# Techniques d'analyse et d'optimisation pour la synthèse architecturale de systèmes temps réel embarqués distribués : problèmes de placement, de partitionnement et d'ordonnancement

Asma Mehiaoui

**THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
*sous le sceau de l'Université européenne de Bretagne*

pour obtenir le titre de
**DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
*Mention : Informatique*
**École Doctorale SICMA**


présentée par

Asma MEHIAOUI

Préparée au Laboratoire d'Ingénierie dirigée par les modèles pour les Systèmes Embarqués (LISE), CEA LIST


# Techniques d'analyse et d'optimisation pour la synthèse architecturale de systèmes temps réel embarqués distribués

## Problèmes de placement, de partitionnement et d'ordonnancement

**Thèse soutenue le 16/06/2014**
devant le jury composé de :

**Emmanuel GROLLEAU**
Professeur, ISAE-ENSMA LIAS / *Rapporteur*

**Frédéric MALLET**
Maître de Conférences HDR, UNS INRIA I3S /*Rapporteur*

**Laurent PAUTET**
Professeur, TELECOM ParisTech / *Président du jury*

**Laurent LEMARCHAND**
Maître de Conférences, UBO Lab-STICC / *Examinateur*

**Jean-Philippe BABAU**
Professeur, UBO Lab-STICC / *Directeur de la thèse*

**Sara TUCCI PIEGIOVANNI**
Docteur, CEA-LIST / *Encadrant de la thèse*

# *Résumé*

Dans le cadre industriel et académique, les méthodologies de développement logiciel exploitent de plus en plus le concept de "modèle" afin d'appréhender la complexité des systèmes temps réel critiques. En particulier, celles-ci définissent une étape dans laquelle un modèle fonctionnel, conçu comme un graphe de blocs fonctionnels communiquant via des échanges de signaux de données, est déployé sur un modèle de plateforme d'exécution matérielle et un modèle de plateforme d'exécution logicielle composé de tâches et de messages. Cette étape appelée étape de déploiement, permet d'établir une architecture opérationnelle du système nécessitant une validation des propriétés temporelles du système. Dans le contexte des systèmes temps réel dirigés par les évènements, la vérification des propriétés temporelles est réalisée à l'aide de l'analyse d'ordonnançabilité basée sur l'analyse des temps de réponse. Chaque choix de déploiement effectué a un impact essentiel sur la validité et la qualité du système. Néanmoins, les méthodologies existantes n'offrent pas de support permettant de guider le concepteur d'applications durant l'exploration de l'espace des architectures possibles. L'objectif de ces travaux de thèse consiste à mettre en place des techniques d'analyse et de synthèse automatiques permettant de guider le concepteur vers une architecture opérationnelle valide et optimisée par rapport aux performances du système.

Notre proposition est dédiée à l'exploration de l'espace des architectures en tenant compte à la fois des quatre degrés de liberté déterminés durant la phase de déploiement, à savoir (i) le placement des éléments fonctionnels sur les éléments de calcul et de communication de la plateforme d'exécution, (ii) le partitionnement des éléments fonctionnels en tâches temps réel et des signaux de données en messages, (iii) l'affectation de priorités d'exécution aux tâches et aux messages du système et (iv) l'attribution du mécanisme de protection des données partagées pour les systèmes temps réel périodiques. Nous nous intéressons principalement à la satisfaction des contraintes temporelles et celles liées aux capacités des ressources de la plateforme cible. De plus, nous considérons l'optimisation des latences de bout-en-bout et la consommation mémoire. Les approches d'exploration architecturale présentées dans cette thèse sont basées sur la technique d'optimisation PLNE (programmation linéaire en nombres entiers) et concernent à la fois les applications activées périodiquement et celles dont l'activation est pilotée par les données.

Contrairement à de nombreuses approches antérieures fournissant une solution partielle au problème de déploiement, les méthodes proposées considèrent l'ensemble du problème de déploiement. Les approches proposées dans cette thèse sont évaluées à l'aide d'applications génériques et industrielles.

**Mots-clé :** systèmes temps réel critiques distribués, optimisation, analyse des temps de réponse, programmation linéaire en nombres entiers, exploration de l'espace des architectures, ingénierie dirigée par les modèles.

# *Abstract*

Modern development methodologies from the industry and the academia exploit more and more the "model" concept to address the complexity of critical real-time systems. These methodologies define a key stage in which the functional model, designed as a network of function blocks communicating through exchanged data signals, is deployed onto a hardware execution platform model and implemented in a software model consisting of a set of tasks and messages. This stage so-called deployment stage allows establishment of an operational architecture of the system, thus it requires evaluation and validation of the temporal properties of the system. In the context of event-driven real-time systems, the verification of temporal properties is performed using the schedulability analysis based on the response time analysis. Each deployment choice has an essential impact on the validity and the quality of the system. However, the existing methodologies do not provide support to guide the designer of applications in the exploration of the operational architectures space. The objective of this thesis is to develop techniques for analysis and automatic synthesis of a valid operational architecture optimized with respect to the system performances.

Our proposition is dedicated to the exploration of architectures space considering at the same time the four degrees of freedom determined during the deployment phase, (i) the placement of functional elements on the computing and communication resources of the execution platform, (ii) the partitioning of function elements into real time tasks and data signals into messages, (iii) the priority assignment to system tasks and messages and (iv) the assignment of shared data protection mechanism for periodic real-time systems. We are mainly interested in meeting temporal constraints and memory capacity of the target platform. In addition, we are focusing on the optimization of end-to-end latency and memory consumption. The design space exploration approaches presented in this thesis are based on the MILP (Mixed Integer Linear programming) optimization technique and concern at the same time time-driven and data-driven applications.

Unlike many earlier approaches providing a partial solution to the deployment problem, our methods consider the whole deployment problem. The proposed approaches in this thesis are evaluated using both synthetic and industrial applications.

**Keywords :** critical distributed real-time systems, optimization, response time analysis, mixed integer linear programming, design space exploration, Model-driven engineering.

# *Remerciements*

*Je tiens tout d'abord à remercier le directeur de thèse, M. Jean-Philippe BABAU, pour m'avoir soutenu et pour m'avoir aidé à surmonter les difficultés rencontrées pendant ces trois dernières années. Je le remercie aussi pour ses remarques pertinentes, son humour et sa gentillesse qui m'ont permis de reprendre mon souffle et de me ressaisir.*

*Mes remerciements s'adressent également à Mme Sara TUCCI-PIERGIOVANNI qui a encadré cette thèse. Sara m'a donné l'opportunité d'effectuer un travail de recherche et d'approfondir mes connaissances. Je la remercie pour ses conseils avisés, ses idées intéressantes et sa rigueur scientifique.*

*Je remercie les membres du jury, qui ont accepté d'évaluer mon travail de thèse. Merci à M. Emmanuel GROLLEAU et M. Frédéric MALLET d'avoir accepté d'être les rapporteurs de ce manuscrit, de m'avoir fait l'honneur de juger ce travail avec intérêt. Merci également à M. Laurent PAUTET et M. Laurent LEMARCHAND d'avoir accepté d'examiner ce travail et de faire partie de mon jury de thèse. Je remercie également M. Laurent LEMARCHAND pour son aide, sa disponibilité et ses discussions utiles. À tout le jury, j'exprime ma reconnaissance pour leurs remarques constructives.*

*Un grand merci à M. Haibo Zeng et M. Marco DI-NATALE qui m'ont donné l'opportunité d'intégrer la communauté des systèmes temps réel, ce fut un honneur de vous connaître et de travailler avec vous. Je voudrais remercier Chokri MRAIDHA pour son amabilité et sa disponibilité.*

*Je remercie chaleureusement mes collègues, membres de l'équipe DILS pour l'ambiance très favorable qu'ils ont su créer autour de moi. Plus particulièrement, je remercie Rania Mzid, Fadoi Lakhal, Lamine Boukhanoufa, Yassamine Seladji, Olivier Mullier, Ernest Wozniak, Ahmed Daghsen, Amel Belaggoun, Wafaa et Zaina avec qui j'ai passé de précieux moments.*

*J'adresse un immense merci à mes amis Djawida DIB, Fadéla FASNAOUI, Wahida HANDOUZI, Abdelhakim YOUCEF et Rafik SEKKAL avec qui je me suis lancée dans une si belle aventure, et je souhaite que nous partagions vivement encore d'autres expériences.*

*Un merci infini à mes chers parents que j'adore et qui ont su rester patients malgré la distance. Merci pour l'amour et la confiance que vous m'avez accordé tout au long de ma vie et un grand merci pour vos prières et votre soutien durant mes études. J'adresse un merci très particulier à ma chère petite sœur Amina pour sa présence, sa compréhension et sa complicité.*

*Je remercie fortement ma chère grand-mère, mes chers oncles et tantes ainsi que tous mes cousins et cousines pour leurs prières et leur soutien. Une pensée particulière à mes correcteurs de fautes d'orthographes de ce manuscrit, mon cousin Yassine et mes cousines Sarah et Inès.*

*Je tiens aussi à remercier profondément mes beaux-parents, ma belle-sœur Asmaa et mes*

*beaux-frères Imad, Réda et Anès pour leur sympathie et leur encouragement.*

*Enfin, mes remerciements chaleureux vont à mon cher époux Zakaria pour sa patience et son soutien quotidien indéfectible. Tu n'as pas cessé de me conforter et tu as toujours su me remonter le moral.*

Chartres, le 26 Juin 2014.

# Table des matières

# Table des figures

# Liste des tableaux

CHAPITRE 1

# Introduction générale

## 1.1 Contexte

Dans le monde actuel, les systèmes temps réel embarqués jouent un rôle de plus en plus important dans de nombreux domaines d'applications, tels que l'automobile, la robotique, l'industrie ou les télécommunications. Un système temps réel est un système pour lequel la correction ne dépend pas uniquement de l'exactitude des résultats, mais également de la date à laquelle ces résultats sont fournis [4]. Les systèmes temps réel sont qualifiés de critiques si le dépassement de la durée de temps dans laquelle les résultats doivent être produits conduit à des conséquences dramatiques. Ils sont qualifiés d'embarqués (ou enfouis) lorsqu'ils sont intégrés à un dispositif où les ressources d'exécution et de communication disponibles sont limitées. La mise en œuvre des systèmes temps réel embarqués modernes s'oriente à présent vers une architecture distribuée. Un système temps réel embarqué doit être construit afin de réaliser une fonctionnalité dédiée dans le délai spécifié. De plus, il doit pouvoir être produit et entretenu à bas prix ainsi qu'être suffisamment flexible pour être étendu à de nouvelles fonctions dédiées quand cela est nécessaire.

L'Ingénierie Dirigée par les Modèles (IDM) offre un cadre méthodologique outillé qui se base sur des modèles abstraits plutôt que sur des concepts d'algorithmique et de programmation. Elle permet ainsi de mieux maîtriser la complexité du développement logiciel de ces systèmes et de valider les choix de conception avant la phase de codage, impliquant une réduction du coût et de la durée de développement. Dans une démarche de développement à base de modèles, initialement le système est présenté par un modèle d'exigences abstrait qui est par la suite raffiné et complété jusqu'à un modèle d'implémentation à partir duquel le code peut être généré. La phase de conception est une étape intermédiaire qui consiste à associer un modèle fonctionnel à un modèle d'architecture matérielle. Le modèle fonctionnel décrit les fonctions du système et leurs interactions (représentées par des signaux de données) et le modèle d'architecture matérielle représente les ressources permettant d'exécuter les fonctionnalités du système. Le résultat de cette association est une architecture opérationnelle prête à être raffinée vers un modèle d'implémentation, conduisant ainsi à du code logiciel. La tâche principale du concepteur est de garantir une association correcte qui assure à la fois la non violation des contraintes du système et son bon fonctionnement. Les contraintes temporelles font parties des contraintes les plus importantes à considérer durant la phase de conception. Dans le cas des systèmes critiques, ces contraintes peuvent être garanties soit à l'aide d'un ordonnancement cyclique statique où l'exécution répétitive de l'ensemble des opérations du système est ordonnée dans le temps, soit en se basant sur une analyse d'ordonnançabilité dynamique qui se focalise sur le calcul du pire scénario. On distingue deux types d'analyse d'ordonnançabilité dynamique : à base de priorités dynamiques, telle que Earliest Deadline First (EDF) et à base de priorités fixes comme Rate Monotonic (RM). Cependant, en raison de l'efficacité des ressources, la plupart des systèmes temps réel embarqués sont conçus en se basant sur l'ordonnancement dynamique à base de priorités fixes.

## 1.2 Problématique et motivation

Maîtriser la complexité, réduire le temps et le coût de développement, garantir les niveaux de qualité exigés sont aujourd'hui des enjeux majeurs pour les industriels des systèmes temps réel embarqués. Comme nous l'avons mentionné précédemment, l'Ingénierie Dirigée par les Modèles a été adoptée afin d'adresser ces enjeux. Les standards existants [5, 6, 7, 8] offrent un large éventail de concepts nécessaires à la modélisation de l'architecture du système. Certains outils tels que Metropolis/Metro II [181, 182, 183] présentent un environnement méthodologique pour la modélisation, la conception et

l'analyse de ces systèmes. D'autre part, quelques méthodologies ont été proposées pour la transformation de modèles dans le but de générer automatiquement le code logiciel du système. Par exemple, les travaux présentés dans [9] se focalisent sur la génération automatique d'un modèle d'implémentation et du code logiciel à partir d'un modèle d'architecture opérationnelle. Ces solutions restent néanmoins partielles à cause de la génération manuelle de l'architecture opérationnelle du système durant la phase de conception représentant ainsi une source d'erreurs conduisant à une perte de temps et à une augmentation du coût. Dans le cadre des systèmes distribués, la génération d'une architecture opérationnelle consistant en l'association du modèle fonctionnel au modèle de la plateforme d'exécution se présente principalement en trois phases : (i) l'affectation des blocs fonctionnels aux tâches et des signaux aux messages, (ii) l'attribution des tâches aux nœuds d'exécution et des messages aux bus de communication, et enfin (iii) le calcul de l'ordonnancement des tâches et des messages. Dans les systèmes avec ordonnancement à priorités fixes, ce calcul revient à l'assignation de priorités d'exécution pour les tâches et les messages. L'objectif de cette thèse est de combler la lacune des méthodologies à base de modèles en proposant une transformation automatique adressant l'ensemble des trois phases citées ci-dessus. Le problème d'association du modèle fonctionnel au modèle de plateforme d'exécution, défini par les trois phases ci-dessus, est aussi connu sous le terme de "problème de déploiement" ou "problème de synthèse". En raison de la complexité des systèmes temps réel embarqués, il existe plusieurs alternatives pour l'intégration d'un modèle fonctionnel dans un modèle de plateforme d'exécution et chaque alternative a un impact sur les performances du système conçu. L'exploration de l'espace des solutions dans le but de trouver l'association adéquate vis-à-vis des performances du système est une tâche difficile pour le concepteur. Idéalement, ce dernier doit essayer toutes les configurations possibles et choisir ensuite la meilleure par rapport à des exigences spécifiques du système. Malheureusement, une telle approche n'est pas possible à cause du nombre élevé des paramètres de conception conduisant ainsi à un très large espace de solutions empêchant une recherche exhaustive. Par conséquent, une technique d'exploration automatique performante est indispensable pour trouver une alternative de bonne qualité. Généralement, les techniques d'exploration automatique font référence aux techniques d'optimisation. Par ailleurs, il est probable que l'utilisation de ces techniques pour l'exploration de tout l'espace des solutions ne garantisse pas que des solutions satisfaisantes soient fournies dans un intervalle de temps raisonnable. Classiquement, la solution est d'explorer indépendamment des sous espaces de l'espace global des solutions, comme cela a été considéré par l'approche proposée dans [10] adressant d'autres problèmes de conception que le problème de déploiement. Cependant, le problème majeur de cette démarche est que l'exploration n'est pas toujours bien guidée, autrement dit, elle peut exclure des solutions de bonne qualité à chaque étape de l'exploration, ce qui implique forcément une dégradation de la qualité de la solution retournée pour le problème global initial. Ce problème est dû au fait que l'exploration et la vérification des contraintes du système (réalisée par des méthodes d'analyse) sont faites séparément. Par conséquent, la qualité de la technique d'exploration dépend fortement de la stratégie utilisée pour décomposer/réduire l'espace des solutions. Dans le cadre du problème de déploiement, plusieurs solutions basées sur la stratégie de décomposition ont été proposées [11, 12, 10, 2, 13, 14, 15, 16, 17, 18, 19]. Bien que la quantité des travaux existants semble être convaincante, la résolution de l'ensemble du problème de déploiement reste encore une question ouverte. En effet, ces travaux font des hypothèses soit sur l'affectation des fonctions aux tâches et l'affectation des signaux de données aux messages [10, 14, 15, 16, 17, 18, 19] , soit sur leur attribution respective aux nœuds d'exécutions et aux bus de communication [11, 12, 2, 13] ; ces décisions étant souvent faites en se basant sur l'expertise du concepteur. Elles n'abordent donc que partiellement le problème de déploiement.

La problématique à laquelle s'intéresse cette thèse est la génération automatique d'une

architecture opérationnelle optimisée pour les systèmes temps réel critiques, embarqués et distribués, et ceci dans un cadre de développement logiciel à base de modèles.

## 1.3 Contribution

Le travail présenté dans cette thèse a pour objectif d'assister le concepteur des systèmes temps réel embarqués durant la synthèse d'une architecture opérationnelle à partir d'une architecture fonctionnelle et d'un modèle de plateforme d'exécution. Plus particulièrement, il s'intéresse à automatiser le déploiement des éléments fonctionnels sur les entités matérielles et leur association aux éléments logiciels, et ce, tout en considérant l'optimisation des performances du système. La phase d'analyse faisant partie du problème de synthèse, les travaux présentés dans cette thèse se focalisent tout d'abord sur l'analyse temporelle du système, puis sur l'exploration d'espace des architectures (DSE : Design Space Exploration) qui est basée sur la programmation linéaire mixte en nombres entiers (PLMNE). Notre contribution par rapport à chacun de ces deux aspects est donnée par la suite.

### Analyse temporelle du système

L'analyse temporelle a pour objectif de valider l'exactitude du système et d'évaluer ses performances par rapport aux propriétés temporelles. L'analyse temporelle considérée dans cette thèse concerne l'analyse d'ordonnançabilité. Cette dernière consiste à vérifier si toutes les tâches du système achèvent leur exécution avant leur échéance. À cet effet, il est nécessaire de déterminer le pire temps de réponse (la plus longue durée entre l'instant d'activation et l'instant de fin d'exécution) des tâches du système. La littérature est très riche en analyse des temps de réponse [20] de différents degrés de pessimisme et qui sont intégrées dans plusieurs outils d'analyse [21, 22]. Bien que l'analyse des temps de réponse ait été largement abordée, des interrogations persistent sur son utilisation et son intégration dans un processus de synthèse. En particulier, la communauté des systèmes temps réel considère traditionnellement les tâches comme étant les unités de base pour l'analyse des temps de réponse et suppose que les éléments fonctionnels sont masqués. Or, dans un processus de conception à base de modèles, le modèle fonctionnel représente le modèle d'entrée pour le problème de synthèse. Sachant que la répartition des fonctions sur une tâche peut avoir différents impacts sur le pire temps de réponse de la tâche selon la sémantique d'exécution considérée pour cette dernière, nous proposons une extension d'une analyse des temps de réponse existante afin de prendre en compte le niveau fonctionnel du système durant la validation des propriétés temporelles. Par ailleurs, la majorité des analyses existantes considèrent des flots d'activations linéaires. Autrement dit, les éléments constituant le modèle d'analyse sont des listes d'actions. L'analyse présentée dans cette thèse adresse les systèmes avec des dépendances non-linéaires où un élément peut avoir plusieurs prédécesseurs et plusieurs successeurs. D'un autre point de vue, les techniques d'analyse apportées pour l'ordonnancement temps réel des graphes fonctionnels se limitent à des systèmes mono-processeur et/ou à une implémentation sur des tâches strictement périodiques et qui sont parfois supposées indépendantes. Notre contribution vis-à-vis de l'analyse temporelle possède un double intérêt : d'une part, elle permet d'appliquer les théories fondamentales d'ordonnancement temps réel durant la phase de synthèse dans une approche de conception à base de modèles, et d'autre part, elle offre la possibilité de considérer des systèmes ayant une structure proche de celle des systèmes réels (des systèmes avec des dépendances non-linéaires). Cette contribution a été rendue nécessaire en vue des besoins de l'exploration d'espace des architectures.

### Exploration d'espace des architectures

L'exploration d'espace des architectures est au cœur de cette thèse. Elle représente le processus permettant d'analyser et d'évaluer différentes alternatives d'architecture opérationnelle afin de sélectionner celle optimisant les performances du système. Les approches proposées considèrent le problème de déploiement dans son intégralité : on part d'un modèle fonctionnel pour aller vers un modèle opérationnel distribué. Notre proposition consiste à fournir un cadre de synthèse facilitant l'exploration automatique de l'espace des architectures par rapport aux trois principaux degrés de liberté qui sont : le placement des blocs fonctionnels sur les processeurs et des signaux sur les bus de communication, le partitionnement des blocs fonctionnels sur des tâches et des signaux sur des messages, et finalement l'attribution des priorités aux tâches et aux messages. Ceci augmente considérablement la complexité du problème de synthèse en comparaison avec les approches existantes, car les décisions faites pour les différents degrés de liberté sont fortement liées. Par rapport à notre objectif, nous estimons que les avancées mathématiques récentes dans les formulations corrélées et les solveurs offrent une solution attractive pour le problème de synthèse. L'idée est d'utiliser un cadre d'optimisation basé sur la programmation mathématique pour traiter l'ensemble du problème de synthèse. Dans ce travail, nous considérons principalement l'optimisation du temps de réponse du système. Pour cela, nous proposons une formulation mathématique considérant simultanément les trois étapes principales du problème. En effet, le coût d'une solution optimale peut être très élevé pour les systèmes de taille moyenne, nous proposons donc une alternative qui résout le problème de synthèse en deux étapes, lui permettant ainsi de réduire le coût de résolution. Celle-ci adresse d'une façon liée les trois sous problème de synthèse, conduisant à des solutions acceptables. Notre proposition a l'avantage d'être compatible avec les méthodologies modernes de conception des systèmes temps réel embarqués puisqu'elle part d'un modèle de haut niveau d'abstraction et non pas d'un modèle intermédiaire dans le processus de conception.

Les contributions adressent à la fois les systèmes dont les activations sont pilotées par les données, et ceux ayant des activations périodiques. Dans ce dernier cas, il est indispensable de protéger les données partagées par les blocs fonctionnels à cause des activations asynchrones de ces derniers. Par conséquent, nos travaux abordent un quatrième degré de liberté qui concerne l'affectation du mécanisme de protection à ces données partagées. Ceci conduit à une optimisation multiobjectif vis-à-vis de la minimisation des latences et de la consommation mémoire.

## 1.4 Plan de la thèse

Le contenu de cette thèse est organisé en quatre chapitres selon le plan ci-dessous :

**Chapitre2 :** ce chapitre est structuré en trois parties essentielles. La première présente le contexte général de la thèse. Elle inclut les concepts de base sur les systèmes temps réel embarqués ainsi que les terminologies liées à leur ordonnancement et à leur vérification et validation. Elle discute également le développement logiciel de tels systèmes et plus particulièrement leur conception. Ensuite, cette partie met l'accent sur le processus d'exploration d'architectures qui est au cœur de cette thèse. Les deux autres parties sont dédiées aux travaux connexes. La seconde partie présente les différentes méthodes d'analyse d'ordonnancement temps réel et plus particulièrement l'analyse des temps de réponse et l'analyse des graphes fonctionnels. La dernière partie expose les approches d'exploration d'architectures qui sont classifiées selon les degrés de liberté abordés. Ces deux dernières

parties s'intéressent aussi au positionnement des travaux de cette thèse par rapport aux travaux connexes cités auparavant.

**Chapitre3 :** ce chapitre est consacré aux différentes techniques d'analyse et d'optimisation proposées dans cette thèse. Il commence par décrire le modèle de systèmes considéré dans ce travail, puis présente l'extension de l'analyse des temps de réponse apportée afin de supporter ce modèle de systèmes. Cette analyse concerne, d'une part, les systèmes ayant un modèle d'activation périodique, et d'autre part, ceux avec des activations pilotées par les données. Ce chapitre introduit par la suite le problème de déploiement adressé par nos travaux. Enfin, il termine par la présentation des deux alternatives proposées pour guider le processus de conception durant la synthèse automatique d'architectures opérationnelles du système. Ces deux approches sont basées sur la technique d'optimisation appelée "la programmation linéaire mixte en nombres entiers" (PLMNE).

**Chapitre4 :** ce chapitre concerne l'expérimentation et l'évaluation des propositions de cette thèse. Il présente les résultats obtenus par nos approches pour un ensemble d'applications synthétiques générées d'une façon aléatoire. Ces derniers concernent, d'une part, la consommation de ressources en temps de calcul et en mémoire, et d'autre part, la qualité des solutions renvoyées vis-à-vis de la moyenne des latences de bout-en-bout et de la consommation mémoire. Ce chapitre présente également une étude comparative avec les approches existantes afin de démontrer la valeur ajoutée de cette thèse. Par ailleurs, nos approches sont illustrées à l'aide d'un ensemble de systèmes automobiles, à savoir le système "Brake-By-Wire", le système "Anti-lock Brake" et le système "Cruise Control".

**Chapitre5 :** ce chapitre retrace les travaux réalisés par rapport à la synthèse automatique des architectures opérationnelles pour les systèmes temps réel critiques, embarqués et distribués. Il présente quelques améliorations qui pourraient être apportées vis-à-vis de la complexité algorithmique des approches proposées ainsi que de l'analyse des temps de réponse utilisée. Enfin, ce chapitre ouvre la voie sur une poursuite des travaux.

CHAPITRE 2

# État de l'art

## 2.1 Introduction

Cette thèse concerne principalement l'analyse et l'optimisation des applications temps réel distribuées permettant ainsi de guider le processus d'exploration de l'espace de conception. Ce chapitre est constitué de trois parties principales. Dans la première partie (partie 2.2), nous présentons le contexte de travail de cette thèse. Nous commençons par la présentation des notions basiques liées aux systèmes temps réels et à la théorie de l'ordonnancement temps réel. Puis, nous mentionnons les approches de vérification et de validation d'aspects temporels appliquées dans ce contexte. Enfin, afin de mieux situer le niveau auquel cette thèse intervient, nous abordons la méthodologie de développement des systèmes temps réel embarqués sur laquelle s'appuie ce travail. Nous discutons également le problème d'exploration architecturale qui se pose lors de la phase de conception ainsi que les différentes techniques d'optimisation utilisées dans la littérature pour traiter ce problème. La seconde partie (partie 2.3) de ce chapitre est dédiée à la présentation des approches de la littérature qui traitent l'analyse des systèmes temps réel critiques. Nous présentons des méthodes d'analyse temps réel existantes puis nous identifions les limites de chaque méthode vis-à-vis du modèle des systèmes ciblé par cette thèse. Dans la dernière partie (partie 2.4), nous nous intéressons aux approches de la littérature qui abordent l'exploration architecturale pour les systèmes temps réel embarqués. Initialement, nous citons les différentes solutions apportées au problème d'exploration d'espace de conception par rapport à différents degrés de liberté, puis nous mentionnons le positionnement de nos approches. Enfin, nous concluons ce chapitre dans la partie 2.5 en synthétisant les objectifs et les contributions de cette thèse.

## 2.2 Domaine et contexte de l'étude

Comme déjà indiqué précédemment, les systèmes temps réel embarqués constituent la cible de ce travail de thèse. Pour cette raison, la partie 2.2.1 apporte une introduction générale aux systèmes temps réel et présente le domaine de l'ordonnancement temps réel. Elle évoque également les différentes méthodes concernant la validation des propriétés temporelles. À la fin de cette partie, un ensemble de standards et de protocoles pour les plateformes d'exécution et de communication utilisées dans le domaine de cette thèse est présenté. Un aperçu sur le cadre de développement logiciel de ce type de systèmes et quelques généralités et variations sont ainsi présentés dans la partie 2.2.2. Nous mettons ensuite le point sur la méthodologie suivie par le travail de cette thèse. Cette partie détaille aussi le problème de l'exploration architecturale pour les systèmes temps réel distribués et cite un ensemble de techniques d'optimisation permettant de l'adresser.

### 2.2.1 Généralités sur les systèmes temps réel

Les systèmes temps réel sont présents dans de nombreux domaines d'activités comme le transport, les télécommunications et l'industrie. Un système temps réel se présente comme un système réactif ayant une spécification temporelle. Il est vu comme un environnement à contrôler adjoint d'un système informatique de contrôle (figure 2.1). Plus précisément, un système temps réel est en interaction permanente avec son environnement externe, représenté généralement par un procédé physique, afin de contrôler son comportement évoluant dans le temps. Le système doit être capable de détecter les changements d'états de l'environnement et de réagir sur celui-ci, en réalisant des traitements permettant de produire un changement d'état à la sortie, dans un intervalle de temps limité. L'interaction du système avec l'environnement se fait par acquisition d'informations provenant des

capteurs sous forme d'interruptions ou de mesures, et par réaction sur l'environnement en envoyant des affichages ou des commandes via les actionneurs [4].

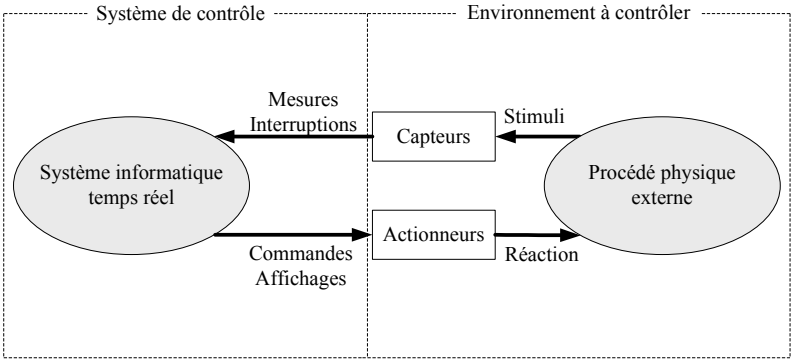



FIGURE 2.1 – Représentation schématique d'un système temps réel

**Définition** *Un système temps réel est défini comme étant un système de contrôle-commande dans lequel l'exactitude des applications ne dépend pas seulement du résultat mais aussi du temps auquel ce résultat est produit. Si les contraintes temporelles de l'application ne sont pas respectées, on parle de défaillance du système [4].*

#### 2.2.1.1 Structure interne d'un système temps réel

Un système informatique temps réel est composé de deux couches, la première concerne l'architecture matérielle et la seconde représente l'architecture logicielle. La figure 2.2 illustre la structure générale (gros grain) d'un système temps réel. De bas en haut, la couche matérielle représente l'ensemble des ressources physiques utilisées pour exécuter la partie logicielle du système. Celle-ci englobe les ressources de calculs ou de traitements (processeurs), les ressources de communication (réseaux), les ressources de stockage (mémoires) et les périphériques d'entrée-sortie (capteurs et actionneurs). Le rôle de la couche logicielle est d'assurer le fonctionnement correct du système temps réel en garantissant une bonne gestion de ressources matérielles. La couche logicielle comprend l'exécutif temps réel (système d'exploitation temps réel SETR) qui est composé de différents modules lui permettant de fournir des services de communication, de synchronisation et d'exécution. Parmi les exécutifs temps réel existants, on trouve ceux qui sont libres comme MicroC/OS-II, FreeRTOS et RTEMS, et ceux qui sont commerciaux comme VxWorks, QNX, LynxOS, Chorus, Nucleus RTOS, WindowsCE. D'autre part, la couche logicielle contient le programme informatique responsable du contrôle de l'environnement. Ce programme est constitué d'un ensemble d'entités d'exécution et de structuration appelées tâches et processus.

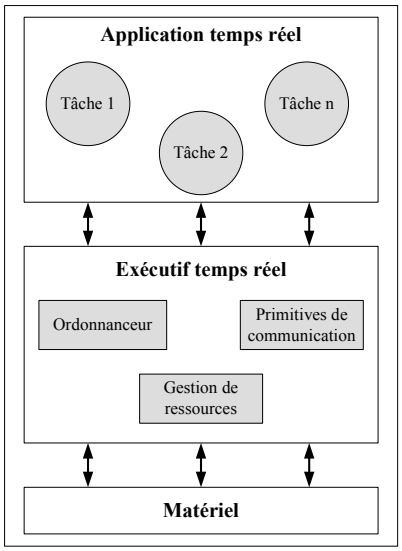



FIGURE 2.2 – Structure interne d'un système temps réel

Du point de vue du syst-me d'exploitation, le comportement d'une tâche est capturé par un automate à états finis représenté par la figure 2.3. Initialement une tâche se met en attente du processeur, elle est donc "prête" à s'exécuter. Une fois que le processeur lui est attribué, la tâche change son état et devient "active" c'est-à-dire en cours d'exécution. À ce niveau, trois situations peuvent se produire soit : (i) la tâche a besoin d'une ressource non disponible, elle passe donc à l'état "bloquée" pour que la disposition de la ressource la ramène à l'état "prête" par la suite, (ii) la tâche est interrompue pour passer le processeur à une autre tâche et devient par la suite "prête" à s'exécuter, soit (iii) la tâche a terminé son exécution et elle se met dans l'état "terminée". Cependant, à partir de cet état la tâche peut se réveiller une nouvelle fois pour demander le processeur.

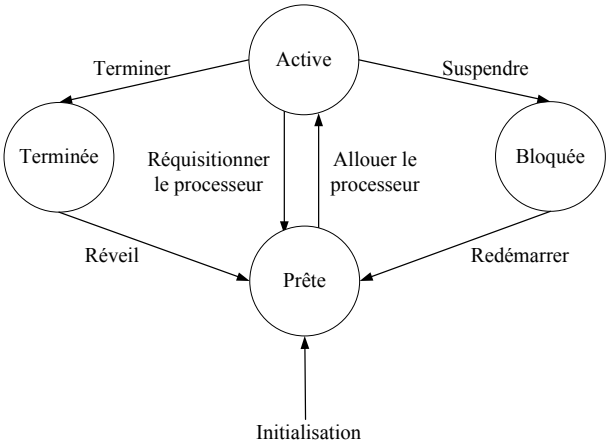



FIGURE 2.3 – États d'une tâche

Les tâches temps réel sont caractérisées par leurs propriétés temporelles. On distingue les tâches périodiques, apériodiques et sporadiques [23, 24] :

1. Une tâche périodique est une tâche récurrente dont le délai entre deux activations successives est constant et connu au préalable. La figure 2.4 résume le modèle canonique d'une tâche périodique dont les principales caractéristiques temporelles sont :

- $C_i$ **:** représente la durée d'exécution maximale de la tâche, appelé aussi le pire temps d'exécution (WCET).
- $T_i$ **:** désigne la durée entre deux activations successives de la tâche.
- $D_i$ **:** correspond à l'échéance relative de la tâche (relative deadline). Elle se définit comme étant la durée maximale attribuée à l'exécution de la tâche. C'est-à-dire qu'au-delà de cette durée, l'exécution n'est pas valide.
- $a_{ij}$ **:** détermine la $j^{\text{ème}}$ date d'activation de la tâche.
- $s_{ij}$ **:** représente la $j^{\text{ème}}$ date du début d'exécution de la tâche.
- $f_{ij}$ **:** est la $j^{\text{ème}}$ date de fin d'exécution de la tâche.
- $R_{ij}$ **:** est le temps de réponse de la $j^{\text{ème}}$ activation de la tâche.
- $d_{ij}$ **:** désigne la $j^{\text{ème}}$ échéance absolue de la tâche.

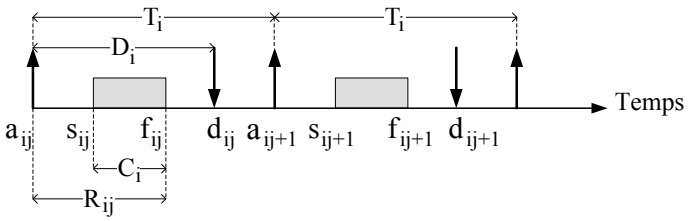



FIGURE 2.4 – Modèle d'une tâche périodique

Les tâches périodiques peuvent être caractérisées selon la relation qui existe entre leur échéance relative et leur période. Une tâche périodique à échéance sur requête représente le cas où l'échéance est égale à la période ($T_i = D_i$). Lorsque l'échéance relative est inférieure ou égale à la période, la tâche est dite à échéance contrainte. Enfin, la tâche périodique est à échéance arbitraire s'il n'existe pas de relation d'ordre entre sa période et son échéance relative.

2. Une tâche sporadique est caractérisée par une fréquence d'activation inconnue a priori. En revanche, sa période est toujours supérieure à une durée minimale ($dmin$) séparant deux activations successives ($T_i > dmin$).

3. Une tâche apériodique est un cas particulier dont les instants d'activation ainsi que le laps de temps minimal entre deux activations successives sont totalement inconnus.

Dans une application temps réel, les tâches peuvent avoir des relations de dépendance soit via le partage de ressources, soit à travers des relations de précédence. Cette dernière représente la situation où une tâche est sujette au résultat d'exécution d'une autre tâche avant de pouvoir démarrer son exécution. En d'autres termes, une tâche (réceptrice) est mise en attente d'une condition (un message) jusqu'à la satisfaction de cette condition (arrivée du message). Ainsi, l'exécution de la tâche réceptrice doit être précédée par l'exécution de la tâche émettrice, ce qui introduit des contraintes de précédence entre les deux tâches.

#### 2.2.1.2 Classification des systèmes temps réel

Selon les contraintes temporelles, les systèmes temps réel peuvent être organisés en deux catégories principales :

- **Systèmes temps réel à contraintes strictes ou dures (hard real-time)** : ce sont des systèmes pour lesquels toutes les contraintes temporelles doivent être impérativement respectées dont le non-respect peut entrainer de lourdes conséquences [25]. Ils sont aussi appelés "systèmes critiques". Des exemples de tels systèmes sont les : systèmes avioniques, systèmes spatiaux, systèmes automobiles et systèmes de transport ferroviaire.

- **Systèmes temps réel à contraintes relatives, lâches ou souples (soft real-time) :** à l'inverse des systèmes stricts, les systèmes souples tolèrent la rare violation des contraintes temporelles sans que celle-ci engendre de graves conséquences [26, 27]. Les exemples typiques d'applications ayant des contraintes souples sont les applications multimédias, les jeux vidéo et la téléphonie mobile.
- Les systèmes temps réel peuvent être à contraintes mixtes dont les contraintes temporelles sont un mélange entre les contraintes strictes et les contraintes relatives [28].

Les systèmes temps réel peuvent aussi être encapsulés dans le dispositif (un dispositif différent d'un ordinateur classique) qu'il contrôle, ils sont alors appelés *"systèmes temps réel embarqués (enfouis)"*. Ces systèmes n'ont pas d'intervention humaine directe (pas de modification du programme ou de ses paramètres) [4] et ils sont souvent conçus et créés spécifiquement pour une application donnée. On parle alors de système temps réel dédié. En plus des contraintes temporelles, les systèmes temps réel embarqués sont caractérisés par des ressources limitées, des contraintes liées à l'espace mémoire, une puissance de calcul finie et une contrainte sur la consommation d'énergie.

Nous pouvons également classifier les systèmes temps réel vis-à-vis de leur modèle de concurrence. Les systèmes dont l'architecture logicielle est mono-tâche sont particulièrement simples car le programme informatique de ces systèmes comprend une seule tâche exécutant séquentiellement les acquisitions, les traitements et les réactions. L'architecture multitâche est mise en place afin de répondre à un comportement parallèle de l'environnement possédant des rythmes variés. Dans les systèmes temps réel multitâche, l'application est constituée d'un ensemble de tâches qui coopèrent pour réaliser l'interaction avec l'environnement. La coopération peut prendre différentes formes, telles que la synchronisation, la communication et/ou le partage de ressources. Les tâches du système peuvent réagir et s'exécuter de deux façons différentes :

- Périodiquement, à des moments déterminés par un temps de référence interne au système (voir la figure 2.5). Le système est piloté par le temps (time-driven system). Étant donné que la prise en compte d'un évènement survenant de l'environnement, c'est à dire que l'exécution des tâches, se fait à des intervalles réguliers fixés par le système, le système est alors dit synchrone. Autrement dit, l'interaction avec l'environnement est synchronisée par rapport au déroulement du système.

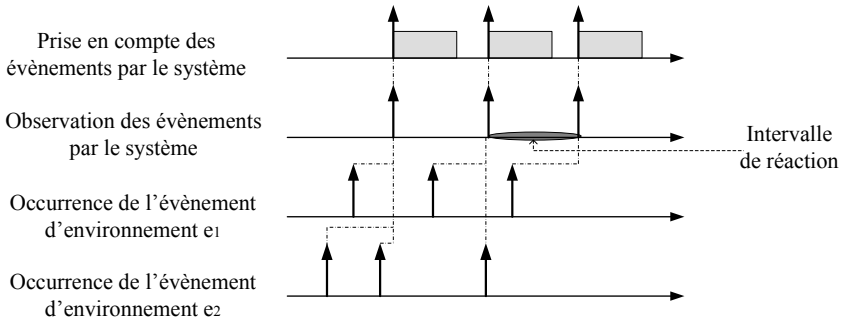



FIGURE 2.5 – Système synchrone (piloté par le temps)

- À la réception d'évènements d'interruption, le système est dans ce cas piloté par évènements (event-driven system). Le système est dit asynchrone car les évènements externes, provenant de l'environnement, sont produits à des instants aléatoires par rapport au déroulement du système et sont immédiatement pris en compte par celui-ci (voir la figure 2.6). Ces évènements peuvent être générés périodiquement par

des temporisateurs (timer) ou apériodiquement à l'acquisition de données. Pour la première situation, on dit que les tâches sont activées selon le modèle périodique, tandis que pour le deuxième cas l'activation des tâches est apériodique pilotée par les données (data-driven system).

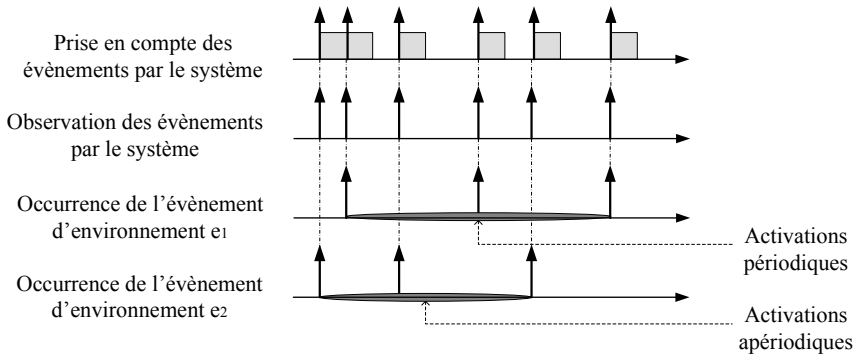



FIGURE 2.6 – Système asynchrone (piloté par évènements)

#### 2.2.1.3 Théorie de l'ordonnancement temps réel

Dans un système temps réel multitâche, les tâches ont un accès concurrent aux processeurs. De ce fait, le rôle essentiel du système est d'organiser l'exécution des tâches dans le temps et de gérer leur concurrence au sein de chaque processeur. Ce processus est appelé ordonnancement temps réel des tâches. Cet ordonnancement doit garantir la satisfaction des contraintes de temps imposées à l'exécution de l'application. Le système est dit ordonnançable lorsque toutes les contraintes de temps sont respectées. Nous définissons un ordonnanceur comme étant un composant (un programme) permettant de choisir l'ordre d'exécution des tâches sur un processeur selon une stratégie appelée politique d'ordonnancement. Nous distinguons deux classes d'ordonnancement : l'ordonnancement statique, appliqué à des tâches pilotées par le temps, et l'ordonnancement dynamique appliqué à des tâches pilotées par évènements.

1. **Ordonnancement statique :** est un ordonnancement fait hors-ligne lors de la phase de compilation. Il repose sur la définition d'une table statique représentant une séquence de planification de l'exécution des tâches. Cet ordonnancement peut être appliqué à des systèmes synchrones dont les tâches sont pilotées par le temps. Dans ce cas, le comportement du système est prévisible puisque les contraintes temporelles sont garanti au moment de la construction de la table et que l'environnement n'influence pas le déroulement de l'ordonnanceur, le système est déterministe. Cette approche se prête bien aux systèmes complexes tels que les systèmes temps réel distribués [29]. Un autre intérêt de cet ordonnancement est que le surcoût d'exécution de l'ordonnanceur est très faible. Cependant, cet ordonnancement reste très rigide à l'adaptation aux changements de l'environnement survenant pendant l'exécution du système puisqu'il suppose que tous les paramètres, y compris les dates de réveil des tâches, sont fixés. De plus, la synchronisation des horloges locales, une tâche difficile à assurer en pratique, est indispensable pour un déroulement cohérent du plan d'exécution [30].
2. **Ordonnancement dynamique :** le choix de la tâche à exécuter est déterminé en ligne en se basant sur les demandes produites au cours de l'exécution du système. C'est-à-dire en fonction des paramètres temporels (mesurant le degré d'urgence à l'exécution) des tâches prêtes, l'ordonnanceur choisit la prochaine tâche à exécuter. Ce choix peut être remis en cause lors de l'occurrence d'un nouvel évènement imprévu. Cet ordonnancement peut être appliqué à des systèmes asynchrones dont les tâches sont

pilotées par évènements. L'avantage principal de cet ordonnancement est qu'il permet au système de réagir à des évènements non prévus, et donc de s'adapter aux évolutions de l'environnement. Il est plus complexe à mettre en œuvre qu'un ordonnanceur statique et il nécessite un surcoût d'exécution pour pouvoir gérer les files de tâches à ordonnancer et déterminer le moment où une décision d'ordonnancement est nécessaire. Selon les mesures de degré d'urgence considérées par l'ordonnanceur, il existe différentes politiques pour l'ordonnancement dynamique, celle-ci résident principalement en deux classes d'algorithmes d'ordonnancement :

- ***L'algorithme d'ordonnancement à priorités fixes :*** l'ordonnancement à base de priorités fixes considère que chaque tâche est munie d'une priorité statique exprimant le degré de son urgence à l'exécution. À chaque instant l'ordonnanceur élit la tâche de plus forte priorité qui possèdera le processeur tant qu'elle n'est pas terminée ou bloquée, et tant qu'il n'y a pas l'arrivée d'une tâche plus prioritaire. Une fois une tâche plus prioritaire arrivée, la tâche en cours d'exécution s'arrête pour passer le processeur à cette dernière. Les priorités peuvent être fixées selon des règles prédéfinies comme RM (Rate Monotonic) [35] où une tâche est d'autant plus prioritaire que sa période est petite ou DM (Deadline Monotonic) [37] où une tâche est d'autant plus prioritaire que son délai critique (échéance relative) est petit.
- ***L'algorithme d'ordonnancement à priorités dynamiques :*** fonctionne avec le même principe que l'ordonnancement à base de priorités fixes, c'est-à-dire l'élection des tâches se base sur leurs priorités. La différence réside dans le fait que les priorités des tâches sont variées au cours de l'exécution. En d'autres termes, la priorité d'une tâche n'est pas fixée au préalable mais elle est calculée par rapport au temps écoulé au moment de l'exécution. Il existe différentes règles qui sont utilisées pour trouver les priorités des tâches pendant l'exécution du système, nous citons EDF (Earliest Deadline First) [35] qui consiste à attribuer la plus grande priorité à la tâche prête avec l'échéance absolue la plus proche de la date courante, et LLF (Least Laxity First) [89] où la tâche de plus haute priorité à un instant donné est la tâche qui dispose de la plus petite laxité (la durée entre la fin d'exécution de la tâche et son échéance, la fin d'exécution d'une tâche est calculée comme étant la somme de la durée du traitement restant et le temps courant [31]).

