôt 37, 97
Au plus tard 79
Automate état fini 58
Automate à état fini 52

B

Back Pressure Protocol 51
Balancé 21
Bloc fonctionnel 48
Brute force 98

C

Canal 33
Canal instantané 42
Canaux 33, 74, 99
Capacité 42, 43, 74, 76, 99
Capacité instantanée 45
Channel 50
Circuit 55
Circuits synchrones élastiques 60
Composant IP 7
Conception insensible aux latences 47
Construction 23
Contre-exemple 43
Cplex 65
Création 30
Critique 34
Cycle 34
Cycles 63

D

Débit 34, 43, 64
Delta 27
Dynamique 36

E

Egalisé 36
Egalisation 62, 73
Elastiques 60
Equipollent 36, 90
Etat 34, 52, 82
Event Graph 11
Event graph 33
Exécutable 36
Exécution 36, 97
Existence 76
Expansion 10, 40, 43, 45

F

Fil d'interconnexion 48
Finite State Machine 52
Fork 57
Formel 52
FR 67
Fractional synchronizer 60
FSM 52

G

Graphe d'événement 33

Graphe d'évenements 11

H

Hold 69

I

Implémentation formelle 58
Initialisation 67, 83, 98
Instantané 42
Instantanée 45
IP 7

J

Jetons 90
Join 57

K

K-périodicité 38, 64

L

Latence 13, 40, 64
Latency Insensitive Design 47
Latency insensitive design 12
LID 12, 47
Lister 63
Liveness 39
Luca Macchiarulo 58

M

Mario Casu 58
Marked Graph 11
Marked graph 33
Marquage 34, 82, 90
Marquage équipollent 36
Marquage balancé 38
Michael Theobald 57
Minimum cycle mean ratio 71
Modèle d'exécution 37
Montek Singh 57
Mot Balancé 21
Mot balancé 18
Mot binaire 18, 19
Mot périodique 19

N

N-égalisé 36
Nœud 33
NCT 122

O

Optimisation 71
Orbite 26
Ordonnancement 14, 37, 65, 80, 86
Ordonnancement balancé 37

P

Période 38
Périodicité 38, 64
Partie fortement connexe 35
Pearl 50
Perle 50
Plateforme Conception 4
Politique d'exécution 37
Pré-égalisé 36
Pré-égalisation 13
Précédence 27
Processus 63, 99
Protocole de back-Pressure 59
Protocole de rétroaction 51, 59

Q

Q-égalisé 36

R

Réduction 25
Régime permanent 97
Répartition 79
Réseau de processus 33
Résumé 1
Registre fractionnaire 15, 66, 67, 74, 77, 81, 83, 88
Relay-Station 50
Relay-station 52
Remerciements 4

Repositionnement 79
Retard 39, 77, 81, 83
Retarder 39
Rotation 20

S

Safety 39
SELF 60
Shell 55
Shell-Wrapper 48, 55
Simulation 65
SoC 7
Solveur d'inéquations linéaires 65, 71, 72
Split 57
ST Microelectronics 4
Station de relais 52
Sync-Chart 52, 68
Synchrone 37
Sys2RTL 4
Système insensible aux latences 12

T

Tau 28
Timing closure 7
Transposition 21, 27

U

Unicité 26
Unité fonctionnel 48

V

Vérification 54, 55, 70, 73
Validation 73
Valorisation de compétence 122
Vecteur de retards 78

Annexe A

Valorisation des compétences des docteurs

Ce travail a été fait dans le cadre d'une formation proposé par l'Association Bernard Gregory (www.abg.asso.fr). Le but de cette formation est de nous faire prendre conscience des compétences professionnelles que l'on a développées pendant notre doctorat. Pour cela nous avons analysés nos trois ans de recherche comme un projet professionnelle qui serait arrivé à terme : Analyse des coûts, des contraintes, des résultats, des retombés économiques, ...