De plus, nous distinguons l'ordonnanceur selon qu'il autorise l'arrêt de la tâche élue en cours d'exécution ou pas. Nous distinguons donc l'ordonnancement préemptif de l'ordonnancement non-préemptif :

- **L'ordonnancement préemptif :** avec un ordonnancement préemptif, l'exécution d'une tâche peut être interrompue à n'importe quel instant au profit d'une autre tâche jugée plus urgente.
- **L'ordonnancement non préemptif :** dans ce cas d'ordonnancement, l'exécution d'une tâche ne peut être interrompue qu'à des instants spécifiques.
- **L'ordonnancement à préemptivité mixte :** dans cette catégorie d'ordonnancement les deux modes, préemptif et non préemptif, sont utilisés en même temps pour des tâches différentes. Chaque tâche possède un attribut indiquant son mode d'ordonnancement. L'ordonnancement non préemptif des tâches dans un contexte préemptif reste utile dans certaines situations comme le cas de tâches avec un temps d'exécution voisin du temps de changement de contexte, si l'espace mémoire RAM doit être économisé ou encore le cas où la non-préemption est nécessaire.

***Dans le présent travail de thèse nous adressons les systèmes temps réel distribués asynchrones à contraintes strictes. Nous considérons un ordonnanceur dynamique à priorités fixes car en pratique la plupart des systèmes temps réel asynchrones ont***

***recours à l'ordonnancement à priorités fixes. De plus, les fonctionnalités nécessaires à l'ordonnancement à priorités fixes sont fournies par les standards et les supports d'exécution les plus courants et les plus utilisés [24]. Nous nous intéressons également aux deux catégories des tâches pilotées par évènements : les tâches activées à la réception de données (tâches pilotées par les données "data-driven") et les tâches activées à l'arrivée d'un signal d'horloge périodique (tâches périodiques).***

#### 2.2.1.4 Validation temporelle des systèmes temps réel

La validation des systèmes temps réel concerne d'une part l'aspect fonctionnel ou comportemental, un aspect classique à toute application informatique, et d'autre part, l'aspect temporel qui consiste à vérifier les délais auxquels le système réagit aux évolutions de son environnement. Le sujet d'étude de cette thèse s'intéresse à la validation temporelle sous l'hypothèse que le comportement du système a bien été vérifié et validé auparavant. La validation des propriétés temporelles consiste à s'assurer que durant toute la vie du système, les tâches de ce dernier ne s'exécuteront jamais au-delà des échéances requises. Il s'agit donc de répondre à la question : "Est-ce que le système réalisé est ordonnançable ?" Lorsque le système est de nature synchrone, la vérification et la validation temporelles sont faites par construction au moment de la définition de la table d'ordonnancement statique. Par ailleurs, il existe différentes techniques de vérification et de validation temporelles pour les systèmes asynchrones. Celles-ci se distinguent en trois catégories : les approches à base de simulation, les approches centrées automates et les approches analytiques. Dans la suite, nous détaillons ces derniers.

Les approches analytiques se basent sur la définition de conditions d'ordonnançabilité fondées sur les critères temporels des tâches du système. Ces conditions peuvent être nécessaires, suffisantes ou nécessaires et suffisantes et elles dépendent des particularités des systèmes auxquels elles sont appliquées. D'après [32, 33, 34] il existe trois classes de techniques analytiques. La première classe représente les techniques opérant sur le facteur d'utilisation des processeurs (Processor Utilization Analysis). La seconde classe est centrée sur la demande du processeur (Processor Demand Analysis). Enfin, les approches analytiques basées sur l'analyse des temps de réponse.

- **Analyses de l'utilisation processeur :** le facteur d'utilisation du processeur se définit comme la fraction de temps que le processeur passe à exécuter des tâches. Celui-ci est calculé selon la formule $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, où $n$ est le nombre de tâche. Dans le cadre de systèmes mono-processeur de tâches périodiques, simultanées, indépendantes, à échéance sur requête et ordonnancées par l'ordonnanceur préemtif à priorités fixes RM (Rate Monotonic), le facteur d'utilisation permet de déterminer une condition d'ordonnançabilité suffisante qui est d'avoir une utilisation inférieure ou égale à $n \cdot (\sqrt[n]{2} - 1)$ [35]. Récemment, Bini et al. [187] ont amélioré cette condition en proposant la borne hyperbolique $\prod_{i=1}^{n} (\frac{C_i}{T_i} + 1) \leq 2$. Notons que pour ce type de système, dans le cas des algorithmes à priorités fixes, l'affectation de priorités aux tâches selon l'algorithme RM est optimale [35, 36]. Cependant, lorsqu'il existe au moins une tâche avec une échéance inférieure à sa période dans ce type de systèmes, c'est l'algorithme DM qui prouve son optimalité [37]. Dans ce dernier cas la condition suffisante devient $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n \cdot (\sqrt[n]{2} - 1)$. Il existe d'autres algorithmes optimaux pour différents types de systèmes à priorités fixes comme l'algorithme d'Audsley qui est optimal pour les tâches indépendantes asynchrones [38]. Pour le cas d'ordonnancement à

priorités dynamiques c'est l'ordonnanceur EDF qui est optimal pour les systèmes mono-processeur de tâches indépendantes à échéances inférieures ou égales aux périodes [39, 40]. La condition nécessaire et suffisante pour ce cas consiste à avoir une utilisation inférieure ou égale à 1. Dans un environnement multiprocesseur où les tâches sont synchrones à échéance sur requête, en classifiant les tâches selon leur facteur d'utilisation ($\frac{C_i}{T_i} \geq \frac{C_{i+1}}{T_{i+1}}$) alors la condition nécessaire et suffisante requise est donnée par la relation $\max\left[\max_{j\in[1,m]}\left[\frac{1}{j}\sum_{i=1}^{j}\frac{C_i}{T_i}\right]; \frac{1}{m}\sum_{i=1}^{n}\frac{C_i}{T_i}\right] \leq 1$, où $n$ et $m$ sont respectivement le nombre de tâches et le nombre de processeurs [4]. Cependant, pour d'autres types de systèmes, les systèmes distribués de tâches dépendantes par exemple, les conditions déterminées par le facteur d'utilisation du processeur restent des conditions nécessaires et non pas suffisantes.

- **Analyses de la demande processeur :** cette analyse est basée sur le calcul de la demande cumulée des exécutions des tâches. Elle consiste à tester pour tout intervalle de temps $[t_1, t_2]$, que la durée maximale cumulée des exécutions des tâches réveillées et terminées dans cet intervalle n'excède pas la longueur de l'intervalle, qu'elle est donc inférieure à $t_2 - t_1$ [32]. Lorsque le système est mono-processeur et composé de $n$ tâches indépendantes, périodiques, à échéances sur requêtes, synchrones et ordonnancées avec RM, une condition d'ordonnançabilité nécessaire et suffisante est définie par : $\forall i \in [1,n] : \min_{t\in S_i}\sum_{j=1}^{i}\frac{C_j}{t}\lceil\frac{t}{T_i}\rceil \leq 1$, où $S_i = KT_j, 1 \leq j \leq i : K = 1,..,\lfloor\frac{T_i}{T_j}\rfloor$ [41].
  Si les tâches sont à échéances contraintes et si elles sont ordonnancées par DM alors la condition nécessaire et suffisante reste la même à l'exception de l'ensemble $S_i$ qui devient égale à $D_i \cup KT_j, 1 \leq j \leq i : K = 1,..,\lfloor\frac{D_i}{T_j}\rfloor$. La condition d'ordonnançabilité nécessaire pour qu'un ensemble de $n$ tâches périodiques soient ordonnançables par EDF au sein d'un processeur est de vérifier que la demande processeur causée par les tâches dans tout intervalle de temps est toujours inférieure ou égale à la longueur de l'intervalle, c'est-à-dire $\forall t_1 \in [0, t_2] : \sum_{i=1}^{n} x_{i,t_1,t_2} * C_i \leq t_2 - t_1$, où $x_{i,t_1,t_2}$ représente le nombre d'instances de la tâche $i$ dont l'instant d'activation et l'instant d'échéance sont dans l'intervalle $[t_1, t_2]$ [34].

- **Analyses des temps de réponse :** les approches analytiques basées sur l'analyse des temps de réponse consistent à calculer les temps de réponse de toutes les tâches du système considérant un scénario "pire cas" puis à les comparer aux échéances dédiées (délais critiques). Si tous les temps de réponses au pire cas sont inférieurs aux échéances, alors ceci représente une condition nécessaire et suffisante à la validation temporelle du système. Ces approches reposent donc sur la caractérisation du pire scénario plutôt que sur l'exploration exhaustive de l'espace des états du système. De plus, elles peuvent s'adresser à des systèmes plus complexes comme les systèmes avec des tâches à échéances arbitraires ou avec des tâches dépendantes, ou encore des systèmes distribués. En fonction du modèle de tâches, de l'architecture matérielle et de l'ordonnancement, plusieurs techniques ont été apportées pour l'analyse des temps de réponses. Un sous ensemble de ces techniques, loin d'être exhaustif, est présenté en détail dans la partie 2.3.1 de ce manuscrit.

***Dans cette thèse, nous nous basons sur les approches analytiques et, plus précisément, sur les analyses des temps de réponse pour la validation et la vérification temporelles d' un système temps réel.***

#### 2.2.1.5 Standards et protocoles liés aux systèmes temps réel

Dans le domaine de systèmes temps réel, un certain nombre de normes ont été définies afin de couvrir toute une variété de besoins nécessaires à l'exécution correcte d'un système temps réel. Nous décrivons maintenant les normes liées aux protocoles de communications ainsi qu'aux supports d'exécution (exécutifs temps réel) et qui sont utilisées dans ce travail de thèse.

1. **OSEK :** OSEK [42] *"Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen"*, en français "systèmes ouverts et interfaces correspondantes pour l'électronique des véhicules automobiles", est une proposition de spécifications pour les systèmes automobiles. Ces spécifications concernent le SETR (RTOS en anglais pour Real Time Operating System), la communication et le protocole de gestion de réseau. Le standard OSEK a été créé en 1993 par un consortium de constructeurs automobiles allemands (BMW, Robert Bosch GmbH, DaimlerChrysler, Opel, Siemens et Volkswagen Group) ainsi que par un département de l'université de Karlsruhe. Puis, en 1994, les sociétés françaises Renault et PSA ont développé un projet similaire, VDX "Vehicle Distributed eXecutive" et ont rejoint le groupe OSEK, d'où l'appellation officielle OSEK/VDX. L'ordonnancement des tâches au sein du SETR OSEK est à base de priorités fixes. Cependant, l'ordonnancement peut être préemptif, non préemptif ou de préemptivité mixte. Le SETR OSEK permet de définir le même niveau de priorité pour plusieurs tâches. Par contre le nombre maximal de niveaux de priorité et de tâches autorisé est de 255. Dans OSEK, les tâches peuvent être activées périodiquement par un évènement d'horloge ou explicitement par des évènements en provenance de l'environnement ou d'autres tâches. Elles sont donc activées selon le modèle piloté par évènements [43, 44]. L'accès concurrent des tâches aux ressources partagées dans OSEK est assuré par le protocole PCP *"Priority Ceiling Protocol"* [45] (voir le troisième point de la partie 2.3.1).

2. **CAN :** le bus CAN [**?**] pour *"Controller Area Network"*, est un bus de communication série basé sur la technique du multiplexage. Il a été développé dans les années 80 par Bosch et conçu initialement pour les applications automobiles. Il a ensuite été utilisé dans d'autres domaines comme l'aéronautique, l'automatisation industrielle et le médical. Le bus CAN a été standardisé via des protocoles par l'ISO à partir des années 90. Dans un bus de communication CAN, les messages sont transmis sans préemption en se basant sur leur priorité. Par conséquent, un bus CAN implémente un ordonnanceur non préemptif, dynamique à base de priorités fixes. Suite à l'explosion du nombre de processeurs communicants dans les systèmes récents et menant à une complexité élevée, le bus CAN a été mis en place pour répondre aux inconvénients des connexions traditionnelles point-à-point par rapport au coût élevé de fabrication, d'installation et d'entretien. Cependant, l'inconvénient principal du bus CAN réside dans la bande passante maximale disponible qui peut facilement devenir insuffisante face au grand nombre de dispositifs interconnectés et à l'énorme quantité de données à partager. Afin de surmonter la limite des bus CAN vis-à-vis de la vitesse de transmission, d'autres standards comme le FlexRay ont été proposés. Toutefois, le bus CAN reste l'outil le plus utilisé et une solution séduisante pour les systèmes temps réel distribués et embarqués grâce à son faible coût et à sa large utilisation.

***Dans cette thèse,nous considérons des processeurs munis d'exécutif temps réel basé sur un ordonnancement préemptif à base de priorités fixes, tels que proposé par la norme OSEK, et que la communication entre les processeurs se base sur le protocole CAN.***

### 2.2.2 Développement logiciel des systèmes temps réel embarqués

La nature critique et complexe des applications temps réel embarquées nécessite la mise en place des techniques et des méthodologies de développement rigoureuses permettant ainsi de produire des applications correctes et de bonne qualité. Il existe de nombreuses méthodologies dont l'objectif est d'assister les développeurs et faciliter leur tâche de développement. Dans la suite, nous présentons la méthodologie sur laquelle repose cette thèse, à savoir la méthodologie de développement à base de modèles (partie 2.2.2.1). Ensuite, nous discutons la phase de conception dans une telle méthodologie dans la partie 2.2.2.2. Dans la partie 2.2.2.3, nous abordons l'exploration architecturale des systèmes temps réel embarqués. Enfin, dans la partie 2.2.2.4, nous spécifions les techniques d'optimisation les plus appliquées dans le domaine d'exploration architecturale et nous nous focalisons sur la technique de programmation mathématique qui est choisie pour ce travail.

#### 2.2.2.1 Développement des systèmes temps réel embarqués à base de modèles

Selon Booch [49] une méthodologie est un ensemble de méthodes appliquées tout au long du cycle de développement d'un logiciel, ces méthodes étant unifiées par une certaine approche philosophique générale. Il existe plusieurs méthodes utilisées dans le développement logiciel des systèmes temps réel embarqués telles que les méthodes fonctionnelles structurées, les méthode orientées objet et les méthodes basées sur les machines à états [4]. Les approches de développement basées sur ces méthodes considèrent que la validation du système est faite uniquement sur le test du code. Ceci a l'inconvénient d'augmenter la durée du développement dû au fait que les fautes liées à la conception ne sont découvertes que lors de la première phase d'implémentation. Néanmoins, les approches de développement basées sur le concept de modèle permettent un développement centré sur la phase de conception plutôt que sur la phase d'implémentation. Elles visent la détection des erreurs de conception avant la phase d'implémentation. Dans ce qui suit, nous présentons le domaine général de l'ingénierie dirigée par les modèles et nous nous focalisons sur la méthodologie à base de modèles suivie dans cette thèse.

L'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) en anglais, est une nouvelle technique d'ingénierie qui a été apportée récemment pour le développement de systèmes temps réel. Cette technique, est basée sur l'utilisation systématique des *modèles* tout au long du cycle de développement afin d'affronter la complexité présente dans la production du logiciel. Un modèle est défini comme une simplification d'un système, construite dans une intention particulière [58]. D'après [59], un modèle est une représentation abstraite du système qui se réfère uniquement à certains aspects du système à modéliser. En effet, le principe fondamental de l'IDM réside dans la génération de toute ou partie de l'application à partir d'un modèle de haut niveau. Le passage d'un modèle de haut niveau vers un modèle exécutable, un modèle permettant de générer l'application, se fait généralement en plusieurs phases nécessitant des raffinements et des transformations de modèles. Une transformation de modèle est une technique qui génère un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources conformément à un ensemble de règles de transformations. Dans le contexte de l'IDM, les modèles représentent des éléments productifs ainsi que le pivot du processus de développement contrairement à d'autres approches de développement où le modèle est un simple élément visuel permettant de faciliter la communication entre le concepteur et l'utilisateur. Les principes de l'IDM ont été utilisés dans le développement de différents types d'applications comme les applications ubiquitaires, les applications temps réel, les applications embarquées et les applications web [60].

Suite aux efforts apportés pour l'IDM, le groupe OMG (pour Object Management Group) en 2000 a proposé l'initiative MDA (pour Model Driven Architecture), qui peut être vue comme une variante particulière de l'IDM se reposant sur un ensemble de standards de l'OMG tels que UML (Unified Modeling Language), MOF (Meta-Object Facility), OCL (Object Constraint Language) et QVT (Query View Transformation). L'approche MDA est basée sur la séparation des préoccupations, elle permet donc de gérer l'aspect particulier de dépendance entre le logiciel et la plateforme d'exécution. Le cycle de développement MDA comprend les mêmes étapes qu'un cycle de développement classique à savoir la spécification, l'analyse, la conception, l'implémentation et la validation. La figure 2.7 montre la forme générale d'un cycle de développement MDA [61].

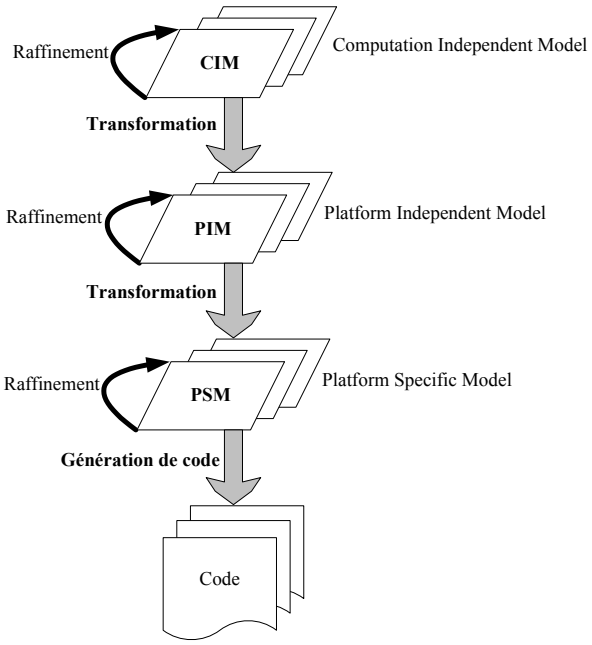



FIGURE 2.7 – Cycle de développement MDA

Le premier modèle produit lors du développement d'une nouvelle application est le modèle *Computer Independent Model* (CIM). Celui-ci permet de spécifier les exigences du client par l'intermédiaire du langage naturel, il ne contient donc aucune information sur la réalisation de l'application. Il est ainsi important que ce modèle représente l'application dans son environnement afin de mettre en clair les réactions entre les différentes entités du système aussi bien que les services devant être fournis par l'application. Une fois les exigences modélisées et validées par le client, celles-ci vont servir d'une base à la définition du modèle *Platform Independent Model* (PIM). Le modèle PIM est de type informatique et se consacre à l'étape d'analyse et de conception. Il représente une spécification formelle de la structure du système. Cependant, il décrit le système du point de vue métier indépendamment de toute plateforme technique et de toute technologie. L'étape suivante dans le cycle de développement MDA consiste à générer le modèle *Platform Specific Model* (PSM) à partir des modèles d'analyse et de conception. La différence majeure entre le modèle PIM et le modèle PSM est que le modèle PSM est lié à une plateforme d'exécution. Ce dernier est destiné à la mise en œuvre de l'application car il permet de générer automatiquement le code qui est par la suite raffiné et testé par le développeur.

Dans le contexte de l'IDM plusieurs travaux ont adapté le cycle de développement en "Y" [62, 63, 64]. La différence avec le cycle MDA est que le nouveau cycle en "Y" comporte un modèle supplémentaire appelé *Platform Description Model* (PDM), un modèle décrivant

la plateforme d'exécution. L'idée derrière ce cycle en "Y" est de construire parallèlement le modèle PIM et le modèle PDM et ensuite les joindre afin d'obtenir un modèle PSM (voir la figure 2.8). Selon ce processus de développement, les modèles CIM et PIM sont déterminés pendant la phase de spécification et la phase d'analyse. Initialement, le modèle CIM est créé et ensuite raffiné indépendamment de l'implémentation dans le but de produire un modèle PIM. Ce dernier spécifie les interactions entre les différentes entités du système. En même temps, le modèle PDM est produit afin de décrire les détails des caractéristiques techniques de la plateforme cible qui sont liées à l'implémentation. Un exemple de caractéristiques techniques est le type d'ordonnancement supporté par la plateforme. Par la suite, dans la phase de conception, les modèles PIM et PDM sont fusionnés résultant ainsi dans un modèle PSM qui est raffiné jusqu'à l'obtention du code [65].

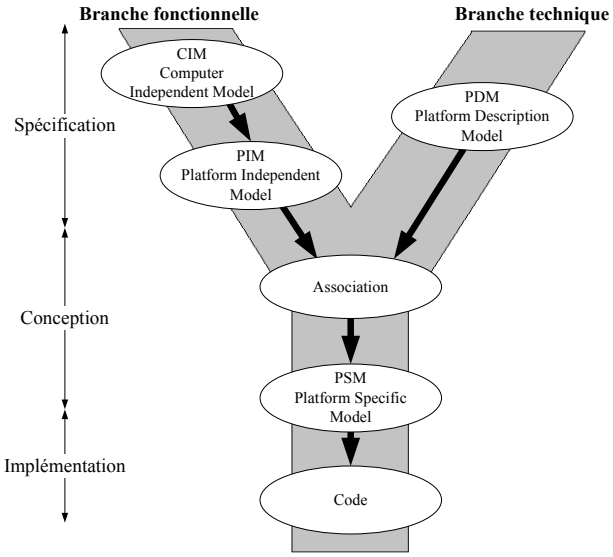



FIGURE 2.8 – Cycle de développement en "Y" selon l'approche MDA

Ces dernières années, plusieurs travaux [66, 67, 68, 69, 70] ont opté pour l'utilisation d'une approche MDA lors de développement logiciel de systèmes. La méthode Accord/$_{UML}$ [69] représente l'une des méthodologies de développement de systèmes temps réel basées sur l'approche MDA. Cette méthode a été proposée afin de démontrer la possibilité et l'efficacité d'employer une méthodologie basée sur le MDA pour le développement des systèmes temps réel. L'objectif principal de cette méthode est de faciliter la construction et le raffinement des modèles à chaque étape du cycle de développement. Pour cela un ensemble de règles de modélisation et de raffinement sont définies dans le profil UML ACCORD. Une autre méthodologie plus récente basée sur le MDA est la méthodologie Autosar [70] qui est dédiée au développement d'applications automobiles temps réel. L'approche MDA, étant basée sur les notions de l'IDM, représente un atout prometteur pour le développement logiciel des systèmes temps réel critiques [71]. Ses principaux avantages résident dans des durées de développement réduites avec des coûts peu élevés. En effet, le raisonnement sur des modèles de plus haut niveau que le code permet de se focaliser sur la structure du système tout en faisant omission des détails d'implémentation non pertinents. Ceci a le bénéfice de maitriser la complexité et de faciliter la communication entre les différentes personnes impliquées dans le développement. De plus, l'exploitation des modèles permet de valider l'implémentation à la phase de conception c'est-à-dire avant même de générer le prototype de l'application. En conclusion, l'approche MDA permet d'une part, la réduction du temps et du coût de développement, et d'autre part, le contrôle de la complexité du système et ce, grâce aux principes d'abstraction et de raffinement. Bien que le cycle de développement MDA ait eu beaucoup de succès pendant ces dernières années, il reste

encore des points ouverts sur la liaison de modèles représentant différents aspects d'une application et l'automatisation du passage d'un modèle vers un autre [65].

Une méthodologie de développement est souvent accompagnée d'un paradigme lui permettant de produire une spécification du logiciel à différents niveaux du développement. La notion de paradigme de développement se définit comme la manière de représenter le système afin de concevoir un programme informatique. Un paradigme introduit un ensemble de concepts qui sont appliqués par des langages accompagnés d'outils facilitant la spécification du système utilisant le paradigme sous-jacent. Il est extrêmement important de noter qu'une approche à base de modèles peut utiliser différents paradigmes de développement comme le flot de données, le composant, l'objet et l'architecture. Dans cette thèse, nous avons choisi de décrire le comportement de l'application en cours de développement par le paradigme de flot de données (dataflow) [50]. Un programme ou un modèle de flot de données est un graphe orienté où les nœuds, appelés acteurs, représentent les fonctions et où les arcs sont des canaux FIFO transmettant les données sous forme de jetons. Le buffer de chaque canal est de taille limitée. La figure 2.9 présente un exemple de la structure d'un graphe de flot de données. Les carrées sont les acteurs, les flèches sont les canaux de transmission et les cercles désignent les jetons. Le comportement d'un graphe de flot de données est une séquence de déclenchements (firings) de ses acteurs. Un acteur lorsqu'il est déclenché, consomme un nombre fini de jetons sur ses arcs d'entrée et produit un nombre fini de jetons sur ses arcs de sortie. Un ensemble de règles de déclenchement indique la condition sous laquelle les acteurs peuvent être déclenchés. En d'autres termes, une règle de déclenchement détermine tout simplement les jetons qui doivent être disponibles à l'entrée de l'acteur pour qu'il soit activé. En fonction de la règle de déclenchement, un ensemble de graphes de flot de données spécifiques est déterminé. Nous citons comme exemple de tels graphes, le graphe de flot de données synchrone (SDF), le graphe de flot de données synchrone homogène (HSDF), le graphe Cyclo-Static (CSDF) et le graphe aux taux fractionnaires (FRDF) [51].

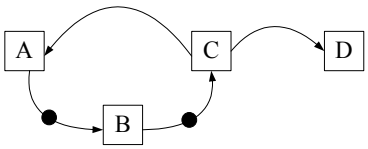



FIGURE 2.9 – Exemple d'un graphe de flot de données

Le paradigme de flot de données synchrone (SDF) [52] est le paradigme le plus utilisé dans la spécification des systèmes temps réel embarqués. Celui-ci exige que chaque acteur consomme et produit un nombre fixe de jetons qui est toujours le même pour toutes ses activations. Lorsque le nombre de jetons consommés et produits sur chaque arc est égale à 1, alors le graphe est dit homogène (HSDF).

***Le travail apporté dans cette thèse se situe dans le contexte de développement logiciel de systèmes temps réel suivant le cycle en "Y" selon l'approche MDA. Le paradigme utilisé pour représenter le modèle de l'application est le paradigme de flot de données synchrone et homogène.***

#### 2.2.2.2 Conception des systèmes temps réel

La phase de conception est au cœur du processus de développement de systèmes temps réel. Elle produit un modèle qui est une abstraction de l'implémentation du

système, d'où son impact important sur la qualité et la correction du logiciel produit. Dans le cadre du développement logiciel basé sur le cycle en "Y" selon l'approche MDA, la phase de conception (voir le flux de conception présenté par la figure 2.10) consiste à définir une architecture *opérationnelle* à partir d'une architecture *fonctionnelle* et d'une architecture *matérielle*. L'architecture fonctionnelle est une description de la structure de l'application à l'aide de fonctions et de leurs interactions . D'après [72], une architecture fonctionnelle, appelée parfois architecture logique, est un modèle de la structure et du comportement des fonctions de l'application. Elle spécifie les différents services demandés par l'application comme les services de stockage et ceux de calcul et elle inclue également les diverses exigences extraites lors de la phase de spécification. L'architecture matérielle ou l'architecture technique, correspond d'une part, à l'organisation de l'ensemble d'équipements permettant de supporter l'architecture fonctionnelle tels que les ressources de calcul, de stockage et de communication, d'autre part, elle correspond au logiciel impliqué dans la gestion de ces équipements, comme par exemple l'algorithme d'ordonnancement utilisé par l'exécutif. Ces équipements ont des caractéristiques spécifiques décrivant leur capacité. Notons que l'architecture fonctionnelle est différente de l'architecture logicielle. L'architecture logicielle représente l'ensemble des tâches et des messages offert par l'architecture matérielle et qui est utilisé pour implémenter les éléments de l'architecture fonctionnelle.

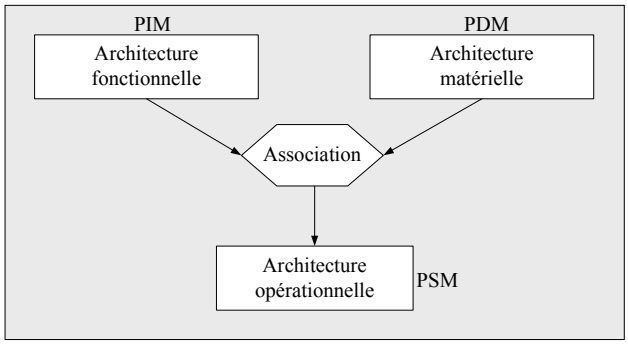



FIGURE 2.10 – Flot de conception selon la philosophie en "Y"

Lors de la phase de conception, la spécification de l'application et celle de la plateforme d'exécution sont initialement séparées. L'architecture opérationnelle est le résultat de l'association de l'architecture fonctionnelle et de l'architecture matérielle. Cette intégration correspond au placement des composants de l'application (fonctions et leurs interconnexions) sur les entités d'exécution et de communication (processeurs, réseaux, bus, tâches, messages) et au paramétrage nécessaire à la mise en place des services liés à l'exécution de l'application comme la configuration des priorités qui est nécessaire à l'ordonnancement de l'application. Au final, l'architecture opérationnelle détermine l'ensemble des éléments nécessaires à l'implémentation du système, sous forme de programmes concurrents et communicants via le partage des ressources communes. L'architecture opérationnelle à laquelle le concepteur a abouti subira différentes évaluations permettant de déterminer la qualité des choix de conception considérés lors de l'intégration. Dans ce qui suit, nous présentons le problème qui se produit en raison de plusieurs façons de déterminer une architecture opérationnelle ainsi que les solutions apportées.

#### 2.2.2.3 Exploration d'espace des architectures

Durant le développement d'un système temps réel, les concepteurs sont confrontés à de nombreuses questions telles que : quel est le nombre nécessaire et le type des ressources matérielles ? Comment distribuer les fonctionnalités du système ? Comment

respecter au mieux les contraintes de temps, de fiabilité et de robustesse ? Le défi majeur pour les concepteurs de systèmes temps réel critiques consiste à analyser de nombreuses possibilités de conception afin de pouvoir répondre aux différentes questions posées. Généralement ces possibilités diffèrent en fonction des paramètres de conception tels que le dimensionnement des ressources et leur structure, la taille de la mémoire, les politiques et les paramètres d'ordonnancement, le choix de l'architecture logicielle, etc. Chaque choix fait sur l'un des paramètres de conception peut avoir une influence importante sur les performances du système produit. L'ensemble des alternatives possibles forme l'espace de conception. Il existe plusieurs mesures d'intérêt comme le temps, l'utilisation des ressources, la consommation d'énergie, le coût, etc. permettant d'évaluer les performances obtenues. Toutefois, les différents aspects caractérisant une application temps réel, tels que la concurrence, le partage de ressources et le comportement dynamique, ainsi que l'orthogonalité des mesures d'intérêts rendent la prise de décision sur les paramètres de conception souvent très difficile à réaliser. Traditionnellement, les concepteurs se basent sur leur intuition ou une expérience métier pour effectuer un choix particulier. Après avoir pris des décisions sur les paramètres de conception, le concepteur doit s'assurer que les choix faits mènent à un système performant par rapport aux mesures d'intérêts considérées et que ce dernier respecte bien les exigences. Cette démarche de conception basée sur l'expertise du concepteur est appelée conception par essai-erreur. Une stratégie d'essai-erreur est souvent manuelle et par conséquent longue et couteuse. De plus, elle ne permet généralement d'aborder qu'une petite sous partie de solutions parmi toutes les alternatives possibles. La figure 2.11 représente le flot de conception enrichi par une description de la stratégie d'essai-erreur. Une fois que l'architecture opérationnelle est définie, celle-ci est analysée afin de vérifier sa justesse et d'évaluer ses performances. Les résultats de l'analyse sont utilisés soit pour valider l'architecture opérationnelle, soit pour améliorer ses performances. Cette dernière situation consiste à modifier la spécification de l'application, celle de la plateforme matérielle et/ou leur association.

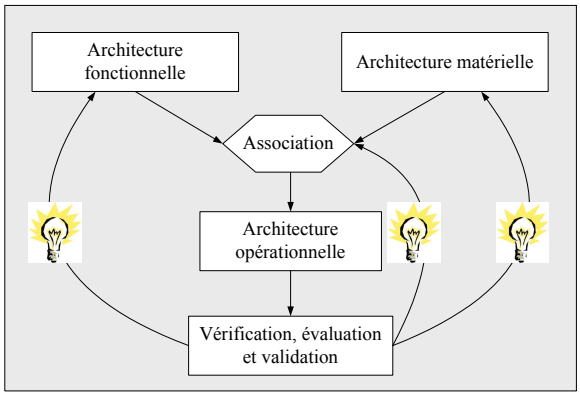



FIGURE 2.11 – Exploration de l'espace des architectures basée sur la stratégie d'essai/erreur [1]

Afin de remédier aux limites de la stratégie d'essai/erreur, l'automatisation de l'exploration de l'espace de conception devient cruciale. Le processus d'exploration de l'espace de conception est nommé "l'exploration architecturale". L'exploration architecturale permet d'assister les concepteurs dans leur prise de décision effective sur les différents paramètres de conception, notamment lorsque l'espace des solutions est vaste. L'idée est de fournir une méthode permettant d'explorer efficacement les différentes alternatives de conception et de retenir par la suite la (les) solution(s) de conception satisfaisant les exigences du système et améliorant ses performances.

***Dans le cadre de cette thèse nous supposons que les architectures (fonctionnelle***

***et matérielle) sont fixées et on s'intéresse à l'exploration architecturale vis-à-vis de leur association. Autrement dit, on se focalise sur l'étape d'intégration de l'architecture fonctionnelle à l'architecture matérielle qui représente une étape particulière de la phase de conception, on appelle cette étape "étape de déploiement". Plus particulièrement, notre objectif est de proposer une méthode qui offre une assistance aux concepteurs lors de l'allocation des éléments de l'architecture fonctionnelle (fonctions et leurs interconnexions) sur les ressources matérielles et logicielles (processeurs, bus, tâches, messages) tout en considérant la configuration de l'ordonnancement qui consiste en l'affectation de priorités aux entités logicielles.***

Dans la littérature, l'exploration architecturale pour la phase de conception en général et pour la phase de déploiement en particulier, a souvent été menée à l'aide des techniques d'optimisation combinatoire. Dans ce contexte, une technique d'optimisation est une stratégie permettant de chercher une architecture opérationnelle qui vérifie les exigences spécifiées et qui est optimisée par rapport à une mesure d'intérêt donnée. Dans la suite nous discutons les différentes techniques d'optimisation utilisées dans le cadre de l'exploration architecturale puis nous nous focalisons sur la présentation de la technique choisie pour la thèse.

#### 2.2.2.4 Techniques d'optimisation pour l'exploration architecturale

Un problème d'optimisation consiste à trouver les meilleures valeurs pour un ensemble donné de variables, appelées variables de décision, par rapport à certains critères donnés, exprimés par une fonction objectif, tout en respectant les conditions spécifiées, représentées comme des contraintes [73]. Lorsque le domaine des valeurs des variables de décision est discret, le problème est dit un problème d'optimisation combinatoire. La résolution d'un tel problème repose sur deux questions : la modélisation du problème et l'utilisation de la méthode informatique adéquate. À ce type de problèmes, différentes techniques de résolution ont été apportées. Dans la suite, nous donnons un aperçu des principales techniques utilisées dans le contexte de l'exploration d'espace des architectures. Nous mentionnons tout d'abord les approches de la programmation mathématique et certaines méta-heuristiques, puis nous nous focalisons sur la programmation linéaire mixte en nombres entiers.

1. **Aperçu d'une gamme de techniques d'optimisation utilisées pour l'exploration architecturale des systèmes temps réel embarqués :** la programmation mathématique représente la première catégorie de techniques d'optimisation étudiée. Celle-ci se base principalement sur la construction d'un modèle mathématique décrivant le problème d'optimisation considéré. Elle permet de trouver une solution optimale à un problème de taille raisonnable lorsque celle-ci existe ou de déterminer l'écart entre la solution obtenue et la solution optimale. Cependant, malgré les progrès réalisés en matière de calcul parallèle et distribué, les méthodes de la programmation mathématique rencontrent généralement des difficultés avec les applications de taille industrielle à cause du temps de calcul qui risque d'augmenter exponentiellement avec la taille du problème. À l'opposé de la recherche exhaustive, ces méthodes parcourent l'espace des solutions à l'aide d'une énumération implicite. Les techniques de programmation mathématique se distinguent généralement par rapport à la forme du problème d'optimisation à considérer (voir le tableau 2.1).

   L'algorithme du Simplexe [75] est le plus utilisé dans la résolution des problèmes PL. Il permet d'obtenir la solution optimale en parcourant uniquement les sommets du polyèdre représentant l'espace des solutions admissibles. Pour les problèmes PLNE et PLNEM, un algorithme d'évaluation et de séparation (Branch-and-Bound) est

Tableau 2.1 – Classification des techniques de la programmation mathématique

| Forme des contraintes et de la fonction objectif | Type des variables | Type de problème | Propriétés |
|---|---|---|---|
| Fonctions mathématiques linéaires | $\in \mathbb{R}$ | PL | Convexe $\Rightarrow$ l'espace des solutions admissibles formé par les contraintes du problème est un polyèdre convexe |
| | $\in \mathbb{N}$ ou $\in \{0,1\}$ | PLNE | Non convexe |
| | $\in (\mathbb{R} \cup \mathbb{N})$ | PLMNE | |
| Fonctions mathématiques non linéaires | $\in \mathbb{R}$ | PNL | Un problème non linéaire peut être convexe ou non (tel qu'un PNLNE) [74] |
| | $\in \mathbb{N}$ ou $\in \{0,1\}$ | PNLNE | |
| | $\in (\mathbb{R} \cup \mathbb{N})$ | PNLMNE | |

**PL :** problème de programmation linéaire [75]
**PNL :** problème de programmation non linéaire [76]
**PLNE :** problème de programmation linéaire en nombres entiers [77]
**PNLNE :** problème de programmation non linéaire en nombres entiers [76]
**PLMNE :** problème de programmation linéaire mixte en nombres entiers [77]
**PNLMNE :** problème de programmation non linéaire mixte en nombres entiers [76]

le plus utilisé puisqu'il permet une optimisation globale des programmes linéaires non-convexes [77]. L'optimalité de la solution trouvée par "Simplexe" et "Branch-and-Bound" est garantie uniquement lorsque l'algorithme achève son parcours. Le principe de ces algorithmes est détaillé dans la partie suivante. D'autre part, les méthodes de Newton, de Lagrange et de points intérieurs ont été proposées pour résoudre un problème d'optimisation non linéaire. Celles-ci ne garantissent pas une solution optimale au problème global mais sous certaines conditions, un minimum (maximum) local d'une fonction objectif convexe (concave) sur un espace des solutions admissibles convexe représente aussi un minimum (maximum) global du problème non linéaire [74]. Généralement les méthodes d'optimisation globale d'un problème non linéaire non convexe sont basées sur le principe de relaxation continue [78, 79]. Cependant, elles requièrent des types spécifiques de contraintes et de fonction comme des fonctions factorisables et/ou non séparables.

Il existe un autre ensemble de techniques d'optimisation qui peuvent être appliquées à des problèmes ne possédant pas une formulation algébrique appropriée. Celles-ci se contentent de trouver une solution réalisable dans un intervalle de temps raisonnable sans disposer en retour d'information sur l'optimalité ou la qualité des solutions obtenues. Ces techniques résident dans les heuristiques qui représentent d'une part les heuristiques dédiées, et d'autre part, les méta-heuristiques. Les heuristiques dédiées fournissent des règles simples permettant d'indiquer la solution qui semble être la plus efficace par rapport à un objectif fixé parmi plusieurs solutions alternatives. Cependant, la plupart d'entre elles sont spécifiques à un problème donné et ne peuvent pas être adaptées, avec seulement des modifications mineures, à d'autres types de problèmes d'optimisation [80]. À l'opposé, les méta-heuristiques sont destinées et adaptables à un grand nombre de problèmes d'optimisation. Néanmoins, les nombreux paramètres qui les contrôlent sont délicats à régler et influencent directement la qualité de la solution retenue. De plus, la modification des contraintes

exige une reconfiguration des paramètres de la méthode. Les méta-heuristiques les plus fréquentes dans le domaine de l'exploration architecturale sont :

- **Les algorithmes génétiques** (GA pour Genetic Algorithm) [81] font partie des approches évolutionnaires et ils sont inspirés de la sélection naturelle dans l'évolution biologique des espèces. Leur principe clé consiste à créer aléatoirement une population initiale d'individus puis à faire évoluer la population itérativement à l'aide des méthodes de sélection et des opérateurs basiques de croisement et de mutation pour explorer l'espace de recherche. Plus précisément, une population représente l'espace de recherche pour l'algorithme génétique et chaque individu (appelé aussi chromosome) de la population consiste en une solution possible pour le problème d'optimisation considéré. Un chromosome est un ensemble de gènes où chaque gène code par exemple une variable de décision du problème. Chaque individu de la population est d'une certaine performance mesurée par une fonction d'évaluation (fitness). Les résultats de la fonction d'évaluation permettent de construire la prochaine population à partir de la population courante. Ceci consiste à sélectionner ou rejeter un individu de la population courante et de reproduire d'autres bons individus pour ne garder au final que les individus ayant une bonne performance.
- **Le recuit simulé** [82] est une méthode probabiliste dont le nom et le principe sont inspirés du processus de recuit utilisé en métallurgie. Le recuit est une opération consistant à laisser refroidir lentement un métal pour améliorer ses qualités. L'idée physique est d'éviter un refroidissement brutal qui peut bloquer le métal dans un état peu favorable. L'idée du recuit simulé en optimisation est d'éviter que l'algorithme ne reste piégé dans des minima locaux. Elle consiste aussi à accepter une dégradation de la fonction objectif, avec une certaine probabilité, pendant des premières phases de recherche de solution puisque cette dégradation peut entraîner une amélioration ultérieure.
- **Les algorithmes gloutons** [83] se basent sur le principe de construction incrémentale de la solution finale. À chaque étape, une décision est faite par rapport à un optimum local dans l'espoir d'obtenir un résultat optimum global et les choix effectués dans les étapes précédentes n'étant jamais remis en cause.

2. **Programmation linéaire mixte en nombres entiers (PLMNE) :** nous avons vu précédemment que les méthodes utilisées pour aborder l'exploration architecturale lors de la phase de déploiement sont nombreuses. Cependant, le travail de cette thèse repose sur la technique de programmation linéaire mixte en nombres entiers. L'existence de méthodes de résolution performantes et efficaces en pratique ainsi que la disponibilité d'outils informatiques génériques et puissants, tels que CPLEX, GLPK, GAMS et MATLAB, permettant d'appliquer aisément ces méthodes peuvent justifier notre choix. S'ajoutant à cela, la programmation linéaire mixte en nombres entiers permet une modélisation plus flexible en comparaison aux autres techniques d'optimisation citées. Autrement dit, l'ajout, la suppression et la modification de contraintes ainsi que la modification de l'objectif d'optimisation se font facilement sans la nécessité d'une reformulation ou d'une reconfiguration particulière. Comparativement à la programmation linéaire, la programmation linéaire mixte en nombres entiers permet de modéliser une classe plus large de problèmes. Elle est ainsi mieux adaptée à notre problème de déploiement.

   - **Définition :** un programme linéaire en nombres entiers permettant d'exprimer un problème de minimisation/de maximisation peut s'écrire respectivement sous les formes standards suivantes [76] :

$$minimiser \quad c^T x \qquad\qquad -maximiser \quad -c^T x$$

$$\text{Soumis à} \quad Ax \leq b \qquad\qquad \text{Soumis à} \quad Ax \leq b$$
$$x \geq 0 \qquad\qquad x \geq 0$$

Où $x = (x_1, x_2, ..., x_n)$ est un vecteur de variables de décisions entières positives. Lorsqu'on parle de programme linéaire mixte en nombres entiers, un sous ensemble de variables de ce vecteur sont des réelles positives. $A$ est une matrice constante de $m$ lignes et $n$ colonnes. $b$ et $c$ sont des vecteurs constants de dimension $n$. La première ligne représente la fonction objectif à optimiser. Les lignes suivantes sont les contraintes du problème.

- **Résolution :** la majorité des solveurs modernes, comme CPLEX, utilisent la méthode de "Branch-and-Cut" pour résoudre des problèmes d'optimisation linéaire en nombres entiers. CPLEX utilise aussi une variante de "Branch-and-Cut" qui est la recherche dynamique. Cette méthode est basée sur la méthode de séparation et d'évaluation (Branch-and-Bound) et sur la méthode des plans sécants (Cutting-plane). Elle utilise l'analyse de propriétés du problème pour éviter l'énumération de mauvaises solutions afin de ne sélectionner que des solutions potentiellement bonnes [84, 85]. La méthode de séparation et d'évaluation repose, d'une part sur la séparation (branch) de l'ensemble des solutions en sous ensembles plus petits, et d'autre part, sur l'évaluation des solutions d'un sous ensemble en appliquant des bornes (bound) à la valeur de la meilleure solution de ce sous ensemble, ce qui permettra d'explorer uniquement les sous ensembles susceptibles de contenir de meilleures solutions. Comme l'illustration par l'exemple est un bon moyen pour comprendre un concept, nous considérons pour la suite un problème de programmation linéaire en nombres entiers (P) comprenant quatre variables de décision binaires $(x_1, x_2, x3, x_4)$. Soit $z$ le coût de la fonction objectif à minimiser. Veuillez noter que l'énumération explicite pour P résulte en $2^4 = 16$ solutions possibles. L'utilisation d'un arbre est la manière classique permettant une énumération totale des solutions possibles, comme indiqué ci-dessous :

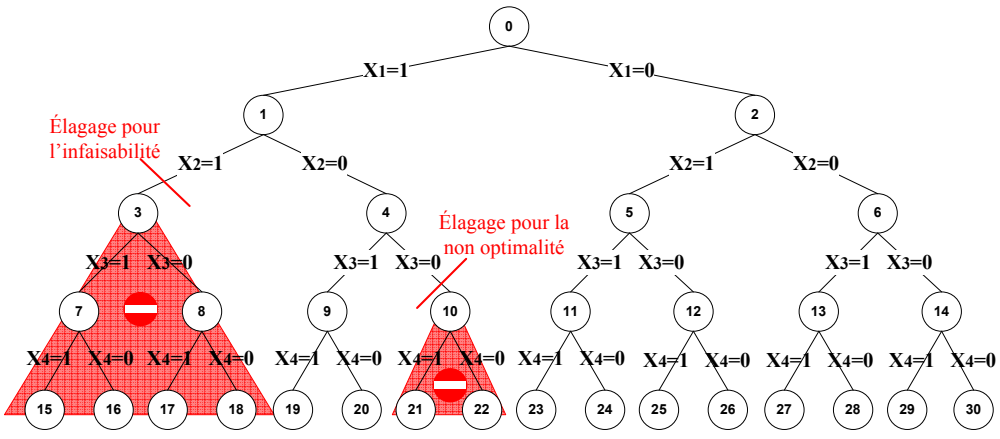



FIGURE 2.12 – Arbre d'énumération explicite pour un problème à quatre variables binaires permettant d'illustrer la méthode de séparation et d'évaluation

Le nœud racine de l'arbre représente le point de départ où aucune variable n'est décidée. À ce niveau, on dit que l'on "sépare sur le premier élément", qui dans notre cas correspond à la variable $x_1$. En d'autres termes, $x_1$ peut prendre les valeurs 1 ou 0. Nous créons donc deux branches, chacune représente une solution partielle où $x_1$ est fixé et où les autres variables ne le sont pas encore. Dans tous les nœuds descendants, la variable $x_1$ prendra toujours la même valeur : celle fixée sur la branche considérée. Par exemple, au niveau des nœuds 1, 3-4, 7-10 et 15-22, la solution considère la valeur 1 pour la variable $x_1$. Cette opération est répétée pour le

niveau suivant de l'arbre où la séparation est faite sur la variable $x_2$ et ainsi de suite jusqu'au dernier niveau de l'arbre où les feuilles (les nœuds 15-30) représentent des solutions finales au problème P. Le principe de la méthode de séparation et d'évaluation est basé sur l'arbre d'énumération explicite. Cela consiste à énumérer implicitement toutes les alternatives possibles. L'idée est la suivante : dans l'arbre d'énumération explicite, à chaque nœud $x$, s'il y a un moyen de détecter que la solution optimale ne peut pas se produire dans l'un de ses descendants, alors il n'est pas nécessaire d'examiner les solutions dans les branches initialisées par ce nœud. On dit alors que nous avons élagué l'arbre au niveau du nœud $x$. L'élagage au niveau du nœud $x$ peut être dû à deux raisons. La première est que la solution partielle à ce niveau n'est pas réalisable (elle viole une ou plusieurs contraintes). Supposons que le problème $P$ possède une contrainte qui est $3x_1 + 4x_2 \leq 6$, alors l'arbre peut être élagué au niveau du nœud 3 puisque la solution où $x_1 = 1$ et $x_2 = 1$ ne respecte pas cette contrainte. La deuxième raison d'élagage est que le coût de la solution partielle au niveau de ce nœud est moins attractif que le coût de la solution courante. Admettons que nous ayons examiné toute la branche du nœud 0 jusqu'au nœud 19 et que le coût pour la solution obtenue (la solution $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$) est $z = 15$. Cette solution représente une solution courante au problème P. Puis, pendant le parcours d'une autre branche, nous croisons le nœud 10 pour lequel le coût de la solution partielle (la borne inférieure) est $z = 17$. Ceci implique qu'il n'est pas nécessaire de continuer à parcourir les branches initiées par le nœud 10 puisque le coût des solutions aux nœuds 21 et 22 n'est pas meilleur que la borne inférieure.

Le calcul des bornes (inférieures pour un problème de minimisation et supérieures pour un problème de maximisation) se fait à l'aide de l'algorithme du Simplexe appliqué au problème relaxé du problème original. La relaxation continue (ou linéaire) $R_P$ d'un problème de programmation linéaire en nombres entiers $P$ consiste à interpréter d'une façon continue un problème d'optimisation de variables discrètes. Elle est obtenue par le relâchement des contraintes d'intégrité ($\forall i : x_i \in \mathbb{N}$ devient $\forall i : x_i \in \mathbb{R}$). Pour un problème à variables binaires la contrainte $\forall i : x_i \in 1, 0$ devient $\forall i : x_i \in [0, ..., 1]$. Le résultat de la relaxation est un problème de programmation linéaire qui peut être résolu d'une façon optimale par l'algorithme du Simplexe. Si $F(P)$ est la région des solutions admissibles pour le problème $P$ alors $F(P) \subseteq F(R_P)$. Dans la suite, $x^*$ et $\overline{x}$ représentent respectivement la solution optimale de $P$ et de $R_P$. La solution optimale pour $R_P$ est une borne pour $P$. Afin d'illustrer l'algorithme du Simplexe et le principe de relaxation, nous considérons le programme linéaire en nombres entiers $P$ suivant dont la relaxation résulte dans le programme linéaire $R_p$.

$$P : \begin{cases} minz = -3x_1 - 5x_2 \\ x_1 + 2x_2 \leq 3 \\ 6x_1 + 8x_2 \leq 15 \\ x_1, x_2 \in \mathbb{N} \end{cases} \qquad R_P : \begin{cases} minz = -3x_1 - 5x_2 \\ x_1 + 2x_2 \leq 3 \\ 6x_1 + 8x_2 \leq 15 \\ x_1, x_2 \in \mathbb{R} \end{cases}$$

La résolution graphique du problème $R_P$, montrée par le graphe droit de la figure 2.13, consiste tout d'abord à représenter graphiquement toutes les contraintes du problème par des demi-plans (les lignes en pointillés). L'intersection de ces demi-plans est un ensemble convexe déterminant la région des solutions admissibles. La solution optimale fait donc partie de cet ensemble. Pour le problème de programmation en nombres entiers $P$, les solutions admissibles sont discrètes et sont représentées par des ronds noir sur le graphe gauche de la figure. Les solutions

pour le problème relaxé sont quant à elles continues. Les figures nous montrent sans ambiguïté que $F(P) \subseteq F(R_P)$. Par la suite, il nous faut chercher à l'intérieur de la région des solutions admissibles, la solution $(x_1, x_2)$ minimisant $z$. Dans l'exemple, la minimisation de $z$ revient à la maximisation de $x_1$ et $x_2$. Pour cela, nous définissons graphiquement la fonction objectif par une ligne droite, à savoir la droite $-3x_1 - 5x_2 = 0$ passant par l'origine et donnant une solution dont la valeur du coût est nulle. La maximisation de $x_1$ et $x_2$ consiste à éloigner la droite de l'origine en la déplaçant vers le haut. Pour respecter toutes les contraintes du problème, cette droite ne doit pas dépasser les limites de la région de solutions admissibles. Le dernier point atteint par la droite dans la région est la solution optimale. Celle-ci représente le point $\overline{x} = (\frac{3}{2}, \frac{3}{4})$. Nous remarquons que la solution optimale se trouve nécessairement sur le pourtour de la région des solutions admissibles.

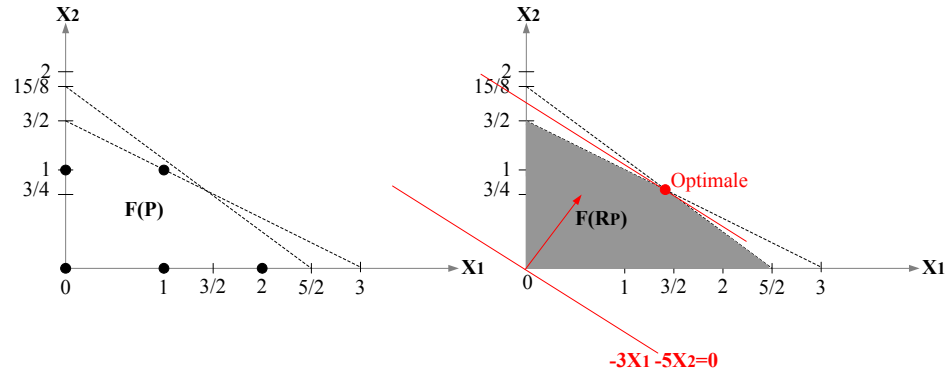



FIGURE 2.13 – Résolution graphique de $R_P$

La résolution graphique d'un problème à deux dimensions (deux variables) permet de comprendre l'algorithme du Simplexe dans le cas général (plusieurs variables) sans détailler son aspect mathématique. Considérons un espace à $n$ dimensions, les contraintes délimitent un polyèdre convexe qui représente la région des solutions admissibles. La fonction objectif est un hyperplan que l'on déplace le plus loin possible de l'origine, jusqu'à l'extrême limite. Étant donné que la solution optimale se trouve forcément sur le pourtour du polyèdre, la méthode du simplexe consiste à itérer sur tous les sommets du polyèdre en sélectionnant le sommet optimisant la fonction objectif. En conclusion, la solution optimale $\overline{x}$ pour $R_P$ représente une borne pour la solution optimale du problème $P$.

L'algorithme (l'algorithme 1) et l'organigramme suivants décrivent le principe général de la méthode "Branch-and-Cut".

**Algorithme 1 :** L'algorithme "Branch-and-Cut"

**Data** : L = { P } /* la liste des problèmes à traiter */
$x^* =$ inadmissible /* la solution optimale */
$z^* = \infty$ /* le coût de la solution $x^*$ pour un problème de minimisation */
**while** *true* **do**
  **if** $L = \emptyset$ **then**
    $x^*$ est la solution optimale pour $P$ ;
    Break ;
  **else**
    Choisir un sous problème $Q$ de $L$ ;
    $L = L \setminus Q$ ;
    résoudre $R_Q$ par Simplexe et obtenir la solution $\overline{x}$ de coût $\overline{z}$ ;
  **end**
  **if** $F(R_Q) = \emptyset$ **then**
    élagage car pas de solution;
  **else if** $\overline{z} \geq z^*$ **then**
    élagage car $Q$ ne contient pas une solution meilleure que la solution courante $z^*$;
  **else if** $\overline{x} \in \mathbb{N}$ *et* $\overline{z} < z^*$ **then**
    $x^* = \overline{x}$ ;
    solution courante= $x^*$ ;
    $z^* = \overline{z}$ ;
  **else**
    faire appel à l'algorithme des plans sécants pour la solution fractionnaire $\overline{x}$ ;
    choisir une variable $\overline{x_j}$ tq $\overline{x_j} \notin \mathbb{N}$ ;
    générer deux sous problèmes $(Q_1, Q_2)$ de $Q$ en ajoutant les contraintes $x_j \geq \lceil \overline{x_j} \rceil$ et $x_j \leq \lfloor \overline{x_j} \rfloor$, respectivement à chaque sous problème ;
    $L = L \cup \{Q_1, Q_2\}$ ;
  **end**
**end**

Afin de compléter l'explication de la méthode "Branch-and-Cut", nous présentons l'algorithme des plans sécants. Ce dernier est utilisé pour trouver une solution optimale entière à un problème d'optimisation linéaire. Dans le cadre de la méthode "Branch-and-Cut", cet algorithme consiste à trouver une solution optimale entière pour le problème relaxé. La solution trouvée ($\overline{x}$) est la solution optimale ($x^*$) pour le problème original $P$. L'explication au préalable de quelques définitions est nécessaire avant de présenter la méthode des plans sécants. Nous débutons tout d'abord par la notion de contrainte valide pour un problème donné. Une contrainte $C$ est valide pour $P$ si pour toute solution $x \in F(P)$, $C$ est vérifiée. Puis, pour le concept de "plan coupant" (voir la figure 2.15), il s'agit de munir le problème $P$ d'une contrainte valide $C$ de telle sorte que $F(R_{P \cup C}) \subset F(R_P)$, on dit donc que $C$ est un plan coupant pour $P$. Enfin, $C$ est une coupe valide pour $\overline{x} \in R_P$ si $C$ est un plan coupant pour $P$ et $\overline{x}$ ne vérifie pas $C$. L'illustration graphique de coupe valide est donnée par la figure 2.16.

L'algorithme présentant la méthode des plans sécants appliquée au problème P (l'algorithme 2) est décrit ci dessous :

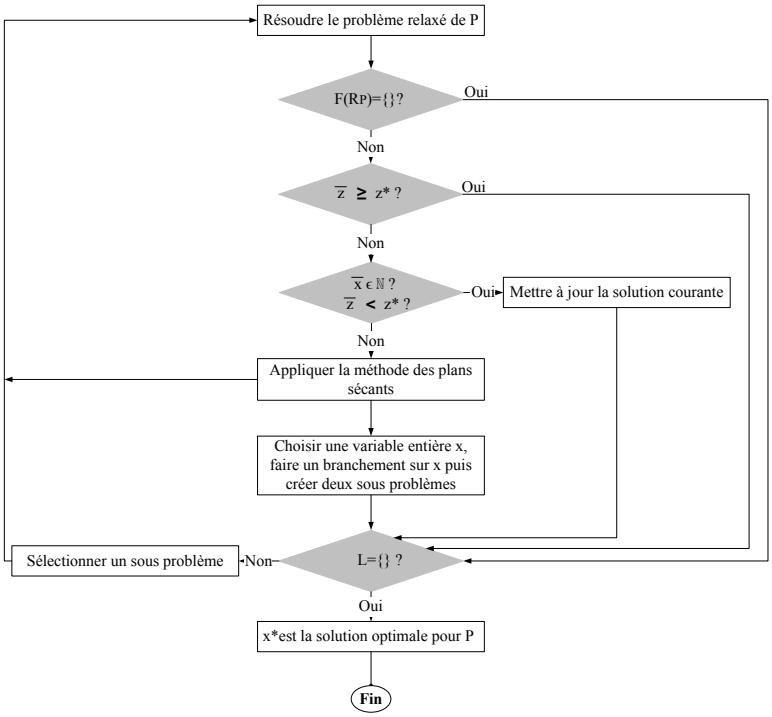



FIGURE 2.14 – Organigramme de la méthode "Branch-and-Cut"

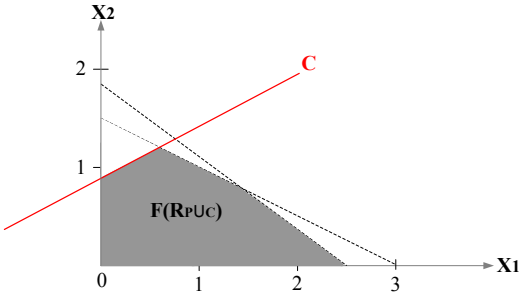



FIGURE 2.15 – Illustration graphique d'un plan coupant

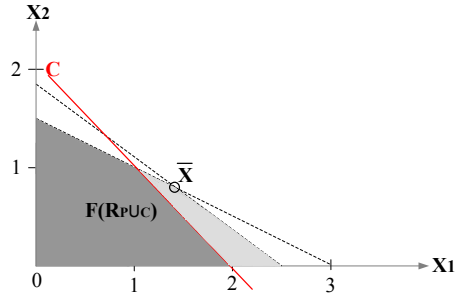



FIGURE 2.16 – Illustration graphique d'une coupe valide

**Algorithme 2 :** Algorithme des plans sécants

**while** $\overline{z}$ *ne change pas beaucoup* **do**
  Résoudre $R_P$ et obtenir la solution $\overline{x}$;
  **if** *toutes les variables de* $\overline{x}$ *sont des entiers* **then**
    Le problème est résolu et $\overline{x}$ est la solution entière trouvée ;
    Break;
  **else**
    Construire une coupe valide $C$ pour $\overline{x}$ et $P$;
    Ajouter la contrainte $C$ à la formulation $P$;
  **end**
**end**

- **Complexité théorique :** la complexité de résolution d'un problème de programmation linéaire mixte en nombres entiers dépend du nombre de variables, du nombre de contraintes ainsi que de la forme de la fonction objectif et des contraintes. Le type

des variables peut également complexifier la résolution. Dans [86], deux mesures différentes ont été définies pour évaluer la complexité des problèmes de la PLMNE. La première mesure qui représente la taille du problème, est indiquée via le nombre total de variables ($n$), le nombre de variables binaires ($d$), et le nombre total de contraintes ($m$). La deuxième mesure est le coût de calcul en termes de temps CPU nécessaire à la résolution du PLMNE. Dans L'algorithme "Branch-and-Bound", le problème étant décomposé en un problème entier (IP) et en une série de problèmes linéaires subordonnés (LP), la complexité a été définie pour LP et IP séparément. La relation entre la taille du problème et la complexité moyenne d'un PL est encore une question ouverte. Cependant, il a été constaté expérimentalement que généralement le nombre d'itérations dépend beaucoup plus de $m$ que de $n$, et qu'il est souvent entre $m$ et $3.m$. Pour les problèmes de grande taille, il a été rapporté qu'une estimation plus réaliste est de $O(m)$ pour le nombre d'itérations et de $O(m.n)$ pour le nombre d'opérations arithmétiques à chaque itération. Par ailleurs, Hocks [87] a donné une estimation de la complexité de l'algorithme du Simplexe par rapport à la taille du problème et il a précisé que sa complexité théorique est de $O(2^m)$ (voir le tableau 2.2).

| Taille du problème | Dimension | Complexité |
|---|---|---|
| Petite | $m < 100$ | $O(m)$ |
| Moyenne | $100 \leq m < 10000$ | $O(m^2)$ |
| Grande | $m \geq 10000$ | $O(m^4...m^5)$ |
| Pire cas | | $O(2^m)$ |

Tableau 2.2 – Complexité de l'algorithme du Simplexe selon Hocks

La complexité de l'approche classique permettant de résoudre un problème IP avec $d$ variables binaires, qui consiste à appliquer l'algorithme de "Branch-and-Bound" et un ensemble de relaxations linéaires associées aux nœuds de l'arbre de recherche, est théoriquement exponentielle [88]. Dans le meilleur des cas, la complexité de l'algorithme de "Branch-and-Bound" est linéaire par rapport au nombre de variables binaires $d$, et dans le pire des cas, une recherche exhaustive sur l'arbre doit être effectuée. Une relation exacte entre le nombre de variables binaires et la complexité moyenne des problèmes en pratique est encore inconnue, mais la complexité moyenne varie de $O(d)$, $O(d^2)$ jusqu'à $O(2^d)$.

Après avoir présenté le contexte de la thèse, nous discutons dans les deux parties suivantes de ce chapitre, des approches d'analyse et d'optimisation existantes pour les systèmes temps réel embarqués. Durant cette discussion, nous nous focalisons d'une part sur l'analyse temps réel (partie 2.3), et d'autre part, sur l'optimisation de la phase de déploiement (partie 2.4). Nous mentionnons également l'apport de notre travail pour la communauté des systèmes temps réel par rapport à chacun des aspects d'analyse et d'optimisation du déploiement.

## 2.3 Méthodes d'analyse pour les systèmes temps réel

Dans un premier temps, nous nous intéressons dans la partie 2.3.1 à l'analyse des temps de réponse des systèmes asynchrones (pilotés par évènements) et plus particulièrement les systèmes ordonnancés à base de priorités fixes. Ensuite, dans la partie 2.3.2, nous étendons notre présentation aux techniques d'analyse d'ordonnancement temps réel apportées aux applications modélisées par un graphe de flot de données. Enfin, nous indiquons dans la

partie 2.3.3 les lacunes des méthodes d'analyse existantes vis-à-vis du modèle des systèmes considéré par la présente thèse.

### 2.3.1 Analyses des temps de réponse pour les systèmes à priorités fixes

Comme déjà mentionné précédemment, l'analyse des temps de réponse sert à s'assurer que l'exécution des tâches d'un système temps réel se déroule dans les délais spécifiés. Celle-ci valide un ordonnancement si et seulement si, dans le pire scénario, le temps de réponse de chaque tâche du système est inférieur ou égal à son échéance. Dans cette partie, nous présentons certaines techniques d'analyse des temps de réponse apportées aux systèmes ordonnancés par un algorithme à base de priorités fixes. Nous commençons par étudier les analyses dédiées à un environnement d'exécution mono-processeur, puis nous présentons différentes analyses pour un environnement multiprocesseur distribué. Initialement, nous considérons que les tâches ne partagent pas de ressources mis à part le processeur. Ensuite, nous étendons notre présentation à la situation où un ensemble de ressources est partagé par les tâches du système.

1. **Environnement d'exécution mono-processeur :** les études apportées initialement pour l'analyse des temps de réponse des systèmes mono-processeur à priorités fixes [35] considèrent un système de tâches périodiques et indépendantes dans le sens où il n'existe pas de communication entre les tâches. Le principe de ces études est de déterminer les périodes d'activité représentant les intervalles de temps où le processeur est pleinement occupé à exécuter des tâches sans temps creux. La notion de période d'activité permet de déterminer le scénario "pire cas" pour les tâches du système. Dans le contexte d'un ordonnancement préemptif à priorités fixes, les travaux [35, 89, 90] ont prouvé que le temps de réponse au pire cas (WCRT) d'une tâche dans un système de tâches périodiques ou sporadiques, est obtenu dans le scénario où toutes les tâches du système sont activées d'une façon synchrone à l'instant critique[1] $t = 0$ et avec leur rythme maximal. [35] a montré que le WCRT d'une tâche périodique avec échéance sur requête correspond au temps de réponse de sa première instance activée dans la période d'activité. Plus tard, [90] a élargi ce résultat pour le cas de tâches périodiques avec échéances inférieures ou égales aux périodes. Une extension de ces derniers travaux a été menée dans [91], celle-ci consiste à prouver que dans le cas général (le cas de tâches avec échéances arbitraires), le WCRT d'une tâche de priorité $i$ se produit dans la plus longue période d'activité de niveau $i$. Une période d'activité de niveau $i$ est une durée pendant laquelle le processeur est pleinement occupé par l'exécution de tâches de priorité supérieure ou égale à $i$ [91]. La plus longue période d'activité de niveau $i$ est initiée par l'instant critique survenant lorsqu'une tâche de priorité $i$ est activée en même temps que toutes les tâches plus prioritaires ou de même priorité. Étant donné qu'une tâche périodique $x$ de priorité $i$ peut s'activer plusieurs fois dans une période d'activité de niveau $i$, le WCRT de cette tâche est le maximum des temps de réponse de toutes ses instances qui sont activées dans la période d'activité. Le temps de réponse de la $k^{\text{ième}}$ instance de $x$ $(R_{x,k})$ est calculé comme étant la différence entre l'instant de fin d'exécution et l'instant d'activation de cette instance. Son instant d'activation est égal à $(k-1) * T_x$, où $T_x$ est la période d'activation de la tâche $x$. La formule ci-après permet de calculer l'instant de fin d'exécution de la $k^{\text{ième}}$ instance $(W_{x,k})$ qui correspond à la longueur de la période d'activité de niveau $i$ considérant

1. L'instant critique est l'instant initiant une période d'activité [35]. Dans le cadre de tâches périodiques et indépendantes, l'instant critique de la plus longue période d'activité se produit lors d'une activation simultanée de toutes les tâches du système.

seulement les instances activées à ou avant $(k-1)T_x$ :

$$W_{x,k}^{q+1} = kC_x + \sum_{j \in hp(i)} \left\lceil \frac{W_{x,k}^{q}}{T_j} \right\rceil C_j \tag{2.1}$$

Où, $hp(i)$ représente l'ensemble des tâches avec priorité supérieure à $i$. La formule 2.1 est itérative jusqu'à l'arrivée à un point fixe où $W_{x,k}^{q+1} = W_{x,k}^{q} = W_{x,k}$. Sachant que la valeur initiale de $W_{x,k}$ est égale à $k * C_x$ ($W_{x,k}^{0} = k * C_x$). Par conséquent, le temps de réponse de la $k^{\text{ième}}$ instance de la tâche $x$ est donné par :

$$R_{x,k} = W_{x,k} - (k-1)T_x \tag{2.2}$$

Supposons que $Q$ est le nombre d'instances de la tâche $x$ activées dans la période d'activité de niveau $i$, le WCRT de la tâche $x$ est donné par :

$$R_x = \max_{k=1,2,..,Q} R_{x,k} \tag{2.3}$$

Le nombre d'instances d'une tâche $x$ activées dans la période d'activité ($Q$) est égal au numéro de l'instance $\alpha$ pour laquelle l'instant de fin d'exécution est inférieure ou égale à la période de la tâche $x$ ($Q = \alpha$ tel que $W_{x,\alpha} \leq T_x$). Dans le contexte de tâches à échéances sur requêtes ou à échéances contraintes ($R_x \leq D_x$) [90, 35], le calcul du WCRT revient à appliquer les formules (2.1), (2.2) et (2.3) uniquement à la première instance ($k = 1$). Ainsi, le WCRT de la tâche $x$ devient $R_x = W_{x,1}$.

Le scénario "pire cas" considéré par les travaux précédents pour les tâches indépendantes reste trop pessimiste pour le cas de tâches dépendantes. Ce pessimisme peut être soulevé grâce au fait que certaines tâches ne pourront jamais s'activer simultanément puisqu'il existe des relations de décalage entre leur activation. Ces relations , appelées relations d'Offset ou relations de précédence, sont capturées par un modèle appelé "modèle de transaction" qui consiste à regrouper un ensemble de tâches déclenchées par un même évènement (voir la figure 2.17). Le principe à retenir pour un modèle de transactions est qu'une tâche ne peut jamais s'activer avant l'écoulement de son offset depuis l'activation de la transaction. Étant donné qu'une tâche $i$ appartenant à la transaction $\Gamma_j$ ($\tau_{i,j}$) peut s'activer plusieurs fois dans une période d'activité, la $k^{\text{ième}}$ activation se produit à $O_{i,j} + (k-1)T_{\Gamma_i}$, où l'offset $O_{i,j}$ de la tâche $\tau_{i,j}$ est l'intervalle de temps entre l'instant d'activation de la transaction ($T_{\Gamma_i}$) qui est égale à l'instant d'arrivée de l'évènement externe (réveil sporadique évènementiel d'une transaction) et l'instant d'activation de la tâche. La figure 2.17 présente un exemple de transaction $\Gamma_i$ activée toutes les 16 unités de temps ($T_{\Gamma_i} = 16$) et constituée de trois tâches $\tau_{i,1}$ , $\tau_{i,2}$ et $\tau_{i,3}$ dont le pire temps d'exécution est $C_{i,1} = 3, C_{i,2} = 2$ et $C_{i,3} = 4$, respectivement. Les tâches $\tau_{i,1}$ , $\tau_{i,2}$ et $\tau_{i,3}$ sont activées respectivement après $2, 7$ et $12$ unités de temps ($O_{i,1} = 2, O_{i,2} = 7, O_{i,3} = 12$) de l'arrivée de l'évènement externe. $D_{i,j}$ indique l'échéance relative de $\tau_{i,j}$.

Dans la littérature, de nombreux travaux ont été apportés pour l'analyse des temps de réponses d'un modèle de transactions. Les travaux de Tindell [92, 93, 94] ont été les premiers travaux proposés, ils englobent la définition du modèle de transactions et l'identification du pire scénario d'exécution d'une tâche permettant de trouver son WCRT. Dans [93, 95], il a été prouvé qu'un instant critique initiant une pire (plus longue) période d'activité se produit lorsqu'une tâche plus prioritaire que la tâche analysée de chaque transaction s'active simultanément à la tâche analysée et arrive avec son rythme maximal après le début de la période d'activité. La procédure de calcul du WCRT consiste à identifier tous les facteurs participant au retard de l'exécution de la tâche analysée. Ces facteurs correspondent d'une part aux interférences causées par les tâches plus prioritaires de toutes les transactions, et

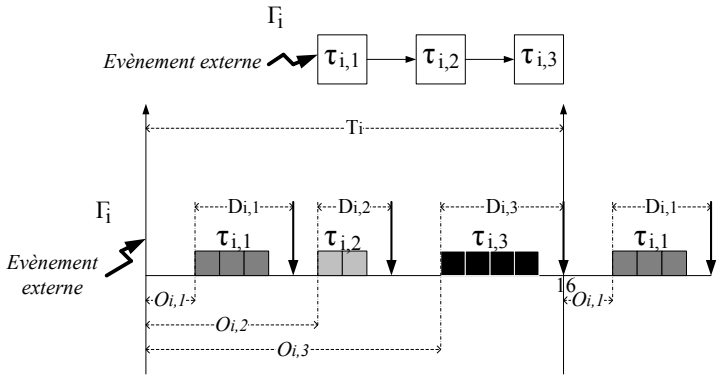



FIGURE 2.17 – Exemple de transaction

d'autre part, aux interférences causées par les instances de la tâche analysée qui sont activées avant l'instance pour laquelle l'analyse se fait. En effet, l'étude de toutes les périodes d'activité possibles sur laquelle se basent les travaux apportés dans [93, 95] permet une analyse exacte du pire temps de réponse. Mais malheureusement, la complexité est exponentielle ce qui la rend inapplicable pour les systèmes de taille réaliste. Par la suite Tindell [93] a fourni une méthode d'analyse avec une complexité pseudo-polynomiale permettant de déterminer une borne supérieure du WCRT. Cette approche a été améliorée plus tard par Turja et Nolin [96, 97, 98, 99], une amélioration qui consiste à réduire le pessimisme de la borne supérieure définie par Tindell. La valeur de la borne supérieure est garantie dans le sens où elle ne peut jamais être inférieure à la valeur exacte du WCRT d'où la garantie de l'ordonnançabilité du système par ces méthodes approchées.

Dans le contexte de conception des systèmes temps réel, Saksena [100] a développé une analyse des temps de réponse pour le modèle de transactions. Cependant, les transactions considérées sont composées de fonctions élémentaires appelées "actions" au lieu de tâches et ces actions sont supposées être encapsulées dans des objets actifs communiquant via des échanges de messages synchrones et asynchrones. L'analyse apportée considère deux types d'implémentation possibles : une implémentation mono-tâche, où toutes les actions du système sont exécutées par la même tâche, et une implémentation multitâche pour laquelle les actions représentant le système peuvent être regroupées dans des tâches différentes. La particularité de cette analyse est qu'elle calcule le WCRT des éléments composant une tâche et non pas le WCRT des tâches. De plus, elle prend en considération le fait que ces mêmes éléments sont organisés autrement via des objets. En revanche, le modèle considéré nécessite des efforts de traitement supplémentaires vis-à-vis de l'extraction des chaines de bout-en-bout et de la prise en compte des communications synchrones. S'ajoutant à cela, d'autres traitements sont introduits dûs à la non préemptivité entre les actions appartenant à la même tâche ou au même objet.

Après le survol des différentes méthodes d'analyse des temps de réponse existantes pour les systèmes mono-processeur, nous évoquons dans la suite celles qui sont destinées aux systèmes multiprocesseur distribués.

2. **Environnement d'exécution multiprocesseur distribué (réparti) :** une architecture multiprocesseur distribuée représente une bonne solution pour certains problèmes liés aux systèmes temps réel. Par exemple elle permet d'augmenter les chances de respecter les contraintes temporelles ou encore de réduire les défaillances du système

et ce, grâce à l'exécution parallèle sur différents équipements. Pour ce contexte, nous présentons les principales analyses des temps de réponse, notamment les plus pertinentes concernant ce travail de thèse.

Une première solution qui peut venir à l'esprit pour l'analyse des temps de réponse d'un modèle de transactions dans un environnement distribué est d'appliquer l'analyse apportée aux systèmes mono-processeur (comme celle proposée par Lehoczky [91] et qui est destinée aux systèmes de tâches avec échéances arbitraires) à chacune des ressources (processeur ou réseau de communication) du système distribué. Cependant, il est nécessaire de prendre en compte le fait que l'activation des tâches et celle des messages sont dépendantes. Basées sur ce principe, les premières analyses des temps de réponse dédiées aux systèmes distribués sont appelées "analyses holistiques". Cette appellation vient du fait que ces analyses considèrent à la fois les tâches et les messages échangés entre ces tâches. L'analyse holistique a été introduite par Tindell et Clark en 1994 [101]. Étant donné que les dates effectives d'activation des tâches peuvent être différées dans un environnement distribué à cause de l'attente de messages, Tindell et Clark propose de capturer les décalages sur l'activation des tâches par la notion de gigue (jitter) temporelle. L'analyse holistique permet donc de calculer le WCRT global des tâches et celui des messages en se basant sur les gigues. Le WCRT global est calculé comme étant la différence entre l'instant de la fin d'exécution de la tâche ou du message et l'instant d'activation de la transaction (dans la suite, nous nous référons au WCRT global simplement par WCRT). Classiquement le calcul du WCRT d'une tâche de priorité $i$, là aussi, repose sur la définition de la plus grande période d'activité de niveau $i$. D'après [102], la plus grande période d'activité se produit lorsque la tâche analysée est activée simultanément avec toutes les tâches plus prioritaires après avoir été retardées par une valeur maximale de gigue. Ainsi les tâches s'activent après l'instant critique à leur rythme maximal. Dans le cadre des tâches avec gigue, la longueur de la période d'activité ($W_{x,k}$) pour la tâche $x$ considérant seulement les instances activées à ou avant $(k-1)T_x$ se définit comme le point fixe de l'équation (2.4). Le point fixe est obtenu lorsque $W_{x,k}^{q+1} = W_{x,k}^{q} = W_{x,k}$.

$$W_{x,k}^{q+1} = kC_x + \sum_{j \in hp(i)} \left\lceil \frac{W_{x,k}^{q} + J_j}{T_j} \right\rceil C_j \tag{2.4}$$

La valeur initiale pour $W_{x,k}$ est égale à $kC_x (W_{x,k}^{0} = kC_x)$. L'équation itérative (2.4) est appliquée pour les valeurs de $k$ égales à $1, 2, ..,$ jusqu'à ce que la condition $W_{x,k} \leq kT_x$ soit vérifiée. Le WCRT d'une instance $k$ de la tâche $x$ relatif à l'instant d'activation de la transaction est obtenu par la formule suivante :

$$R_{x,k} = W_{x,k} - (k-1)T_x + J_x \tag{2.5}$$

Utilisant le résultat de la formule (2.5), le WCRT d'une tâche $(R_x)$ est obtenu comme étant le WCRT maximum de toutes les instances considérées $(R_x = \max_{k=1,2,..,} R_{x,k})$.

Du point de vue des messages, Tindell et Clark [102] ont proposé une analyse des temps de réponse pour des messages ordonnancés selon le protocole TDMA (Time-Division Multiple Access). Cette dernière a été étendue par Tindell dans [103, 104, 105] afin de considérer les messages au sein des bus CAN. Rappelons que l'ordonnancement des messages selon le protocole CAN se base sur les priorités des messages et il est de nature non préemptive. Pour ce dernier cas, le calcul du WCRT (voir l'équation 2.6) d'une instance $k$ d'un message $m$ $(R_{m,k})$ activée à la date $(k-1)T_m$ dépend de : i) la gigue du message $(J_m)$, qui correspond à la plus longue durée entre l'instant où le message est déclenché et le moment où il est prêt à être transmis sur le

bus, ii) la plus longue durée que l'instance du message peut rester dans la file d'attente avant qu'elle soit élue pour une transmission sur le bus $(W_{m,k})$ et iii) pire temps de transmission que le message peut avoir sur le bus $(C_m)$.

$$R_{m,k} = J_m + W_{m,k} - (k-1)T_m + C_m \tag{2.6}$$

Le WCRT d'un message est donc calculé par la formule :

$$R_m = \max_{k=1,..,} R_{m,k} \tag{2.7}$$

La durée d'attente d'une instance dans la file des messages prêts $(W_{m,k})$ est déterminée en considérant la plus longue période d'activité pour le message $m$. Celle-ci se produit lorsque le message $m$ de priorité $i$ est prêt à être transmis en même temps que tous les messages ayant une priorité supérieure et que ces messages sont réactivés avec leur rythme maximal [47]. La plus grande durée entre le début de la période d'activité et le moment où la transmission de l'instance $k$ commence est donnée par :

$$W_{m,k}^{q+1} = (k-1)C_m + B_m + \sum_{j \in hp(i)} \left\lceil \frac{W_{m,k}^{q} + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{2.8}$$

Où, $\tau_{bit}$ est le temps nécessaire à la transmission d'un bit de donnée sur le bus. $B_m$, désignant le temps de blocage, représente le retard causé par les messages moins prioritaires lorsqu'ils possèdent l'accès au bus tout juste avant que le message $m$ soit prêt à être transmis. Ce terme est dû à la non préemptivité des bus de communication. Le temps de blocage maximal est calculé selon la formule (2.9) où $lp(i)$ est l'ensemble des messages ayant une priorité inférieure à $i$ (priorité du message $m$).

$$B_m = \max_{j \in lp(i)} C_j \tag{2.9}$$

La formule récursive (2.8) commence par la valeur $(k-1)C_m + B_m$, c'est-à-dire $W_{m,k}^0 = (k-1)C_m + B_m$, et se termine lorsque $W_{m,k}^{q+1} = W_{m,k}^q$. Le nombre d'activations $k$ à considérer dans la période d'activité pour le calcul du WCRT du message $m$ est obtenu lorsque $W_{m,k} \leq kT_m$. En d'autres termes, $k$ va de $1$ jusqu'à la dernière instance dans la période d'activité. Celle-ci représente l'instance vérifiant la condition $W_{m,k} \leq T_m \cdot k$.

Lorsque le système est à échéances contraintes, Davis et al. [47] ont montré que l'analyse proposée par Tindell est erronée puisqu'elle peut fournir des WCRT optimistes pour les messages. Autrement dit, ils ont démontré que dans le cas où les échéances sont inférieures aux périodes, il ne suffit pas de prendre en considération uniquement la première instance du message $(k = 1)$ comme c'est le cas pour les tâches pour obtenir le scénario "pire cas". Le remède proposé par Davis [47] consiste soit à évaluer le temps de réponse de toutes les instances du message dans la période d'activité, comme cela se fait d'ailleurs pour le cas d'échéances arbitraires, soit à considérer le plus grand retard possible dû à la non préemption. Cette dernière solution est plus simple mais plus pessimiste. Elle est simple car elle ne nécessite pas le calcul de plusieurs temps de réponse pour un message dans le cas des échéances contraintes et pessimiste car en pratique cette situation se produit rarement [106]. Concrètement, celle-ci revient à remplacer le terme $B_m$ dans la formule (2.8) par $\max\{B_m, C_m\}$. Elle peut être encore plus simpliste si le terme $B_m$ est remplacé par le temps de transmission du plus long message transmis sur le même bus ou encore par le temps de transmission du plus long message CAN (l'équivalent de $8$ octets).

À ce niveau nous avons défini le calcul du WCRT des tâches et celui des messages, viens ensuite la définition du calcul des gigues. La gigue d'activation d'un message

est égale au WCRT de la tâche émettrice tandis que la gigue d'une tâche est égale au WCRT du message reçu. Cependant, la gigue de la première tâche activée en réponse à un événement externe est nulle. Comme nous pouvons le constater, il existe une dépendance entre le calcul des WCRT et le calcul des gigues. La solution à ce problème consiste à répéter itérativement le calcul des WCRT et celui des gigues à partir des valeurs nulles pour les gigues [102]. La dépendance monotone du WCRT par rapport à la gigue permet une convergence de l'analyse vers une solution stable.

En 1997, Palencia et al. [107] ont apporté une extension de l'analyse holistique afin de prendre en compte la vérification des échéances absolues (échéances locales). Celles-ci sont considérées par rapport à l'activation de la tâche ou du message et non pas par rapport à l'activation de la transaction comme c'est le cas des échéances relatives. Après en 1998, Palencia et Gonzalez [95] ont proposé une borne supérieure du WCRT qui est plus optimiste que la borne supérieure apportée par l'analyse holistique de Tindell. L'analyse proposée est une extension pour le modèle à offset dynamique. Plus tard, Palencia et Gonzalez [108] ont utilisé les relations de précédence entre les tâches d'une même transaction afin de fournir une estimation plus optimiste et significativement faible des WCRT. Plus récemment, une analyse holistique a été développée par Pop et al. [109]. Cette analyse considère des tâches communicantes par un bus FlexRay et leur activation est soit pilotée par le temps (time-triggered) soit par évènements (event-triggered). Pop et al. ont aussi proposé dans [10] une approche holistique qui est capable de gérer des applications distribuées sur une plateforme matérielle hétérogène. L'hétérogénéité concerne différentes politiques d'ordonnancement pour les processeurs et les bus de communication comme un mélange de bus CAN, FlexRay, TTP ou encore des processeurs avec ordonnancement : statique cyclique, préemptif à base de priorités fixes, à base de priorités dynamiques EDF, etc. Récemment, Traore et al. [110] ont proposé une analyse des temps de réponse exacte pour les transactions monotones.

En se basant sur l'analyse holistique proposée par Tindell [103, 104, 105] et sur l'analyse des bus CAN affinée par Davis [47], les auteurs de [111, 112, 113, 16, 114] ont déterminé le calcul des *latences de bout-en-bout* dans le pire des cas pour les transactions du système. Une latence de bout-en-bout au pire cas (appelée simplement latence dans la suite) associée à une transaction est définie comme la plus grande durée nécessaire au traitement de la donnée fournie par l'évènement externe. Ceci correspond à la propagation d'une donnée de la première tâche de la transaction jusqu'à l'autre extrémité de la transaction qui représente sa dernière tâche. Les auteurs ont considéré trois situations où : (i) les tâches d'une transaction sont activées périodiquement, (ii) les activations des tâches d'une transaction sont pilotées par l'arrivée des données, et (iii) les activations des tâches d'une transaction sont un mélange entre les modèles d'activation précédents. La latence $L_{\Gamma_i}$ de la transaction $\Gamma_i$ pour le cas d'activations périodiques est calculée en additionnant les WCRT ($R_j$) et les périodes ($T_j$) de tout objet $j$ appartenant à la transaction (voir la formule (2.10)) [111]. Un objet représente soit une tâche soit un message.

$$L_{\Gamma_i} = \sum_{j \in \Gamma_i} (R_j + T_j) \tag{2.10}$$

En raison d'horloges non synchronisées, dans le pire des cas, la donnée attendue par un objet $j$ est arrivée juste après l'activation de ce dernier. Cette donnée ne va donc être prise en compte par l'objet que pendant sa prochaine activation et le résultat est produit après son WCRT, d'où $R_j + T_j$ unités de temps depuis l'arrivée de l'évènement externe. Le même raisonnement s'applique à tous les objets de la transaction. Étant donné que les activations des tâches (messages) exécutées sur la même ressource sont en réalité

synchronisées puisqu'elles font référence à la même horloge, une autre valeur moins pessimiste a été définie pour $L_{\Gamma_i}$. Celle-ci est donnée par l'équation (2.11) où $G$ est l'ensemble des messages transmis sur le réseau. $L_{\Gamma_i}$ représente la somme des WCRT de toutes les tâches $(\tau_j)$ et les messages $(m_j)$ transmis sur le réseau de la transaction, ainsi que les périodes de tous les messages transmis sur le réseau et celle de leur tâche destinatrice $(\tau_l)$.

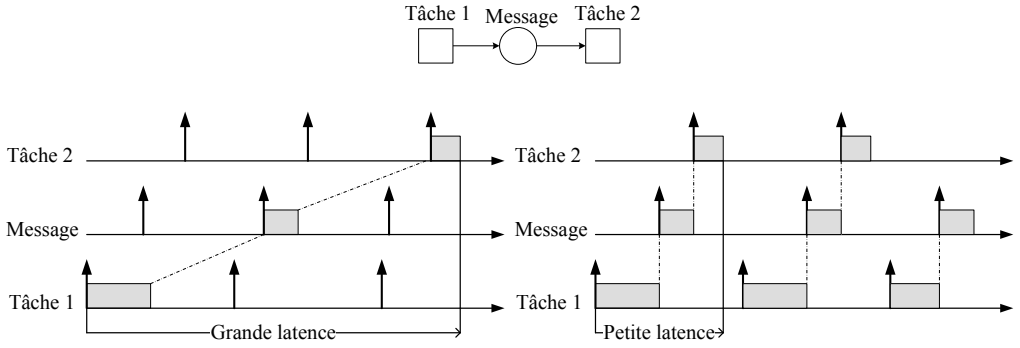
$$L_{\Gamma_i} = \sum_{\tau_j \in \Gamma_i} R_{\tau_j} + \sum_{m_j \in \Gamma_i : m_j \in G} (R_{m_j} + T_{m_j} + T_{\tau_l}) \tag{2.11}$$

De même, il est parfois possible de synchroniser la mise en file d'attente d'un message à transmettre avec l'exécution de la tâche émettrice. Cela permet donc de diminuer la valeur de la latence de l'équation (2.11) par une quantité égale à la période des messages transmis sur le réseau $(T_{m_j})$ [16].

Dans le cas d'activations pilotées par les données, $L_{\Gamma_i}$ est calculée comme étant le WCRT de la dernière tâche de la transaction. Celle-ci est exprimée par la formule suivante où $j$ est un objet (tâche ou message) [111] :

$$L_{\Gamma_i} = \sum_{j \in \Gamma_i} W_j \tag{2.12}$$

Considérant les deux modèles d'activations, la figure 2.18 illustre le calcul de la latence d'une transaction composée de deux tâches et d'un message. Dans le cas d'activations pilotées par les données, la valeur des gigues s'incrémente au fur et à mesure que le traitement se propage au long de la transaction. Cette situation est appelée propagation des gigues. La grande valeur des gigues des tâches et des messages prioritaires peut augmenter le WCRT des tâches et des messages de faible priorité qui sont exécutées/transmis sur la même ressource. Néanmoins, en raison des activations asynchrones entre les tâches et les messages dans le cas d'activations périodiques, la latence est souvent plus importante dans ce cas comparée au cas d'activations pilotées par les données. Pour plus de détails sur le compromis entre les deux modèles d'activations voir [111]. Enfin, pour un modèle d'activations mixtes, la




**(a) Modèle d'activation périodique** **(b) Modèle d'activation pilotée par données**

FIGURE 2.18 – Comparaison de la latence pour les systèmes temps réel asynchrones

latence peut être calculée par la formule suivante, où $j$ est un objet (tâche ou message) de la transaction $\Gamma_i$ [114] :

$$L_{\Gamma_i} = \sum_{j \in \Gamma_i} (J_j + W_j) + \sum_{l \in \Gamma_i \wedge \text{l est périodique}} (T_l) \tag{2.13}$$

Les analyses pour les systèmes distribués présentées jusqu'à présent considèrent

un modèle de transaction appelé *linéaire*. Dans un modèle de transaction linéaire, chaque tâche a au plus une seule tâche prédécesseur et une seule tâche successeur. Cependant, ce modèle ne permet pas d'exprimer de complexes interactions entre les tâches du système, d'où la proposition de [115] qui consiste à transformer un système représenté par un modèle de transaction non linéaire, dans lequel les tâches peuvent avoir plusieurs prédécesseurs et successeurs, en un système équivalent décrit par un modèle linéaire afin de pouvoir appliquer les analyses existantes.

Chaque technique d'analyse a son modèle de tâches adéquat et son propre modèle de communication. Ceci résulte en une collection large et hétérogène de méthodes d'analyse holistique ce qui rend difficile leur application en pratique [116]. Cette difficulté a été relevée avec l'apparition de plusieurs analyseurs d'ordonnançabilité. Parmi les outils d'analyse les plus connus nous citons "MAST : Modeling and Analysis Suite for Rea-Time Applications" [21] et "CHEDDAR" [22].

Une autre solution que les analyses holistiques a été proposée pour l'analyse des temps de réponse des systèmes distribués. Cette dernière est appelée "analyse compositionnelle" et elle s'adresse à des systèmes complexes et hétérogènes tels que les systèmes avec différentes politiques d'ordonnancement, des systèmes dont les tâches ont des dépendantes cycliques, ou encore des systèmes avec communication conditionnée [117, 118, 119, 120].