A.1 Cadre général et enjeux de ma thèse

A.1.1 Présentation succincte

L'objet scientifique de mon travail est d'étudier la faisabilité d'une nouvelle méthode de conception de système sur puce afin de pouvoir profiter entièrement des capacités (dimension, rapidité, autonomie) que nous offrent les dernières avancées en matière de fabrication de puce. Un écart se creuse entre les systèmes que l'on conçoit et l'incroyable capacité des puces qui sont censées accueillir ces systèmes. L'exemple le plus connu de système sur puce est le microprocesseur.

Le système implanté dans une puce est un ensemble de bloc de calcul et de bloc de stockage qui communiquent et échangent des données afin de réaliser une ou plusieurs fonctions. Mon étude concerne les problèmes liés aux communications à l'intérieur d'une puce qui sont apparus avec la dernière technologie de fabrication de puce. Celle-ci permet de réaliser des systèmes plus complexes, plus rapides, plus économes mais nécessite une nouvelle approche concernant la gestion des communications inter-bloc d'une puce. Expérimentalement, plusieurs approches existent, mon travail consiste à en choisir une, à l'améliorer et à la rendre utilisable par des industriels.

Mener à bien mon projet permettra aux entreprises qui me financent de pouvoir produire des systèmes sur puce plus complexes, plus performants dans les mêmes délais que les systèmes actuels. Cela renforcera leur compétitivité. Ce qui tend à des retombées économiques favorables pour la région car ces entreprises y

sont implantées.

A.1.2 Contexte de ma thèse

Place de la thèse dans le projet global de l'équipe INRIA AOSTE

L'axe de recherche principal de mon équipe INRIA/AOSTE (Institut National de Recherche en Informatique et Automatique/ Analyse et Optimisation des Systèmes Temps réel et Embarqués) couvre le domaine de la conception des systèmes temps réel embarqués. Par "conception" nous réunissons des activités de : Modélisation de haut niveau Transformation et Analyse Implantation sur des Plates-formes embarquées Les systèmes sur puce sont le cœur ou plutôt le cerveau des plates formes embarquées. Si le but de mon travail touche à la troisième activité de l'équipe : « Implantation sur des Plates-formes embarquées », la manière de résoudre le problème concerne la seconde « Transformation et Analyse » et les outils utilisés, la première : « Modélisation de haut niveau ».

Compétences scientifiques, techniques et humaines nécessaires au projet

Mon projet de thèse est conduit sous la direction du Dr Robert de Simone. L'équipe est composée du Dr Charles André, du Dr Fréderic Mallet, et du Dr Marie-Agnès Peraldi-Frati. Il y a deux ingénieurs experts : Benoit Ferrero et le Dr Julien Boucaron. Nous sommes ensuite quatre doctorants : Aamir Mehmood Kahn, Anthony Coadou, Jean-Francois Le Tallec, et moi-même. Et enfin, un stagiaire : Anthony Gaudino. Pour mon projet, les compétences techniques nécessaires sont : programmation haut et bas niveau, architecture des ordinateurs, microélectronique, informatique théorique, théorie de l'ordonnancement, théorie des graphes. Les relations avec les autres personnes de l'équipe sont de deux natures : certains m'aident à atteindre les objectifs que je me suis fixés, d'autres discutent avec moi de la clairvoyance des ces objectifs et m'aident à les redéfinir si nécessaire. Une assistante de projet sert d'interface entre les services administratifs, financiers et légaux et les besoins de l'équipe dans ce domaine : Patricia Lachaume.

Réseau de collaboration scientifique et industrielle : CIMPACA

Mon équipe est engagée dans un centre régional de promotion de la recherche dans le domaine de conception des systèmes embarqués temps réel. Ce centre nommé CIMPACA (Centre Intégré de Microélectronique en région PACA) est financé par la région PACA, par un ensemble d'industriels présents dans la région (Texas Instrument, ST Microelectronics, Esterel Technologies, Scaleo Chips, Synopsis, NXP) et par des équipes de recherches locales (INRIA Sophia Antipolis, CNRS I3S, LEAT, Eurecom). CIMPACA est géré par une association loi 1901 : ARCSIS. Il est divisé en trois plateformes dont deux sont situées dans les Bouches-du-Rhône et une, celle dont je dépens, à Sophia-Antipolis. La plateforme Conception, est divisée en quatre groupes. Mon groupe, Sys2RTL, a la vocation de développer des outils pour concevoir des systèmes sur puces compatibles avec les procédés de fabrication de demain. Il regroupe actuellement cinq projets de recherche complémentaires. Le budget de CIMPACA sur la période 2005-2008 est de 40 millions d'euros dont 20 millions d'aide régionale.

La concurrence

Beaucoup d'autres équipes de recherches s'intéressent aux mêmes problèmes que ceux soulevés dans mon travail, le projet Mozart du LEAT/CNRS, l'équipe ESPRESSO de l'INRIA Rennes, l'équipe PROVAL INRIA, le département d'informatique de l'université de Turin, Italie, le département d'informatique de l'université de Columbia, New-York USA, ou enfin le département d'informatique et d'électronique de Virginia Tech, Blacksburg USA. Comme pour mon projet, ces équipes sont supportées par des industriels qui attendent des résultats. Quel que soit le critère que l'on prenne en compte : quantité et qualité des publications, coopérations internationales ou industrielles, appartenance à des réseaux de recherche, l'équipe AOSTE est comparable à ses concurrents.

A.1.3 Moi dans ce contexte

Après un Bac S et un DUT en informatique génie logiciel à l'Université de Nice Sophia-Antipolis, je suis entré en école d'ingénieur à Fontainebleau (77) : l'ESIGETEL. Durant ma 3eme année d'école d'ingénieur (BAC+5), je me suis spécialisé dans les systèmes embarqués. Les systèmes embarqués sont à la frontière entre l'informatique et la microélectronique. Cette dernière nous donne les contraintes fortes d'architectures en terme de puissances de calculs, espaces de stockage, ou consommation en énergie et le challenge consiste à réaliser l'application informatique la plus performante adaptée à l'architecture électronique considérée. C'est-à-dire que l'on doit faire le maximum avec un minimum de ressources. Ce travail résonne en moi comme ma contribution personnelle à la lutte contre le réchauffement global. Parallèlement, je me suis inscrit en DEA informatique à l'Université de Marne la Vallée afin de découvrir le monde de la recherche. A l'époque, l'idée de consacrer mon temps à la création de savoir et de technologie était, et est toujours, une perspective agréable. Pour mon stage de DEA, j'ai choisi l'équipe AOSTE de l'INRIA Sophia Antipolis tout d'abord car sa thématique correspond parfaitement avec ma formation et mes centres d'intérêts et ensuite pour des raisons géographiques : ma famille est originaire du pays Niçois. Suite à ce stage, étant donné que la réalité du travail de chercheur à l'INRIA Sophia Antipolis me convenait parfaitement, j'ai choisi de continuer l'aventure en thèse.

L'intitulé de mon projet étant assez vague et vaste : « Etude de la modélisation des systèmes GALS pour la conception de systèmes sur puce », il n'y avait pas de rôle à jouer dans sa définition. Par contre, la programmation de mon projet est restée tout au long sous mon entière responsabilité. Les seules contraintes extérieures sont les deadlines de soumission de papier technique, relatant mes derniers résultats et les réunions trimestrielles d'avancement de CIMPACA. De plus, les directions que j'ai prises et les choix que j'ai faits dépendent aussi de ma responsabilité bien que j'ai suivi les avis de mes référents : directeur de thèse, chercheurs INRIA et industriels de CIMPACA.

A.2 Déroulement, gestion et coût de mon projet

A.2.1 Préparation et cadrage de mon projet

Facteurs de succès et de risques de mon projet

Mon projet de recherche est favorisé par un certain nombre de facteurs qui peuvent contribuer à son succès :