3. **Prise en compte de partage de ressources :** rares sont les applications temps réel dont les tâches ne partagent pas de ressources critiques matérielles (telles que la mémoire et les périphériques) ou logicielles (telles que les variables et les fichiers). En effet, deux problèmes peuvent apparaître lors de l'ordonnancement préemptif d'un système de tâches partageant des ressources à savoir le phénomène d'interblocage [121] et le phénomène d'inversion de priorité non borné [122]. La prise en compte des accès concurrents aux ressources partagées consiste d'une part à prévenir les situations d'interblocage, et d'autre part, à borner l'inversion de priorité afin d'intégrer cette borne dans les algorithmes d'ordonnancement. La borne de l'inversion de priorité est aussi appelée "le temps de blocage", et représente le retard sur l'exécution qui peut être causé pas les tâches moins prioritaires. Le rôle des protocoles d'accès aux ressources ou protocole de synchronisation est donc de trouver la plus petite valeur possible du pire temps de blocage. Il existe deux protocoles principaux : le protocole à priorités héritées (PIP pour Priority Inheritance Protocol) et le protocole à priorité plafond (PCP pour Priority Ceiling Protocol) [45]. Dans cette thèse, nous nous intéressons au protocole PCP car bien que le protocole PIP permette d'éloigner l'effet de l'inversion de priorité, il reste malheureusement impertinent pour le problème d'interblocage. De plus, avec PIP le temps de blocage est élevé dans le cas où plusieurs ressources critiques sont utilisées par des tâches.

***Protocole à priorité plafond (PCP pour Priority Ceiling Protocol) :*** ce protocole proposé par Sha et al. [45], représente une amélioration du protocole PIP permettant d'éviter les situations d'interblocage. L'idée consiste à attribuer à chaque ressource une priorité plafond qui est égale à la plus haute priorité parmi les priorités des tâches pouvant demander cette ressource. Par exemple, s'il existe trois tâches utilisant une ressource $x$, et la première tâche a la plus grande priorité alors la priorité plafond de la ressource $x$ est la priorité de cette première tâche. Ensuite, une tâche accède à une ressource sous la condition que sa priorité est strictement supérieure aux priorités plafonnées de toutes les ressources en cours d'utilisation. Si cette condition n'est pas satisfaite ou si la ressource demandée est occupée alors la tâche est bloquée et la tâche bloquante hérite de la priorité de la tâche bloquée la plus prioritaire. Ainsi, la plus

grande priorité que peut hériter une tâche pendant qu'elle détient une ressource est égale à la priorité plafond de cette ressource. Une tâche utilisant une ressource ne peut être préemptée que par les tâches plus prioritaires qui ne demandent pas d'accès à cette même ressource [4, 123, 124]. Dans le cadre du protocole PCP, une tâche a trois sources de blocage : i) le blocage direct qui survient lorsqu'une tâche prioritaire demande une ressource détenue par une tâche moins prioritaire, ii) le blocage d'héritage de priorité qui survient lorsqu'une tâche de priorité moyenne est bloquée par une tâche moins prioritaire qui a hérité d'une priorité plus forte et iii) le blocage de plafond de priorité qui se produit lorsqu'une tâche ne peut pas accéder à une ressource libre en raison d'une autre ressource utilisée et ayant une priorité plafond supérieure à la priorité de la tâche [124]. Ce dernier permet d'empêcher la situation d'interblocage. Soit $R$ une ressource critique et $C(R)$ sa priorité plafond, le temps de blocage maximale pour la tâche $x$ de priorité $i$ dans le cadre du protocole PCP est égale à :

$$B_x = \max_{y,R} S(R,y) \text{ t.q } i > l \text{ et } i \leq C(R) \tag{2.14}$$

Où, $S(R,y)$ est la durée de l'utilisation de $R$ par la tâche $y$ et $l$ est la priorité de la tâche $y$. De ce fait, la prise en compte de la dépendance des tâches via l'accès simultané aux ressources critiques lors de la validation d'une application temps réel consiste tout simplement à ajouter le terme de blocage $B_x$, donné par le protocole considéré, au calcul de la longueur de la période d'activité $W_x$. Les formules (2.1) et (2.4) deviennent ainsi :

$$W_{x,k}^{q+1} = kC_x + B_x + \sum_{j \in hp(i)} \left\lceil \frac{W_{x,k}^q}{T_j} \right\rceil C_j \;\; ; \;\; W_{x,k}^{q+1} = kC_x + B_x + \sum_{j \in hp(i)} \left\lceil \frac{W_{x,k}^q + J_j}{T_j} \right\rceil C_j \tag{2.15}$$

Dans cette partie, nous avons présenté les principales solutions apportées pour l'analyse des temps de réponse d'une application représentée par un modèle de tâches. Dans la partie qui suit, nous présentons les techniques d'analyse amenées aux applications temps réel modélisées par un graphe de flot de données.

### 2.3.2 Analyses d'ordonnancement temps réel pour les graphes de flot de données

Dans le contexte des systèmes temps réel embarqués, d'autres approches différentes de l'analyse des temps de réponse ont été proposées pour la validation du comportement temporel de ces systèmes. Ce sont des approches qui s'adressent à des applications représentées par un graphe de flot de données (un graphe fonctionnel) où le niveau opérationnel désigné par les tâches et les messages n'est pas présent. Contrairement aux analyses des temps de réponse qui se basent sur les paramètres temporels du modèle de tâches pour déterminer les WCRT, ces approches suivent une démarche différente qui consiste à trouver les paramètres temporels pour les éléments du graphe fonctionnel (tels que les périodes, les instants de début et de fin d'exécution, la taille des buffers, etc.) permettant à ce dernier de tenir dans le modèle de tâche ciblé. Enfin, les analyses des temps de réponse existantes peuvent être appliquées au modèle de tâches résultant.

Plusieurs travaux sur l'ordonnancement temps réel des graphes de flot de données ont été publiés. Parks et Lee [125] ont étudié l'applicabilité d'un ordonnancement RM (Rate Monotonic) non préemptif à des applications temps réel représentées par un graphe de flot de données synchrone (SDF). Ils ont déterminé un modèle d'exécution temps réel multitâche pour l'application considérée ainsi qu'une condition d'ordonnançabilité suffisante. Néanmoins, les éléments du graphe fonctionnel sont structurés en tâches périodiques indépendantes.

Dans [126], Goddard a utilisé l'algorithme d'ordonnancement EDF pour analyser des applications de flot de données modélisées par un graphe PGM (Processing Graph

Method) et implémentées par un modèle de tâches de type RBE (Rate-Based Execution). La particularité de ce modèle de tâches est que chaque tâche est caractérisée par une période de la forme de $x$ activations dans un intervalle de $y$ unités de temps. Initialement l'auteur propose d'identifier les taux auxquels les nœuds du graphe PGM doivent exécuter les signaux de données. Ensuite, chaque nœud est associé à une tâche différente dans un modèle de tâches RBE et les paramètres des tâches sont ainsi déterminés. Enfin, une condition suffisante mais pas nécessaire a été apportée pour déterminer l'ordonnançabilité de l'application selon l'algorithme EDF. Cette approche prend en considération les contraintes liées à la mémoire et à la latence de bout-en-bout. L'analyse d'ordonnançabilité est effectuée suivant la demande du processeur et la plateforme d'exécution est mono-processeur.

Bekooij et al. [1] ont déterminé les temps de réponse au pire cas de chaque fonction d'un graphe SDF exécuté dans un environnement multiprocesseur et ordonnancé selon l'algorithme TDM (Time Division Multiplexing). Leur travail cible les systèmes temps réel souples pour lesquels l'analyse du graphe de flot de données se base sur la simulation. Les auteurs ont montré que la considération d'une politique d'ordonnancement prévisible, telle que TDMA, permet de réduire les efforts de la simulation et d'augmenter la confiance des résultats de la simulation. Toutefois, les tâches implémentant le graphe de flot de données sont indépendantes et la plateforme d'exécution n'est pas distribuée.

Les auteurs dans [127] ont abordé l'ordonnancement temps réel des applications de flot de données modélisées sous forme de graphes SRDF (Single-Rate Data flow). Ils ont proposé de combiner l'ordonnancement statique avec l'ordonnancement TDM. Leur stratégie consiste à ordonnancer les tâches au sein de chaque processeur selon l'algorithme TDM, ensuite à exécuter les fonctions d'une tâche statiquement suivant un ordre déterminé. Dans ce travail, les auteurs ont aussi apporté des techniques permettant de définir les paramètres de l'ordonnancement TDM (tels que la période du cycle TDM pour chaque processeur et l'intervalle de temps dédié à chaque tâche appartenant au processeur) et de fixer l'ordre d'exécution entre les fonctions d'une tâche de telle sorte que les contraintes temporelles soient respectées. Leur expérimentation a montré comment la combinaison de l'ordonnancement TDM et de l'ordonnancement statique résulte en une stratégie d'ordonnancement améliorant les latences de bout-en-bout trouvées dans le cas de l'ordonnancement TDM apporté dans [1]. Néanmoins, les auteurs supposent que toutes les fonctions appartenant à la même tâche possèdent le même taux d'activation, autrement dit, le partitionnement des fonctions sur les tâches est à base de périodes. Par ailleurs, le calcul des latences reste pessimiste puisqu'il s'agit de trouver les instants de début et de fin d'exécution pour chaque tâche sans autoriser sa préemption par une autre tâche. Le système considéré est donc non préeemptif.

Plus tard, Bamakhrama et al. [128] ont étendu l'approche de Goddard aux graphes CSDF (pour Cyclo-Static Data Flow) qui représentent des graphes plus expressifs que les graphes PGM. Les auteurs se sont basés sur un modèle de tâches périodiques avec échéances implicites pour implémenter une application CSDF. Initialement, un vecteur de répétition est utilisé pour exprimer les périodes en termes de la période d'une fonction donnée, puis pour chaque fonction la période minimale est calculée. Ensuite, les instants de début d'exécution des fonctions sont déterminés en tenant compte des jetons initiaux sur chaque canal. En se basant sur les paramètres temporels trouvés, la taille minimale est estimée pour chacun des buffers de l'application. La dernière étape de l'approche présentée consiste à utiliser la condition sur la capacité d'utilisation définie pour n'importe quel algorithme d'ordonnancement multiprocesseur afin de calculer le nombre minimal des processeurs nécessaires à l'exécution de l'application. Cependant, les auteurs considèrent que chaque fonction est implémentée par une tâche périodique différente. Par conséquent, le partitionnement n'est pas abordé.

Bouazak et al. [129] se sont basés sur [128] pour déterminer l'ordonnancement des

graphes de flot de données par un ensemble de tâches périodiques, préemptives et exécutées en se basant sur leur priorité. Tout d'abord, ils ont proposé de calculer un ordonnancement abstrait "affine" pour le graphe de flot de données. Pour cela, ils ont défini une relation affine entre chaque paire de fonctions connectées dans le graphe. Ensuite, l'ordonnancement trouvé est concrétisé en assignant chaque fonction à une tâche périodique et en déterminant les périodes et les phases d'activation de chaque tâche de telle sorte que l'application reste ordonnançable sur une plateforme mono-processeur munie d'un ordonnanceur basé sur l'algorithme EDF ou l'algorithme RM. Cependant, l'ordonnançabilité du système a été vérifiée en s'appuyant sur la condition liée au facteur d'utilisation du processeur qui est suffisante pour les systèmes mono-processeur. Les auteurs n'ont donc pas apporté une nouvelle analyse des temps de de réponse pour les graphes de flot de données puisque chaque fonction du graphe est considérée comme étant une tâche différente.

### 2.3.3 Positionnement

Dans la partie précédente, nous avons donné un aperçu d'une gamme d'analyses des temps de réponses utilisées pour la vérification et la validation des applications temps réel considérant un ordonnancement à priorités fixes. Nous avons également discuté les principales études apportées à l'ordonnancement temps réel des graphes de flot de données. Malheureusement, aucune des approches existantes n'a permis de répondre à notre besoin. Par rapport à l'analyse des temps de réponse, la communauté temps réel a souvent considéré un modèle d'analyse dont les éléments constructifs sont des tâches, masquant ainsi la structure du niveau fonctionnel. Or, la phase de conception d'une manière générale ou la phase de déploiement plus particulièrement, est initiée par le modèle fonctionnel et est fortement basée sur la façon dont les éléments fonctionnels sont organisés sur les entités du modèle de tâches. Ces éléments fonctionnels sont ignorés par la majorité des analyses des temps de réponse apportées aux systèmes à priorités fixes, à l'exception de l'analyse de Saksena et al. [100]. Cependant, cette dernière est assez complexe pour être intégrée à un processus de conception puisqu'elle considère des fonctions non seulement avec des priorités d'exécution, mais aussi avec des seuils de préemption utilisés pour réduire le temps de blocage dû à la sémantique "*run-to-completion*" considérée pour les tâches et pour les objets. S'ajoutant à cela, cette analyse s'adresse à des systèmes mono-processeur et suppose que le modèle fonctionnel est spécifié à l'aide du paradigme orienté objet, alors que les applications d'aujourd'hui sont souvent distribuées et s'expriment par un modèle de flot de données.

Du point de vue des techniques d'ordonnancement temps réel apportées au modèle fonctionnel désigné par un graphe de flot de données, le tableau 2.3 montre les limites de chacune et met en évidence l'intérêt de l'analyse proposée dans cette thèse. Nous constatons à partir du tableau que les approches existantes font des hypothèses concernant le partitionnement des fonctions sur les tâches. Certaines approches (celles proposées dans [126, 128, 129]) supposent que chaque fonction du graphe est partitionnée sur une tâche différente à celle du reste des fonctions. Ceci revient donc tout simplement à aborder un modèle de tâches. D'autres approches (celles apportées dans [125, 1, 127]) exigent qu'une tâche soit un ensemble de fonctions ayant la même période d'activation. Dans notre cas, nous ne faisons aucune de ces hypothèses de partitionnement, ce qui conduit à des solutions plus pertinentes comme cela est monté dans le chapitre 4. Par rapport au modèle de tâches ciblé, les approches existantes optent toujours pour un modèle de tâches périodiques qui est parfois restreint à des tâches indépendantes et/ou à des tâches non préemptives ([125, 126, 1]). Toutefois, notre analyse adresse à la fois le modèle de tâches avec activations périodiques et celui ayant des activations pilotées par les données. Les tâches dans les deux cas sont préemptives et peuvent être dépendantes. Un autre intérêt de notre analyse consiste à considérer une plateforme d'exécution multiprocesseur et distribuée. Comme

nous pouvons le remarquer sur le tableau 2.3, les solutions présentées dans [125, 126, 129] sont destinées à des plateformes mono-processeurs. Par ailleurs, les auteurs dans [1, 128, 127] supposent que la plateforme d'exécution est multiprocesseur mais pas distribuée, autrement dit, la communication entre les processeurs est faite via une mémoire partagée et donc les messages ne sont pas pris en compte et l'analyse ne nécessite pas un effort concernant la synchronisation des horloges. Néanmoins, la proposition dans [127] consiste à calculer une table d'ordonnancement statique définie au moment de la compilation, en d'autres termes, le système analysé est supposé être synchrone. Cette démarche a également été suivie par [1]. Par ailleurs, les auteurs dans [126, 128, 129] se sont contentés de vérifier l'ordonnançabilité du système à l'aide des conditions de faisabilités suffisantes déterminées pour un ordonnancement selon EDF ou selon RM.

Toutes les raisons citées ci-dessus se réunissent pour déterminer le besoin d'une analyse de temps de réponse, considérant le modèle fonctionnel ainsi que son association au modèle de tâches et permettant d'aborder la problématique de cette thèse, à savoir le déploiement des systèmes temps réel distribués. Dans la suite, nous discutons les travaux liés à cette problématique.

Tableau 2.3 – Positionnement par rapport aux approches d'analyse des graphes de flots de données existantes

| **Critères de comparaison** / **Approches** | **Restrictions de partitionnement** | | | **Modèle de tâches** | | | | **Plateforme d'exécution** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Pas de restriction | Une fonction est une tâche | Une période est une tâche | Activations pilotées par les données | Activations périodiques | | | Mono-processeur | Multi processeurs | Multip rocesseur distribuée |
| | | | | | Tâches indépendantes | Tâches non préemptives | Pas de restriction | | | |
| Parks, 95 [125] | | | X | | X | X | | X | | |
| Goddard(*), 98 [126] | | X | | | X | | | X | | |
| Bekooij, 05 [1] | | | X | | X | | | | X | |
| Moreira, 07 [127] | | | X | | | | X | | X | |
| Bamakhrama , 11 [128] | | X | | | | | X | | X | |
| Bouazak, 12 [129] | | X | | | | | X | X | | |
| **Notre proposition** | X | | | X | | | X | | | X |

(*) Le modèle d'activation des tâches est un cas particulier du modèle périodique (voir la partie 2.3.2).

## 2.4 Exploration d'architectures pour les systèmes temps réel embarqués

Dans la littérature, plusieurs approches et solutions ont été apportées pour l'exploration de l'espace de conception lors du développement logiciel des systèmes temps réel embarqués. Dernièrement, [130] a présenté une étude exhaustive sur les méthodes existantes pour l'optimisation des systèmes temps réel embarqués. Il a classifié 188 travaux selon différents axes, à savoir les objectifs et les contraintes de conception, les degrés de liberté résolus, le type de techniques utilisées, etc. Il existe aussi des travaux adressant d'autres degrés de liberté liés à la conception que ceux considérés dans cette thèse. Par exemple, dans [113] et [114], les auteurs ont abordé respectivement l'optimisation de l'affectation des périodes et celle des modèles d'activations aux tâches et aux messages de l'application.

Dans la suite, nous présentons principalement des travaux autour de l'exploration architecturale des systèmes temps réel. Nous nous focalisons sur les approches de déploiement représentant le contexte de notre travail. Parmi les solutions de déploiement existantes, nous trouvons des solutions adressant le sous problème de placement et d'ordonnancement (partie 2.4.1) et des solutions traitant le sous problème de partitionnement et d'ordonnancement (partie 2.4.2). Dans ce qui suit, nous présentons les approches de ces deux catégories, puis nous discutons dans la partie 2.4.3 les travaux considérant un autre degré de liberté qui consiste dans l'affectation de mécanismes de protection aux données partagées.

### 2.4.1 Approches de placement et d'ordonnancement

Dans cette partie, nous présentons les travaux qui traitent le placement et l'ordonnancement d'un modèle de tâches sur une architecture matérielle distribuée. Le problème de la répartition des tâches sur les processeurs et de leur ordonnancement (un problème NP-difficile [37]) a été largement discuté dans la littérature. Tindell et al. [131] est parmi les premiers travaux qui ont fait face au problème de placement d'un modèle de tâches. Pour cela, il a utilisé la technique de recuit simulé et a considéré l'optimisation de la communication sur le réseau, l'utilisation de mémoire et le temps de réponse des tâches tout en tenant compte des échéances locales des tâches. L'affectation de priorités a été réalisée par l'algorithme DM (Deadline Monotonic). Cependant, ce travail ne s'adresse pas au modèle de tâches transactionnel et il ne considère qu'un type de communication selon le protocole d'anneau à jeton.

Bastaricca et al. [132] ont développé deux approches permettant l'optimisation du placement d'un modèle à composants sur une plateforme distribuée. Cette dernière n'impose pas des contraintes liées à la topologie du réseau de communication dans le sens où deux processeurs qui ne sont pas directement reliés peuvent communiquer via d'autres processeurs intermédiaires. La première approche est un modèle de programmation en nombres entiers et la seconde consiste en un algorithme génétique. L'optimisation dans ces deux approches concerne la minimisation du nombre de messages envoyés sur le réseau. Celui-ci comprend la fréquence de chaque message transmis sur le réseau ainsi que le nombre de sauts permettant à ce message de traverser tous les processeurs intermédiaires pour arriver à sa destination finale. Les auteurs se sont basés sur une application de type client-serveur pour étudier le compromis entre l'optimalité et le passage à l'échelle de chaque approche. En revanche, seulement les contraintes liées au placement (comme forcer le placement de deux composants particuliers sur le même processeur) ont été considérées, les contraintes temporelles ne l'ont pas été.

Dans [133], les auteurs ont proposé une approche permettant de guider les

concepteurs de systèmes temps réel critiques distribués durant les phases de placement et d'ordonnancement. Celle-ci consiste en une méthode de séparation et d'évaluation qui tient compte de plusieurs règles de branchement et dans laquelle l'analyse holistique est utilisée pour vérifier l'ordonnançabilité des tâches. L'architecture matérielle considérée est similaire à la nôtre, elle est composée de processeurs munis d'un système d'exploitation temps réel avec un ordonnanceur à priorités fixes et des bus de communication de type CAN. Cependant, le modèle de tâches est linéaire, contrairement à notre modèle, ce qui limite l'applicabilité de l'approche à un grand nombre de systèmes temps réel modernes. Les résultats expérimentaux ont montré le passage à l'échelle de l'approche proposée. Toutefois, les auteurs supposent que les tâches sont initialement affectées à un sous ensemble de processeurs c'est-à-dire chaque tâche dispose uniquement d'un sous ensemble de processeurs candidats ce qui résulte en une restriction de l'espace de recherche et donc un meilleur passage à l'échelle de l'approche. Enfin, cette proposition n'aborde pas l'aspect optimisation et se contente de trouver une solution ordonnançable.

L'équipe AOSTE de l'INRIA a proposé l'outil Syndex pour soutenir les concepteurs de systèmes temps réel embarqués distribués. L'outil se base sur la méthodologie AAA (pour Algorithme Architecture Adéquation) [134] qui couvre l'ensemble du cycle de développement depuis la spécification des fonctions de l'application jusqu'à l'implémentation. Syndex vise à optimiser l'implémentation par rapport à la latence et la cadence tout en tenant compte des contraintes temporelles et des contraintes de placement. Initialement, l'application est représentée par un graphe "*Algorithme*" qui est obtenu comme le résultat de transformation de la spécification de l'application. Cette transformation concerne le partitionnement de chaque fonction en une tâche différente. La plateforme est exprimée par un graphe appelé "*Architecture*". Par la suite, le problème de placement et d'ordonnancement des tâches et des messages est référencé par l'étape "*Adéquation*". Cette étape est réalisée pas une heuristique gloutonne qui consiste à trouver dans la première itération le meilleur placement pour la première tâche de chaque transaction par rapport à son temps de réponse. Ensuite, dans les itérations suivantes, le meilleur placement est recherché pour les tâches qui succèdent et ainsi de suite jusqu'à ce que toutes les tâches de l'application soient placées. Lorsqu'une tâche est considérée par l'heuristique à un moment donné, toutes ses tâches prédécesseurs ont déjà été placées et ordonnancées et aucune des tâches successeurs n'a été traitée à ce moment-là. Par ailleurs, pendant qu'une tâche est placée sur un processeur, elle n'est ordonnancée qu'après toutes les autres tâches qui ont déjà été placées sur ce même processeur. De cela, on déduit que la proposition de ce travail tend à définir un ordonnancement statique et non pas une affectation de priorités et par conséquent le modèle de tâches considéré est synchrone.

L'approche de Pop et al. [10] se focalise sur l'optimisation des systèmes temps réel embarqués distribués sur une plateforme multi-clusters (un cluster est un groupe de processeurs connectés) reliés entre eux par des passerelles. Les auteurs ont présenté une approche heuristique pour attribuer une application, modélisée comme un ensemble de graphes polaires orientés acycliques composés de tâches et messages, sur une architecture constituée de processeurs hétérogènes connectés par des bus de communication TDMA. L'hétérogénéité réside dans le fait qu'un processeur est muni de trois ordonnanceurs structurés hiérarchiquement : à base d'un cycle périodique statique, à base de priorités fixes et à base de priorités dynamiques. De plus, le cycle de communication TDMA est un mélange entre une partie dynamique et une autre statique. Chaque graphe de l'application possède une période d'activation et une échéance de bout-en-bout. L'objectif de l'optimisation est de trouver : un placement des tâches sur les processeurs, une affectation d'une politique d'ordonnancement à chaque tâche, une attribution de priorités aux tâches ordonnancées selon la politique à priorités fixes et une manière d'accès au bus de communication, de telle sorte que les temps de réponse des tâches sont minimisés. Initialement une solution aléatoire est donnée pour chaque sous problème, puis l'approche proposée s'appuie sur deux

phases basée chacune sur un algorithme glouton. La première phase traite les trois premiers degrés de liberté et la seconde aborde l'optimisation de l'accès au bus. Dans la partie expérimentations, les auteurs ont montré que leur stratégie d'optimisation était capable de trouver des configurations valides pour presque $80\%$ des tests générés aléatoirement. Cependant, cette heuristique est restreinte dans le sens où si un choix, fait à un instant donné de l'approche, mène à une solution non ordonnançable alors le processus d'exploration s'arrête. Par conséquent, l'approche ne donne pas la possibilité d'explorer d'autres solutions ou d'améliorer une solution valide trouvée lors de chaque phase. Ceci est dû au fait que la validation est indépendante du processus d'exploration. On retient néanmoins de cette approche l'idée d'une exploration en deux étapes.

Une approche basée sur la technique de recuit simulé a été présentée dans [135]. Celle-ci adresse à la fois le problème de placement des tâches et des messages sur une architecture matérielle distribuée et le problème d'affectation de priorités aux tâches et aux messages représentant le système temps réel. Les tâches et les messages constituant l'application s'activent suivant le modèle d'activations pilotées par les données et s'exécutent selon un ordonnancement à priorités fixes préemptif pour les tâches. Cette approche considère les contraintes temporelles ainsi que les contraintes liées à la topologie du réseau de communication. D'autre part, l'approche utilise une fonction d'équilibrage de la charge des processeurs pour permettre au processus d'exploration d'éviter les solutions non ordonnançables. Cependant, ce travail se focalise sur l'optimisation de la flexibilité du système. Celle-ci se mesure par la quantité du temps par laquelle le WCET des tâches peut être augmenté sans perturber l'ordonnançabilité du système. Dans le cas où le système n'est pas ordonnançable, cette métrique permet de quantifier de combien le système n'est pas ordonnançable. Les auteurs se sont basés sur l'intégration de scénarios permettant de forcer les solutions à être flexible vis-à-vis des changements suggérés par ces scénarios. Par ailleurs, le modèle de tâches considéré est formé de transactions linéaires.

[14] présente une formulation PLMNE pour le problème de placement des tâches et des messages ainsi que leur affectation de priorités. Les tâches et les signaux représentant l'application sont activés périodiquement et organisés sur des chemins dont l'échange de données se fait via des variables partagées. Dans le modèle considéré, une tâche peut initier plusieurs chemins, ce qui signifie qu'elle peut envoyer le même signal ou des signaux différents à plusieurs tâches. Les auteurs se sont intéressés à l'optimisation de la somme des latences tout en considérant les contraintes temporelles et celles d'utilisation des ressources. L'architecture matérielle est composée d'un ensemble de processeurs munis d'ordonnanceurs à priorités fixes et tous connectés à un seul bus CAN. Ce travail s'est aussi intéressé au partitionnement des signaux sur les messages. Pour cela, ce travail introduit la notion d'ensemble de messages agissant comme des conteneurs possibles pour les signaux transmis entre chaque couple de processeurs. Le nombre de conteneurs étant inconnu au préalable, ceci nécessite une détermination d'une borne supérieure pour ce nombre avant le début de résolution. Les auteurs définissent un ensemble de messages entre chaque couple de processeurs et pour chaque période possible puisqu'un signal peut être partitionné uniquement sur un message ayant la même période. De plus, la taille de chaque ensemble doit être égale au nombre de signaux de l'application ayant la période sous-jacente. Par conséquent, le problème avec cette stratégie de partitionnement est que le nombre de messages conteneurs est souvent pessimiste puisqu'il englobe même les signaux qui peuvent être transmis localement à un processeur. Ceci augmente le nombre de variables de la formulation PLMNE et donc sa complexité. Par ailleurs, nous croyons que le passage à l'échelle de l'approche devrait être pire lorsque l'on considère plusieurs bus puisqu'il s'agit de définir ce même nombre pessimiste de messages pour chacun des bus. D'autre part, les auteurs ont proposé une approche de déploiement en deux étapes afin de réduire la complexité de la formulation PLMNE intégrale. Cette approche consiste à diviser l'ensemble du problème en deux sous problèmes tels que le premier concerne la

répartition des tâches sur les processeurs et leur affectation de priorités et la seconde est dédiée au partitionnement des signaux et à l'attribution des priorités aux tâches et aux messages. Les auteurs ont montré l'applicabilité et les avantages de leur approche en deux étapes à l'aide d'un sous système de l'automobile composé de neuf processeurs, quarante et une tâches et quatre-vingt-trois signaux. Néanmoins, l'évaluation de la formulation PLMNE intégrale n'a pas été fournie. S'ajoutant à cela, aucune mesure vis-à-vis du partage de données n'a été prise en compte dans ce travail. Plus tard dans [16], l'approche en deux étapes a été améliorée et étendue afin d'optimiser l'extensibilité des tâches et de considérer une architecture matérielle à plusieurs bus de communication. Dans cette version, l'approche en deux étapes consiste à décider, en premier lieu, le placement des tâches à l'aide d'une formulation PLMNE, puis le partitionnement des signaux sur les messages et le placement des messages sur les bus en se basant sur une heuristique dédiée. Enfin, une méthode itérative a été utilisée pour attribuer les priorités aux tâches et aux messages. Cette approche offre la possibilité d'améliorer la solution obtenue en réaffectant les tâches aux processeurs et en répétant ainsi le processus. Cependant, l'amélioration n'est pas garantie puisque la réaffectation des tâches aux processeurs dans l'itération suivante se fait d'une manière arbitraire en se basant sur l'expertise du concepteur. Une comparaison avec une approche à base de recuit simulé a été fournie. Une fois de plus, l'approche a été étendue dans [17] pour adresser une plateforme avec des processeurs représentant des passerelles entre d'autres processeurs. Les études présentées dans [14, 16, 17] ont été intégrées et validées au sein de l'outil de conception Metropolis/Metro II [181, 182, 183].

Dans le cadre d'une méthodologie de développement centrée modèles, [15] a proposé une approche permettant d'automatiser le processus de déploiement incluant les phases de placement et d'ordonnancement. À chaque phase, le problème est modélisé par un programme linéaire en nombres entiers (PLNE). Initialement, l'approche suppose que le partitionnement des fonctions sur les tâches a été déterminé grâce aux méthodes de la théorie des graphes. Ensuite, le placement des tâches sur les processeurs est réalisé de telle sorte que toutes les contraintes liées aux capacités des ressources et à l'architecture de la plateforme soient respectées. Par ailleurs, le placement est optimisé vis-à-vis de la métrique du coût (nombre de processeurs) et les différentes métriques concernant l'utilisation des ressources (communication, mémoire, utilisation, etc.). Enfin, dans la seconde phase, un ordonnancement statique est déterminé de telle sorte que les temps de fin d'exécution pour les tâches soient minimaux. Toutefois, cette approche s'adresse à des systèmes synchrones et considère chaque sous problème indépendamment des autres. De plus, chacun des sous problèmes est optimisé par rapport à différentes métriques.

Un autre type de démarches a été suivi dans [136] pour le placement d'un ensemble de tâches périodiques indépendantes sur une plateforme multiprocesseur homogène. Celle-ci consiste à modéliser le problème de placement en tant qu'un problème de Bin Packing puis à appliquer un ensemble de quatre heuristiques standards apportées à ce dernier. Étant donné que ces heuristiques suivent le principe d'une affectation séquentielle, les auteurs ont choisi de trier les tâches par ordre croissant puis par ordre décroissant selon quatre attributs, à savoir la période, l'utilisation, l'échéance et la densité. Par conséquent, huit méthodes de tris ont été considérées. Par ailleurs, les deux types d'ordonnancement, à priorités statiques et à priorités dynamiques, ont été pris en compte dans ce travail. L'objectif de cette proposition est d'évaluer chaque combinaison "une heuristique, une méthode de tri, un ordonnancement" par rapport au nombre de processeurs utilisés, au nombre de tâches ordonnançables et à l'espace libre en terme de charge pour chaque processeur, afin de déterminer le meilleur triplet. Cette proposition ne considère pas l'aspect communication entre les tâches et par conséquent les processeurs sont indépendants.

Une attention particulière a été consacrée à l'allocation et l'ordonnancement des partitions, définies comme un sous ensemble de fonctions élémentaires, dans les systèmes

avioniques [137]. Les auteurs considèrent des partitions structurées en chaines et activées périodiquement. Cependant, il est possible d'avoir des boucles à l'intérieur de la chaine dans le sens où une partition peut envoyer une donnée à la partition qui précède dans la chaine. L'approche proposée consiste en une formulation PLMNE qui vise à déterminer un processeur et un créneau de temps pour chaque partition tout en maximisant la possibilité de supporter les évolutions futures dans les temps d'exécution des partitions. De plus, les contraintes d'allocation, d'utilisation de mémoire, de chevauchement des exécutions des partitions et celles de capacité maximale des processeurs vis-à-vis du nombre de partitions supportées ont bien été considérées durant la résolution du problème. Par ailleurs, les auteurs ont suggéré d'accélérer le temps de résolution de la formulation via une phase de prétraitement basée sur la théorie des graphes. Les expérimentations ont montré que la formulation présentée peut gérer jusqu'à trente partitions sur quatre processeurs. Néanmoins, ce travail considérant la définition d'un ordonnancement statique pour les partitions, ne présente pas une formulation PLMNE de l'analyse des temps de réponse.

Dans le contexte de la méthodologie AUTOSAR [5], Wei et al. [138] ont fourni une approche pour placer les composants sur les calculateurs tout en tenant compte des exigences liées à la capacité des ressources et de l'ordonnançabilité. Cette approche vise à optimiser la communication sur le réseau. Initialement, les auteurs ont traité les composants qui sont censés être situés sur le même calculateur comme étant un seul élément atomique. Ensuite, ils ont abordé les composants qui ont des contraintes d'allocation tels que les composants logiciels de type capteur-actionneur qui sont couplés avec les ressources matérielles. Enfin, le reste des composants sont attribués aux calculateurs à l'aide d'un algorithme génétique. En se basant sur un cas d'étude de vingt-deux composants et de trois calculateurs, les auteurs ont démontré l'efficacité de leur approche en comparant leur solution avec trois autres solutions valides données par un expert. Cependant, la condition d'ordonnançabilité est modélisée en se basant sur les propriétés de l'utilisation des calculateurs ce qui représente une condition nécessaire mais pas suffisante.

Azketa et al. ont tout d'abord présenté un algorithme génétique avec une solution d'encodage permutationnel qui, en se basant sur l'analyse holistique, permet d'attribuer des priorités aux tâches et aux messages d'un système temps réel distribué [139]. Ils considèrent des tâches structurées en transactions linéaires et dont l'activation est pilotée par les données. Les auteurs se sont intéressés à trouver une affectation de priorités valide de telle sorte que l'indice d'ordonnançabilité, représentant la somme de la soustraction entre l'échéance et la latence de toutes les transactions, soit maximisé. Ce travail montre que l'algorithme génétique proposé peut trouver de meilleures solutions en comparaison avec les résultats de l'heuristique HOPA (Heuristic Optimized Priority Assignment)[140].Plus tard dans [141], les auteurs ont utilisé la segmentation du réseau pour augmenter les chances de trouver un système ordonnançable puisque celle-ci peut parfois diminuer la latence de certaines transactions. Les expérimentations ont montré l'efficacité de l'algorithme par rapport au temps moyen de calcul et au taux moyen de solutions valides trouvées. Toutefois, la répartition des tâches sur les processeurs est réalisée en utilisant une approche gloutonne simple dans laquelle un processeur est sélectionné de manière aléatoire et une tâche est placée sur ce processeur si et seulement si ce dernier garantit les exigences temporelles de cette tâche, sinon un autre processeur est sélectionné et ainsi de suite jusqu'à ce que toutes les tâches soient placées. Azketa et al. ont également étendu l'algorithme génétique proposé pour considérer l'optimisation du placement des tâches et des messages sur la plateforme d'exécution [142]. Ils se sont intéressés à la minimisation du nombre de processeurs utilisés, des temps de réponse des tâches et de l'utilisation moyenne des ressources de communication, de mémoire et de calcul. En comparaison avec une formulation en PLMNE, les auteurs ont démontré le passage à l'échelle de l'algorithme génétique pour un système industriel de cinquante tâches et huit processeurs reliés par trois réseaux de communication.

Le travail dans [143] s'intéresse à guider les concepteurs de systèmes temps réel durant

la phase de placement dans une méthodologie de développement conforme à AUTOSAR. Initialement, les auteurs supposent que le concepteur a extrait les composants logiciels à partir du modèle de l'application. Ensuite, une fois l'application modélisée par un ensemble de composants communicants, un algorithme génétique est appliqué pour trouver un placement optimisé des composants et de leurs interconnexions sur les contrôleurs et les bus de la plateforme d'exécution. L'aspect optimisation concerne, d'une part, la minimisation de la charge des calculateurs et celle du réseau de communication et, d'autre part, la minimisation des temps de réponse. L'algorithme génétique se lie à l'outil d'analyse et de validation SymTA/S [117, 118] pour l'estimation de la valeur de la fonction objectif et de la qualité d'une solution donnée. Ensuite l'algorithme génétique utilise différentes valeurs de la fonction objectif pour calculer la fonction objectif Pareto. Les auteurs ont utilisé le système "steering-by-wire" pour illustrer leur approche. Cependant, cette proposition se base sur l'expertise du concepteur pour la définition des composants logiciels et n'aborde pas le problème de partitionnement et d'affectation de priorités. S'ajoutant à cela, le passage à l'échelle de l'approche n'a pas été étudié.

Récemment, un intérêt particulier a été apporté aux systèmes multi-cœurs. Dans [144], une approche itérative permettant le placement des tâches sur les cœurs de la plateforme a été présentée. Le modèle de l'application est un modèle Simulink [145] constitué d'un ensemble de tâches qui peuvent être activées périodiquement ou selon le modèle d'activations pilotées par les données. Dans la plateforme d'exécution, les cœurs peuvent avoir des ordonnanceurs basés sur des politiques d'ordonnancement différentes. Au début de l'approche proposée, une solution de placement initiale est fournie par le concepteur puis une simulation est appliquée à cette solution. À partir des résultats de la simulation, les valeurs de la charge et les informations sur les échéances violées sont recueillies et utilisées pour trouver une autre solution de placement. Ensuite, une analyse de performance est réalisée afin de comparer la solution courante avec la solution trouvée dans l'itération précédente. La comparaison est réalisée par rapport au nombre de transactions ordonnançables et à la charge maximale de chaque cœur. Le processus est répété jusqu'à l'arrivée à une solution de bonnes performances en termes de rapidité et d'équilibrage de la charge. Toutefois, les auteurs adressent les systèmes temps réel à contraintes souples où une validation par simulation est suffisante.

Dans ce même contexte d'AUTOSAR, les auteurs dans [146] ont abordé le problème du placement des composants sur les contrôleurs en se basant sur la théorie des graphes. Dans ce travail, l'application est représentée par un ensemble de composants logiciels communicants via des signaux activés soit périodiquement, soit sporadiquement. Les auteurs se sont intéressés à une solution de placement optimisée par rapport à la communication sur le réseau d'une part, et à l'échange des signaux sporadiques sur le réseau, d'autre part. Ce dernier critère est dû au fait que l'ordonnancement des signaux sporadiques est plus dur que celui des signaux périodiques. L'approche proposée consiste tout d'abord à munir les arêtes, représentant les relations entre les composants de l'application, de différents poids dont chaque poids représente l'impact de ce signal sur le critère sous-jacent dans la solution finale, puis à appliquer un algorithme de partitionnement de graphes. Celui-ci consiste à commencer par placer tous les composants sur le même contrôleur, puis si la solution est valide en termes de capacités des ressources alors le placement est optimal puisqu'aucune communication n'est faite sur le réseau. Sinon, l'algorithme divise le graphe en deux sous ensembles de composants de telle sorte que le coût de communication entre ces deux sous ensembles soit minimisé. Enfin, chaque sous ensemble est placé sur un contrôleur différent puis cette opération est répétée jusqu'à ce qu'un compromis entre le respect des contraintes liées aux ressources et l'optimisation du coût de communication soit trouvé. Au final, la solution a été évaluée par rapport à la communication sur le réseau, la communication au sein d'un calculateur, le nombre de processeurs utilisés ainsi que leur coût, l'utilisation des processeurs et celle de la

mémoire. Les résultats ont montré que pour un exemple donné, l'approche a réussi à retrouver la solution optimale trouvée par la technique d'optimisation de programmation mathématique. Par ailleurs, les auteurs ont montré que l'approche proposée passe mieux à l'échelle que l'approche à base de programmation mathématique. Cependant, cette approche n'est pas capable de tenir compte des contraintes d'allocations comme le placement d'un composant sur un contrôleur particulier. D'autre part, cette proposition ne s'intéresse pas à la validation temporelle pendant le processus de placement.

Dans le cadre de l'outil de développement à base de modèles AUTOFOCUS3 [147], Voss et Schätz ont développé une approche permettant, d'une part, d'allouer les tâches du système sur une architecture multi-cœurs, et d'autre part, de calculer l'ordonnancement pour les tâches et les messages du système. L'application est exprimée par un graphe orienté acyclique dont les nœuds représentent les tâches et les arcs sont les messages échangés par ces tâches. La ressource de calcul cible est constituée d'un ensemble de cœurs communicant via un bus TDMA et partageant une ressource de stockage permettant l'échange des messages entre les différents cœurs. L'approche proposée s'appuie sur une formulation en un problème de satisfiabilité SMT (Satisfiability Modulo Theory) qui est résolue par le solveur YICES [148]. L'objectif de cette approche est de trouver un placement pour les tâches et les messages sur les cœurs et de déterminer les instants de début et de fin d'exécution pour chaque tâche et message selon un ordonnancement non préemptif, de telle sorte que les contraintes temporelles et celles des ressources et de la capacité de mémoire sont satisfaites. Cette approche vise aussi à minimiser la latence de bout-en-bout. Les auteurs ont évalué les performances et le passage à l'échelle de leur approche en se basant d'une part sur un système automobile, et d'autre part, sur un ensemble de tests générés aléatoirement. Les résultats ont montré que l'approche était capable de résoudre d'une manière optimale le problème pour un système constitué de $50$ tâches et $15$ cœurs en quatre heures. En outre, ce travail s'adresse aux systèmes temps réel synchrones, ce qui implique une formulation d'ordonnancement plus simple en comparaison avec la formulation de l'analyse d'ordonnançabilité considérée pour les systèmes asynchrones.

AQOSA (pour Automated Quality-driven Optimization of Software Architecture) [149, 19] est un outil générique qui cible la génération automatique d'architectures opérationnelles optimisées pour un système temps réel en se basant sur les algorithmes évolutionnistes de type algorithmes génétiques. Il permet, tout d'abord, de modéliser l'application à l'aide de composants périodiques ou sporadiques échangeant des signaux de données, puis d'explorer l'espace de conception par rapport à plusieurs degrés de libertés en utilisant trois algorithmes génétiques différents. Les principaux degrés de liberté considérés sont le choix du nombre de processeurs et de bus utilisés, le choix de la topologie du réseau de communication et le placement des composants sur les processeurs. AQOSA offre la possibilité d'optimiser plusieurs métriques comme le temps de réponse, l'utilisation des processeurs et celle des bus, la sécurité et le coût du système lié au nombre d'éléments de l'architecture matérielle utilisés. Le module d'optimisation est soumis à certaines contraintes comme les contraintes temporelles et les contraintes de déploiement qui consistent à établir une liaison entre des éléments particuliers de l'architecture matérielle. Le module d'analyse comprend différents outils permettant de mesurer les attributs de performance d'une architecture donnée. Les auteurs ont comparé les solutions apportées par AQOSA avec celles obtenues avec une recherche aléatoire [149]. Puis ils ont étudié le passage à l'échelle de leur approche en se basant sur un cas d'étude industriel du domaine de l'automobile [19]. Cependant, la validation et l'évaluation temporelles sont basées sur les approches par simulation.

### 2.4.2 Approches de partitionnement et d'ordonnancement

Dans un second temps, nous présentons les principaux travaux sur la transformation automatique d'une description fonctionnelle en un modèle de tâches en considérant l'ordonnancement de ce dernier.

Saksena et al. ont abordé dans [11] le problème de la synthèse automatique d'un modèle de tâches valide à partir d'un modèle d'application représentant le comportement fonctionnel du système fourni avec des exigences temporelles de bout-en-bout. Les auteurs considèrent un modèle d'application composé d'un ensemble d'objets en collaboration où chaque objet encapsule un ensemble d'actions représentant les fonctions atomiques du système. L'interaction entre les actions se fait par signaux synchrones et asynchrones. Il existe trois types d'actions : (1) une action qui envoie un signal asynchrone à une autre action, (2) une action qui appelle une autre action à l'aide d'un signal synchrone, elle se met en attente d'un accusé de réception et (3) une action qui envoie un signal accusant la réception d'un appel synchrone. L'approche proposée vise des systèmes mono-processeur ayant des contraintes temporelles critiques. Pour résoudre le problème de synthèse, les auteurs ont utilisé une approche de décomposition, dans laquelle ils commencent par attribuer les priorités aux actions en appliquant la stratégie DM (Deadline Monotonic). Ensuite ils cherchent à trouver un partitionnement valide d'actions sur des tâches à l'aide d'un algorithme de "Branch-and-Bound". Ce dernier est utilisé pour minimiser le nombre de tâches utilisées, d'échange de signaux entre les tâches et d'objets partitionnés sur différentes tâches. L'approche a été évaluée en comparaison avec une implémentation mono-tâche. Dans un deuxième temps, les auteurs ont proposé de simplifier l'approche en imposant des restrictions sur l'architecture telles que le partitionnement de toutes les actions appartenant au même objet ou à la même transaction sur la même tâche ou encore le partitionnement d'actions ayant la même priorité sur la même tâche. Toutefois, cette proposition pose quelques problèmes. D'une part, la méthodologie considère l'affectation de priorités indépendamment du partitionnement, ce qui peut diminuer la qualité de la solution finale. D'autre part, les objets et les tâches sont exécutés selon le principe de "*run-to-completion*" qui consiste à interdire la préemption du traitement d'un évènement par l'arrivée d'un autre évènement sur la file d'entrée de la même tâche ou le même objet ce qui implique un temps de blocage potentiellement important.

Dans le contexte du développement logiciel à base de modèles, Kodase et al. [12] ont proposé une méthode de transformation générique pour convertir un modèle structurel en un modèle d'implémentation. Le modèle structurel capturant le comportement du système est modélisé par un ensemble de composants logiciels en interaction. Ces derniers contiennent chacun un ensemble d'actions de contrôle produisant la réponse aux évènements générés. La communication entre les composants est supposée être asynchrone. Le modèle d'implémentation est un ensemble de tâches temps réel s'exécutant à base de priorités attribuées de manière statique sur une plateforme mono-processeur. Le processus de transformation proposé passe par l'identification des transactions dans le modèle structurel, puis l'affectation des priorités aux actions de chaque transaction, et enfin le partitionnement d'actions sur des tâches. Les auteurs se sont basés sur la technique de recuit simulé pour attribuer les priorités aux actions tout en respectant les contraintes temporelles. Du point de vue de l'optimisation, les auteurs se sont intéressés à la minimisation des niveaux de priorités, des temps de réponse et de la communication entre les tâches. L'ensemble des tâches est construit après l'attribution de priorités à toutes les actions du système, de telle sorte que toutes les actions ayant la même priorité sont affectées à la même tâche. Les auteurs indiquent que cette approche de partitionnement ne remet pas en cause la validation faite vis-à-vis de l'ordonnancement lors de l'étape d'attribution de priorité. De plus, les auteurs ont montré l'intérêt de leur approche de partitionnement par rapport aux approches à base d'objets et celles à base de transactions [150, 151].

Dans [2], les auteurs ont développé deux algorithmes permettant de générer automatiquement un modèle de tâches à partir d'un modèle fonctionnel représentant le comportement du système ainsi que les différentes contraintes temporelles associées. Cette génération consiste d'une part à affecter les blocs fonctionnels sur des tâches, et d'autre part, à attribuer des périodes et des échéances à ces tâches. Pour éviter les lacunes des solutions de partitionnement extrêmes, à savoir l'implémentation mono-tâche et l'implémentation multitâches dans laquelle chaque fonction est exécutée par une tâche différente [152], cette proposition a pour objectif d'aider le concepteur à trouver un compromis. Le modèle fonctionnel considéré est un modèle logique composé de blocs fonctionnels interagissant par l'échange d'événements asynchrones. Chaque bloc fonctionnel est exécuté une fois pour chaque jeton d'activation reçu sur un chemin donné et si plusieurs jetons sont transmis à un bloc fonctionnel, la sémantique d'activation est de type "OU". Cependant, la sémantique d'activation de type "ET", n'a pas été considérée. Dans ce travail, la plateforme d'exécution est mono-processeur avec un ordonnanceur à priorités dynamiques EDF. Néanmoins, l'algorithme d'ordonnancement EDF est difficile à implémenter sur le matériel et la majorité des outils commerciaux n'implémente pas des tâches avec priorités dynamiques. De plus, les algorithmes présentés ne font partie d'aucune méthodologie de développement en comparaison avec notre travail qui s'inscrit dans le cadre de méthodologies à base de modèles. En se basant sur un exemple, les auteurs ont montré l'intérêt d'utiliser des algorithmes de partitionnement basés sur l'ordonnancement EDF par rapport à ceux considérant l'ordonnancement à base de priorités fixées selon les approches RM et DM. S'ajoutant à cela, la solution proposée n'est pas flexible dans le sens où un seul modèle de tâche est évalué et donc elle n'offre pas de possibilité d'explorer d'autres alternatives. Par ailleurs, l'aspect optimisation n'est pas pris en compte et l'analyse d'ordonnançabilité n'est appliquée qu'après la génération du modèle de tâches ce qui implique que le partitionnement et l'ordonnancement sont réalisés indépendamment.

La proposition dans [13] est une approche en deux étapes qui génère automatiquement un modèle de tâches ordonnançable à partir d'une architecture logicielle implémentant les fonctions de contrôle du système. Cette dernière est composée d'un ensemble de composants logiciels interconnectés dont chaque composant s'exécute suivant le paradigme "*run-to-completion*" et il s'active à la réception de tous ses signaux d'entrée, autrement dit, la sémantique d'exécution est de type "ET". Dans la première étape de l'approche, les contraintes temporelles pour les composants sont extraites à partir des contraintes temporelles de bout-en-bout. Puis, dans la deuxième étape, les composants sont itérativement fusionnés sur des tâches et leurs exécutions sont ensuite séquencées. Plus précisément, l'idée de la première étape est de définir un intervalle de temps pour l'exécution de chaque composant tout en garantissant un ordonnancement valide. Pour cet objectif, les auteurs déterminent l'instant de début d'exécution et l'instant limite de fin d'exécution pour chaque composant en se basant sur les contraintes temporelles de bout-en-bout. Le processus de fusion dans la seconde étape est basé sur la similarité des périodes et sur le chevauchement des exécutions. Ce dernier consiste à éviter qu'une tâche ait un temps d'inactivité, un temps libre entre l'instant de fin d'exécution d'un composant et l'instant de début d'exécution du composant suivant. Par la suite, une méthode "Branch-and-Bound" a été adoptée pour déterminer la séquence des composants à l'intérieur de la tâche. L'approche proposée envisage de respecter la relation de précédence entre les composants ainsi que les contraintes temporelles de bout-en-bout tout en minimisant le surcoût d'exécution via la minimisation du nombre total de tâches. Afin d'évaluer leur approche, les auteurs se sont basés sur l'algorithme qui consiste à affecter sur la même tâche tous les composants de la même transaction s'exécutant avec la même période sur le même processeur. Les résultats ont illustré le passage à l'échelle de l'approche vis-à-vis du nombre d'étapes de l'algorithme et de son efficacité par rapport au nombre total de tâches qui en résultent. Cependant, l'approche de génération de tâches apportée dans ce travail manipule les composants à gros

grains plutôt que les fonctions, ce qui restreint les fonctions au sein d'un même composant à être partitionnées sur la même tâche. De plus, bien que le modèle fonctionnel soit activé d'une manière asynchrone, les auteurs se sont intéressés à trouver un ordonnancement statique au lieu de profiter des avantages de l'analyse des temps de réponse pour ce type d'activation. En outre, les auteurs supposent que la répartition des composants sur les ressources de calcul est prédéterminée par le concepteur.

De son côté, [153] considère le même modèle d'architecture logicielle que celui proposé dans [13]. Leur objectif est aussi de construire un modèle de tâches à partir d'un modèle à composants tout en considérant les relations de précédence et les contraintes temporelles. Cependant, l'approche proposée consiste en un algorithme basé sur les quatre règles suivantes : (1) chaque composant doit être affecté à exactement une tâche, (2) si deux composants appartiennent au même chemin avec une relation de dépendance linéaire, alors ils sont affectés à la même tâche, (3) s'il n'existe pas un chemin entre deux composants, alors ils sont affectés à différentes tâches, enfin (4) la génération des tâches doit être indépendante des temps d'exécution des composants. Après la génération du modèle de tâches, les auteurs ont déterminé les échéances absolues pour chaque tâche puis ils ont utilisé l'analyse de la demande processeur pour valider le modèle de tâche produit. Cette proposition est très proche de celle de Bartolini et al. dans [2] et suppose de même un ordonnancement de type EDF. Néanmoins, le placement des composants sur les processeurs est censé être défini par le concepteur et les phases de partitionnement, d'ordonnancement et de validation sont faites séparément, ce qui limite cette approche pour explorer d'autres solutions lorsque le modèle de tâche généré n'est pas ordonnançable.

Dans le contexte de la méthodologie AUTOSAR [5], Rongshen et al. [154] ont fourni six règles pour regrouper les "*runnables*" (représentant les plus petits fragments de code dans les composants) dans des tâches. L'idée est de trouver une solution de partitionnement qui réduit à la fois le surcoût du changement de contexte lié au nombre élevé de tâches, et le temps de retard causé par les *runnables* appartenant à la même tâche. Cependant, pour une application donnée, l'utilisation de ces règles aboutit à de nombreux choix possibles de partitionnement. Afin de choisir la meilleure combinaison, les auteurs ont proposé une équation pour évaluer la performance d'une solution de partitionnement donnée. Cette équation est un faible indicateur de performance puisqu'elle ne tient pas compte des propriétés temporelles. De plus, le surcoût du changement de contexte n'est pas formulé, il est simplement exprimé par le nombre de tâches. Par ailleurs, les auteurs n'ont fourni aucun processus pour l'exploration de l'espace des combinaisons possibles. Les auteurs ont présenté une étude de cas permettant de montrer l'applicabilité des règles proposées et les résultats ont montré que la solution obtenue était optimale par rapport aux nombre de tâches. Malheureusement une évaluation plus générale ainsi que l'évaluation du passage à l'échelle de l'approche ne sont pas présentes.

### 2.4.3 Approches d'attribution de mécanismes de protection pour les données partagées

Dans certaines applications temps réel, la communication entre différentes tâches locales à un processeur se réalise par le partage de données stockées en mémoire. Ces applications résident généralement dans les applications asynchrones activées selon le modèle périodique. L'objectif des mécanismes de protection est d'assurer un comportement temporel déterministe et une cohérence des données en imposant par exemple un accès exclusif aux variables de communication. En raison de plusieurs mécanismes de protection possibles et de leurs différents impacts sur les attributs qualitatifs du système, l'attribution d'un mécanisme de protection aux données partagées peut être vue comme un autre degré de liberté pour la conception des systèmes temps réel. Dans cette partie, nous présentons

les travaux adressant l'affectation du mécanisme de protection s'inscrivant dans le cadre de l'exploration architecturale pour le problème de déploiement.

Dans le contexte de la méthodologie AUTOSAR, [155] discute les différentes stratégies permettant de protéger les données partagées entre les tâches appartenant au même processeur. Ce travail a également mentionné l'impact de chaque stratégie sur le temps de réponse et la consommation mémoire, pour enfin soulever la question de l'optimisation de l'architecture opérationnelle vis-à-vis du compromis temps-mémoire. En revanche, aucun algorithme d'optimisation n'est utilisé. La première solution proposée pour la protection des données partagées est de démontrer l'absence de préemption. Celle-ci se base sur les temps de réponse et consiste à garantir que pour chaque paire de fonctions, la préemption de la fonction moins prioritaire par la fonction plus prioritaire ne se produira en aucun cas. La deuxième solution est de désactiver la préemption entre les fonctions. Il s'agit tout simplement d'autoriser la préemption uniquement entre l'instant de fin d'exécution d'une fonction et l'instant de début d'exécution d'une autre fonction. Cette solution a un impact négligeable sur la mémoire mais nécessite l'intégration d'un terme de blocage dans le calcul des temps de réponse. Cependant, elle résulte en un temps de blocage excessif lorsque les temps d'exécution des fonctions sont significatifs. La troisième solution réside dans l'utilisation de buffers pour le stockage des différentes copies d'une donnée. Par conséquent, elle nécessite des zones mémoires supplémentaires tandis que son coût par rapport au temps est négligeable. Enfin, la dernière solution se base sur l'utilisation des primitives "verrouiller/déverrouiller" des sémaphores. De ce fait, le protocole PCP est appliqué pour calculer le terme de blocage que subit une fonction suite à l'exécution de la primitive "verrouiller" par une fonction moins prioritaire partageant la même donnée protégée par un sémaphore.

L'étude apportée dans [155] a conduit à plusieurs contributions. Dans [18], une approche à base d'algorithmes génétiques a été proposée pour, d'une part, le placement des composants sur les contrôleurs de la plateforme d'exécution, et d'autre part, l'affectation du mécanisme de protection pour chaque donnée partagée. Cette approche s'appuie sur deux étapes, dont la première adresse l'optimisation de placement par rapport à la charge du bus de communication, suivie de l'optimisation du choix de mécanisme de protection vis-à-vis de la consommation mémoire. Ce choix se fait entre l'utilisation des sémaphores et celle des buffers. Les contraintes temporelles ont été considérées durant les deux étapes. Cependant, le partitionnement des *runnables* sur les tâches est supposé être réalisé en se basant sur les périodes des *runnables* de telle sorte que toutes les *runnables* ayant la même période et appartenant au même contrôleur sont exécutées par la même tâche. De plus, les priorités sont attribuées aux tâches par l'algorithme RM. Par conséquent, ceci implique un partitionnement et une affectation de priorités fixés. Par ailleurs, un exemple de six composants et de deux contrôleurs a été utilisé pour illustrer l'approche proposée. Le passage à l'échelle de l'approche n'a pas été testé.

Plus tard, Zeng et Di Natale [156] ont proposé une formulation en PLMNE (Programme Linéaire Mixte en Nombres Entiers) pour : (1) l'affectation d'un mécanisme de protection, parmi les sémaphores et les buffers, à chaque communication inter-tâches, (2) la détermination de l'ordre d'exécution des *runnables* au sein d'une tâche, et (3) l'attribution d'un seuil de préemption à chaque *runnable* permettant de désactiver la préemption à certains endroits de l'exécution. En d'autres termes, l'exécution d'une *runnable* peut être interrompue par une autre *runnable* seulement si la priorité de cette dernière est supérieure au seuil de la *runnable* en cours d'exécution et non pas à sa priorité. En effet, dès qu'une *runnable* commence son exécution, elle augmente sa priorité au même niveau que la valeur de son seuil. Il est donc clair que le seuil de préemption d'une *runnable* est supérieur à sa priorité. L'approche prend en considération les contraintes temporelles associées à l'exécution des *runnables* et à celle des tâches. Elle s'intéresse aussi à l'optimisation de la consommation mémoire. En plus de l'espace mémoire demandé par le mécanisme de

protection, l'espace mémoire nécessaire à la tâche pour effectuer un changement de contexte entre ses *runnables* est aussi formalisé et intégré à l'approche. S'ajoutant à cela, les auteurs ont considéré un temps d'exécution variable pour les *runnables*. Celui-ci dépend du mécanisme de protection choisi, il inclut son WCET, son temps d'exécution pour accéder à un signal et le surcoût lié à l'exécution des primitives verrouiller/déverrouiller dans le cas de l'utilisation des sémaphores. Les auteurs ont présenté une étude de cas qui a montré le passage à l'échelle de leur approche. Celle-ci était capable de résoudre un problème de 90 *runnables* et 16 tâches en quatre heures. En revanche, cette proposition s'adresse à des systèmes mono-processeur et suppose que le partitionnement des fonctions et les priorités des tâches sont connus a priori.

Dans [157], les auteurs ont également abordé l'optimisation de la mémoire. Initialement, ils ont proposé un algorithme de recuit simulé permettant d'attribuer les priorités aux tâches. Ensuite, ils ont appliqué un algorithme existant (l'algorithme d'affectation du seuil de préemption maximal) pour définir les seuils de préemption pour les tâches. Cet algorithme a été appliqué aux différentes solutions obtenues par d'autres heuristiques d'affectation de priorités dans la littérature afin d'évaluer l'algorithme de recuit simulé proposé vis-à-vis de la consommation mémoire. Dans un second temps, les auteurs ont considéré que le partitionnement des fonctions et les priorités des tâches sont déterminés au préalable, pour ensuite proposer un algorithme permettant de définir d'une façon optimale l'ordre d'exécution des *runnables* de chaque tâche ainsi que leur seuil de préemption. Cet algorithme se base sur le calcul de la borne du temps de blocage et vise à respecter l'ordonnançabilité du système et à optimiser l'utilisation de la mémoire. Enfin, cette proposition a été intégrée à un algorithme de recuit simulé adressant le partitionnement des *runnables* sur des tâches et l'affectation de priorités à ces dernières. Les auteurs ont tout d'abord montré que grâce à la considération de l'ordre d'exécution des *runnables* au sein d'une tâche et à l'utilisation des seuils de préemption, l'algorithme est capable de trouver de meilleures solutions par rapport à l'utilisation de la mémoire. Ensuite, ils ont montré que la considération du partitionnement et des priorités comme étant des degrés de liberté supplémentaires, permet d'augmenter encore plus la qualité de la solution en comparaison avec l'utilisation d'heuristiques existantes comme RM et le partitionnement à base de périodes.

Pour conclure, il est possible de classifier les différentes approches de déploiement en deux catégories principales : les approches intégrales et les approches en plusieurs étapes. Les approches intégrales comme celles proposées dans [133], [135] , [142], [134] , [137] , [147] ,[14], [157], [156], considèrent un espace de recherche englobant l'ensemble des degrés de liberté abordés. Par ailleurs, la complexité de la conception des systèmes temps réel distribués fait que les approches en plusieurs étapes sont souvent utilisées dans le domaine de l'exploration architecturale [15] ,[10], [16], [17], [19], [11],[12], [2], [13],[18]. Celles-ci considèrent l'espace de recherche, pour le problème traité, par rapport à chaque degré de liberté indépendamment des autres degrés de liberté.

### 2.4.4 Positionnement

Après avoir présenté l'ensemble des travaux de la littérature lié à l'optimisation du déploiement des systèmes temps réel, nous positionnons notre travail.

L'étude précédente a montré que les approches d'exploration architecturale proposées dans la littérature possèdent de nombreux aspects positifs mais sont néanmoins parfois limitées. Celles-ci n'abordent que partiellement le problème d'optimisation du déploiement. Comme le montre le tableau 2.4, il existe une partie des travaux qui s'est intéressée uniquement aux sous problème de placement et d'ordonnancement. Une autre partie des travaux ne considère que le problème de placement (le $\sim$ indique que le partitionnement a été considéré uniquement pour les signaux). Ces approches se répartissent en deux

classes. La première classe suppose que le partitionnement des fonctions sur les tâches a été réalisé simplement en considérant chaque fonction comme étant une tâche (exp :[134, 17]) ou en considérant toutes les fonctions ayant la même période comme une seule tâche (exp :[18]). Par rapport aux approches adressant le placement en isolation, les priorités des tâches sont généralement supposées fixées par les algorithmes classiques RM ou DM (exp :[18, 131]). Cependant, RM et DM ne sont optimaux que sous certaines conditions bien spécifiques (voir la partie 2.2.1.4) qui généralement ne sont pas satisfaites dans ces approches. La deuxième classe, représente les approches réalisant le placement en se basant sur les propriétés de l'architecture matérielle, comme la charge ou la mémoire, sans considérer l'aspect temporel. Par conséquent, ces approches ne nécessitent pas la définition des tâches et de leur priorité (exp :[146, 132]). Par ailleurs, un sous ensemble des travaux de déploiement existants ne s'est intéressé qu'au sous problème de partitionnement (exp :[153, 154]) sans aborder le problème de placement. Certains d'entre eux ont aussi considéré le problème d'ordonnancement (exp :[11, 12, 2, 157, 13]). Cette catégorie de travaux suppose une plateforme d'exécution mono-processeur ([11, 12, 2]) ou un placement défini a priori en se basant sur la consommation de ressources ([153, 13]). Le tableau 2.4, mentionne également le type de systèmes considéré par chaque approche. Ceci permet de distinguer les approches abordant l'ordonnancement par une affectation de priorité (si le système est asynchrone) de celles définissant uniquement une table d'ordonnancement statique (si le système est synchrone). Dans [132], aucun type de systèmes ne doit être spécifié puisque l'approche proposée est générique et ne dépend pas du modèle de l'application. D'autre part, les modèles d'activation pour les systèmes asynchrones permettent d'indiquer les travaux nécessitant un mécanisme de protection pour les données partagées, c'est-à-dire les travaux ayant un modèle d'activation périodique. Nous constatons que rare sont les solutions de déploiement prenant en compte la protection des données partagées lorsque les activations du modèle de l'application sont périodiques ([156, 157, 18]). S'ajoutant à cela, certaines d'entre elles considèrent un système mono-processeur [156, 157] ou un système dont le partitionnement et les priorités ont déjà été déterminés par le concepteur ([156, 18]).

En conclusion, dans toutes les approches de déploiement existantes, une optimisation a été apportée pour un sous ensemble de degrés de liberté constituant l'espace d'exploration du problème de déploiement, et ce indépendamment des autres. Autrement dit, à notre connaissance aucune des approches de déploiement existantes n'aborde à la fois l'optimisation du placement, du partitionnement, de l'ordonnancement et de l'affectation du mécanisme de protection pour les données partagées, ce qui est l'objectif de la thèse.

Une première idée qui vient à l'esprit est de combiner les solutions proposées pour les différents sous problèmes. D'ailleurs, cette démarche a été suivie par Daghsen dans sa thèse [158]. Il a proposé une approche qui consiste à appliquer une suite d'optimisations à une architecture opérationnelle donnée initialement. Chaque optimisation, à une étape donnée, est réalisée par un algorithme génétique. La première étape de l'approche est dédiée à l'optimisation du placement des composants logiciels sur les contrôleurs de la plateforme vis-à-vis du coût, de la charge du réseau et de l'utilisation maximale des contrôleurs. Cette étape considère les contraintes liées aux ressources en même temps que les contraintes temporelles. Dans la seconde étape, la solution de placement trouvée précédemment est utilisée pour optimiser, localement au sein de chaque contrôleur, la configuration des tâches. Celle-ci consiste à décider du type de chaque tâche entre "préemptive" et "non préemptive" et de lui attribuer une priorité d'exécution. Cette étape considère les latences comme métriques d'optimisation. Enfin, la dernière étape consiste à optimiser le partitionnement des *runnables* au sein de chaque contrôleur dans des tâches. Celle-ci vise à optimiser la consommation mémoire en minimisant le nombre de changements de contexte. Ce travail se situe dans le contexte d'AUTOSAR et il est destiné aux systèmes asynchrones dont les activations sont périodiques.

En fait, la nature NP-difficile de chaque sous problème est la raison pour laquelle le

problème de déploiement n'a pas été traité d'une manière entière. L'idée de restreindre l'espace de recherche pour accélérer le processus d'exploration reste la solution la plus répandue dans la littérature. Celle-ci consiste à diviser l'ensemble de l'espace de recherche en parties indépendantes pour effectuer ensuite une exploration hiérarchique.

Toutefois, la décomposition du problème de déploiement et sa résolution en plusieurs étapes semble une stratégie pratique et plus réalisable, tandis qu'en réalité celle-ci peut résulter en mauvaises solutions de déploiement [15]. Les auteurs dans [159] ont clairement signalé cette question et discuté l'inconvénient d'une exploration en plusieurs étapes. En effet, bien que cette pratique permette de réduire le problème d'optimisation global à un problème d'optimisations locales moins complexes, le découplage des dépendances conduit à une incapacité de l'algorithme d'exploration à trouver de bonnes solutions pour le problème global. Concrètement, si par exemple une solution de placement, trouvée dans la première étape de la résolution, est utilisée pour chercher une solution de partitionnement dans la seconde étape, alors ceci ne signifie en aucun cas qu'il n'existe pas d'autres solutions de placement conduisant à une meilleure solution globale. D'autre part, si initialement le placement a été optimisé par rapport à une métrique donnée, alors il se peut que pour la solution de placement trouvée il n'existe pas de solutions respectant la métrique considérée (différente de celle de la première étape) dans la seconde étape de partitionnement. Ceci est dû au fait que les métriques liées aux systèmes temps réel sont souvent orthogonales et que les décisions de placement et de partitionnement ont un impact sur la métrique évaluant la qualité du système.

| Critères de comparaison / Approches | Type de systèmes | | | Sous problèmes de déploiement | | | |
|---|---|---|---|---|---|---|---|
| | Synchrones | Asynchrones | | Placement | Partitionnement | Ordonnancement | Protection de données |
| | | Activations périodiques | Activations pilotées par données | | | | |
| Tindell, 92 [131]<br>Shoukry, 13 [146]<br>Lupu, 10 [136] | | X | | X | | | |
| Bastaricca, 01 [132] | | | | X | | | |
| Richard, 03 [133]<br>Bate, 06 [135]<br>Azketa, 11 [139, 141, 142] | | | X | X | | X | |
| Sorel, 04 [134]<br>Kugele, 09 [15]<br>Al Sheikh, 10 [137]<br>Voss, 13 [147] | X | | | X | | X | |
| Pop, 04 [10] | X | | X | X | | X | |
| Zheng, 08[14]<br>Zhu, 09 [16]<br>Zhu, 10 [17] | | X | | X | ~ | X | |
| Peng, 10[138]<br>Daghsen, 12[143]<br>Feljan, 12 [144] | | X | X | X | | | |
| Li, 11 [149]<br>Etemaadi, 13[19] | X | | | X | | | |
| Saksena, 00 [11]<br>Kodase, 03 [12]<br>Bartolini, 05 [2] | | | X | | X | X | |
| Wang, 06 [13] | X | | | | X | X | |
| Li, 08 [153] | X | | | | X | | |
| Long, 09 [154] | | X | X | | X | | |
| Zhang, 11 [18] | | X | | X | | | X |
| Zeng, 12[157] | | X | | | X | X | X |
| Zeng, 12[156] | | X | | | | | X |
| **Notre proposition** | | X | X | X | X | X | X |

Tableau 2.4 – Positionnement par rapport aux approches de déploiement de l'état de l'art

## 2.5 Synthèse et conclusion

Suite aux limites des solutions de déploiement existantes, les objectifs de cette thèse sont :

- La prise en compte du problème de déploiement d'une façon intégrale.
- Une approche de résolution suffisamment efficace pour aborder le déploiement des systèmes dont la taille est comparable à ceux utilisés dans la littérature.

Les contributions principales de la présente thèse se résument dans les points suivants :

- Une analyse des temps de réponse capable de tenir compte du modèle fonctionnel encapsulé dans le modèle de tâches, et ce pour des systèmes distribués. Cette analyse possède une formulation linéaire lui permettant d'être intégrée à un programme linéaire en nombre entiers (PLMNE). Nous rappelons que la formulation en PLMNE de l'analyse des temps de réponse pour le modèle d'activation piloté par les données n'a jamais été apportée auparavant même dans le cadre du problème de placement des tâches où le partitionnement des fonctions sur les tâches est constant. En effet, celle-ci nécessite plus d'effort de formalisation que celle considérant un modèle périodique, notamment lorsque les transactions de l'application ne sont pas linéaires. Cet effort est dû au fait que les activations des éléments du modèle d'analyse sont dépendantes. L'analyse que nous proposons est capable de préserver la sémantique d'exécution pilotée par les données du graphe fonctionnel pendant la synthèse du modèle de tâches.
- Une formulation mathématique (PLMNE) du problème de déploiement permettant de résoudre simultanément et d'une façon optimale le placement, le partitionnement et l'affectation de priorités. Enfin, la résolution du problème de déploiement d'une façon intégrale conduit le processus d'exploration à des améliorations significatives vis-à-vis de la métrique d'optimisation considérée. Ceci est illustré par les expérimentations apportées dans la chapitre 4.
- Une approche en deux étapes permettant de résoudre les sous problèmes du problème de déploiement d'une façon dépendante de telle sorte qu'elle puisse traiter des systèmes ayant une taille similaire à ceux abordés dans la littérature. La particularité de notre approche est qu'elle est apte à considérer les dépendances entre les différents degrés de liberté durant l'exploration de l'espace de recherche. Ces dépendances sont souvent ignorées par les approches de déploiement en plusieurs étapes existantes.
- Pour le cas d'un modèle d'activation périodique, l'optimisation de l'affectation du mécanisme de protection pour les données partagées a lieu en même temps que l'optimisation du déploiement.
- Un effet de changement de contexte modélisé dans l'analyse des temps de réponse, permettant ainsi l'évaluation précise d'une solution de déploiement en comparaison avec l'évaluation basée uniquement sur le nombre de tâches.

Enfin, nous rappelons que nous avons choisi la programmation mathématique (PLMNE) comme technique d'exploration architecturale parmi celles utilisées dans la littérature (les algorithmes génétiques, le recuit simulé, les algorithmes gloutons, etc.) et que nous considérons des systèmes dont l'ordonnancement est à base de priorités fixes. Nous précisons que dans ce travail nous considérons principalement l'optimisation du temps de réponse. Celle-ci est combinée avec l'optimisation de la consommation mémoire pour le cas des systèmes activés périodiquement.

Dans ce chapitre de l'état de l'art, nous avons présenté d'une part, le contexte de notre thèse, et d'autre part, les travaux connexes à notre travail. Cette présentation était centrée sur la conception des systèmes temps réel dans le cadre d'une méthodologie de développement à base de modèles. Nous avons abordé principalement deux aspects, à savoir la validation à base d'analyse temps réel et l'exploration architecturale réalisée par les techniques d'optimisation. Nous avons également discuté les travaux de la littérature par rapport à chacun de ces aspects et nous nous sommes focalisés sur l'analyse des temps de réponse pour les systèmes à priorités fixes, l'analyse d'ordonnancement temps réel pour les graphes de flot de données et sur l'exploration architecturale pour le problème de déploiement. Dans notre contexte, ce dernier consiste à associer une architecture fonctionnelle à une architecture matérielle et logicielle, tout en considérant la configuration du modèle de déploiement résultant vis-à-vis des priorités d'exécution. Enfin, nous avons identifié le besoin des systèmes temps réel en termes d'approches permettant un déploiement automatique optimisé. Dans la partie suivante, nous présentons nos solutions permettant de répondre à la problématique posée.

CHAPITRE 3

# Analyse et optimisation des systèmes temps réel distribués

## 3.1 Introduction

Nous présentons dans ce chapitre les contributions que nous avons apportées pour l'analyse et l'optimisation des systèmes temps réel embarqués. Initialement, nous exposons dans la partie 3.2 le modèle de systèmes auquel s'intéresse ce travail de thèse. Ensuite, dans la partie 3.3, nous décrivons l'extension proposée pour l'analyse des temps de réponse nécessaires aux types de systèmes visés par les travaux de cette thèse. Enfin, nous détaillons dans la partie 3.4 les deux solutions adressant l'exploration de l'espace des architectures lors de la phase de déploiement. Celles-ci sont basées sur la technique d'optimisation de programmation linéaire mixte en nombre entiers (PLMNE). La première solution est une méthode exacte qui consiste en une description formelle du problème de déploiement en PLMNE. La seconde solution est une optimisation en deux étapes basée sur les stratégies de décomposition et d'améliorations itératives. Du point de vue applicatif, ces approches ont été intégrées à l'outil de modélisation et d'analyse Qompass [173] qui a été développé au sein de l'environnement Papyrus [177], représentant un modeleur UML/MARTE [7], par l'équipe CEA/LISE.

## 3.2 Modélisation du système et notations

Les travaux de la présente thèse s'inscrivent dans le cadre de développement logiciel basé sur le cycle en "Y" selon l'approche MDA. Nous avons donc besoin de déterminer d'une part le modèle fonctionnel représentant le modèle indépendant de la plateforme (PIM), et d'autre part, le modèle de la plateforme d'exécution (PDM). La première partie décrit les caractéristiques de la plateforme d'exécution (partie 3.2.1). Ensuite, la partie 3.2.2 est dédiée à la définition de la structure et de la sémantique d'exécution pour le graphe fonctionnel des systèmes adressés.

### 3.2.1 Modèle de la plateforme d'exécution

Le modèle de la plateforme d'exécution, indiquant la topologie du réseau, est représenté par un graphe non orienté $G = (C,\beta)$, où $C$ est l'ensemble des processeurs $C = \{c_1, c_2,..., c_c\}$ et $\beta$ est l'ensemble des bus de communication par lesquels les processeurs sont connectés $\beta = \{\beta_1, \beta_2,..., \beta_\beta\}$. Chaque processeur offre un ensemble de tâches temps réel [1] permettant d'effectuer le calcul requis par les fonctions du système. Chaque tâche se caractérise par l'ensemble des fonctions qu'elle exécute et $\tau_i$ désigne la $i^{\text{ème}}$ tâche du système. $\tau_{f_i}$ retourne la tâche à laquelle appartient la fonction $f_i$. À leur tour, les bus de communication concernent un ensemble de messages réalisant la transmission des signaux. Chaque message est défini par l'ensemble des signaux qu'il transmet et $\mu_i$ représente le $i^{\text{ème}}$ message du système. $\mu_{s_i}$ retourne le message transmettant le signal $s_i$. Les tâches/messages partageant la même ressource (processeur/bus) peuvent demander l'accès à cette ressource au même instant. Afin d'arbitrer les conflits au sein d'une ressource, un ordonnanceur lui est associé. Ce dernier se base sur une certaine politique d'ordonnancement pour sélectionner la tâche ou le message parmi l'ensemble des tâches/signaux actifs auquel il accorde la ressource, pendant que le reste des tâches/messages actifs doit attendre son tour. Dans notre modèle, chaque processeur exécute un système d'exploitation temps réel avec un ordonnancement préemptif à priorités fixes. Les bus de communication sont censés être basés sur la technologie CAN (Controller Area Network), dont l'arbitrage est à base de priorités. Par conséquent, les tâches

1. Services d'exécution simultanés fournis par un système d'exploitation temps réel (RTOS pour Real Time Operating System). Une tâche dans notre modèle peut être implémentée comme un processus/thread.

et les messages sont caractérisés par une priorité d'exécution. Les processeurs et les bus de communication ont une capacité d'utilisation maximale, respectivement $\eta_{c_i}$ et $\eta_{\beta_i}$, qui ne doit pas être dépassée. Ces derniers peuvent être hétérogènes dans le sens où ils peuvent avoir des vitesses de calculs (pour les processeurs) et de transmission (pour les bus) différentes. De plus, chaque processeur possède une mémoire interne de taille limitée. La capacité de mémoire maximale pour un processeur $c_i$ est notée par $Mem_{c_i}$.

### 3.2.2 Modèle de l'architecture fonctionnelle

Le modèle fonctionnel décrit le processus opérationnel du système dans lequel les évènements générés par des utilisateurs ou par des capteurs déclenchent le calcul d'un ensemble de fonctions de contrôle définissant éventuellement la réponse du système. Ce comportement peut être capturé par un modèle de flot de données synchrone homogène. Il est décrit comme un graphe orienté $G^{'} = (F \cup \Phi, E)$, où $F$ est l'union de l'ensemble des fonctions $F$ =$\{f_1$,$f_2$,...,$f_f\}$ et $\Phi$ est l'ensemble des signaux de données $\Phi$ =$\{s_1$,$s_2$,...,$s_s\}$. Les fonctions marquées par des carrés sur la figure 3.1 représentent les opérations atomiques du système. Les signaux marqués par des cercles sur la même figure représentent leurs interactions. $E$ dénote l'ensemble des connexions entre les fonctions et les signaux. $snd(s_i)$ et $rec(s_i)$ renvoient respectivement la fonction émettrice du signal $s_i$ et la liste de ses fonctions réceptrices contenant au moins une fonction. Une transaction ($\Gamma_i$) est un sous graphe de $G^{'}$. Elle est définie par un nœud source et une liste de nœuds puits qui correspond respectivement à la première et aux dernières fonctions (dans la figure 3.1, $f_1$ et $f_4$ sont des fonctions sources et $f_3$, $f_8$ et $f_9$ sont des fonctions puits). Les fonctions et les signaux au sein d'une transaction sont structurés en un ou plusieurs *chemins* représentant des exécutions de bout-en-bout du système. Un chemin orienté $\rho_{f_i,f_j}$ depuis la fonction source $f_i$ jusqu'à la fonction puits $f_j$ est une séquence de fonctions et de signaux :

$$\rho_{f_i,f_j} = [f_i, s_i, f_{i+1}, s_{i+1}, f_{i+2}, ..., s_{j-1}, f_j]$$

tel que $f_i = snd(s_i)$ et $f_{i+1}$ appartient à $rec(s_i)$. Similairement, $f_{i+1} = snd(s_{i+1})$ et $f_{i+2}$ appartient à $rec(s_{i+1})$ et ainsi de suite jusqu'à la fonction puits sous-jacente $f_j$ qui fait partie de $rec(s_{j-1})$. $\gamma = \{\rho_1$ , $\rho_2$,..., $\rho_\rho\}$ indique l'ensemble des chemins du système. Notez que les fonctions et les signaux appartenant à la même transaction peuvent appartenir à plusieurs chemins et que différents chemins peuvent exister entre un couple de fonctions. C'est ainsi qu'une transaction peut être linéaire avec seulement un seul chemin ou non linéaire avec plus d'un chemin. Dans la figure 3.1, la première transaction ($\Gamma_1$) est linéaire avec $\rho_{f_1,f_3}$ et la seconde est non linéaire avec $\rho_{f_4,f_8}$, $\rho_{f_4,f_9}$ et $\rho_{f_4,f_9}$ (ici, il existe deux chemins différents entre $f_4$ et $f_9$). La fonction source de chaque transaction est déclenchée par un évènement externe $e_i$ qui peut être périodique ou sporadique avec respectivement une période d'activation ou un temps inter-arrivée minimal, notés dans les deux cas par $P_{e_i}$. Lorsque l'activation des fonctions est pilotée par les données (modèle *data-driven*), la période de la fonction source est héritée par toute fonction $f_i$ et tout signal $s_i$ faisant partie de la transaction, elle est nommée respectivement par $P_{f_i}$ et $P_{s_i}$. Autrement dit, les fonctions et les signaux de la même transaction ont la même période d'activation. Par ailleurs, si le modèle d'activation est périodique, les périodes des fonctions et des signaux appartenant à la même transaction peuvent différer de la période de la fonction source. En outre, une échéance $D_i$, désignant la contrainte temporelle de bout-en-bout, est associée à l'exécution de chaque chemin $\rho_i$. Les fonctions et les signaux sont respectivement caractérisés par un vecteur des temps d'exécution au pire cas $\overrightarrow{\omega_{f_i}} = (\omega_{f_i,c_1}, \omega_{f_i,c_2}, ..., \omega_{f_i,c_c})$ (noté dans la suite par WCETs) et un vecteur des temps de transmission au pire cas $\overrightarrow{\omega_{s_i}} = (\omega_{s_i,\beta_1}$,$\omega_{s_i,\beta_2}$,...,$\omega_{s_i,\beta_\beta})$ (noté par WCTTs), où $\omega_{f_i,c_c}$ et $\omega_{s_i,\beta_\beta}$ sont, respectivement, le WCET de $f_i$ sur le processeur $c_c$ et le WCTT de $s_i$ sur le bus $\beta_\beta$. Chaque fonction $f_i$ est exécutée par rapport à sa priorité statique $\pi_{f_i}$ héritée

de sa tâche. Les fonctions au sein d'une tâche possèdent donc la même priorité. De manière similaire, les signaux transmis par le même message héritent de sa priorité. $\pi_{s_i}$ dénote la priorité du signal $s_i$.

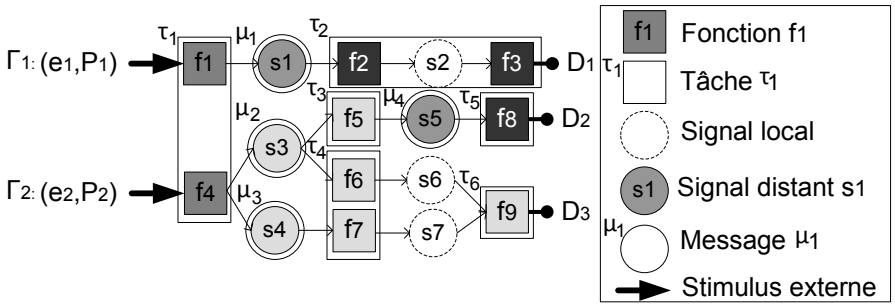



FIGURE 3.1 – Un exemple de modèle de système

Un signal est transmis localement via un processeur (cercles en pointillés sur la figure 3.1) si et seulement si sa fonction émettrice et toutes ses fonctions réceptrices sont placées sur le même processeur. Sinon, il est transmis à distance sur le réseau de communication (cercles à trait continu dans la figure 3.1). Il est à noter qu'un signal local peut représenter soit, une communication inter-tâches où toutes les fonctions réceptrices sont exécutées par la même tâche qui est différente de celle de la fonction émettrice, soit une communication intra-tâche où la fonction émettrice et toutes les fonctions réceptrices appartiennent à la même tâche. Dans ce travail, nous considérons un temps de transmission négligeable pour les signaux locaux et que chaque communication locale inter-tâches est réalisée par l'intermédiaire d'une ressource partagée. $\zeta_{s_i}$ est la ressource partagée associée à la communication inter-tâches qui est effectuée par le biais du signal local $s_i$. Nous définissons ainsi $DS(s_i)$ comme étant la taille du signal de données $s_i$. Sur la figure 3.1, la tâche (resp. le message) pour chaque fonction (resp. signal distant) est indiquée par un encadrement. Le placement des fonctions (resp. signaux) sur les processeurs (resp. bus) est illustré par différentes nuances de gris, autrement dit, les fonctions (resp. signaux) ayant la même nuance appartiennent au même processeur (resp. bus).

Rappelons que dans le cas d'activations pilotées par les données, la fonction source de chaque transaction est déclenchée à l'arrivée de l'évènement externe et les fonctions suivantes sont activées à la réception de leurs signaux entrants. La réception d'un signal correspond à la fin d'exécution de sa fonction émettrice si le signal est local, ou à l'arrivée du message délivrant les données de ce signal si le signal est distant. L'envoi d'un message démarre lorsque toutes les fonctions émettrices des signaux transmis par ce message ont terminé leur exécution. Une fonction appartenant à plusieurs chemins (par exemple : la fonction $f_9$ de la figure 3.1) peut avoir différentes sémantiques d'activation : une sémantique "ET" ou une sémantique "OU". Sous la sémantique "ET", la fonction doit attendre l'arrivée de tous les signaux entrants pour qu'elle puisse s'activer, tandis que pour la sémantique "OU", la fonction est activée à l'arrivée de n'importe quel signal entrant. Un signal appartenant à plusieurs chemins est un signal ayant plusieurs fonctions réceptrices. D'autre part, si les activations sont périodiques, alors chaque fonction (resp. signal) est activée à l'arrivée de l'évènement d'horloge indépendamment des autres fonctions et signaux. Du point de vue de la taille des buffers de chaque lien du graphe fonctionnel, nous supposons que pour le cas d'activations périodiques nous disposons d'une taille suffisante qui a été déterminée au moment où les périodes sont spécifiées. Par contre, lorsque les activations sont pilotées par les données, la taille des buffers est bornée par le test d'ordonnançabilié. Dans le cas où les échéances sont contraintes, des buffers d'une taille égale à 1 sont suffisants pour exécuter un système ayant des activations pilotées par les données.

Le tableau 3.1 résume les notations utilisées dans la présentation du modèle de systèmes

temps réel considéré dans les travaux de cette thèse.

| | |
|---|---|
| $C$ | Ensemble des processeurs du système |
| $\beta$ | Ensemble des bus de communication du système |
| $c_i$ | Le $i^{\text{ème}}$ processeur |
| $\beta_i$ | Le $i^{\text{ème}}$ bus de communication |
| $\eta_i$ | La capacité d'utilisation maximale du processeur/ bus de communication $i$ |
| $F$ | Ensemble des fonctions du système |
| $\Phi$ | Ensemble des signaux du système |
| $f_i$ | La $i^{\text{ème}}$ fonction du système |
| $s_i$ | Le $i^{\text{ème}}$ signal du système |
| $\tau_i$ | La $i^{\text{ème}}$ tâche du système |
| $\mu_i$ | Le $i^{\text{ème}}$ message du système |
| $rec(s_i)$ | Ensemble des fonctions réceptrices du signal $s_i$ |
| $snd(s_i)$ | La fonction émettrice du signal $s_i$ |
| $e_i$ | Le $i^{\text{ème}}$ évènement externe du système |
| $P_i$ | La période d'activation de l'évènement externe/la fonction/le signal $i$ |
| $\rho_{f_i,f_j}$ | Le chemin de la fonction $f_i$ à la fonction $f_j$ |
| $\rho_i$ | Le $i^{\text{ème}}$ chemin du système |
| $\gamma$ | Ensemble des chemins du système |
| $D_i$ | L'échéance de bout-en-bout du chemin $\rho_i$ |
| $\omega_{i,j}$ | WCET/WCTT de la fonction/du signal $i$ sur le processeur/bus $j$ |
| $\overrightarrow{\omega_i}$ | Le vecteur des WCET/WCTT pour la fonction $f_i$/ le signal $s_i$ |
| $\pi_i$ | La priorité de la fonction/signal $i$ |
| $\tau_{f_i}$ | La tâche à laquelle appartient la fonction $f_i$ |
| $\mu_{s_i}$ | Le message transmettant le signal $s_i$ |
| $\zeta_{s_i}$ | La ressource partagée associée au signal local $s_i$ |
| $DS(s_i)$ | La taille du signal $s_i$ |
| $Mem_{c_i}$ | La capacité de mémoire maximale du processeur $c_i$ |

Tableau 3.1 – Résumé des notations du modèle de système

## 3.3 Analyse des temps de réponse appliquée au modèle fonctionnel

Après avoir présenté le modèle des systèmes temps réel considéré par ce travail de thèse, nous abordons dans cette partie l'analyse des temps de réponse. Dans la partie 3.3.1, nous présentons la technique d'analyse des temps de réponse proposée pour le cas d'un modèle fonctionnel ayant des activations pilotées par les données. Pour cela, nous nous sommes basés sur l'analyse des temps de réponse avec propagation des gigues présentée dans [101] et sur les solutions proposées dans [115] pour le traitement des modèles non linéaires. Ensuite, pour le modèle d'activation périodique, nous discutons dans la partie 3.3.2 la différence par rapport à l'analyse apportée précédemment. Pour ce type de modèle d'activation, nous nous sommes appuyés sur l'analyse présentée dans [17]. Dans ce travail, quel que soit le modèle d'activation considéré, nous supposons qu'une fonction (un signal) ne peut se réactiver qu'après la fin d'exécution de son activation précédente, ce qui indique un modèle d'activations à échéances contraintes. Nous reviendrons plus tard dans la partie 3.4.2 sur la raison conduisant à cette hypothèse et nous indiquons les moyens permettant de considérer un cas plus général où les échéances peuvent être supérieures aux périodes sous-jacentes.

Les notations utilisées pour l'analyse des temps de réponse ont été rassemblées dans le tableau 3.2.

| | |
|---|---|
| $W_i$ | Le temps de complétion de la fonction $f_i$ ou le temps d'attente du signal $s_i$ dans la file |
| $J_i$ | La gigue d'activation de la fonction/ du signal $i$ |
| $R_i$ | Le WCRT de la fonction/ du signal $i$ |
| $dp_{i,j}$ | La relation de précédence entre les fonctions/signaux $i$ et $j$ |
| $hp(i)$ | Ensemble des fonctions/signaux avec une priorité supérieure à la priorité de la fonction/du signal $i$, appartenant au même processeur/bus que $i$ et à différentes tâches/ différents messages |
| $\varrho(i)$ | Ensemble de chemins auxquels la fonction/ le signal $i$ appartient |
| $Eo_{f_i}$ | L'ordre de séquence de la fonction $f_i$ à l'intérieur de sa tâche |
| $\Omega_{s_i}$ | Le bus auquel le signal $s_i$ est affecté |
| $B_i$ | Le temps de blocage du signal/fonction $i$ |
| $cs$ | Le coût de changement de contexte |

Tableau 3.2 – Résumé des notations d'analyse des temps de réponse

### 3.3.1 Analyse des temps de réponse pour le cas d'activations pilotées par les données

1. **Phase de prétraitement :** avant de calculer le pire temps de réponse (noté par WCRT), nous appliquons la transformation proposée dans [115] pour les fonctions ayant une sémantique d'activation de type "OU". Cette transformation consiste à séparer les séquences associées à chacun des différents signaux d'entrée. Toute la séquence des fonctions et des signaux débutant de la fonction ayant la sémantique "OU " est répliquée et le nombre de réplication est égal au nombre de signaux entrants. Cette transformation se traduit par un graphe fonctionnel sans fonctions ayant la sémantique d'activation "OU". La figure 3.2 montre un exemple d'une telle transformation. Les répliques des fonctions et des signaux sont ajoutées respectivement à $F$ et $\Phi$.

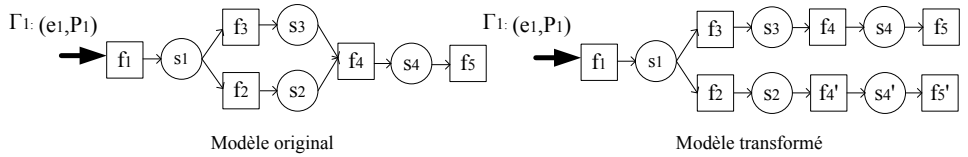



FIGURE 3.2 – Illustration de la transformation de la sémantique d'activation "OU"

2. **Calcul des pires temps de réponse :** dans l'analyse des temps de réponse avec propagation de gigues, les instants d'activation des signaux dépendent des instants de terminaison des fonctions déclencheuses. La même chose se produit pour les fonctions activées à la réception des signaux. Nous rappelons que l'instant d'activation est le moment auquel la fonction/le signal est prêt à s'exécuter et que l'instant de terminaison est le temps de fin d'exécution (voir la figure 3.3).