- Des résultats majeurs d'informatique théorie et de la théorie des automates sont en adéquation avec mon travail. Nous devons ces résultats à P. Chretienne et J. Carlier (chercheurs CNRS) d'une part et à G. Cohen, F. Baccelli, G. J. Olsder, et JP. Quadrat (chercheurs INRIA).
- Le point de base de mon projet est une théorie réalisée par un chercheur de renommée internationale : Alberto Sangiovanni-Vincentelli et son étudiant de l'époque maintenant en poste de professeur à l'Université de Columbia : Luca Carloni. Ce travail a mis en ébullition toute notre communauté de chercheur, mon équipe y compris, et de nombreux papiers scientifiques sont venus complémenter cette théorie.
- La coopération entre industriels et laboratoires de recherche à l'intérieur de CIMPACA me permet d'être au plus proche des problèmes des industriels. En revanche, mon affectation en laboratoire de recherche me permet le recul nécessaire face à ces problèmes.
- Le regroupement de plusieurs projets de recherches complémentaires dans le même groupe nous permet, les autres doctorants et moi-même, de nous faire une idée précise des contraintes à respecter pour que notre travail rentre dans un flot de conception de système sur puce global.

Les facteurs de risques sont les suivants :

- Le risque majeur est la concurrence car ce domaine de recherche fait l'objet d'un intérêt mondial. Il est donc possible qu'un laboratoire « concurrent » trouve une solution meilleure que la mienne.
- Les choix que je fais durant mes recherches peuvent me mener à une solution restrictive quand au type de systèmes sur puce concernés.

Stratégie de maitrise des risques

J'ai limité les risques ci-dessus en me tenant informé des avancées des laboratoires concurrents et en discutant le plus souvent possible des options qui s'offrent à moi avec les personnes capables de me guider comme mon directeur de thèse, des industriels ou même d'autres chercheurs de CIMPACA ou d'ailleurs.

Partenaires industriels

Les acteurs industriels de CIMPACA sont des agences régionales de grandes multinationales du secteur. Concernant les chercheurs académiques de CIMPACA, ils appartiennent à des laboratoires régionaux d'instituts nationaux de réputation internationale tels l'INRIA ou le CNRS.

Contrat de propriété intellectuelle

La totalité des résultats de mon projet appartient à chacun des membres (institutions et entreprises) de CIMPACA. Libre à chacun des industriels, par la suite, d'utiliser les découvertes selon leurs intérêts et enjeux.

A.2.2 Conduite de mon projet

Les étapes du projet

Durant mon stage de DEA, j'ai effectué le travail préliminaire nécessaire à la réalisation de mon projet c'est-à-dire le choix de la solution que j'allais devoir améliorer et adapter aux besoins des industriels. Pendant la première année, j'ai exploré trois théories concurrentes pour le même problème : le « Latency-Insensitive Design » (LID), la « De-synchronization », et « l'Iso/Endochronie ». J'ai choisi le « LID » et j'ai commencé à rendre cette théorie pratique, utilisable par des industriels. Durant la deuxième année, partant du choix précédent, j'ai cherché à améliorer cette théorie tout en finissant de l'adapter aux besoins des industriels. En troisième année, j'ai commencé par fixer le point d'arrivée afin de savoir jusqu'où je devais aller. J'ai fini d'améliorer la théorie, et maintenant je suis en train de finir d'adapter cette nouvelle théorie à la pratique.

De plus, une attention particulière est donnée à la communication de mes travaux de recherche durant les trois ans d'existence du projet. Pour cela, l'équipe AOSTE et moi-même avons publié dans des conférences ou des journaux les résultats que nous avons obtenus.

Relations avec les partenaires industriels

Des réunions trimestrielles d'avancement de projet ont rythmé les travaux des doctorants du groupe Sys2RTL car elles nous obligeaient à présenter oralement l'avancement de nos travaux ainsi que les futurs axes d'étude. Ces réunions nous permettent de prendre du recul tous les trimestres et de faire des bilans d'étapes, de regarder le travail accompli et le travail qu'il reste à fournir.

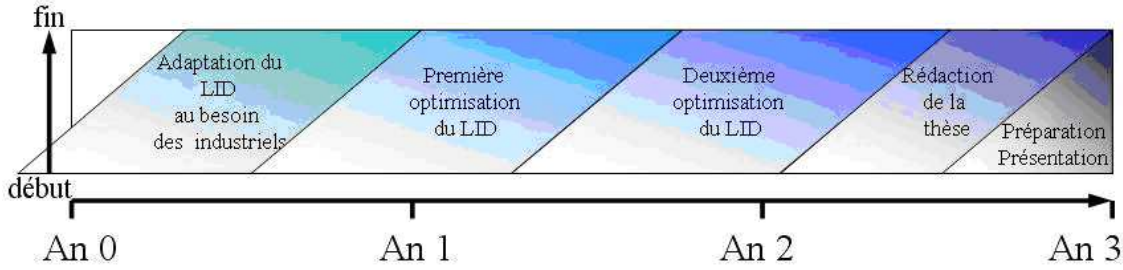


FIG. A.1 – Histogramme du déroulement de mon projet

Problèmes rencontrés

La principale difficulté que j'ai rencontrée est technique. Quand on obtient un résultat, il est de bon goût de démontrer mathématiquement que ce résultat est correct. Cette étape peut s'avérer plus compliquée et surtout plus longue qu'on ne l'imagine. Afin de ne pas passer tout mon temps sur l'élaboration de ces preuves, j'ai reçu l'aide d'un ingénieur expert de l'équipe AOSTE : Benoit Ferrero.

A.2.3 Evaluation et prise en charge du coût du projet

A.2.4 Ressources humaines

- Mon allocation de recherche doctorant : 98474.17E (elle se décompose en mon salaire de base : 66431.88E plus un complément donné par l'INRIA Sophia Antipolis : 32042,29E)
- Benoit Ferrero (Ingénieur expert) : 12.5% de son temps donc 12.5% de son salaire sur ces 1,5 ans de contrat : 5512.50E
- Julien Boucaron (autre Doctorant d'AOSTE) : Une partie de son travail de recherche recoupe le mien. J'estime cette part à 16.6%. Donc 16.6% de son salaire sur 1,5 ans : 8206,18E
- Robert de Simone (Mon directeur de thèse) : Etant un homme très occupé, je n'occupe pas plus de 3,3% de son temps donc 3,3% de son salaire : 6300.00E
- Patricia Lachaume (Assistante de projet) : Elle s'occupe de deux équipes, chaque équipe contient une quinzaine de personne donc 3,3% de son temps : 2400.00E
- Les personnels administratifs et techniques de l'INRIA Sophia Antipolis : Cela peut se cumuler à 1/2 journée par mois. Soit 1,63% du temps d'une personne : 2065.57E
- Les personnels administratifs de l'université de Nice Sophia-Antipolis : Soit un quart de journée par mois : 1032.78E

Frais de fonctionnement

- Deux postes de travail : 4000.00E
- Un rétroprojecteur (un 1/4 de son utilisation) 500.00E

Frais associés

- 14 missions : 9850.00E
- 120 heures de formations : 3857.00E

Frais d'infrastructure

- Electricité, communication téléphonique, Internet, réseau, chauffage, climatisation, place de parking, bureau, salle de réunion : coût d'environnement estimé à 30% de mon salaire brut : 22724,80E
- Subvention déjeuner : 1524.60E

L'origine du financement se répartit entre

L'Université de Nice (UNSA) : 2 318.45E décomposé en frais de comptabilité : 1032.78E et en financement de 40 heures de formations sur 120 : 1285.67E.
La région PACA, la moitié de mon salaire de base : 33215.94E.
ST Microélectronique, la moitié de mon salaire de base : 33215.94E.
L'INRIA Sophia Antipolis : tous les autres frais : 97 697,27E.

	CHARGES	RESSOURCES
Ressources humaines :	123 991.20€	Région PACA: 33 215.94€ ST Micro. : 33 215.94€ INRIA : 56 526,54€ UNSA : 1 032.78€
Frais d'infrastructure :	24 249,40€	INRIA: 24 249,40€
Frais de fonctionnement :	4 500,00€	INRIA : 4 500,00€
Frais associés :	13 707,00€	INRIA : 12 421,33€ UNSA : 1 285.67€
Total :	**166 447,60€**	**166 447,60€**