   Nous définissons $R_{f_i}$ et $R_{s_i}$ comme étant les temps de réponse global de la fonction $f_i$ et du signal $s_i$ respectivement dans le pire scénario. Il représente la durée entre l'instant d'arrivée de l'évènement externe (l'instant d'activation de la fonction source) et l'instant auquel l'exécution de $f_i$ ou de $s_i$ est terminée. Le WCRT global pour une fonction $f_i$ est calculé comme la somme de la gigue $J_{f_i}$, représentant le délai entre

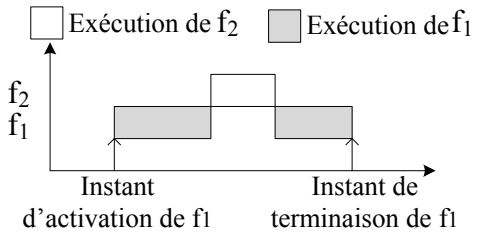



FIGURE 3.3 – Définition des instants d'activation et de terminaison

l'instant d'arrivée de l'évènement externe et l'instant d'activation de la fonction, et le temps de complétion $W_{f_i}$. $W_{f_i}$ est le plus grand délai possible entre les instants d'activation et de terminaison (voir l'équation (3.1)). Cependant, le calcul du WCRT global pour un signal $s_i$ est légèrement différent de celui des fonctions. Il est égal à la somme de sa gigue $J_{s_i}$, de son pire temps de transmission $\omega_{s_i,\beta_j}$, du temps de transmission de tous les signaux au sein du même message que $s_i$ et de la plus longue durée d'attente dans la file avant son élection pour une transmission sur le bus $\beta_j$. Cette durée d'attente est aussi notée par $W_{s_i}$ (voir l'équation (3.2)).

$$R_{f_i} = J_{f_i} + W_{f_i} \tag{3.1}$$

$$R_{s_i} = J_{s_i} + \omega_{s_i,\beta_j} + \sum_{\substack{s_k \in \{\Phi \setminus s_i\}: \\ \mu_{s_i} = \mu_{s_k}}} \omega_{s_k,\beta_j} + W_{s_i} \tag{3.2}$$

Par exemple, dans la figure 3.4, le signal $s_4$ n'est activé qu'après la fin d'exécution de la fonction $f_5$ c'est-à-dire que la gigue de $s_4$ est égale au WCRT global de $f_5$. Similairement, la fonction $f_6$ n'est activée qu'après la fin d'exécution de $s_4$, donc la gigue de $f_6$ est égale au WCRT global de $s_4$. Au final, le WCRT global de $f_6$ est le délai écoulé depuis l'arrivée de l'évènement externe $e_2$ jusqu'à la fin d'exécution de $f_6$.

3. **Calcul des gigues et des temps de complétion pour les fonctions :** le calcul des gigues et des temps de complétion pour les fonctions dépend de la sémantique de synchronisation des tâches. Dans ce qui suit, nous présentons trois sémantiques différentes ainsi que le calcul de $J_{f_i}$ et de $W_{f_i}$ approprié. Ensuite, nous discutons le calcul de $J_{s_i}$ et de $W_{s_i}$. Nous illustrons chaque sémantique à l'aide d'un exemple simple présenté par la figure 3.4. Cet exemple consiste en deux processeurs communiquant par un bus. Chaque processeur contient une tâche comprenant trois fonctions. La tâche $\tau_1$ et la tâche $\tau_2$ contiennent respectivement les fonctions $f_1, f_2, f_5$ et les fonctions $f_3, f_4, f_6$. Le bus contient deux messages chacun possédant un signal. Les fonctions au sein d'une tâche sont structurées en deux chemins.

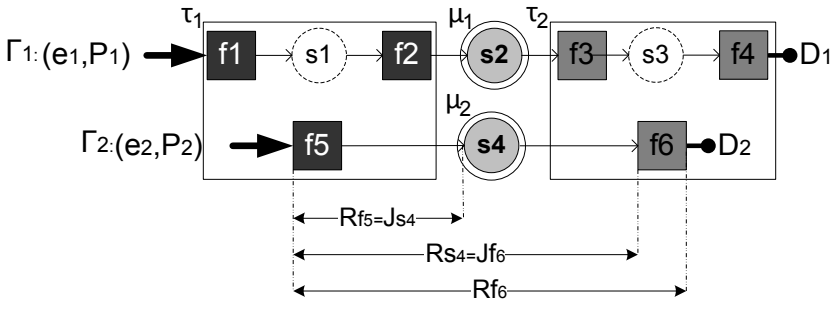



FIGURE 3.4 – Exemple basique illustrant les sémantiques de synchronisation des tâches

Les différentes sémantiques se distinguent relativement aux conditions d'activation de la tâche et aux instants auxquels celle-ci renvoie ses résultats.

A) **La sémantique 'Input N-of-N / output N-of-N' :** ce cas correspond au déclenchement d'une tâche à l'arrivée de toutes ses données d'entrée. Autrement dit, la tâche ne commence son exécution qu'après la disponibilité de toutes ses données d'entrée. En outre, la tâche génère simultanément tous ses résultats en sortie à la fin de son exécution.

La gigue de la fonction $f_i$ est égal au WCRT maximum de tous les signaux reçus par les fonctions appartenant à la tâche de $f_i$ (y compris $f_i$). En d'autres termes, $f_i$ doit attendre l'arrivée des signaux entrants de toutes les fonctions au sein de sa tâche et l'arrivée de ses propres signaux d'entrée (voir l'équation (3.3)).

$$J_{f_i} = \max_{\substack{f_j \in F: \\ \tau_{f_i} = \tau_{f_j}}} \left[ \max_{\substack{s_k \in \Phi: \\ f_j \in rec(s_k)}} [R_{s_k}] \right] \tag{3.3}$$

Dans le cas d'un modèle d'activation à échéance contrainte, le temps de complétion de la fonction $f_i$ comprend son WCET, le WCET de toutes les fonctions appartenant à la même tâche que $f_i$, et les temps d'interférences provenant des fonctions plus prioritaires qui sont exécutées sur le même processeur (voir l'équation 3.4). $hp(f_i)$ représente l'ensemble des fonctions ayant une priorité supérieure à celle de $f_i$ et qui résident dans le même processeur qu'elle.

$$W_{f_i} = \omega_{f_i,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k}}} \omega_{f_k,c_j} + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil \cdot \omega_{f_k,c_j} \tag{3.4}$$

***Illustration basée sur l'exemple :*** considérant cette sémantique, il existe trois scénarios possibles pour la tâche $\tau_2$ de notre exemple illustratif de la figure 3.4. Elles conduisent pourtant toutes au même résultat par rapport aux WCRTs des fonctions. En d'autres termes, l'ordre d'exécution des fonctions à l'intérieur de la tâche n'influence pas le WCRT des fonctions. Dans les trois scénarios, l'activation de $f_3$ dépend de ses données d'entrée et de celles de $f_6$. De même, l'activation de $f_6$ dépend de ses propres données d'entrée et des données d'entrée de $f_3$. Par ailleurs, les données sortants de $f_3$, $f_4$ et $f_6$ ne sont fournies qu'après l'exécution de toutes les fonctions de la tâche $\tau_2$ ($f_3$, $f_4$ et $f_6$). Du point de vue de l'analyse des temps de réponse, cela implique que toutes les fonctions d'une tâche ont la même gigue et le même temps de complétion, et donc le même WCRT. Par conséquent, au niveau de $\tau_2$, $J_{f3} = J_{f4} = J_{f6} = \max(R_{s_2}, R_{s_4})$, représenté par $J$ sur la figure 3.5, et $W_{f3} = W_{f4} = W_{f6} = \omega_{f_3,c_2} + \omega_{f_4,c_2} + \omega_{f_6,c_2}$.

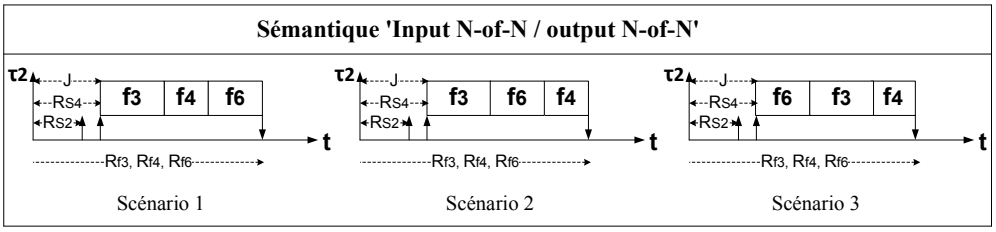



FIGURE 3.5 – Illustration de la sémantique 'Input N-of-N / output N-of-N'

B) **La sémantique 'Input N-of-N / output 1-of-N' :** afin d'être activée, une tâche doit attendre l'arrivée de toutes ses données d'entrée (la même hypothèse que celle de

la sémantique précédente). Ensuite, dès qu'une donnée d'entrée est traitée, son résultat est obtenu sans l'attente du traitement des autres données d'entrée.

Pour cette sémantique, l'ordre d'exécution à l'intérieur d'une tâche a un impact sur le WCRT des fonctions. Par conséquent, l'analyse des temps de réponse doit prendre en considération l'ordre des fonctions au sein des tâches comme il est montré par la suite. Pour ce cas, la gigue $J_{f_i}$ de la fonction $f_i$ est calculée de la même façon que dans la formule (3.3). Le temps de complétion de la fonction $f_i$ comprend son WCET, le WCET des fonctions appartenant à la tâche de $f_i$ et ayant un ordre de séquence plus élevé que celui de $f_i$ et le temps de préemption provenant des fonctions plus prioritaires appartenant au même processeur que $f_i$, c'est-à-dire les fonctions de $hp(f_i)$. Dans l'équation (3.5), $Eo_{f_i}$ se réfère à l'ordre de séquence de la fonction $f_i$ dans sa tâche.

$$W_{f_i} = \omega_{f_i,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ Eo_{f_k} > Eo_{f_i}}} \omega_{f_k,c_j} + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil \cdot \omega_{f_k,c_j} \tag{3.5}$$

***Illustration basée sur l'exemple :*** selon cette sémantique, pour l'exemple illustratif, nous illustrons deux scénarios possibles pour la tâche $\tau_1$ (voir la figure 3.6). Dans ces scénarios, l'activation de $f_1$ dépend de ses données d'entrée et des données d'entrée de $f_5$. De même pour l'activation de $f_5$ qui dépend de ses propres données d'entrée et des données de $f_1$.

- **Scénario 1 :** dans ce scénario, $f_1$ et $f_5$ deviennent prêtes à s'exécuter au même instant, juste après l'arrivée des deux évènements d'entrée $e_1$ et $e_2$. Ceci implique donc que $J_{f_1} = J_{f_2} = J_{f_5} = 0$. L'ordre d'exécution à l'intérieur de la tâche $\tau_1$ est fixé de telle sorte que $f_1$ soit exécutée avant $f_2$ et qu'à son tour $f_2$ soit exécutée avant $f_5$. Le résultat de la fonction $f_2$ est retourné avant que la fonction $f_5$ commence son exécution ce qui implique que $J_{s_2} = R_{f_2} < R_{f_5} = J_{s_4}$. Dans ce scénario, $W_{f_1} = \omega_{f_1,c_1}, W_{f_2} = \omega_{f_1,c_1} + \omega_{f_2,c_1}$ et $W_{f_5} = \omega_{f_1,c_1} + \omega_{f_2,c_1} + \omega_{f_5,c_1}$. Similairement à la sémantique précédente, $J_{f3} = J_{f4} = J_{f6} = \max(R_{s_2}, R_{s_4})$.
- **Scénario 2 :** la différence pour ce scénario comparé au scénario précédent, est que les fonctions de la tâche $\tau_1$ s'exécutent dans l'ordre suivant : $f_1$ , $f_5$ puis $f_2$. La sortie de la fonction $f_5$ est renvoyée avant que $f_2$ commence son exécution, et donc $J_{s_4} = R_{f_5} < R_{f_2} = J_{s_2}$ et $W_{f_1} = \omega_{f_1,c_1}, W_{f_2} = \omega_{f_1,c_1} + \omega_{f_2,c_1} + \omega_{f_5,c_1}$ et $W_{f_5} = \omega_{f_1,c_1} + \omega_{f_5,c_1}$.

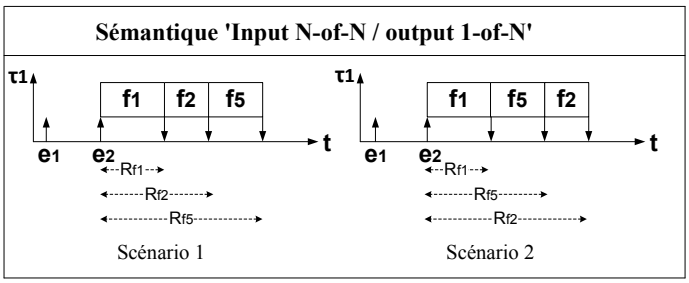



FIGURE 3.6 – Illustration de la sémantique 'Input N-of-N / output 1-of-N'

Il est à noter que quelque soit le scénario choisi pour cette sémantique, $f_2$ ne pourra jamais s'exécuter avant $f_1$ puisque son exécution est soumise à l'exécution de $f_1$. Autrement dit, les données d'entrée pour $f_2$ n'arrivent qu'après l'exécution

de $f_1$.

C) **La sémantique 'Input 1-of-N / output 1-of-N' :** c'est le cas où l'exécution d'une tâche peut être déclenchée à l'arrivée de n'importe quelle donnée d'entrée. La tâche est donc activée à chaque fois qu'une donnée en entrée est disponible. La tâche fournit le résultat correspondant au traitement d'une donnée particulière sans attendre le traitement des autres données (la même hypothèse que la sémantique précédente). Ici, l'exécution de la tâche suit le principe de "*run-to-completion*" c'est-à-dire qu'elle ne se réactive pas à l'arrivée d'une donnée si le traitement de la donnée précédente n'est pas encore fini.

D'un point de vue de l'analyse des temps de réponse, le scénario "pire cas" pour chaque fonction de la tâche doit être considéré. Le pire scénario pour la fonction $f_i$ se produit lorsque la donnée d'entrée de la tâche qui cause l'exécution de $f_i$ arrive après l'arrivée de toutes les autres données d'entrée de la tâche. Par la suite, nous présentons le calcul des gigues et celui des temps de complétion correspondant à ce scénario. Si la fonction $f_i$ est une fonction source déclenchée par l'évènement externe alors sa gigue est nulle. Sinon elle est égale au WCRT maximum parmi tous les WCRTs des signaux reçus.

$$J_{f_i} = \begin{cases} 0 & f_i \text{ est une fonction source} \\ \max\limits_{\substack{s_i \in \Phi: \\ f_i \in rec(s_i)}} [R_{s_i}] & \text{sinon} \end{cases} \tag{3.6}$$

Le temps de complétion d'une fonction $f_i$ inclue son WCET et le temps d'interférences provenant des fonctions prioritaires appartenant au même processeur que celui de $f_i$, c'est-à-dire les fonctions dans $hp(f_i)$. De plus, le temps de complétion doit intégrer le WCET de toutes les fonctions $f_k$ qui sont exécutées par la même tâche que $f_i$ et qui, soit appartiennent à différents chemins soit précédent $f_i$ dans le même chemin. Dans l'équation (3.7), $dp_{k,i} = 1$ indique que l'exécution de $f_i$ dépend de l'exécution de $f_k$ sur le même chemin. L'ajout de ce dernier terme est dû à la considération du scénario "pire cas" pour la fonction $f_i$. Une explication à base d'exemple du scénario "pire cas" est fournie dans le point suivant. $\varrho(f_i)$ représente l'ensemble des chemins auxquels la fonction $f_i$ appartient.

$$W_{f_i} = \omega_{f_i,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ \varrho(f_i) \cap \varrho(f_k) = \emptyset}} \omega_{f_k,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ dp_{k,i} = 1}} \omega_{f_k,c_j} \quad + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil \omega_{f_k,c_j} \tag{3.7}$$

***Illustration basée sur l'exemple :*** par rapport à l'exemple illustratif, nous fournissons deux scénarios possibles pour la tâche $\tau_2$ considérant cette sémantique (voir la figure 3.7).

- **Scénario 1 :** dans ce scénario, la donnée d'entrée de $f_3$ est supposée être présente avant celle de la fonction $f_6$. C'est ainsi que $f_3$ devient prête à s'exécuter avant $f_6$. Par conséquent, $R_{s_2} = J_{f_3} < R_{s_4} = J_{f_6}$. Par ailleurs, $f_4$ débute son exécution avant même que $f_6$ ne devienne prête à s'exécuter. Le résultat de la fonction $f_4$ est retourné avant que la fonction $f_6$ commence son exécution, et donc $W_{f3} = \omega_{f_3,c_2}, W_{f4} = \omega_{f_4,c_2} + \omega_{f_3,c_2}$ et $W_{f6} = \omega_{f_4,c_2} + \omega_{f_3,c_2} + \omega_{f_6,c_2}$.
- **Scénario 2 :** ce scénario présente le cas où l'entrée de $f_6$ est arrivée avant celle

de $f_3$ ($R_{s_4} = J_{f_6} < R_{s_2} = J_{f_3}$). La fonction $f_3$ ne commence son exécution qu'après l'achèvement de la fonction $f_6$ et cette dernière renvoie son résultat avant que $f_3$ commence son exécution, et donc $W_{f_6} < W_{f_3}$ puisque $W_{f3} = \omega_{f_3,c_2} + \omega_{f_6,c_2}, W_{f4} = \omega_{f_4,c_2} + \omega_{f_3,c_2} + \omega_{f_6,c_2}$ et $W_{f6} = \omega_{f_6,c_2}$.

Nous pouvons noter que le scénario "pire cas" pour $f_3$ et $f_4$ est le deuxième scénario alors qu'il s'agit du premier pour $f_6$.

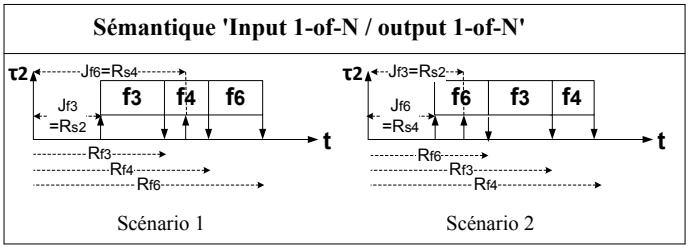



FIGURE 3.7 – Illustration de la sémantique 'Input 1-of-N / output 1-of-N'

De ces premières études, nous pouvons noter que dans la situation où une transformation est appliquée à la sémantique d'exécution de type "OU", une certaine prudence en ce qui concerne l'analyse des temps de réponse pour les fonctions est requise. Cette précaution consiste soit à s'assurer que la fonction et sa réplique soient exécutées par la même tâche et, par conséquent, assurer leur exécution séquentielle, soit à ajouter un temps de blocage (correspondant au temps d'exécution de la fonction) au temps de complétion de la fonction et à celui de sa réplique. Ce temps de blocage exprime l'empêchement de la préemption d'une fonction par sa réplique et vice versa. De plus, pour les deux premières sémantiques présentées (sémantiques avec Input N-of-N), nous ne faisons aucune hypothèse sur la période d'activation des évènements déclencheurs de la tâche ayant plusieurs entrées contrairement à ce qui est supposé dans [115]. Les auteurs justifient cette hypothèse par l'accumulation illimitée des évènements les plus fréquents qui peut se produire dans le cas de périodes différentes. Dans notre cas, nous considérons un point de synchronisation à l'entrée de la tâche qui est exprimé via le calcul de la gigue des fonctions de cette tâche. Ceci peut entrainer une perte de données lorsque la tâche est censée traiter uniquement la dernière donnée produite à une entrée particulière. Autrement dit, la taille du buffer pour chaque entrée de la tâche est de 1. Néanmoins, pour notre situation, il n'y a pas de perte de données puisque nous considérons un modèle d'activation à échéance contrainte où l'évènement externe ne se représente pas une nouvelle fois avant le traitement complet de l'instance précédente.

4. **Discussion :** afin de confronter les unes aux autres les différentes sémantiques présentées ci-dessus, nous nous basons sur les quatre cas décrits dans la figure 3.8.

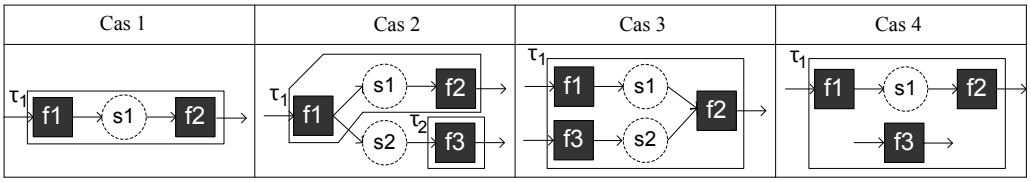



FIGURE 3.8 – Une liste de cas permettant la comparaison des différentes sémantiques de synchronisation des tâches

- **Cas 1 :** lorsque toutes les fonctions de la tâche sont positionnées séquentiellement sur un même chemin, quelle que soit la sémantique appliquée, le résultat par rapport au pire temps de réponse de la tâche est toujours le même. De ce fait, le WCRT de la fonction $f_2$ est identique pour les trois sémantiques.
- **Cas 2 :** pour ce cas, la considération de la sémantique 'Input N-of-N / output 1-of-N' ou de la sémantique 'Input 1-of-N / output 1-of-N' mène au même résultat par rapport aux WCRTs des fonctions. Pour ces deux sémantiques, la gigue de la fonction $f_3$ est égale au WCRT de la fonction $f_1$, celui-ci n'inclut pas le WCET de la fonction $f_2$. Cependant, pour la sémantique 'Input N-of-N / output N-of-N', la gigue de $f_3$ est plus pessimiste puisqu'elle comprend en plus le WCET de $f_2$. Par ailleurs, sachant que la priorité de $f_2$ est la même que celle de $f_1$, le WCET de $f_2$ est indirectement considéré dans le WCRT de la fonction $f_3$ pour les autres sémantiques 'Input N-of-N / output 1-of-N' et 'Input 1-of-N / output 1-of-N', sauf que cette fois-ci, il est inclut dans le temps de complétion de $f_3$ plutôt que dans sa gigue. En conclusion pour ce deuxième cas, les différentes sémantiques aboutissent également au même pire temps de réponse pour les deux tâches. Ceci est vrai car $f_2$ préempte l'exécution de $f_3$ uniquement une seule fois puisque la période de $f_3$ est égale à la période de $f_2$ et la gigue de $f_3$ est forcément supérieure ou égale à celle de $f_2$. Enfin, il est évident que si les fonctions $f_1$, $f_2$ et $f_3$ sont implémentées par la même tâche, alors les sémantiques 'Input N-of-N / output N-of-N' et 'Input 1-of-N / output 1-of-N' renvoient le même WCRT pour toutes les fonctions. Ce résultat reste plus pessimiste que celui obtenue par la sémantique 'Input N-of-N / output 1-of-N'.
- **Cas 3 :** pour ce cas, bien que la considération de la sémantique 'Input 1-of-N / output 1-of-N' réduit la propagation de la gigue, les trois sémantiques vis-à-vis du WCRT de la fonction $f_2$ donnent le même résultat. Néanmoins, la réduction de la gigue apportée par la sémantique 'Input 1-of-N / output 1-of-N' peut avoir un impact positif sur le WCRT des fonctions moins prioritaires puisque la gigue influence le nombre de fois qu'une préemption des fonctions moins prioritaires a lieu.
- **Cas 4 :** si les fonctions d'une tâche sont réparties sur plusieurs chemins, alors nous avons deux situations : le cas où les gigues des premières fonctions déclenchées dans la tâche sont égales (ici $J_{f_1} = J_{f_3}$) et le cas opposé où celles-ci sont différentes. Lorsque les gigues sont égales, les sémantiques 'Input N-of-N / output N-of-N' et 'Input 1-of-N / output 1-of-N' retournent la même valeur pour le WCRT des fonctions puisque le temps de complétion des fonctions dans les deux sémantiques est identique. La sémantique 'Input N-of-N / output 1-of-N' reste meilleure par rapport aux WCRTs des fonctions puisque le temps de complétion dans cette sémantique (grâce à l'ordre d'exécution défini à l'intérieur de la tâche) ne présente pas une attente bilatérale comme c'est le cas dans les autres sémantiques. En d'autres termes, soit $W_{f_3}$ contient $\omega_{f_2}$ soit inversement, $W_{f_2}$ contient $\omega_{f_3}$, produisant alors une amélioration sur l'un des chemins. Dans le cas des gigues différentes, le phénomène de propagation des gigues est réduit pour la sémantique 'Input 1-of-N / output 1-of-N' en comparaison aux autres sémantiques. Par exemple, si la gigue de $f_1$ est plus grande que celle de $f_3$ alors $f_3$ n'a pas à attendre plus longtemps que sa gigue. Ici, il existe un compromis entre la sémantique 'Input 1-of-N / output 1-of-N' et la sémantique 'Input N-of-N / output 1-of-N' par rapport aux WCRTs des fonctions. L'une ('Input N-of-N / output 1-of-N') cause un calcul pessimiste lié à la gigue et l'autre ('Input 1-of-N / output 1-of-N') cause un calcul pessimiste dû à l'attente bilatérale au sein de la tâche. Cependant, la sémantique 'Input N-of-N / output N-of-N' comprend les deux sources de pessimisme, d'une part la gigue et d'autre part l'attente bilatérale.

5. **Calcul des gigues et des temps de complétion pour les signaux :** la sémantique de synchronisation des messages est de type 'Input N-of-N / output N-of-N'. Du point de vue de l'analyse des temps de réponse, ceci implique que tous les signaux transmis par un message donné ont les mêmes instants de début et de fin de transmission. Si un signal $s_i$ est transmis sur le réseau, alors sa gigue est égale au maximum WCRT parmi tous les WCRTs des fonctions émettrices des signaux transmis par ce même message contenant $s_i$. La gigue d'un signal transmis localement à un processeur est égale soit au WCRT de sa fonction émettrice lorsqu'il représente une transmission inter-tâches, soit à la gigue de cette dernière s'il est transmis intra-tâche.

$$J_{s_i} = \begin{cases} \max\limits_{\substack{s_j \in \Phi: \\ \mu_{s_i} = \mu_{s_j}}} [R_{snd(s_j)}] & s_i \text{ est transmis inter-processeurs} \\ R_{snd(s_i)} & s_i \text{ est transmis inter-tâches} \\ J_{snd(s_i)} & s_i \text{ est transmis intra-tâche} \end{cases} \tag{3.8}$$

Le délai d'attente dans la file pour un signal $s_i$ comprend son temps de blocage $B_{s_i}$ dû à la non préemptivité ainsi que les temps d'interférences provenant des signaux prioritaires qui sont en attente d'une transmission sur le même bus que $s_i$ (les signaux de $hp(s_i)$). Ici, les interférences représentent le fait que les signaux plus prioritaires sont placés devant le signal $s_i$ dans la file d'attente.

$$W_{s_i} = B_{s_i} + \sum_{s_k \in hp(s_i)} \left\lceil \frac{W_{s_i} + J_{s_k}}{P_{s_k}} \right\rceil \omega_{s_k,\beta_j} \tag{3.9}$$

Le pire temps de blocage d'un signal $s_i$ représente le plus grand WCTT de n'importe quel message différent de celui de $s_i$, ayant une priorité inférieure et partageant le même bus de communication que le message contenant $s_i$. Notez que les signaux $s_i$, $s_j$ et $s_k$ sont tous transmis sur le bus $\beta_l$.

$$B_{s_i} = \max_{\substack{s_j \in \Phi: \\ \pi_{s_i} > \pi_{s_j} \\ \Omega_{s_i} = \Omega_{s_j} \\ \mu_{s_i} \neq \mu_{s_j}}} [\omega_{s_j,\beta_l} + \sum_{\substack{s_k \in \{\Phi \setminus s_j\}: \\ \mu_{s_j} = \mu_{s_k}}} \omega_{s_k,\beta_l}] \tag{3.10}$$

Comme dans [47], si le pire temps de transmission du message contenant le signal $s_i$ dépasse le pire temps de blocage de ce signal, alors ce dernier est remplacé par le pire temps de transmission de son message. Ceci est dû au fait que nous nous situons dans un cadre de systèmes à échéances contraintes. Par conséquent, la formule (3.10) devient :

$$B_{s_i} = \max \left[ \max_{\substack{s_j \in \Phi: \\ \pi_{s_i} > \pi_{s_j} \\ \Omega_{s_i} = \Omega_{s_j} \\ \mu_{s_i} \neq \mu_{s_j}}} [\omega_{s_j,\beta_l} + \sum_{\substack{s_k \in \{\Phi \setminus s_j\}: \\ \mu_{s_j} = \mu_{s_k}}} \omega_{s_k,\beta_l}]; \sum_{\substack{s_j \in \Phi: \\ \mu_{s_j} = \mu_{s_i}}} \omega_{s_j,\beta_l} \right] \tag{3.11}$$

6. **Amélioration et raffinement des temps de complétion :** le calcul des temps de réponse présenté ci-dessus est assez pessimiste pour le cas d'un modèle d'activation à échéance

contrainte. Ceci est dû au fait qu'une fonction $f_i$ subit deux fois les interférences provenant des fonctions prioritaires $f_k$ qui la précèdent dans le même chemin ($dp_{k,i} = 1$) : une fois dans la gigue et une autre fois dans le calcul du temps de complétion. En fait, si le modèle d'activation est à échéance contrainte, alors les fonctions prioritaires qui précèdent $f_i$ dans le même chemin ne peuvent se réactiver qu'après la terminaison de $f_i$, et ne peuvent donc pas préempter $f_i$. Pour tenir compte de cette amélioration, il suffit de supprimer toute fonction $f_l$ appartenant au même chemin que $f_i$ de l'ensemble $hp(f_i)$ et, par conséquent, $f_k \in hp(f_i)$ dans les formules (3.4), (3.5) et (3.7) est remplacé par $f_k \in hp(f_i) \setminus f_l : \varrho(i) \cap \varrho(l) \neq \emptyset$. De même, lorsque les activations sont à échéance contrainte, deux signaux appartenant au même chemin ne s'activent jamais au même temps et par conséquent la formule (3.9) devient :

$$W_{s_i} = B_{s_i} + \sum_{\substack{s_k \in hp(s_i): \\ \varrho(s_i) \cap \varrho(s_k) = \emptyset}} \left\lceil \frac{W_{s_i} + J_{s_k}}{P_{s_k}} \right\rceil \omega_{s_k,\beta_j} \tag{3.12}$$

Généralement, les analyses des temps de réponse ignorent le temps d'exécution requis par l'ordonnanceur et assument un changement de contexte instantané. Cependant, le coût de changement de contexte a un impact sur la solution de déploiement puisqu'il est en relation proportionnelle avec le nombre de tâches du système. À cet effet, l'analyse des temps de réponse présentée peut être raffinée afin de prendre en considération le surcoût du changement de contexte lors du calcul des temps de complétion. Selon [160], trois blocs doivent être considérés. Premièrement, le temps d'exécution de l'ordonnanceur pour choisir la prochaine tâche à exécuter. Ensuite, le temps qu'il lui faut pour charger le contexte de la tâche choisie. Enfin, le temps nécessaire à la décharge du contexte de la tâche en cours d'exécution. Dans [161], l'effet du changement de contexte est simplement pris en compte dans l'analyse par l'ajout au WCET d'une quantité égale à $(2.cs)$, où $cs$ est le pire temps de changement de contexte. En se basant sur [160] et [161], nous ajoutons $(2.cs)$ au premier et au dernier termes de l'équation présentant le calcul de $W_{f_i}$. Le premier terme $\omega_{f_i,c_j}$ devient $(\omega_{f_i,c_j} + 2.cs)$ et $\omega_{f_k,c_j}$ dans le dernier terme devient $(\omega_{f_k,c_j} + 2.cs)$. La considération de l'amélioration et du raffinement présentés dans cette partie implique le remplacement des équations (3.4), (3.5) et (3.7), respectivement, par les équations suivantes :

$$W_{f_i} = (\omega_{f_i,c_j} + 2.cs) + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k}}} \omega_{f_k,c_j} + \sum_{\substack{f_k \in \{hp(f_i) \setminus f_l\}: \\ \varrho(i) \cap \varrho(l) \neq \emptyset}} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil (\omega_{f_k,c_j} + 2.cs) \tag{3.13}$$

$$W_{f_i} = (\omega_{f_i,c_j} + 2.cs) + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ Eo_{f_k} > Eo_{f_i}}} \omega_{f_k,c_j} + \sum_{\substack{f_k \in \{hp(f_i) \setminus f_l\}: \\ \varrho(i) \cap \varrho(l) \neq \emptyset}} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil (\omega_{f_k,c_j} + 2.cs) \tag{3.14}$$

$$W_{f_i} = (\omega_{f_i,c_j} + 2.cs) + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ \varrho(f_i) \cap \varrho(f_k) = \emptyset}} \omega_{f_k,c_j} + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ dp_{k,i} = 1}} \omega_{f_k,c_j} + \sum_{\substack{f_k \in \{hp(f_i) \setminus f_l\}: \\ \varrho(i) \cap \varrho(l) \neq \emptyset}} \left\lceil \frac{W_{f_i} + J_{f_k}}{P_{f_k}} \right\rceil (\omega_{f_k,c_j} + 2.cs) \tag{3.15}$$

### 3.3.2 Analyse des temps de réponse pour le modèle d'activation périodique

Après avoir détaillé l'analyse des temps de réponse pour le modèle d'activations pilotées par les données, nous présentons dans cette partie les différences pour le modèle d'activation périodique. La différence principale réside dans le fait que la valeur pour toutes les gigues, celles des fonctions et des signaux, devient nulle puisque dans le modèle périodique, l'exécution d'une fonction ou la transmission d'un signal dépend de l'arrivée des évènements d'horloge et non pas des données d'entrée. En effet, l'exécution de la tâche est périodique selon la plus petite période parmi les périodes de ses fonctions. À chaque activation, la tâche exécute soit un sous ensemble, soit la globalité de ses fonctions. Par conséquent, pour ce modèle d'activation le scénario "pire cas" se produit lorsque toutes les fonctions de la tâche s'activent en même temps. Cette situation correspond à l'instant d'activation de la fonction la moins fréquente. D'autre part, pour ce modèle d'activation, une synchronisation en entrée des tâches n'a pas de sens puisque celle-ci revient à exécuter la tâche uniquement à la période la moins fréquente, ce qui résulte en une perte de données d'une part et un dépassement d'échéance pour les fonctions avec des périodes plus fréquentes d'autre part. Par exemple, considérons une tâche comprenant deux fonctions $f_1$ et $f_2$ avec les périodes 10 et 20, respectivement. À l'instant 10, la fonction $f_1$ s'active et celle-ci doit s'exécuter avant l'instant 20 où il y aura lieu au même temps sa prochaine activation et l'activation de la fonction $f_2$. Par conséquent, si l'instance de $f_1$ activée à 10 attend l'activation de $f_2$, elle ne peut s'exécuter qu'à partir de l'instant 20, et donc $f_1$ dépasse son échéance qui est égale à 20. Par ailleurs, il existe deux cas de figures pour l'instant auquel la tâche fournit le résultat correspondant à l'exécution d'une fonction donnée : i) l'ordre d'exécution des fonctions de la tâche est méconnu, au pire des cas chaque fonction doit donc attendre l'exécution de toutes les autres fonctions de cette tâche et ii) l'ordre d'exécution est déterminé au préalable et donc la fonction attend uniquement l'exécution des fonctions ayant un ordre d'exécution plus élevé. Le pire temps de réponse pour une fonction $f_i$ considérant le premier cas de figure est donné par l'équation (3.16) :

$$R_{f_i} = W_{f_i} = (\omega_{f_i,c_j} + 2.cs) + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k}}} \omega_{f_k,c_j} + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i}}{P_{f_k}} \right\rceil (\omega_{f_k,c_j} + 2.cs) \tag{3.16}$$

Par rapport au deuxième cas de figure, le pire temps de réponse d'une fonction $f_i$ activée périodiquement est calculé par :

$$R_{f_i} = W_{f_i} = (\omega_{f_i,c_j} + 2.cs) + \sum_{\substack{f_k \in \{F \setminus f_i\}: \\ \tau_{f_i} = \tau_{f_k} \\ Eo_{f_k} > Eo_{f_i}}} \omega_{f_k,c_j} + \sum_{f_k \in hp(f_i)} \left\lceil \frac{W_{f_i}}{P_{f_k}} \right\rceil (\omega_{f_k,c_j} + 2.cs) \tag{3.17}$$

Comme nous pouvons le constater à partir des équations (3.16) et (3.17), l'amélioration apportée pour le modèle d'activations pilotées par les données ne concerne pas ce modèle d'activation périodique car même si l'activation est à échéance contrainte, les fonctions appartenant au même chemin peuvent s'activer en même temps. Par conséquent, les interférences en provenance des fonctions prioritaires appartenant au même chemin que la fonction en cours d'analyse, sont aussi incluses dans le temps de complétion. Par ailleurs, pour le modèle d'activation périodique, le cas où l'ordre d'exécution des fonctions d'une tâche est connu apporte toujours de meilleurs résultats pour le temps de réponse des fonctions en comparaison aux résultats obtenus par l'autre cas. Celui-ci est moins pessimiste car il ne présente aucune attente bilatérale entre les fonctions de la même tâche : un ordre d'exécution étant défini pour ces fonctions. De ce fait, dans ce travail nous considérons

uniquement le cas où l'ordre d'exécution à l'intérieur de la tâche est connu.

Dans le cas des signaux, nous considérons des transmissions non préemptives basées sur les priorités. Dans ce cas, le pire temps de réponse pour un signal $s_i$ activé périodiquement est obtenu comme suit :

$$R_{s_i} = \omega_{s_i,\beta_j} + \sum_{\substack{s_k \in \{\Phi \setminus s_i\}: \\ \mu_{s_i} = \mu_{s_k}}} \omega_{s_k,\beta_j} + W_{s_i} \; ; \; W_{s_i} = B_{s_i} + \sum_{s_k \in hp(s_i)} \left\lceil \frac{W_{s_i}}{P_{s_k}} \right\rceil \omega_{s_k,\beta_j} \tag{3.18}$$

Là aussi cela ne résulte pas en une amélioration car les signaux appartenant au même chemin peuvent s'activer en même temps.

### 3.3.3 Synthèse

Avant de présenter les solutions de déploiement proposées, nous résumons dans le tableau 3.3 le pseudo code d'une tâche pour chaque modèle d'activation. De plus, pour le cas d'activations pilotées par les données, le code de la tâche est spécifié pour chaque sémantique de synchronisation. La tâche considérée est la tâche $\tau_4$ du modèle de système présenté dans la figure 3.1. La fonction "AttendreTous" indique que la tâche $\tau_4$ doit attendre tous les signaux spécifiés en paramètre avant de commencer son exécution. La fonction "AttendreN'importeLequel" est implémentée de telle sorte que $\tau_4$ attend n'importe quel signal parmi les signaux précisés en paramètre pour commencer son exécution. Lorsque $\tau_4$ est supposée être périodique, elle s'active à la plus petite période parmi les périodes de ses fonctions ($f_6$ et $f_7$), puis en fonction du numéro de l'activation elle décide d'exécuter soit $f_6$ et $f_7$, soit uniquement la fonction ayant la plus grande période d'activation. Cette façon d'implémenter une tâche périodique exige que les fonctions au sein de la tâche aient des périodes harmoniques.

| **Activation pilotée par les données** | | | **Activations périodiques** |
|---|---|---|---|
| Sémantique 'Input N-of-N / output N-of-N' | Sémantique 'Input N-of-N / output 1-of-N' | Sémantique 'Input 1-of-N / output 1-of-N' | |
| AttendreTous($s_3$,$s_4$)<br>Exécuter($f_6$)<br>Exécuter($f_7$)<br>Envoyer($s_6$,$s_7$) | AttendreTous($s_3$,$s_4$)<br>Si ($Eo_{f_6} > Eo_{f_7}$)<br>Exécuter($f_6$)<br>Envoyer($s_6$)<br>Exécuter($f_7$)<br>Envoyer($s_7$)<br>Sinon<br>Exécuter($f_7$)<br>Envoyer($s_7$)<br>Exécuter($f_6$)<br>Envoyer($s_6$) | AttendreN'importe-Lequel($s_3$,$s_4$)<br>Si (recevoir($s_3$))<br>Exécuter($f_6$)<br>Envoyer($s_6$)<br>Si (recevoir($s_4$))<br>Exécuter($f_7$)<br>Envoyer($s_7$) | $i = 0$<br>$P_{\tau_4} = \min[P_{f_6}, P_{f_7}]$<br>$Boucle$<br>Si $(i * P_{\tau_4})$ mod $P_{f_6} = 0$<br>exécuter ($f_6$)<br>envoyer ($s_6$)<br>Si $(i * P_{\tau_4})$ mod $P_{f_7} = 0$<br>exécuter ($f_7$)<br>envoyer ($s_7$)<br>Endormir ($P_{\tau_4}$)<br>i++ |

Tableau 3.3 – Le pseudo code de la tâche $\tau_4$ de la figure 3.1 considérant les différents modèles d'activation

Dans le cas d'activations pilotées par les données, une situation particulière se produit lorsqu'une tâche est activée à l'arrivée des évènements externes. Prenons comme exemple la tâche $\tau_1$ de la figure 3.1. Si celle-ci exécute la fonction "AttendreTous" ou la fonction

"AttendreN'importeLequel", elle se met en attente à l'infini car l'évènement attendu n'est pas envoyé par une fonction mais plutôt déclenché périodiquement par une horloge. L'étude comparative du code source de la tâche $\tau_1$ pour les différentes sémantiques de synchronisation considérées est donnée par le tableau 3.4. Pour les deux premières sémantiques, $\tau_1$ est activée périodiquement à la période la moins fréquente parmi les périodes de ses fonctions. Pour la dernière sémantique, le code de la tâche est identique à celui apporté dans le cas d'activation périodique. L'implémentation pour cette dernière sémantique nécessite que les périodes des fonctions de la tâche soient harmoniques.

| **Sémantique 'Input N-of-N / output N-of-N'** | **Sémantique 'Input N-of-N / output 1-of-N'** | **Sémantique 'Input 1-of-N / output 1-of-N'** |
|---|---|---|
| $P_{\tau_1} = \max[P_1, P_2]$<br>$Boucle$<br>Exécuter($f_1$)<br>Exécuter($f_4$)<br>Envoyer($s_1$)<br>Envoyer($s_3$,$s_4$)<br>Endormir ($P_{\tau_1}$) | $P_{\tau_1} = \max[P_1, P_2]$<br>$Boucle$<br>Si ($Eo_{f_1} > Eo_{f_4}$)<br>Exécuter($f_1$)<br>Envoyer($s_1$)<br>Exécuter($f_4$)<br>Envoyer($s_3$)<br>Envoyer($s_4$)<br>Sinon<br>Exécuter($f_4$)<br>Envoyer($s_3$)<br>Envoyer($s_4$)<br>Exécuter($f_1$)<br>Envoyer($s_1$)<br>Endormir ($P_{\tau_1}$) | $i = 0$<br>$P_{\tau_1} = \min[P_1, P_2]$<br>$Boucle$<br>Si $(i * P_{\tau_1})$ mod $P_1 = 0$<br>exécuter ($f_1$)<br>envoyer ($s_1$)<br>Si $(i * P_{\tau_1})$ mod $P_2 = 0$<br>exécuter ($f_4$)<br>envoyer ($s_3$)<br>envoyer ($s_4$)<br>Endormir ($P_{\tau_1}$)<br>i++ |

Tableau 3.4 – Le pseudo code de la tâche $\tau_1$ de la figure 3.1 considérant les différentes sémantiques de synchronisation pour le cas d'activations pilotées par les données

En conclusion, dans le but d'aborder le problème de déploiement, nous avons été amenés à utiliser une analyse des temps de réponse permettant de valider et d'évaluer différentes solutions pour ce problème. Cependant, les analyses existantes ne peuvent pas être appliquées directement à notre modèle de systèmes à cause des raisons citées dans la partie 2.3.3. Nous avons donc proposé d'étendre et d'adapter les techniques proposées dans la littérature. Nous avons pris en compte deux différents modèles d'activation pour le graphe fonctionnel, à savoir les activations pilotées par les données et les activations périodiques. Dans le cas d'activations pilotées par les données, nous avons montré que l'analyse dépend de la sémantique de synchronisation considérée pour les tâches. Par ailleurs, nous avons formulé et intégré l'effet du changement de contexte dans le calcul des pires temps de réponse.

## 3.4 Approches d'optimisation pour un déploiement automatisé

Cette partie est consacrée tout d'abord à la définition du problème de déploiement représentant le problème principal abordé par la présente thèse (voir la partie 3.4.1). Puis, dans la partie 3.4.2, nous décrivons une première solution de déploiement qui consiste en une formulation intégrale du problème en un programme linéaire mixte en nombres entiers. Dans la dernière partie 3.4.3, nous exposons une alternative permettant de répondre à la complexité du problème de déploiement, via une approche en deux étapes.

### 3.4.1 Présentation du problème de déploiement

Dans ce travail, nous nous intéressons à la recherche d'une association de l'architecture fonctionnelle au modèle de la plateforme d'exécution pour obtenir un modèle de déploiement (Voir la figure 3.9). Le problème qui consiste à trouver cette association est appelé le problème PPO. Ce dernier est composé de trois sous problèmes, à savoir le *Placement*, le *Partitionnement* et l'*Ordonnancement*. Dorénavant le problème de déploiement ou le problème $PPO$ font référence à la même chose.

- **Le placement :** consiste à attribuer chaque fonction du système à un processeur puis à trouver un bus de communication pour les signaux représentant une communication inter-processeurs.
- **Le partitionnement :** consiste à répartir les fonctions du système sur des tâches et les signaux inter-processeurs sur des messages. En d'autres termes, le partitionnement repose sur le choix des fonctions à exécuter par la même tâche et celles à exécuter par des tâches différentes. Il en est de même pour le choix des signaux inter-processeurs à transmettre par le même message ou par différents messages.
- **L'ordonnancement :** est le problème de trouver un ordre d'exécution parmi les tâches d'un même processeur et un ordre de transmission parmi les messages appartenant au même bus de communication. Dans la thèse, ceci consiste à affecter un niveau de priorité à chaque tâche et à chaque message.