FIG. A.2 – Tableau récapitulatif des charges et ressources de mon projet

A.3 Compétences, savoir-faire, qualités professionnelles et personnelles

Compétences scientifiques et techniques

Le sujet de ma thèse concerne la conception de système sur puce. Ce domaine est à la frontière de l'informatique et de la microélectronique. J'ai dû mettre en application mes connaissances générales dans ces domaines et m'élever au niveau d'un spécialiste dans certaines niches comme la programmation haut niveau (JAVA, C/C++), la programmation bas niveau (ASM, langage RTL, langage synchrone), l'informatique théorique, l'architecture des ordinateurs, ou les procédés de fabrication de puce électronique. De plus, mon sujet de thèse a un aspect plus théorique et nécessite des connaissances mathématiques comme la théorie des automates, le model-checking, la théorie de l'ordonnancement, l'algèbre. Heureusement, ma formation d'ingénieur ESIGETEL m'a apporté la plupart de ces connaissances. Pour celles plus spécifiques comme les langages synchrones, j'ai appris au début de ma thèse aussi bien pendant l'étude bibliographique du sujet que pendant les premiers tests.

Comme pour tout travail de recherche, il est important de se tenir informé du travail des laboratoires concurrents. Cela m'a permis de développer mes compétences en veille technologique, en recherche d'information sur Internet, et en anglais (car toutes les publications sont en anglais). La recherche bibliographique et son étude m'a forcé à développer un esprit d'analyse, de synthèse, et de critique de document technique afin de juger par moi-même de l'importance des résultats d'autrui et de pouvoir le communiquer. Pour revenir à l'Anglais, j'ai amélioré ma compréhension de cette langue mais aussi ma capacité à l'écrire et à la parler car si toutes les communications du secteur sont en anglais, les nôtres le sont aussi. Là encore, mes formations précédentes (école d'ingénieur, DEA d'informatique fondamentale et appliqué) m'ont appris ces compétences qui sont des pré-requis au travail de chercheur. Mon projet m'a forcé à les utiliser.

Compétence de conduite de projet

Pendant la durée de ma thèse, j'ai dû apprendre à gérer mon temps sur de grandes périodes, allouer à chaque période des priorités et des objectifs d'étapes en accord avec les objectifs généraux. Mais aussi communiquer à l'écrit comme à l'oral sur l'avancement de mes travaux, sur mes objectifs et sur la valorisation de mes résultats comme lors des réunions trimestrielles CIMPACA ou pour des divers papiers techniques que j'ai publiés. Enfin, j'ai apporté un soin particulier à entretenir le meilleur rapport possible avec les personnels des services de soutien à la recherche (RH, financier, légaux) car ils peuvent nous dégager d'une quantité non négligeable de procédure administrative et ainsi nous permettre de nous consacrer entièrement à notre travail.

Qualités Personnelles :

Conduire ce projet m'a demandé de la rigueur, de l'organisation et de la régularité. Conduire un projet sur trois ans nécessite de poser une pierre à l'édifice tous les jours, aussi petite soit elle ; de plus, les réunions trimestrielles CIMPACA m'ont poussé dans ce sens. Personnellement, je pense que travailler dans la recherche nécessite aussi de l'intuition car explorer toutes les possibilités demande du temps, alors qu'explorer en premier celle qui deviendra la bonne en fait gagner. Une fois celle-ci trouvée, il faut faire preuve de minutie pour évaluer cette solution, prouver chacun de ses résultats comme j'ai dû le faire. Et enfin, la dernière et pas le moindre, la créativité est essentielle. On est dans un domaine où les problèmes présentés n'ont pas encore été résolus ; on doit donc inventer la solution.

Construction de mon réseau professionnel

Bien que je ne vienne de me rendre compte que récemment de l'importance d'un réseau de contact professionnel, la coopération entre industriels et universitaires au sein de CIMPACA me permet de rencontrer et de nouer des liens avec beaucoup de représentants de mon domaine d'activité. Il y a la diversité car la plupart des acteurs principaux sont représentés dans CIMPACA et donc dans mon réseau mais aussi l'importance comme des directeurs d'équipe de recherche : Robert de Simone/INRIA, Michel Auguin/CNRS, Renaud Pacalet/Eurecom , des chefs de département : Gaël Clavé/TI, André Kuntz/CTO, et même un CTO R&D France : Pierre Bricaud/Synopsys France. Auquel on peut ajouter les contacts extérieurs à CIMPACA

comme Olivier Tardieu en poste à IBM, New-York., Stephen Edward, professeur de l'université de Columbia, New-York.