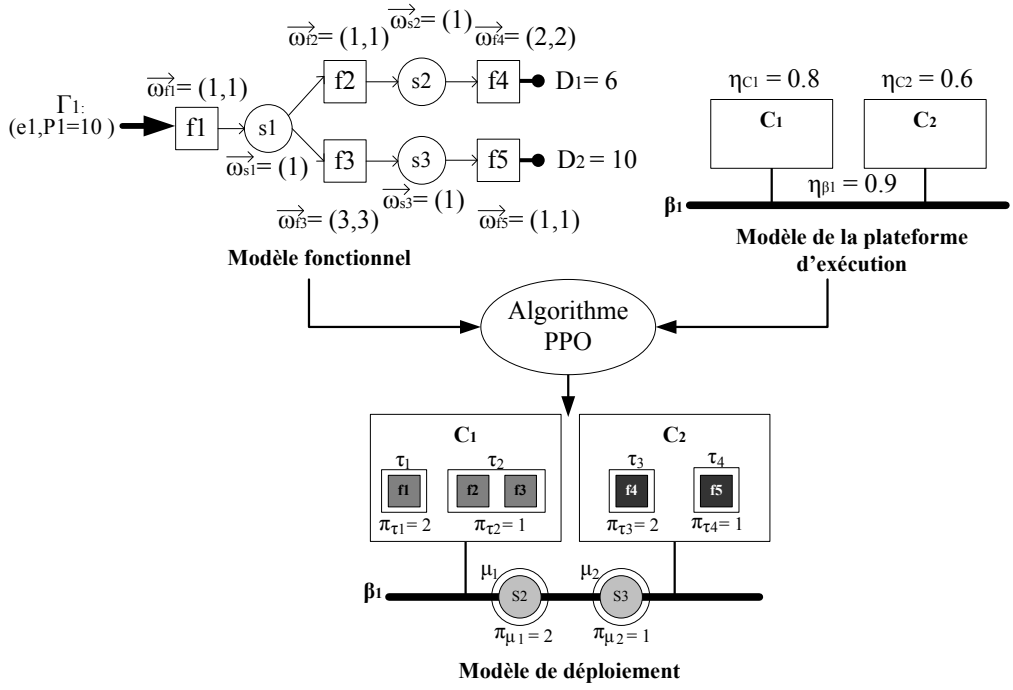



FIGURE 3.9 – Exemple d'un modèle de déploiement obtenue par la résolution du problème PPO

Dans le modèle de déploiement, un ensemble de fonctions est associé à une tâche liée à un processeur. À partir de ce choix, le WCET des fonctions est connu. De plus, la priorité d'une tâche est définie comme la priorité associée à ses fonctions (toutes les fonctions regroupées dans une tâche ont le même niveau de priorité). Veuillez noter qu'étant donné qu'une tâche peut exécuter des fonctions ayant différentes périodes, le temps d'exécution de la tâche varie d'une exécution à une autre. Cependant, comme déjà dit précédemment, pour l'analyse des temps de réponse nous considérons le plus

grand temps d'exécution de la tâche qui correspond au cas d'activations simultanées de toutes les fonctions. Il en est de même pour les messages.

Après la présentation du problème principal de cette thèse qui est le problème PPO, nous allons souligner un problème secondaire qui se manifeste dans le cas du modèle d'activation périodique. Il est évident que la manière dont le placement et le partitionnement des fonctions sont arrangés influence le nombre de ressources partagées correspondantes aux communications inter-tâches, alors il devient nécessaire de protéger les ressources partagées, d'où le problème d'attribution d'un mécanisme de protection adéquat.

- **L'affectation du mécanisme de protection :** les mécanismes de protection sont utilisés pour garantir la cohérence des données partagées entre les tâches. Le problème du choix d'un mécanisme de protection réside dans le fait que celui-ci peut avoir un impact sur le temps de réponse des fonctions ou sur l'utilisation de mémoire des processeurs. Dans ce travail, nous considérons deux mécanismes permettant d'assurer la cohérence des données : les blocs "Rate Transition" et les verrous sémaphore [155].

  – ***Protection par blocs "Rate Transition" :*** l'implémentation de ce mécanisme consiste en l'ajout de zones mémoires supplémentaires pour les buffers de communication. Cette mémoire est dédiée à un ensemble de variables supplémentaires représentant les répliques des variables de communications. Le mécanisme demande aussi un surcoût par rapport au temps mais qui est généralement inclus dans le temps d'exécution des fonctions. Le surcoût global de mémoire ($\delta_{c_j}$) pour un processeur donné $c_j$ est calculé selon l'équation (3.19).

$$\delta_{c_j} = \sum_{s_i \in \Phi'} \delta'_{\zeta_{s_i}} \tag{3.19}$$

Où, $\Phi' \subset \Phi$ est l'ensemble des signaux inter-tâches protégés par blocs "Rate Transition" et qui se trouvent au sein du processeur $c_j$ et $\delta'_{\zeta_{s_i}}$ est le surcoût de mémoire causé par le mécanisme de protection vis-à-vis de la ressource partagée $\zeta_{s_i}$. D'après [156], ce dernier est calculé par l'équation (3.20).

$$\delta'_{\zeta_{s_i}} = DS(s_i) \cdot n_{s_i} \tag{3.20}$$

Où, $n_{s_i}$ représente le nombre de répliques nécessaires au mécanisme "Rate Transition" pour protéger l'accès à $\zeta_{s_i}$. Pour calculer ce dernier terme, nous définissons $rec(s_i)'$ comme étant l'ensemble des fonctions réceptrices de $s_i$ ayant une priorité inférieure à celle de sa fonction émettrice. Comme le montre l'équation (3.21), le nombre de répliques de $s_i$ dépend de la priorité de sa fonction émettrice et de ses fonctions réceptrices.

$$n_{s_i} = \begin{cases} \sharp rec(s_i)' + 1 \text{ si } \forall f_j \in rec(s_i) : \pi_{f_j} < \pi_{snd(s_i)} \\ \sharp rec(s_i)' + 2 \text{ sinon} \end{cases} \tag{3.21}$$

Similairement à [162, 163], toutes les fonctions réceptrices ayant une priorité supérieure à celle de la fonction émettrice utilisent une seule zone mémoire et toutes les autres fonctions réceptrices ont besoin, dans le pire des cas, d'un total de $\sharp rec(s_i)' + 1$ zones. Alors s'il existe au moins une seule fonction réceptrice qui n'est pas dans $rec(s_i)'$, le nombre de zone mémoires est de $\sharp rec(s_i)' + 2$. Veuillez noter

que la formule (3.20) est une simplification de ce qui est proposé dans [156] puisque nous ne considérons pas des seuils de préemption.

– ***Protection par verrous sémaphores :*** pour ce mécanisme, l'accès aux variables de communication est protégé par les opérations verrouiller/déverrouiller. Ce mécanisme a un impact négligeable sur la mémoire puisqu'il ne nécessite pas une réplication de variables de communication. En revanche, son impact sur le temps est considérable. De ce point de vue, l'utilisation des verrous sémaphores résulte en un temps de blocage pour les fonctions. Celui-ci est dû au fait qu'une fonction moins prioritaire peut accéder en première à une ressource partagée et verrouiller par la suite l'accès à cette même ressource pour n'importe quelle autre fonction même si celle-ci est plus prioritaire. Dans cette thèse, le verrouillage des sémaphores est réalisé via le protocole à priorité plafond (PCP) [45]. Le temps de blocage $B_{f_i}$ d'une fonction $f_i$ avec PCP est déterminé par la plus longue section critique des fonctions moins prioritaires, partageant les mêmes ressources avec des priorités plafond supérieures ou égales (Voir la formule (3.22)). Le terme de blocage $B_{f_i}$ est ensuite considéré dans le calcul du WCRT de la fonction $f_i$, il est ainsi ajouté aux formules (3.16) et (3.17).

$$B_{f_i} = \max_{\substack{s_i \in \Phi", f_j \in F: \\ \pi_{f_i} \leq PC(\zeta_{s_i}) \\ f_i \in hp(f_j)}} \omega'_{f_j,\zeta_{s_i}} \tag{3.22}$$

Où, $\Phi" \subset \Phi$ est l'ensemble des signaux inter-tâches au sein du processeur de $f_i$, $PC(\zeta_{s_i})$ est la priorité plafond de la ressource partagée $\zeta_{s_i}$ et $\omega'_{f_j,\zeta_{s_i}}$ représente le pire temps d'exécution de la fonction $f_j$ dans la section critique utilisée pour accéder à la ressource partagée $\zeta_{s_i}$.

Par la suite, nous nous appuyons sur un simple exemple (cf. figure 3.9) pour illustrer l'exploration architecturale lors du déploiement. Le système considéré est constitué de 5 fonctions échangeant 3 signaux et d'une plateforme composée de 2 processeurs communiquant via un seul bus de communication. Pour ce système de petite taille, il existe énormément d'alternatives pour le placement, le partitionnement et l'ordonnancement. Un sous ensemble de ces solutions est donné dans la figure 3.10. De plus, si les fonctions sont activées périodiquement par un signal d'horloge alors il est nécessaire de déterminer le mécanisme de protection de données comme c'est le cas pour les fonctions $f_1$ et $f_2$ qui partagent la donnée du signal $s_1$ dans le cadre de la troisième solution de déploiement présentée. La question est non seulement de trouver une solution au problème $PPO$ mais aussi de considérer la sémantique du modèle fonctionnel, les limites de la plateforme d'exécution, les besoins de l'utilisateur et les exigences du système pour aboutir à un déploiement valide. Chaque choix effectué a un impact sur les performances du système conçu. Par exemple, nous constatons que la deuxième et la quatrième solution de la figure 3.10 sont meilleures en termes du nombre de processeurs utilisés que la première et la troisième solution. Néanmoins, la deuxième solution n'est pas valide puisqu'elle entraîne un dépassement de la capacité maximale pour le deuxième processeur.

***En définitive dans ce travail, en plus des problèmes de placement, de partitionnement et d'ordonnancement, nous nous intéressons aussi à la question du choix du mécanisme de protection pour les ressources partagées dans le cas d'activations périodiques. Par ailleurs, notre objectif est de fournir des décisions valides, c'est-à-dire des décisions qui respectent l'ensemble des contraintes auxquelles le déploiement est soumis.***

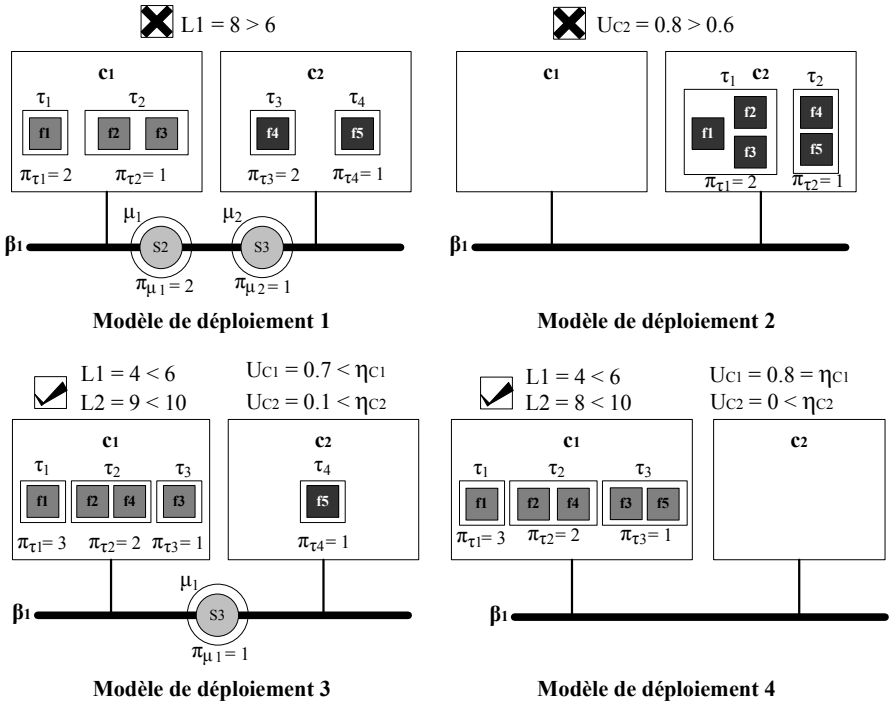



FIGURE 3.10 – Différentes solutions de PPO pour le système de la figure 3.9

Dans les parties suivantes, nous décrivons les solutions apportées à ce problème et nous discutons au fur et à mesure de l'ensemble des contraintes. Nos solutions consistent au développement de méthodologies basées sur la programmation mathématique.

### 3.4.2 Approche intégrale : formulation en PLMNE

Dans ce travail de thèse, la première solution apportée au problème de déploiement correspond à une formulation intégrale dans le cadre général de la programmation linéaire mixte en nombres entiers (PLMNE), où le problème est représenté par des paramètres, des variables de décision, et des contraintes sur les paramètres et sur les variables de décision. Une fonction objectif définie sur le même ensemble de variables, caractérise la solution optimale. Cette formulation se prête à un traitement automatique par le solveur CPLEX [164]. Dans la suite de cette partie, la formulation PLMNE apportée au problème de déploiement est décomposée selon les sous problèmes à modéliser.

La programmation linéaire nous a amené à établir notre hypothèse par rapport au modèle d'activation à échéance contrainte. En effet, si nous considérons un cas plus général où le modèle d'activation est à échéances arbitraires, nous avons besoin de prendre en compte toutes les instances de la fonction ou du signal activées dans la période d'activité pour calculer le pire temps de réponse de cette fonction ou de ce signal. Cependant, la taille de la plus grande période d'activité, et par conséquent le nombre d'instances à considérer dans le calcul de WCRT, n'est à priori pas connue ce qui implique une adaptation impossible avec la formulation PLMNE. Notre hypothèse par rapport aux échéances contraintes ne demande donc pas la considération de plusieurs activations lors du calcul des WCRTs puisque la première instance suffit pour aboutir au scénario "pire cas". En effet, cette restriction peut être soulevée en appliquant les bornes supérieures apportées par [165] au temps de réponse. Néanmoins, cette solution conduit à des temps de réponse pessimistes.

Nous présentons maintenant l'ensemble des formulations PLMNE pour l'ensemble des contraintes de déploiement. La difficulté reste dans l'expression à l'aide d'équations linéaires des éléments décrits dans le contexte et l'état de l'art.

#### 3.4.2.1 Placement dans le cas d'activations pilotées par les données et d'activations périodiques

La formulation PLMNE pour la phase de placement est indépendante du modèle d'activation considéré pour l'architecture fonctionnelle. Cette partie est dédiée à la présentation des contraintes de placement pour les fonctions et les signaux ainsi qu'à la formulation PLMNE associée.

1. **Contraintes de placement :** les contraintes liées à la phase de placement se classent en deux parties. La première partie concerne *l'utilisation de ressources*, soit le respect de la capacité d'utilisation maximale des processeurs et des bus de communication. Nous rappelons que l'utilisation de ressources pour les processeurs est calculée par l'équation de droite et celle pour les bus de communication par l'équation gauche dans (3.23).

$$U_{c_i} = \sum_{f_i \mapsto c_i} \frac{\omega_{f_i,c_i}}{P_{f_i}} \;\; ; \;\; U_{\beta_i} = \sum_{s_i \mapsto \beta_i} \frac{\omega_{s_i,\beta_i}}{P_{s_i}} \tag{3.23}$$

   La relation $\mapsto$ indique que le terme de gauche (fonction/signal) est placé sur le processeur/bus présenté par le terme de droite. La seconde partie est liée à l'allocation et elle inclue les trois points suivants :

   - ***Les allocations exclusives :*** une allocation exclusive impose le placement de chaque fonction sur exactement un processeur et le placement de chaque signal sur un bus de communication au plus.
   - ***Les allocations fixes :*** une allocation fixe impose le fait qu'une fonction doit être placée sur un processeur particulier. Par exemple, une fonction qui est responsable de la collecte de données en provenance d'un capteur doit être forcément placée sur le processeur lié à ce capteur.
   - ***Les allocations sur le réseau :*** une telle allocation exige qu'un signal transmis entre les tâches allouées sur différents processeurs doit être obligatoirement transmis sur un bus de communication. De plus, afin de considérer la topologie du réseau, ce bus doit impérativement relier les processeurs. Par exemple, si les deux processeurs $c_1$ et $c_2$ ne sont pas connectés alors les deux fonctions communicantes $f_1$ et $f_2$ ne peuvent pas être placées respectivement sur $c_1$ et $c_2$.

2. **Placement des fonctions sur les processeurs :** le placement des fonctions sur les processeurs est exprimé par la variable binaire $A_{i,j}$ de telle sorte que :

$$A_{i,j} = \begin{cases} 1 \text{ Si la function } f_i \text{ est mise sur le processeur } C_j \\ 0 \text{ Sinon} \end{cases}$$

   Une allocation exclusive des fonctions est exprimée par l'équation (3.24). Elle implique que chaque fonction peut être placée sur un et un seul processeur. Le paramètre $C(i)$ dans (3.24) détermine l'ensemble des processeurs candidats pour la fonction $f_i$. Il est utilisé pour exprimer les contraintes d'allocations fixes.

$$\sum_{j \in C(i)} A_{i,j} = 1; \sum_{j \in C \setminus C(i)} A_{i,j} = 0 \tag{3.24}$$

Afin d'exprimer la relation entre le placement des différentes fonctions, nous définissons la variable binaire $X_{i,j,k}$ tel que :

$$X_{i,j,k} = \begin{cases} 1 & \text{Si les fonctions } f_i \text{ et } f_j \text{ sont mises sur le même processeur } C_k \\ 0 & \text{Sinon} \end{cases}$$

L'équation (3.25) permet de définir $X_{i,j,k}$ pour le cas où les fonctions $f_i$ et $f_j$ sont placées sur le même processeur $C_k$. Elle contraint donc la variable $X_{i,j,k}$ à 1. Cependant, lorsque $f_i$ et $f_j$ sont sur des processeurs différents, alors (3.25) mettra $X_{i,j,k}$ à 0.

$$0 \leq A_{i,k} + A_{j,k} - (2 \cdot X_{i,j,k}) \leq 1 \tag{3.25}$$

L'équation (3.26) est donnée pour exprimer les contraintes d'utilisation de ressources. Elle permet de garantir le respect de la capacité d'utilisation maximale pour chaque processeur.

$$\forall j \in C : \sum_{i \in F} A_{i,j} \left( \frac{\omega_{i,j}}{P_i} \right) \leq \eta_j \tag{3.26}$$

3. **Placement des signaux sur les bus de communication :** à la suite du placement des fonctions, un signal est placé sur un bus de communication, si et seulement s'il représente une communication entre des fonctions distantes. Dans le cas contraire, il n'est placé sur aucun bus de communication. La variable binaire $AS_{i,j}$ définie ci-après, exprime la relation de placement entre le signal $s_i$ et le bus de communication $\beta_j$.

$$AS_{i,j} = \begin{cases} 1 & \text{Si le signal } s_i \text{ est transmis sur le bus } \beta_j \\ 0 & \text{Sinon} \end{cases}$$

Une autre variable binaire $G_i$ est définie pour indiquer la nature de la communication que représente le signal $s_i$ tel que :

$$G_i = \begin{cases} 1 & \text{Si } s_i \text{ est transmis inter-processeurs} \\ 0 & \text{Si } s_i \text{ est transmis localement à un processeur} \end{cases}$$

L'équation (3.27) force chaque signal à être placé sur un seul bus de communication au plus. Par conséquent, les signaux transmis à distance sont placés exactement sur un seul bus et les signaux locaux ne sont placés sur aucun bus.

$$\sum_{j \in \beta} AS_{i,j} = G_i \tag{3.27}$$

La définition de la variable $G_i$ est donnée par l'équation (3.28). Un signal $s_i$ est placé sur un bus si et seulement si sa fonction émettrice est exécutée sur un processeur différent de celui de l'ensemble de ses fonctions réceptrices.

$$\forall j \in rec(i) : 1 - \sum_{k \in C} X_{snd(i),j,k} = G_i \tag{3.28}$$

Similairement aux fonctions, la relation entre le placement des différents signaux est

exprimée par la variable binaire $XS_{i,j,k}$ de telle sorte que :

$$XS_{i,j,k} = \begin{cases} 1 & \text{Si les signaux } s_i \text{ et } s_j \text{ sont transmis sur le même bus } \beta_k \\ 0 & \text{Sinon} \end{cases}$$

De la même façon que $X_{i,j,k}$, $XS_{i,j,k}$ est définie par l'équation (3.29).

$$0 \leq AS_{i,k} + AS_{j,k} - (2 \cdot XS_{i,j,k}) \leq 1 \tag{3.29}$$

Le placement des signaux doit aussi prendre en compte la topologie du réseau de communication puisque les processeurs sur lesquels résident des fonctions communicantes doivent être connectés. Le paramètre $PB(i)$ représente l'ensemble des processeurs communicants par le biais du bus $\beta_i$. L'équation (3.30) autorise le placement de $s_i$ sur le bus de communication $\beta_b$ si et seulement si sa fonction émettrice et toutes ses fonctions réceptrices sont sur des processeurs liés à travers le bus $\beta_b$.

$$\forall j \in rec(i) : 0 \leq G_i + \sum_{k \in PB(b)} A_{snd(i),k} + \sum_{k \in PB(b)} A_{j,k} - 3 \cdot AS_{i,b} \leq 3 \tag{3.30}$$

La contrainte d'utilisation de ressources pour les bus de communication est exprimée de la même manière que celle pour les processeurs (voir la formule (3.31)).

$$\forall j \in \beta : \sum_{i \in \Phi} AS_{i,j} \left( \frac{\omega_{i,j}}{P_i} \right) \leq \eta_j \tag{3.31}$$

#### 3.4.2.2 Partitionnement et ordonnancement dans le cas d'activations pilotées par les données

Dans cette partie, nous discutons les contraintes liées aux phases de partitionnement et d'ordonnancement pour le cas d'activations pilotées par les données. Nous présentons également la formulation PLMNE appropriée à chaque phase.

1. **Contraintes de partitionnement :** les contraintes de partitionnement comprennent les contraintes de *partitionnement fonctionnel*, de *taux harmoniques* et celles de *communication sur le réseau*.

   - ***Partitionnement fonctionnel :*** celles-ci concernent le partitionnement de chaque fonction en exactement une seule tâche et chaque signal en un seul message au plus.
   - ***Taux harmoniques :*** celles-ci consistent à empêcher le partitionnement de deux fonctions ayant des périodes d'activations non harmoniques sur la même tâche. Ces contraintes sont liées à la phase d'implémentation. Elles sont dues au fait que les tâches activées par des évènements externes selon la sémantique 'Input 1-of-N / output 1-of-N' sont implémentées comme des tâches périodiques ayant une seule période d'activation.
   - ***Communication sur le réseau :*** celles-ci interdisent le partitionnement des signaux sur le même message lorsqu'ils proviennent de différents processeurs c'est-à-dire que deux signaux transmis par le même message doivent avoir des fonctions émettrices co-allouées sur le même processeur. Ces contraintes sont dues logiquement au fait qu'un message n'est construit qu'à partir des signaux appartenant à la même file d'attente et donc en provenance du même processeur puisque chaque file d'attente est propre à un processeur.

2. **Contraintes d'ordonnancement :** les contraintes par rapport à la phase d'ordonnancement concernent d'une part, les contraintes *d'ordre local total* et d'autre part les contraintes *d'ordre fonctionnel*.

   – ***Ordre local total :*** ces contraintes sont liées à la définition apportée pour une tâche et pour un message. Elles consistent à imposer un ordre de priorité total parmi toutes les tâches/ tous les messages appartenant au même processeur/bus de communication. Toutefois, au sein de chaque processeur/bus, toutes les tâches/tous les messages possèdent un ordre de priorité différent.
   – ***Ordre fonctionnel :*** dans le contexte d'échéances contraintes, les contraintes d'ordre fonctionnel consistent à obliger la conservation des flots de données de bout-en-bout. Plus précisément, lorsque l'exécution d'une fonction $f_j$ dépend du résultat d'exécution de la fonction $f_i$, alors la fonction $f_j$ ne peut pas s'exécuter avant $f_i$. Ces contraintes sont implicitement considérées : lorsque les activations sont périodiques, elles sont assurées par la formule de calcul des latences. Si les activations sont pilotées par les données, la conservation des flots de données est garantie par la présence des gigues. L'exception réside dans le cas de la sémantique 'Input N-of-N / output 1-of-N' où la contrainte d'ordre fonctionnel concerne aussi l'ordre de séquence puisqu'à l'intérieur d'une tâche, une fonction $f_j$ dépendante d'une autre fonction $f_i$ ne peut pas s'exécuter en première. Ce dernier cas doit être pris en compte explicitement.
3. **Partitionnement et ordonnancement des fonctions :** le partitionnement et l'affectation des priorités pour les fonctions sont résolues au même temps. Dans ce travail, chaque niveau de priorité au sein d'un processeur représente une tâche. En d'autres termes, une tâche est composée d'un groupe de fonctions ayant le même ordre de priorité. Pour chaque paire de fonctions, nous définissons une variable binaire $\chi_{i,j}$ pour exprimer la relation d'ordre de priorité entre elles.

$$\psi_{i,j} = \begin{cases} 1 & \text{Si la fonction } f_i \text{ a une priorité supérieure à celle de } f_j \\ 0 & \text{Si la fonction } f_i \text{ a une priorité inférieure ou égale à celle de } f_j \end{cases}$$

Lorsque $\psi_{i,j} = 0$ et $\psi_{j,i} = 0$ alors $f_i$ et $f_j$ possèdent le même ordre de priorité, et font donc partie de la même tâche. L'équation (3.32) assure la cohérence de la définition des variables $\psi_{i,j}$ permettant de répondre à la contrainte d'ordre local total.

$$\psi_{i,j} + \psi_{j,i} \leq 1 \tag{3.32}$$

L'ensemble des contraintes suivantes est utilisé pour appliquer les propriétés de symétrie, de transitivité et d'inversion sur la relation d'ordre de priorité et pour s'assurer que chaque fonction est partitionnée sur exactement une seule tâche (contrainte de partitionnement fonctionnel).

$$\psi_{i,j} + \psi_{j,k} - 1 \leq \psi_{i,k} \;\; ; \;\; \psi_{i,j} - (\psi_{j,k} + \psi_{k,j}) \leq \psi_{i,k} \;\; ; \;\; \psi_{j,k} - (\psi_{j,i} + \psi_{i,j}) \leq \psi_{i,k}$$

$$\psi_{i,j} + \psi_{j,i} + \psi_{j,k} + \psi_{k,j} \geq \psi_{i,k} \;\; ; \;\; \psi_{i,j} + \psi_{j,i} + \psi_{j,k} + \psi_{k,j} \geq \psi_{k,i} \tag{3.33}$$

Dans le cas où la sémantique de synchronisation des tâches est de type 'Input N-of-N / output 1-of-N', nous avons besoin de déterminer l'ordre de séquence pour les exécutions des fonctions d'une tâche. Pour cela, nous définissons la variable binaire $So_{i,j}$ qui indique la relation d'ordre de séquence entre les fonctions $f_i$ et $f_j$.

$$So_{i,j} = \begin{cases} 1 & \text{Si la fonction } f_i \text{ a un ordre de séquence supérieure à celui de } f_j \\ 0 & \text{Si la fonction } f_i \text{ a un ordre de séquence inférieure à celui de } f_j \end{cases}$$

L'équation (3.34) assure un ordre de séquence total dans le sens où soit $f_i$ est exécutée avant $f_j$ ($So_{i,j} = 1$) soit l'inverse ($So_{j,i} = 1$).

$$So_{i,j} + So_{j,i} = 1 \tag{3.34}$$

L'équation (3.35) garantit les propriétés d'asymétrie et de transitivité sur la relation d'ordre de séquence.

$$So_{i,j} + So_{j,k} - 1 \leq So_{i,k} \tag{3.35}$$

La contrainte d'ordre fonctionnel est exprimée par l'équation (3.36). Celle-ci empêche l'attribution à la fonction $f_j$ d'un ordre de séquence qui est supérieur à l'ordre de séquence affecté à $f_i$ lorsque l'exécution de $f_j$ dépend du résultat de $f_i$. $dp_{i,j} = 1$ indique que l'exécution de $f_j$ dépend de $f_i$ dans le modèle fonctionnel.

$$So_{i,j} = 1 \text{ Si } dp_{i,j} = 1 \tag{3.36}$$

Selon les contraintes de taux harmoniques, lorsque la sémantique de synchronisation considérée pour les tâches est de type 'Input 1-of-N / output 1-of-N', chaque paire de fonctions activées par des évènements externes ayant des périodes non-harmoniques doit être affectée à différentes tâches et donc avoir différents niveaux de priorité.

$\forall i, j \in F$ tel que ($P_i \geq P_j$), ($P_i$ modulo $P_j$) $\neq 0$ et (i,j sont des fonctions sources) :

$$1 = \psi_{i,j} + \psi_{j,i} \tag{3.37}$$

4. **Partitionnement et ordonnancement des signaux :** la formulation du partitionnement et d'affectation de priorités pour les signaux est équivalente à celle apportée pour les fonctions. Les formules (3.32) et (3.33) peuvent être appliquées directement aux signaux. Dans la suite, nous ajoutons la lettre "S" à chacune des variables pour différencier les variables des fonctions (sans "S") de celles des signaux (avec "S").

   Sachant qu'un signal local peut représenter soit une communication inter-tâches soit une communication intra-tâche, il est nécessaire de fixer le type de communication pour chaque signal local ayant plusieurs fonctions réceptrices. Pour cela, nous définissons certaines variables binaires supplémentaires. La variable $SnHp_{i,j,c}$, définie par la formule (3.38), indique si la fonction $f_i$ est placée sur le même processeur $c_c$ que la fonction $f_j$ et qu'elle possède une priorité inférieure à celle de $f_j$ ($SnHp_{i,j,c} = 1$). Autrement, $SnHp_{i,j,c}$ est fixée à $0$.

   $$0 \leq X_{i,j,k} + \psi_{j,i} - (2 \cdot SnHp_{i,j,c}) \leq 1 \tag{3.38}$$

   La variable $SnDp_{i,j,c}$ dans l'équation (3.39) détermine si les fonctions $f_i$ et $f_j$ sont placées sur le même processeur mais partitionnées sur différentes tâches ($SnDp_{i,j,c} = 1$) ou non ($SnDp_{i,j,c} = 0$).

   $$SnDp_{i,j,k} = SnHp_{i,j,k} + SnHp_{j,i,k} \tag{3.39}$$

   Contrairement à la variable $SnDp_{i,j,c}$, la variable $SnSp_{i,j,c}$, exprimée par la formule (3.40), indique si les fonctions $f_i$ et $f_j$ sont placées sur le même processeur $c_c$ et partitionnées sur la même tâche ($SnSp_{i,j,c} = 1$) ou non ($SnSp_{i,j,c} = 0$).

   $$SnSp_{i,j,k} + SnDp_{i,j,k} = X_{i,j,k} \tag{3.40}$$

L'équation (3.41) garantit que si la fonction émettrice d'un signal est partitionnée sur la même tâche qu'une des fonctions réceptrices, alors toutes les fonctions réceptrices doivent être partitionnées sur cette tâche. Sinon, toutes les fonctions réceptrices sont partitionnées sur des tâches différentes de celle de la fonction émettrice.

$$\forall j,k \in rec(i) : \sum_{l \in C} SnSp_{snd(i),j,l} = \sum_{l \in C} SnSp_{snd(i),k,l} \tag{3.41}$$

Enfin, la contrainte de partitionnement liée à la communication sur le réseau peut être exprimée par la formule suivante :

$$\forall i,j \in \Phi : \sum_{k \in C} X_{snd(i),snd(j),k} \geq \sum_{l \in \beta} SnSpS_{i,j,l} \tag{3.42}$$

#### 3.4.2.3 Partitionnement et ordonnancement dans le cas d'activations périodiques

La majorité des contraintes de partitionnement et d'ordonnancement apportées pour le cas d'activations pilotées par les données sont applicables au cas d'activations périodiques. Celles-ci concernent les contraintes de partitionnement fonctionnel, de communication sur le réseau, d'ordre local total et d'ordre fonctionnel. L'exception réside dans la contrainte de taux harmoniques qui est dans ce cas définie comme suit :

- ***Taux harmoniques :*** celles-ci consistent à empêcher le partitionnement de deux fonctions ou de deux signaux ayant des périodes d'activations non harmoniques respectivement sur la même tâche ou sur le même message. Ces contraintes sont dues au fait que les tâches et les messages considérés sont périodiques avec uniquement une seule période d'activation.

1. **Partitionnement et ordonnancement des fonctions :** la formulation PLMNE pour le partitionnement et l'ordonnancement des fonctions englobe les formules (3.32), (3.33), (3.34) et (3.35). De plus, elle inclut la formule de taux harmoniques définie par l'équation :

$$\forall i,j \in F : 1 = \psi_{i,j} + \psi_{j,i} \text{ Si } (P_i \geq P_j) \text{ et } (P_i \text{ modulo } P_j) \neq 0 \tag{3.43}$$

2. **Partitionnement et ordonnancement des signaux :** similairement, les formules (3.38), (3.39), (3.40), (3.41) et (3.42) sont reportées pour exprimer le partitionnement et l'ordonnancement des signaux périodiques. D'un autre côté, la contrainte de taux harmoniques pour les signaux est exprimée par la formule suivante :

$$\forall i,j \in \phi : 1 = \psi s_{i,j} + \psi s_{j,i} \text{ Si } (P_i \geq P_j) \text{ et } (P_i \text{ modulo } P_j) \neq 0 \tag{3.44}$$

#### 3.4.2.4 Affectation du mécanisme de protection aux données partagées

Cette partie présente les contraintes et la formulation PLMNE liées à l'affectation du mécanisme de protection aux données partagées. Celles-ci concernent uniquement le cas où le modèle d'activation est supposé être périodique.

1. **Contraintes de mémoire processeurs :** les contraintes liées à l'utilisation de la mémoire concernent le respect de la capacité de mémoire maximale de tous les processeurs du système. Plus précisément, la somme du surcoût de mémoire causé par le mécanisme

"Rate Transition" ne doit pas dépasser la capacité de mémoire disponible pour le processeur considéré.

2. **Choix du mécanisme de protection :** comme déjà mentionné, pour le cas d'activations périodiques, les variables de communications représentant une communication inter-tâches au sein d'un processeur, doivent être protégées par l'utilisation des verrous sémaphores ou des blocs "Rate Transition". Les équations (3.45) et (3.46) déterminent la variable binaire $Y_{\zeta_i}$ qui indique si la ressource partagée $\zeta_i$ doit être protégée ($Y_{\zeta_i} = 1$), ou non ($Y_{\zeta_i} = 0$). Lorsque toutes les fonctions réceptrices ainsi que la fonction émettrice d'un signal sont placées sur le même processeur et partitionnées dans la même tâche, alors le signal est envoyé intra-tâche et la ressource partagée associée à ce signal n'a pas besoin d'être protégée. Cette situation est formulé par l'équation (3.45).

$$Y_{\zeta_i} \leq \sum_{j \in rec(i)} \sum_{k \in C} SnDp_{snd(i),j,k} \tag{3.45}$$

À l'intérieur d'un processeur, si le signal $s_i$ représente une communication inter-tâches alors ceci implique qu'il suffit d'avoir une fonction réceptrice avec une priorité différente de celle de la fonction émettrice pour imposer la protection de la ressource partagée. Ce cas est exprimé par la formule (3.46).

$$\forall j \in rec(i) : Y_{\zeta_i} \geq \sum_{k \in C} SnDp_{snd(i),j,k} \tag{3.46}$$

D'autre part, pour spécifier le mécanisme de protection, nous définissons les deux variables binaires $mem_{\zeta_i}$ et $lock_{\zeta_i}$. La variable $mem_{\zeta_i}$ indique si le mécanisme de protection considéré pour la ressource $\zeta_i$ est le mécanisme de blocs "Rate Transition" ($mem_{\zeta_i} = 1$) ou non ($mem_{\zeta_i} = 0$). Similairement, la variable $lock_{\zeta_i}$ vaut 1 lorsque la protection de $\zeta_i$ est faite par le biais des verrous sémaphores. La formule (3.47) permet de déterminer la relation entre les variables $Y_{\zeta_i}, mem_{\zeta_i}$ et $lock_{\zeta_i}$.

$$Y_{\zeta_i} = mem_{\zeta_i} + lock_{\zeta_i} \tag{3.47}$$

3. **Utilisation de mémoire :** le surcoût de mémoire supplémentaire dû au mécanisme de protection par blocs "Rate Transition" est exprimé dans notre formulation PLMNE à l'aide des variables $V_{\zeta_i}$ et $\delta'_{\zeta_i}$. La variable binaire $V_{\zeta_i}$ dépend de la priorité de la fonction émettrice et de celle des fonctions réceptrices du signal $s_i$. La valeur nulle pour cette variable signifie que pour la ressource partagée $\zeta_i$ il n'y a pas de fonctions réceptrices ayant une priorité supérieure à celle de la fonction émettrice, la valeur de $V_{\zeta_i}$ dans ce cas est fixée par l'équation (3.48).

$$V_{\zeta_i} \leq \sum_{j \in rec(i)} \sum_{k \in C} SnHp_{snd(i),j,k} \tag{3.48}$$

Lorsqu'il existe au moins une fonction réceptrice qui a une priorité supérieure à celle de la fonction émettrice, alors $V_{\zeta_i}$ vaut 1. Cette situation est exprimée par l'équation (3.49).

$$\forall j \in rec(i) : V_{\zeta_i} \geq \sum_{k \in C} SnHp_{snd(i),j,k} \tag{3.49}$$

En conclusion, la taille de mémoire supplémentaire $\delta'_{\zeta_i}$ nécessaire à chaque ressource partagée $\zeta_i$ est calculée par l'équation (3.50), où $Z_{j,snd(i),k}$ est une variable binaire qui est égale à $SnHp_{j,snd(i),k} * mem_{\zeta_i}$. Autrement dit, $Z_{j,snd(i),k} = 1$ indique que pour la ressource partagée $\zeta_i$ qui est protégée par le mécanisme de blocs "Rate Transition"

au sein du processeur $c_k$, la fonction réceptrice $f_j$ du signal $s_i$ possède une priorité inférieure à celle de sa fonction émettrice. Par conséquent, elle est utilisée pour calculer le nombre de zones mémoire nécessaires aux fonctions réceptrices ayant une priorité inférieure à celle de la fonction émettrice. $Z'_{\zeta_i}$ est une autre variable binaire qui vaut 1 lorsque $V_{\zeta_i}$ et $mem_{\zeta_i}$ sont toutes les deux égales à 1, autrement elle est égale à 0 ($Z'_{\zeta_i} = V_{\zeta_i} * mem_{\zeta_i}$). Elle détermine le nombre de zones mémoires nécessaires à l'ensemble des fonctions réceptrices ayant une priorité supérieure à celle de la fonction émettrice. $Z_{j,snd(i),k}$ et $Z'_{\zeta_i}$ sont calculées de la même manière que $SnHp_{i,j,k}$ dans (3.38). Le dernier terme dans l'équation (3.50) indique la zone mémoire dédiée à la fonction émettrice.

$$\delta'_{\zeta_i} = DS(i) \cdot \left( \sum_{j \in rec(i)} \sum_{k \in C} Z_{j,snd(i),k} + Z'_{\zeta_i} + mem_{\zeta_i} \right) \tag{3.50}$$

Le surcoût d'utilisation de mémoire par rapport à un processeur donné $c_k$ est ainsi déduit comme dans l'équation (3.51)

$$\delta_k = \sum_{i \in \Phi} \delta_{i,k}" \tag{3.51}$$

Où, $\delta_{i,k}$" est calculée par :

$$\forall j \in rec(i) : \delta_{i,k}" = \delta'_{\zeta_i} \cdot SnDp_{snd(i),j,k} \tag{3.52}$$

Cette contrainte non linéaire est linéarisée grâce à la méthode "Big M" comme suit :

$$\delta'_{\zeta_i} - M(1 - SnDp_{snd(i),j,k}) \leq \delta_{i,k}" \ ; \ \delta_{i,k}" \leq \delta'_{\zeta_i} \ ; \ \delta_{i,k}" \leq M.SnDp_{snd(i),j,k} \tag{3.53}$$

Par conséquent, la contrainte de mémoire pour un processeur s'exprime par :

$$\delta_k + \epsilon_k \leq Mem_k \tag{3.54}$$

Où, $\epsilon_k$ est la mémoire du processeur $c_k$ déjà occupée par les variables du système.

4. **Utilisation des verrous sémaphores :** le surcoût de mémoire peut être évité en utilisant des verrous sémaphores. Néanmoins, cela se traduit par un temps de blocage pour les fonctions calculé selon le protocole PCP. Nous rappelons que $\omega'_{j,\zeta_k}$ est le pire temps d'exécution de la fonction $f_j$ dans la section critique utilisée pour accéder à la ressource $\zeta_k$. L'équation (3.55) exprime le temps de blocage de la fonction $f_i$ en se basant sur la variable binaire $cond_{i,j,\zeta_k}$. Celle-ci représente la condition nécessaire à la considération de la fonction $f_j$ dans le calcul de $B_i$.

$$B_i \geq cond_{i,j,\zeta_k} \cdot \omega'_{j,\zeta_k} \tag{3.55}$$

La définition de cette condition est donnée par l'équation (3.56). Cela consiste à combiner trois sous conditions qui sont : i) $f_j$ possède une priorité inférieure à celle de $f_i$, ii) la ressource partagée $\zeta_k$ est protégé par un verrou sémaphore et iii) la priorité plafond de $\zeta_k$ est supérieure ou égale à la priorité de $f_i$. Cette dernière sous condition est exprimée par la variable binaire $PC_{i,\zeta_k}$.

$$0 \leq PC_{i,\zeta_k} + lock_{\zeta_k} + \sum_{l \in C} SnHp_{j,i,l} - 3 \cdot cond_{i,j,\zeta_k} \leq 2 \tag{3.56}$$

Sachant que la priorité des fonctions qui partagent directement une ressource est

toujours plus petite ou égale à la priorité plafond de cette ressource, $\forall f_i \in \{snd(k) \cup rec(k)\} : PC_{i,\zeta_k} = lock_{\zeta_k}$. Pour une fonction $f_i$ qui ne partage pas la ressource $\zeta_k$, la valeur de $PC_{i,\zeta_k}$ dépend de la priorité de toutes les fonctions $f_j$ qui partagent directement cette ressource. En d'autres termes, s'il existe au moins une fonction $f_j$ qui partage la ressource $\zeta_k$ avec une priorité supérieure ou égale à celle de $f_i$ qui quant à elle ne partage pas $\zeta_k$, alors $PC_{i,\zeta_k} = 1$. Cette situation est fixée par l'équation (3.57). Dans le cas contraire où aucune fonction $f_j$ partageant la ressource $\zeta_k$ ne possède pas une priorité supérieure ou égale à celle de $f_i$, alors $PC_{i,\zeta_k} = 0$ (voir l'équation (3.58)).

$$\forall i \notin \{snd(k) \cup rec(k)\}, j \in \{snd(k) \cup rec(k)\} : PC_{i,\zeta_k} \geq \sum_{l \in C} SnSp_{i,j,l} + \sum_{l \in C} SnHp_{i,j,l} \tag{3.57}$$

$$\forall i \notin \{snd(k) \cup rec(k)\} :$$

$$PC_{i,\zeta_k} \leq \sum_{j \in \{snd(k) \cup rec(k)\}} \sum_{l \in C} SnHp_{i,j,l} + \sum_{j \in \{snd(k) \cup rec(k)\}} \sum_{l \in C} SnSp_{i,j,l} \tag{3.58}$$

#### 3.4.2.5 Formulation de l'analyse des temps de réponse pour le cas d'activations pilotées par les données

Nous présentons dans cette partie la formulation PLMNE correspondant à l'analyse des temps de réponse pour un système ayant des activations pilotées par les données. Nous exprimons également les contraintes temps réel appropriées.