Transférabilité de mes compétences

Si l'on considère mes compétences informatiques et électroniques comme des outils, je pourrais aller travailler dans n'importe quel secteur d'activité qui utilise des simulations dans le flot de développement de ces produits (chimie, physique, médecine). Parmi mes compétences, les plus théoriques peuvent être transférées vers des domaines qui touchent à l'ordonnancement comme la mise en place de chaîne de production. La plupart des compétences et savoir-faire liés à la conduite de projet sont nécessaires au poste de chef de projet ou tout autre poste de cadre, coordinateur ou manager. Et ce quel que soit le domaine de connaissance et d'expertise, elles sont donc totalement transférables vers des domaines d'activités qui nécessitent de conduire un projet.

A.4 Résultats, impacts de la thèse

Résultats et impacts scientifiques et industriels

Sur le plan industriel, dans le meilleur des cas, la méthode de conception des communications dans les systèmes sur puce que j'ai élaborés sera utilisée par les industriels de CIMPACA pour fabriquer leur nouvelle génération de puce. Ceci leur permettra d'être plus compétitif sur leur marché, de prospérer et d'engranger des retombées économiques favorables pour la région. Académiquement parlant, cette réussite contribuera au rayonnement international de mon institut : l'INRIA, et plus localement de mon équipe AOSTE. Grâce à cela, les industriels locaux seront plus enclins à faire appel aux chercheurs et plus précisément à mon équipe pour résoudre des problèmes techniques à moyen ou long terme.

Impact personnel de mon projet

Outre le renforcement de mes compétences scientifiques et techniques et la découverte du monde de la recherche publique française, ce projet m'a permis de mettre en application les compétences nécessaires à la gestion d'un projet de long terme. De plus, étant fraichement sorti de cinq années d'études supérieures, ce projet m'a permis de développer les qualités personnelles nécessaires à la vie et au travail en entreprise. La collaboration avec des industriels du secteur m'a permis, tout en gardant un regard extérieur, d'évaluer leurs besoins et leurs contraintes financières et temporelles. Et ce aussi bien pour des grands groupes comme ST Microélectronique ou Texas Instrument que pour des PME locales comme Esterel Technologie ou Scaleo Chips.

Pistes professionnelles

Concernant mes perspectives professionnelles, mon travail de thèse ainsi que les liens que j'ai établis avec mon réseau me permet d'avoir un point d'entrée comme ingénieur R&D dans les entreprises de CIMPACA ou simplement de candidater sans contact dans d'autres entreprises du domaine comme ingénieur

R&D, chef de projet ou même ingénieur technico-commercial dans des entreprises de conception ou de fabrication de systèmes sur puce. Parallèlement, je peux continuer dans la recherche publique par un post Doc, de préférence à l'étranger pour attester de ma mobilité, puis candidater à un poste de chargé de recherche au CNRS ou à l'INRIA. Contrairement à certains de mes collègues doctorants, je souhaite continuer dans la recherche. Prioritairement dans la recherche publique car bien que l'argent soit le nerf de la guerre, la pression est moins présente au jour le jour dans le public que dans le privé. Etant donné que les postes sont rares, je vais commencer par faire un post Doc en Inde, à Bangalore, au laboratoire de Recherche et Développement de Général Motors sous la direction de S. Ramesh. Mon second choix serait de conduire des recherches dans une entreprise privée de mon domaine d'activité actuel comme TI, ST Microelectronics ou NXP. Une autre possibilité est conditionnée par le fait de trouver une idée intéressante : la création d'entreprise. Pour moi, cette possibilité prévaut sur les autres mais nécessite trois conditions : un minimum d'expérience professionnelle, une idée brillante et des associés de confiance. Cette idée deviendra réalité quand ces ingrédients seront réunis.