1. **Contraintes temporelles :** les contraintes temporelles sont communes au sous problème de placement, de partitionnement, d'ordonnancement et de choix de mécanisme de protection pour les données partagées. Elles consistent à vérifier dans le modèle de déploiement obtenu si toutes les échéances de bout-en-bout sont bien respectées. Pour cela, il suffit de confronter le temps d'exécution de bout-en-bout de chaque chemin $\rho_i \in \gamma$ à son échéance $D_i$. Ce temps d'exécution de bout-en-bout est appelé "latence" et noté par $L_i$. Lorsque le modèle est celui d'activations pilotées par les données, la latence d'un chemin est égale au WCRT de la dernière fonction activée dans ce chemin.
2. **Le calcul des WCRTs pour les fonctions :** le WCRT d'une fonction $f_i$ ($R_i$) se définit comme la somme de sa gigue ($J_i$) et de son temps de complétion ($W_i$) (équation (3.59)). $R_i$, $W_i$ et $J_i$ sont des variables supplémentaires $\in \mathbb{R}$.

$$R_i = W_i + J_i \tag{3.59}$$

Pour calculer $W_i$ dans notre formulation PLMNE, nous commençons par la définition de la variable suivante :

$$\sigma_{i,j} = \begin{cases} n, \in \mathbb{N} & \text{Nombre de préemptions possibles de } f_j \text{ sur } f_i \\ 0 & \text{Sinon} \end{cases}$$

Le calcul de $\sigma_{i,j}$ qui est égale à $\left\lceil \frac{W_i + J_j}{P_j} \right\rceil$ s'exprime par la formule (3.60).

$$0 \leq \sigma_{i,j} - \left( \frac{W_i + J_j}{P_j} \right) < 1 \tag{3.60}$$

Ensuite, nous définissons la variable $I_{i,j,k}$ qui est égale à $\sigma_{i,j}$ lorsque $SnHp_{i,j,k} = 1$, et à $0$ dans le cas opposé. $I_{i,j,k}$ représente donc le nombre de préemptions que subit au pire des cas l'exécution de $f_i$ de la part de la fonction $f_j$ au sein du processeur $c_k$. Elle se définit comme le produit de $\sigma_{i,j}$ et $SnHp_{i,j,k}$ qui est géré en utilisant la méthode du "Big M" pour la linéarisation dans les équations (3.61).

$$\sigma_{i,j} - M(1 - SnHp_{i,j,k}) \leq I_{i,j,k} \; ; \; I_{i,j,k} \leq \sigma_{i,j} \; ; \; I_{i,j,k} \leq M.SnHp_{i,j,k} \tag{3.61}$$

Enfin, le calcul du temps de complétion pour une fonction $f_i$ dépend de la sémantique de synchronisation considérée pour les tâches. Lorsque celle-ci est de type 'Input N-of-N / output N-of-N', il s'exprime par la formule (3.62). Le paramètre $\xi_{i,j} = 1$ indique que les fonctions $f_i$ et $f_j$ sont concurrentes, c'est-à-dire qu'elles ne font pas parties d'un même chemin dans le modèle fonctionnel. Le paramètre $cs$ représente le coût du changement de contexte.

$$W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{j \in \{F \setminus i\}} \sum_{k \in C} SnSp_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1}} \sum_{k \in C} I_{i,j,k}(\omega_{j,k} + 2.cs) \tag{3.62}$$

Dans le cas de sémantique 'Input N-of-N / output 1-of-N' la formule (3.62) est remplacée par la suivante :

$$W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{j \in \{F \setminus i\}} \sum_{k \in C} SPGO_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1}} \sum_{k \in C} I_{i,j,k}(\omega_{j,k} + 2.cs) \tag{3.63}$$

Où, $SPGO_{i,j,k}$ est une variable binaire qui vaut $1$ pour indiquer que la fonction $f_j$ qui se trouve sur le même processeur et sur la même tâche que $f_i$ a un ordre de séquence plus élevé que celui de $f_i$. Elle s'exprime donc par le produit de $SnSp_{i,j,k}$ et $So_{j,i}$ qui peut être formulé de la même façon que dans l'équation (3.38). Lorsque la sémantique de synchronisation pour les tâches est de type 'Input 1-of-N / output 1-of-N', alors la formule du temps de complétion pour une fonction $f_i$ devient :

$$W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i\}: \\ \xi_{i,j}=1}} \sum_{k \in C} SnSp_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in \{F \setminus i\}: \\ dp_{j,i}=1}} \sum_{k \in C} SnSp_{i,j,k} * \omega_{j,k}$$

$$+ \sum_{\substack{j \in F: \\ \xi_{i,j}=1}} \sum_{k \in C} I_{i,j,k}(\omega_{j,k} + 2.cs) \tag{3.64}$$

Afin d'exprimer la gigue des fonctions, nous définissons la variable $\phi_i$ qui représente le plus grand WCRT des signaux d'entrée de la fonction $f_i$ (équation (3.65)).

$$\forall j \in \Phi, i \in rec(j) : \phi_i \geq R_j \tag{3.65}$$

Finalement, si la sémantique de synchronisation des tâches est de type 'Input 1-of-N / output 1-of-N', la gigue d'une fonction $f_i$ est tout simplement égale à $\phi_i$, sinon, elle est égale au plus grand WCRT de tous les signaux reçus par les fonctions au sein de la tâche exécutant $f_i$ (y compris $f_i$). Le calcul de la gigue dans ce dernier cas est représenté par la formule suivante :

$$\forall j \in F : J_i \geq \theta_{i,j} \tag{3.66}$$

Où, $\theta_{i,j} \in \mathbb{R}$ est une variable qui est égale à $\phi_j$ si les fonctions $f_i$ et $f_j$ sont implémentées

par la même tâche, sinon elle est nulle. $\theta_{i,j}$ est définie en termes de $\phi_j$ et $SnSp_{i,j,k}$ en utilisant la méthode "Big M" comme suit :

$$\forall i,j \in F : \phi_j - M\left(1 - \sum_{k \in C} SnSp_{i,j,k}\right) \leq \theta_{i,j} \; ; \; \theta_{i,j} \leq \phi_j \; ; \; \theta_{i,j} \leq M. \sum_{k \in C} SnSp_{i,j,k} \quad (3.67)$$

3. **Le calcul des WCRTs pour les signaux :** le pire temps de réponse pour un signal $s_i$ est représenté par la variable réelle $RS_i$ qui est formulée par l'équation suivante :

$$RS_i = WS_i + \sum_{k \in \beta} AS_{i,k} * \omega_{i,k} + \sum_{j \in \{\Phi \setminus i\}} \sum_{k \in \beta} SnSpS_{i,j,k} * \omega_{j,k} + JS_i \quad (3.68)$$

La variable réelle $WS_i$, représentant le temps d'attente du signal $s_i$ dans la file avant sa transmission sur le réseau, est formulée par l'équation (3.70). Cette formulation nécessite la définition des variables suivantes.

$$\sigma s_{i,j} = \begin{cases} n, \in \mathbb{N} \text{ Nombre de fois que } s_j \text{ passe devant } s_i \text{ dans la file d'attente} \\ 0 \qquad \text{Sinon} \end{cases}$$

Le calcul de $\sigma s_{i,j} = \left\lceil \frac{WS_i + JS_j}{P_j} \right\rceil$, est exprimé de la même façon que $\sigma_{i,j}$ dans la formule (3.60). La deuxième variable $IS_{i,j,k} \in \mathbb{N}$ est définie dans le but de capturer le nombre d'interférences dans la file d'attente tout en tenant compte des conditions de placement et de partitionnement. Elle représente le nombre de fois que le signal $s_j$ interfère le signal $s_i$ sur la file d'attente d'accès au bus $\beta_k$. $IS_{i,j,k}$ est égale à $\sigma s_{i,j}$ si $SnHpS_{i,j,k} = 1$, et à 0 sinon ($IS_{i,j,k} = \sigma s_{i,j} * SnHpS_{i,j,k}$). Sa formulation est identique à celle de $I_{i,j,k}$ présentée par l'équation (3.61). Une dernière variable réelle $BS_i$ est définie pour les signaux représentant une communication inter-processeurs. Elle indique le temps de blocage pour le signal $s_i$ qui est dû à la non préemptivité des messages. L'équation (3.69) permet de calculer $BS_i$, elle représente la formulation PLMNE équivalente à l'équation (3.11).

$$\forall (i,j) \in \Phi \, , k \in \beta : \begin{cases} BS_i \geq SnHpS_{j,i,k} * \omega_{j,k} + \sum_{l \in \{\Phi \setminus (i,j)\}} SnSpS_{j,l,k} * \omega_{l,k} \\ BS_i \geq \sum_{j \in \Phi} \sum_{k \in \beta} SnSpS_{i,j,k} * \omega_{j,k} \end{cases} \quad (3.69)$$

Enfin, le temps de complétion d'un signal $s_i$ peut être calculé comme dans la formule (3.70). Le paramètre $\xi S_{i,j} = 1$ indique que les signaux $s_i$ et $s_j$ sont concurrents, autrement dit, ils n'appartiennent pas au(x) même(s) chemin(s) dans le graphe fonctionnel.

$$WS_i = BS_i + \sum_{\substack{j \in \Phi : \\ \xi S_{i,j} = 1}} \sum_{k \in \beta} IS_{i,j,k} * \omega_{j,k} \quad (3.70)$$

Le calcul de la gigue d'un signal se diffère par rapport à la nature de la communication que représente ce signal. Pour les signaux locaux, nous déterminons deux variables supplémentaires $\alpha_i$ et $\int_i$ comme suit :

$$\alpha_i = \begin{cases} R_{snd(i)} \text{ Si } s_i \text{ représente une communication inter-tâches} \\ 0 \qquad \text{Sinon} \end{cases}$$

$$\int_i = \begin{cases} J_{snd(i)} & \text{Si } s_i \text{ représente une communication intra-tâche} \\ 0 & \text{Sinon} \end{cases}$$

La définition de $\alpha_i$ et $\int_i$ dans le cadre de PLMNE est exprimée respectivement par l'équation (3.71) et (3.72) en se basant sur la méthode "Big M". Une communication inter-tâches/intra-tâche est capturée par la variable $SnDp_{snd(i),j,k}/SnSp_{snd(i),j,k}$ dans l'équation (3.71)/(3.72).

$$\forall j \in rec(i) : \begin{cases} R_{snd(i)} - M\left(1 - \sum_{k \in C} SnDp_{snd(i),j,k}\right) \leq \alpha_i \\ \alpha_i \leq R_{snd(i)} \\ \alpha_i \leq M. \sum_{k \in C} SnDp_{snd(i),j,k} \end{cases} \tag{3.71}$$

$$\forall j \in rec(i) : \begin{cases} J_{snd(i)} - M\left(1 - \sum_{k \in C} SnSp_{snd(i),j,k}\right) \leq \int_i \\ \int_i \leq J_{snd(i)} \\ \int_i \leq M. \sum_{k \in C} SnSp_{snd(i),j,k} \end{cases} \tag{3.72}$$

La gigue d'un signal local est donnée par l'équation (3.73) dans laquelle soit $\alpha_i$ soit $\int_i$ est nulle. Nous pouvons noter que si le signal $s_i$ est transmis sur le réseau, alors $\alpha_i$ et $\int_i$ sont toutes les deux égales à $0$.

$$JS_i \geq \alpha_i + \int_i \tag{3.73}$$

Par ailleurs, la formulation de la gigue d'un signal représentant une communication inter-processeurs est donnée par l'équation (3.74) en se basant sur la variable $\Lambda_{i,j,k}$, définie comme suit :

$$\forall (i,j) \in \Phi : \Lambda_{i,j,k} = \begin{cases} R_{snd(j)} & \text{Si } s_i \text{ et } s_j \text{ sont transmis par le même message sur le bus } \beta_k \\ 0 & \text{Sinon} \end{cases}$$

$$JS_i \geq \Lambda_{i,j,k} \tag{3.74}$$

La formulation PLMNE de $\Lambda_{i,j,k}$ est apportée par les équations dans (3.75).

$\forall i, j \in \Phi, k \in \beta$ :

$$R_{snd(j)} - M\left(1 - SnSpS_{i,j,k}\right) \leq \Lambda_{i,j,k} \; ; \; \Lambda_{i,j,k} \leq R_{snd(j)} \; ; \; \Lambda_{i,j,k} \leq M.SnSpS_{i,j,k} \tag{3.75}$$

4. **Formulation des contraintes temporelles :** après la présentation de toutes les formulations et les définitions précédentes, nous sommes maintenant en mesure de calculer la pire latence pour chaque chemin $\rho_i \in \rho$ et ainsi exprimer les contraintes temps réel. L'équation (3.76) donne le calcul de la pire latence d'un chemin. Pour le cas d'activations pilotées par les données, celle-ci est égale au WCRT de la dernière fonction ($snk_{(i)}$) activée sur le chemin.

$$\forall i \in \gamma : L_i = R_{snk_{(i)}} \tag{3.76}$$

Les contraintes temps réel qui représentent la satisfaction des échéances de bout-en-bout sur chacun des chemins sont exprimées par l'équation (3.77).

$$\forall i \in \gamma : L_i \leq D_i \tag{3.77}$$

#### 3.4.2.6 Formulation de l'analyse des temps de réponse pour le cas d'activations périodiques

Cette partie est consacrée à la formulation PLMNE de l'analyse des temps de réponse pour le cas d'un modèle fonctionnel ayant des activations périodiques. Elle exprime les contraintes temps réel dans ce cas ainsi que le calcul des pires temps de réponse pour les fonctions et les signaux.

1. **Contraintes temporelles :** pour le cas d'un modèle d'activation périodique, la latence d'un chemin est calculée en additionnant les WCRT de toutes les fonctions et de tous les signaux représentant une communication inter-processeurs appartenant à ce chemin, ainsi que les périodes de ces signaux et la période de leur fonction réceptrice vis-à-vis de ce chemin [16] (voir l'équation 3.78). Le calcul pessimiste de la latence dans ce dernier cas est dû à la non synchronisation des horloges d'un système distribué.

$$L_i = \sum_{f_j \in \rho_i} R_{f_j} + \sum_{\substack{s_k \in \rho_i \\ s_k \ est \ inter-processeurs}} (R_{s_k} + P_{s_k} + P_{f_l}) \tag{3.78}$$

   Tel que $f_l$ est la fonction réceptrice de $s_k$ sur le chemin $\rho_i$.

2. **Le calcul des WCRTs pour les fonctions :** le WCRT pour les fonctions activées périodiquement est un cas particulier de ce qui est présenté pour le cas d'activations pilotées par les données où la gigue s'annule. Par conséquent, le temps de réponse de la fonction périodique $f_i$ est exprimé par l'équation suivante :

$$R_i = B_i + \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{j \in \{F \setminus i\}} \sum_{k \in C} SPGO_{i,j,k} * \omega_{j,k} + \sum_{j \in F} \sum_{k \in C} I_{i,j,k}(\omega_{j,k} + 2.cs) \tag{3.79}$$

   Où, $I_{i,j,k}$ est calculée par les équations exprimées dans la formule (3.61). Cette fois-ci $\sigma_{i,j}$, qui est utilisée pour calculer $I_{i,j,k}$, est égale à $\left\lceil \frac{W_i}{P_j} \right\rceil$ plutôt qu'à $\left\lceil \frac{W_i + J_j}{P_j} \right\rceil$. La formule (3.60) est ainsi remplacée par la formule (3.80).

$$0 \leq \sigma_{i,j} - \left( \frac{W_i}{P_j} \right) < 1 \tag{3.80}$$

3. **Le calcul des WCRTs pour les signaux :** si les signaux sont activés périodiquement, alors leurs gigues s'annulent dans toutes les formules permettant de calculer le WCRT. La formule PLMNE exprimant le calcul du WCRT d'un signal périodique est la suivante :

$$RS_i = WS_i + \sum_{k \in \beta} AS_{i,k} * \omega_{i,k} + \sum_{j \in \{\Phi \setminus i\}} \sum_{k \in \beta} SnSpS_{i,j,k} * \omega_{j,k} \tag{3.81}$$

Où, le temps de complétion est donné par :

$$WS_i = BS_i + \sum_{j \in \{\Phi \setminus i\}} \sum_{k \in \beta} IS_{i,j,k} * \omega_{j,k} \tag{3.82}$$

Le pire temps de blocage $BS_i$ d'un signal périodique est formulé similairement à celui des signaux activés sur l'arrivée des données (voir la formule (3.69)). Par ailleurs, dans la formule présentant le calcul de $\sigma s_{i,j}$, utilisée pour exprimer $IS_{i,j,k}$, la valeur des gigues est nulle.

4. **Formulation des contraintes temporelles :** la formulation PLMNE du calcul de la pire latence d'un chemin pour le cas d'activation périodique est donné par :

$$L_i = \sum_{j \in \rho_i} R_j + \sum_{k \in \rho_i} RS_k + (P_l + P_k)G_k \tag{3.83}$$

Tel que $f_l$ est la fonction réceptrice de $s_k$ sur le chemin $\rho_i$. Dans le cas d'un modèle d'activation périodique, en plus de la contrainte temporelle exprimée par la formule (3.77), nous avons également besoin de vérifier que chaque fonction/signal a complété son exécution avant sa prochaine activation (équation (3.84)).

$$\forall i \in F : R_i \leq P_i; \forall i \in \Phi : RS_i \leq P_i \tag{3.84}$$

Avant de présenter la formulation PLMNE des métriques d'optimisation considérées, nous introduisons le tableau 3.5 pour récapituler la formulation PLMNE intégrale proposée pour les contraintes de déploiement, et ce en considérant les différents modèles d'activation et les différentes sémantiques de synchronisation pour le cas d'activations pilotées par les données.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| **Sémantiques de synchronisation** | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| **Contraintes de placement** | (3.24),(3.25), (3.26),(3.27), (3.28),(3.29), (3.30),(3.31) | (3.24),(3.25), (3.26),(3.27), (3.28),(3.29), (3.30),(3.31) | (3.24),(3.25), (3.26),(3.27), (3.28),(3.29), (3.30),(3.31) | (3.24),(3.25), (3.26),(3.27), (3.28),(3.29), (3.30),(3.31) |
| **Contraintes de partitionnement et d'ordonnancement** | (3.32),(3.33), (3.38),(3.39), (3.40),(3.41), (3.42) | (3.32),(3.33), (3.34),(3.35), (3.36),(3.38), (3.39),(3.40), (3.41),(3.42) | (3.32),(3.33), (3.37),(3.38), (3.39),(3.40), (3.41),(3.42) | (3.32),(3.33), (3.34),(3.35), (3.38),(3.39), (3.40),(3.41), (3.42),(3.43), (3.44) |
| **Contraintes d'analyse des temps de réponse** | (3.59),(3.60), (3.61),(3.62), (3.65),(3.66), (3.67),(3.68), (3.69),(3.70), (3.71),(3.72), (3.73),(3.74), (3.75),(3.76), (3.77) | (3.59),(3.60), (3.61),(3.63), (3.65),(3.66), (3.67),(3.68), (3.69),(3.70), (3.71),(3.72), (3.73),(3.74), (3.75),(3.76), (3.77) | (3.59),(3.60), (3.61),(3.64), (3.65)*,(3.68), (3.69),(3.70), (3.71),(3.72), (3.73),(3.74), (3.75),(3.76), (3.77) | (3.61),(3.69), (3.77),(3.79), (3.80),(3.81), (3.82),(3.83), (3.84) |
| **Contraintes d'affectation du mécanisme de protection** | / | / | / | (3.45),(3.46), (3.47),(3.48), (3.49),(3.50), (3.51),(3.52), (3.53),(3.54), (3.55),(3.56), (3.57),(3.58) |

$*$ : indique que $\phi_i$ est remplacée par $J_i$

Tableau 3.5 – Récapitulatif de la formulation PLMNE intégrale pour chaque modèle d'activation

#### 3.4.2.7 Métriques d'optimisation

Une solution valide est un ensemble de décisions sur le placement, le partitionnement, l'ordonnancement et le mécanisme de protection de données partagées satisfaisant toutes les contraintes mentionnées ci-dessus. Par rapport à notre exemple (voir la figure 3.4), nous présentons sur la figure 3.10, deux solutions non valides (modèles de déploiement 1 et 2) et deux autres valides (modèles de déploiement 3 et 4). Le modèle de déploiement 1 viole la contrainte temporelle du premier chemin et le modèle de déploiement 2 viole la contrainte d'utilisation des ressources pour le processeur $c_2$. Les latences sont calculées par rapport à un modèle d'activation piloté par les données en considérant une sémantique de synchronisation de type 'Input N-of-N / output N-of-N' pour les tâches. En pratique, nous pouvons noter qu'il arrive souvent que plusieurs solutions soient valides. Dans

notre exemple, il s'agit des modèles de déploiements 3 et 4. Dans une telle situation, il est préférable de choisir la solution apportant les meilleurs bénéfices. La sélection de la meilleure solution s'appuie sur les métriques d'optimisation. Dans ce travail, nous considérons deux métriques, une première métrique liée au temps de réponse du système et l'autre à l'utilisation de mémoire, comme cela est détaillé ci-après.

La détermination des métriques d'optimisation se base sur les exigences du système. Plusieurs métriques d'optimisation sont le centre d'intérêt des systèmes temps réel critiques. L'une des plus cruciales pour ce type de système est la métrique de temps. Dans ce travail nous nous sommes intéressés principalement à l'optimisation du temps de réponse du système car les performances d'un système temps réel augmentent significativement lorsque son temps de réaction à l'environnement externe est court. Il existe différentes formulations pour exprimer cette métrique de temps [159]. Dans le cadre de cette thèse, nous avons testé deux d'entre elles [166]. La première concerne la maximisation du *"slack" minimal*. Le "slack" d'un chemin donné est défini comme la différence entre son échéance de bout-en-bout et sa latence. La maximisation du "slack" minimal implique la maximisation de la distance minimale entre l'échéance et la latence parmi tous les slacks des chemins du système. La seconde formulation représente la minimisation de la moyenne des latences de tous les chemins du système. Pour les expérimentations de ce manuscrit, nous avons choisi de minimiser la moyenne des latences de tous les chemins du système (voir la formule (3.85)). Il est à noter que l'optimisation du nombre de tâches est implicitement considérée puisqu'elle est incluse dans l'optimisation du temps, et ceci grâce à la modélisation du coût de changement de contexte qui est en relation proportionnelle avec le nombre de tâches.

$$\text{Métrique d'optimisation liée au temps} = \frac{\sum_{\rho_i \in \gamma} L_i}{\sharp\gamma} \tag{3.85}$$

Dans ce travail, nous considérons aussi la métrique d'utilisation de mémoire pour le cas d'un modèle d'activation périodique. Cette optimisation consiste à minimiser le surcoût moyen de mémoire supplémentaire causé par le mécanisme de protection de ressources partagées "Rate Transition" (voir la formule (3.86)).

$$\text{Métrique d'optimisation liée à la mémoire} = \frac{\sum_{c \in C}(\delta_c + \epsilon_c)}{\sharp C} \tag{3.86}$$

Pour le modèle d'activation périodique, il est notamment possible de considérer une optimisation multiobjectif pour laquelle l'idée est de trouver un équilibrage approprié entre le temps et la mémoire lors de la phase de placement, de partitionnement, d'ordonnancement et d'attribution du mécanisme de protection. La fonction objectif dans ce cas peut être exprimée par la formule (3.87) où le premier terme est la normalisation de la formule (3.85) et le second représente la normalisation de la formule (3.86). L'importance de chaque métrique est spécifiée par l'affectation d'un poids noté par $K$.

$$K_1 \frac{\sum_{\rho_i \in \gamma} \frac{L_i}{D_i}}{\sharp\gamma} + K_2 \frac{\sum_{c \in C} \frac{\delta_c + \epsilon_c}{Mem_c}}{\sharp C} \tag{3.87}$$

#### 3.4.2.8 Complexité théorique

La complexité d'un PLMNE en termes de temps de processeurs et de quantité de mémoire nécessaires à la résolution du programme est en réalité imprévisible. Cependant, afin de donner une idée générale sur la complexité théorique de notre formulation PLMNE en termes de la taille du problème, nous présentons le tableau 3.6 dans lequel nous mentionnons le nombre de variables et de contraintes utilisées dans chaque version de

l'approche intégrale apportée au problème $PPO$.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires $(d)$ | $\|F\|^2(4\|C\|+1)+\|\Phi\|^2(4\|\beta\|+1)+\|\Phi\|(\|\beta\|+1)+\|F\|\|C\|$ | $\|F\|^2(5\|C\|+2)+\|\Phi\|^2(4\|\beta\|+1)+\|\Phi\|(\|\beta\|+1)+\|F\|\|C\|$ | $\|F\|^2(4\|C\|+1)+\|\Phi\|^2(4\|\beta\|+1)+\|\Phi\|(\|\beta\|+1)+\|F\|\|C\|$ | $\|F\|^2(6\|C\|+2+\|\Phi\|)+\|\Phi\|^2(4\|\beta\|+1)+\|F\|\|C\|+\|\Phi\|(5+\|F\|+\|\beta\|)$ |
| #var. totales $(n)$ | $\|F\|^2(5\|C\|+3)+\|\Phi\|^2(6\|\beta\|+2)+\|\Phi\|(\|\beta\|+7)+\|F\|(\|C\|+4)+\|\gamma\|$ | $\|F\|^2(6\|C\|+4)+\|\Phi\|^2(6\|\beta\|+2)+\|\Phi\|(\|\beta\|+7)+\|F\|(\|C\|+4)+\|\gamma\|$ | $\|F\|^2(5\|C\|+2)+\|\Phi\|^2(6\|\beta\|+2)+\|\Phi\|(\|\beta\|+7)+\|F\|(\|C\|+3)+\|\gamma\|$ | $\|F\|^2(7\|C\|+3+\|\Phi\|)+\|\Phi\|^2(5\|\beta\|+2)+\|F\|(2+\|C\|)+\|\Phi\|(9+\|F\|+\|\beta\|+\|C\|)+\|\gamma\|+\|C\|$ |
| #contraintes $(m)$ | $5\|F\|^3+5\|\Phi\|^3+\|F\|^2(7\|C\|+\|\Phi\|-11)+\|\Phi\|^2(12\|\beta\|-10)+\|F\|(5+11\|\Phi\|+\|\Phi\|\|\beta\|-3\|C\|)+\|\beta\|(1-3\|\Phi\|)+\|C\|+2\|\gamma\|$ | $6\|F\|^3+5\|\Phi\|^3+\|F\|^2(8\|C\|+\|\Phi\|-10)+\|\Phi\|^2(12\|\beta\|-10)+\|F\|(5+11\|\Phi\|+\|\Phi\|\|\beta\|-3\|C\|)+\|\beta\|(1-3\|\Phi\|)+\|C\|+2\|\gamma\|$ | $5\|F\|^3+5\|\Phi\|^3+\|F\|^2(7\|C\|+\|\Phi\|-11)+\|\Phi\|^2(12\|\beta\|-10)+\|F\|(1+7\|\Phi\|+\|\Phi\|\|\beta\|-3\|C\|)+\|\beta\|(1-3\|\Phi\|)+\|C\|+2\|\gamma\|$ | $7\|F\|^3+5\|\Phi\|^3+\|F\|^2(7\|C\|+3\|\Phi\|+27)+\|\Phi\|^2(8\|\beta\|-10)+\|F\|(2\|\Phi\|-3\|C\|+\|\Phi\|\|\beta\|+4\|\Phi\|\|C\|+11)+\|\Phi\|(15-3\|\beta\|)+2\|\gamma\|+\|\beta\|+3\|C\|$ |

Tableau 3.6 – Complexité théorique de l'approche PLMNE intégrale en termes de la taille du problème

En se basant sur l'étude apportée dans le deuxième point de la partie 2.2.2.4 et sur la complexité en taille du problème qui est indiquée dans le tableau 3.6, nous présentons dans le tableau 3.7 une estimation de la complexité de notre formulation PLMNE intégrale en termes du nombre d'opérations arithmétiques. $O_{PLMNE_{Moy}}$ indique la complexité moyenne déterminée à l'aide de l'analyse empirique basée sur différents problèmes de la littérature [86], tandis que $O_{PLMNE_{Pire}}$ représente la complexité théorique considérant le pire cas.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| $O_{PLMNE_{Moy}} = O(m).O(m.n).O(d^2)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$O((\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$(\lvert F\rvert^2+\lvert\Phi\rvert^2)).$<br>$O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$O((\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$(\lvert F\rvert^2+\lvert\Phi\rvert^2)).$<br>$O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$O((\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$(\lvert F\rvert^2+\lvert\Phi\rvert^2)).$<br>$O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$O((\lvert F\rvert^3+\lvert\Phi\rvert^3).$<br>$(\lvert F\rvert^2+\lvert\Phi\rvert^2)).$<br>$O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ |
| $O_{PLMNE_{Pire}} = O(2^m).O(2^d)$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).$<br>$O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).$<br>$O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).$<br>$O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).$<br>$O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ |

Tableau 3.7 – Complexité moyenne ($O_{PLMNE_{Moy}}$) et théorique ($O_{PLMNE_{Pire}}$) pour l'approche PLMNE intégrale en termes du nombre d'opérations arithmétiques

Dans le tableau 3.7, l'ordre de grandeur utilisé pour présenter la complexité du PLMNE fait en sorte que les quatre versions de l'approche PLMNE intégrale ont la même complexité. La valeur exacte fait une différence pour ces différentes versions. Prenons à titre d'exemple un système avec $16$ fonctions, $15$ signaux, $10$ processeurs, $1$ bus de communication et $4$ chemins. Le tableau 3.8 montre d'une part, la taille du problème représentant ce système, et d'autre part, sa complexité moyenne ainsi que sa complexité théorique en termes du nombre d'opérations arithmétiques, et ce pour les différentes versions de l'approche PLMNE intégrale.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires | 11811 | 14627 | 11811 | 21327 |
| #var. totales | 15716 | 18532 | 15444 | 24849 |
| #contraintes | 56878 | 62034 | 59650 | 91680 |
| $O_{PLMNE_{Moy}}$ | $7,092e^{21}$ | $1,525e^{22}$ | $7,665e^{21}$ | $9,499e^{22}$ |
| $O_{PLMNE_{Pire}}$ | $2^{56878}.2,919e^{3555}$ | $2^{62034}.1,464e^{4403}$ | $2^{59650}.2,919e^{3555}$ | $2^{91680}.1,166e^{6420}$ |

Tableau 3.8 – Complexité moyenne et théorique de l'approche PLMNE intégrale pour un système de $16$ fonctions, $15$ signaux, $10$ processeurs, $1$ bus de communication et $4$ chemins

La consommation effective des ressources de calcul et de stockage pour l'approche de déploiement PLMNE intégrale est étudiée plus profondément dans le chapitre 4.

Enfin, afin de réduire la complexité de l'approche de déploiement et d'améliorer le passage à l'échelle, nous proposons une alternative en deux étapes permettant d'aborder des systèmes ayant une taille similaire à celle des cas d'études utilisés dans la littérature [133, 16, 142].

### 3.4.3 Déploiement en deux étapes

Étant donné que le problème de déploiement doit être pris en compte d'une façon intégrale et que la méthodologie de résolution doit être suffisamment efficace pour aborder des systèmes de la littérature, nous proposons une approche d'optimisation en deux étapes dans laquelle le problème initial est divisé en deux sous problèmes de taille plus malléable. Chaque sous problème est vu comme un problème d'optimisation linéaire en nombres entiers qui peut être résolu à l'aide d'un solveur. L'idée de cette approche en deux étapes est de trouver une solution de déploiement de bonne qualité tout en évitant une exploration exhaustive de l'espace de recherche. L'approche s'oriente donc vers une exploration intelligente de cet espace. Dans cette partie, nous détaillons le principe général de cette approche puis nous présentons la formulation PLMNE apportée à chacun de ces deux sous problèmes.

#### 3.4.3.1 Principe général

Notre approche en deux étapes, exprimée par l'algorithme de la figure 3.11 (le pseudo code est donné dans la figure 3.12), se base sur deux stratégies classiques utilisées en optimisation, à savoir : la stratégie de décomposition et la stratégie d'améliorations itératives.

Du point de vue du principe de décomposition, cela consiste à diviser le problème de déploiement ($PPO$) en deux sous problèmes résolus en cascade de telle sorte que le placement (noté par $PP$) soit traité en premier. Ainsi sont abordés en deuxième lieu le partitionnement et l'ordonnancement ($PO$). Les résultats de chaque sous problème sont ensuite combinés pour représenter la solution apportée au problème de déploiement initial.

Par rapport aux améliorations itératives, l'idée classique est de commencer par une solution donnée quelconque, de modifier ensuite des parties de cette dernière et d'évaluer enfin la nouvelle solution obtenue. Ce processus, comprenant l'étape de modification et celle d'évaluation, est répété itérativement jusqu'à l'obtention d'une solution de qualité convaincante. Cette stratégie est utilisée pour se diriger vers la solution optimale. Dans l'algorithme à deux étapes, nous considérons une amélioration itérative à deux niveaux : la boucle interne et la boucle externe.

Au niveau de la boucle interne, nous essayons d'améliorer une configuration initiale du système en appliquant itérativement une séquence d'optimisations jusqu'à un point de convergence. Le point de convergence est déterminé par le fait que deux solutions successives sont identiques. La boucle interne comprend les deux étapes, l'optimisation du placement suivie de l'optimisation du partitionnement et de l'ordonnancement. L'optimisation des deux sous problèmes est faite par rapport au temps de réponse qui est exprimé par la moyenne des latences et elle est réalisée à l'aide de la PLMNE. Initialement, une configuration quelconque pour le problème $PO$ est fournie. Celle-ci est utilisée dans la première étape de l'approche afin de trouver un placement ($PP$) optimal des fonctions et des signaux respectivement sur les processeurs et les bus de communication. À la fin de cette étape, les tâches et les messages sont implicitement placés sur les processeurs et les bus de communication. L'étape suivante est dédiée au sous problème $PO$. Elle se base sur la solution de placement retournée par l'étape précédente pour trouver une nouvelle solution

de partitionnement et d'ordonnancement pour les fonctions et les signaux. La solution obtenue à la deuxième étape est une amélioration de la solution précédente.

Ensuite, suivant le même principe, la solution pour $PO$ trouvée à la deuxième étape devient une entrée à la première étape de l'itération suivante pour une nouvelle amélioration, et ainsi de suite jusqu'à l'arrivée au point de convergence indiquant la fin de la boucle interne. En fonction de la configuration initiale choisie, la solution pour le problème $PPO$ peut être un optimum local. À travers la boucle externe, notre objectif est de s'éloigner de cet optimum local. Par conséquent, la boucle externe consiste à sélectionner aléatoirement un ensemble de configurations initiales de partitionnement et d'ordonnancement déterminées comme un point de départ pour d'éventuelles boucles internes. La boucle externe est répétée jusqu'à la satisfaction du concepteur ou l'écoulement de la durée spécifiée.

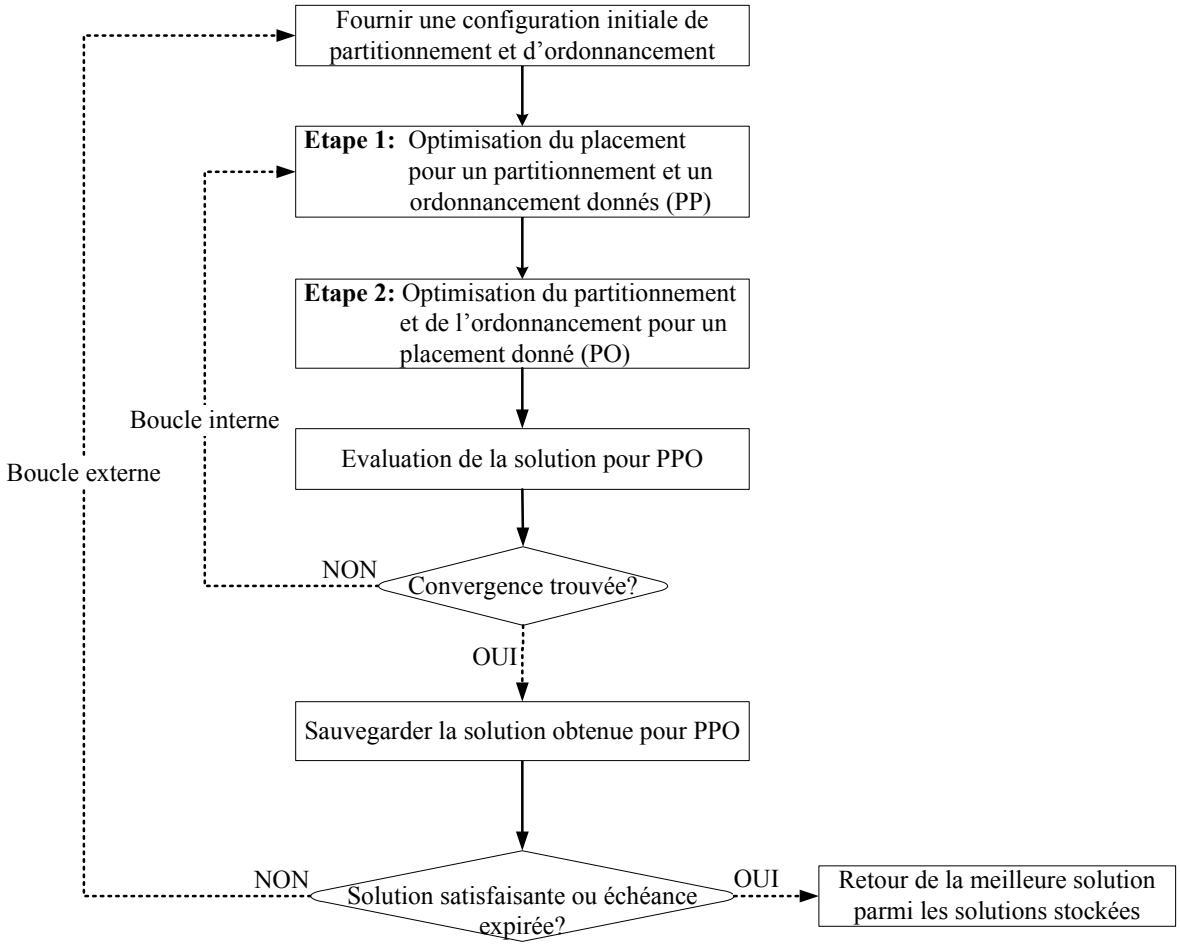



FIGURE 3.11 – Vue d'ensemble de l'approche de déploiement en deux étapes

```
BestSol = null; /* La meilleure solution de déploiement */
Tant que (! Solution satisfaisante et ! Échéance expirée)
    PO = générerPO(); /* Générer une configuration de partitionnement et d'ordonnancement  */
    OldPPO = null; /* Solution de déploiement trouvée dans l'itération précédente */
    PP = optimiser (PO); /* Optimiser le placement en se basant sur PO*/
    CurrentPPO = PO + PP; /* Solution de déploiement trouvée dans l'itération courante */
    Tant que (OldPPO ≠ CurrentPPO)
        PO = optimiser (PP); /* Optimiser le partitionnement et l'ordonnancement en se basant sur PP*/
        OldPPO = CurrentPPO;
        CurrentPPO = PO + PP;
        Si (fonctionObj(BestSol) >  fonctionObj(CurrentPPO))
                BestSol = CurrentPPO;
        Si (OldPPO = CurrentPPO)
                Quitter;
        PP = optimiser (PO);
        OldPPO = CurrentPPO;
        CurrentPPO = PO + PP;
        Si (fonctionObj(BestSol) >  fonctionObj(CurrentPPO))
                BestSol = CurrentPPO;
```

FIGURE 3.12 – Pseudo code de l'approche de déploiement en deux étapes

La stratégie que nous employons pour appliquer le principe d'améliorations itératives constitue le point fort de notre approche en deux étapes en comparaison à l'existant. D'une part, elle garantit un perfectionnement continue de la solution, ou, si aucune amélioration n'est possible, le maintien de la même solution. Ceci est dû au fait que la phase de modification du processus d'amélioration itérative au niveau de la boucle interne ne se fait pas d'une manière aléatoire. La modification s'effectue si et seulement si une amélioration de la solution est réalisée. De plus, elle vise à conserver les améliorations faites précédemment et remet en cause uniquement les décisions promettant des améliorations supplémentaires. La propriété de convergence de la boucle interne est due au fait que la métrique d'optimisation considérée est la même pour les deux sous problèmes. D'autre part, la stratégie proposée donne la possibilité d'éviter un optimum local grâce à la boucle externe. Ces constatations liées aux propriétés de notre approche sont illustrées dans le chapitre 4.3.1.5.

Afin de détailler le principe général de l'approche en deux étapes, la figure 3.13 est utilisée pour montrer le déroulement interne de cette approche vis-à-vis de l'aspect d'exploration architecturale. Soient $\overrightarrow{X_i} = x_1, x_2, ..., x_n$ et $\overrightarrow{Y_i} = y_1, y_2, ..., y_m$ les vecteurs de variables pour les sous problèmes $PP$ et $PO$, respectivement. Dans la figure 3.13, l'arbre de recherche pour chaque sous problème est représenté par un triangle. Un chemin dans l'arbre, représenté par des flèches en gras, est une solution possible pour le sous problème de cet arbre. Initialement, une configuration pour le sous problème $PO$ est donnée en entrée. Il s'agit de spécifier une valeur pour chaque variable du vecteur $\overrightarrow{Y}$, et donc de déterminer un chemin (entre le nœud $0$ et le nœud $J + 1$) dans l'arbre de recherche de $PO$ (voir le triangle de haut-gauche). Ici, toutes les autres branches dans l'arbre de $PO$ sont élaguées. Ensuite, le solveur est lancé sur l'espace de recherche de $PP$ pour retourner les meilleures valeurs possibles pour les variables du vecteur $\overrightarrow{X}$ (meilleure solution pour $PP$) tout en considérant

les valeurs du vecteur $\overrightarrow{Y}$ (voir le triangle de bas-gauche). A ce niveau, nous avons accompli l'étape 1 (l'étape de placement) de l'approche. Le résultat de cette étape est une solution possible pour le problème initial de $PPO$. Dans la prochaine étape de l'approche (l'étape de partitionnement et d'ordonnancement), la solution pour $PP$ trouvée précédemment par le solveur est fixée dans l'arbre de recherche de $PP$ (chemin entre le nœud $0$ et le nœud $J$) et le reste des branches représentant d'autres solutions est élagué (voir le triangle de haut-milieu). Puis, similairement, le solveur est relancé sur l'arbre de recherche de $PO$ pour trouver d'éventuelles valeurs pour les variables du vecteur $\overrightarrow{Y}$ qui améliorent la solution $PO$ considérée dans l'étape précédente (voir le triangle de bas-milieu). À la sortie de cette deuxième étape, nous obtenons une autre solution pour le problème initial de $PPO$. La différence avec la solution obtenue précédemment réside dans les valeurs du vecteur $\overrightarrow{Y}$ qui cette fois-ci résultent en une meilleure solution pour $PPO$ que celles précédentes. A ce stade de l'approche, nous arrivons à la fin de la première itération de la boucle interne. Ce processus est répété pour commencer une autre itération. L'itération suivante se base sur le résultat retourné par l'itération qui précède. Nous remarquons sur le triangle en haut à droite de la figure 3.13 que le chemin fixé dans l'arbre de $PO$ (chemin entre le nœud $0$ et le nœud $J+k$) est différent de celui fixé dans l'itération précédente (chemin entre le nœud $0$ et le nœud $J+1$). Par conséquent, la solution qui correspond au chemin entre le nœud $0$ et le nœud $J+1$ est élaguée dans l'itération $2$. Ceci implique que l'itération $2$ va permettre d'explorer une partie de l'espace de recherche qui est différente de celle explorée dans l'itération $1$. Ce processus s'arrête lorsque la solution pour $PPO$ trouvée à une étape donnée est la même retournée par l'étape qui précède, ce qui indique la terminaison de la boucle interne. La boucle externe de l'approche en deux étapes consiste à répéter le processus correspondant à la boucle interne pour différentes solutions initiales. La boucle externe, contrairement à la boucle interne, ne garantit pas l'exploration d'espaces de recherche différents à chaque itération.

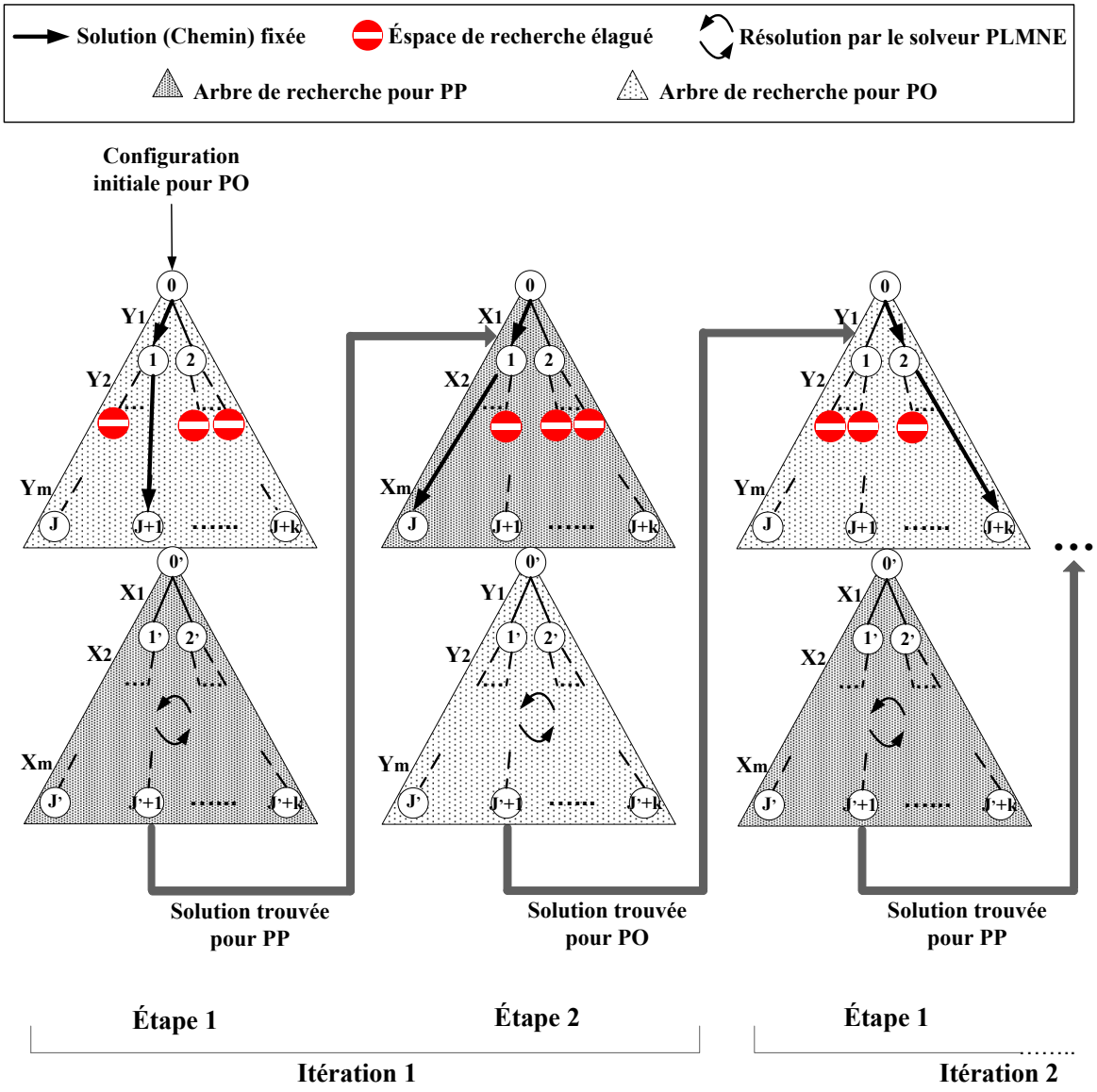



FIGURE 3.13 – Fonctionnement interne de l'approche en deux étapes

### 3.4.3.2 Formulation en PLMNE

Dans cette partie, nous discutons les différences des formulations PLMNE apportées à chaque sous problème de l'approche à deux étapes par rapport à la formulation intégrale présentée précédemment. La formulation complète pour chaque étape de cette approche peut être trouvée dans l'annexe B.

La première différence par rapport à la formulation intégrale réside dans le fait que certaines contraintes sont présentes dans une étape mais pas dans l'autre. Plus précisément, étant donné que dans la première étape nous nous intéressons uniquement au placement des fonctions et des signaux, toutes les contraintes concernant le partitionnement et l'ordonnancement de ces derniers sont écartées de la formulation PLMNE pour cette étape. Par conséquent, toutes les formules de (3.24) jusqu'à (3.31) sont reportées dans la formulation PLMNE pour la première étape de l'approche et toutes les formules de (3.32) à (3.37) ne le sont pas. Inversement, dans la seconde étape, seulement les contraintes de partitionnement et d'ordonnancement sont considérées (les formules (3.32) à (3.37)) pendant

que les contraintes liées au placement (les formules de (3.24) à (3.31)) sont ignorées.

Par ailleurs, certaines formules de l'approche intégrale ne sont considérées dans aucune des formulations PLMNE apportées pour les deux étapes de l'approche. Celles-ci consistent dans les différentes formules permettant de définir les variables $SnHp$ et $SnDp$, à savoir les formules (3.38) et (3.39). Cependant, dans la deuxième étape il est nécessaire de définir la variable binaire $SP_{i,j}$ indiquant que deux fonctions ou deux signaux possèdent le même ordre de priorité. Celle-ci est moins complexe que la variable $SnSp_{i,j,k}$ définie dans l'approche intégrale par la formule (3.40), puisqu'elle est à deux dimensions contre trois pour $SnSp_{i,j,k}$. Par conséquent, toutes les formules telles que (3.41) et (3.42) utilisant la variable $SnSp_{i,j,k}$ sont simplifiées dans la formulation de la deuxième étape. La formule définissant la variable $SP_{i,j}$ est la suivante :

$$1 = SP_{i,j} + \psi_{j,i} + \psi_{i,j} \tag{3.88}$$

Enfin, toutes les formules liées au calcul des WCRTs et à l'attribution du mécanisme de protection sont considérées dans les deux formulations correspondant aux deux étapes de l'approche. Ceci est dû d'une part à l'optimisation du temps de réponse considérée dans les deux sous problèmes, et d'autre part, à l'impact du mécanisme de protection lié aux décisions de placement au même temps qu'à celles de partitionnement et de l'ordonnancement. Néanmoins, ces formulations sont moins complexes dans les deux étapes en comparaison à celles de l'approche intégrale. En effet, dans ces formules toutes les variables liées au placement deviennent des paramètres dans la formulation de la deuxième étape et, réciproquement, toutes les variables de partitionnement sont fixées dans l'étape de placement. Par exemple, la formule (3.62) dans l'approche intégrale est remplacée par la formule (3.89) dans la formulation de la première étape, où la variable $I_{i,j}$ se définit par ( 3.90) pour $\forall i, j \in F$ tel que $\pi_j > \pi_i$.

$$W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i):\} \\ \pi_j = \pi_i}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1 \\ \pi_j > \pi_i}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \tag{3.89}$$

$$\sigma_{i,j} - M(1 - \sum_{k \in C} X_{i,j,k}) \leq I_{i,j}; I_{i,j} \leq \sigma_{i,j}; I_{i,j} \leq M. \sum_{k \in C} X_{i,j,k} \tag{3.90}$$

Cette même formule (3.62) est remplacée par la formule (3.91) dans la deuxième étape, où la variable $I_{i,j}$ se définit dans (3.92) pour $\forall i, j \in F$ tel que $SN_{i,j} = 1$. Cette dernière condition indique que les fonctions $f_i$ et $f_j$ sont placées sur le même processeur.

$$W_i = \omega_{i,k} + 2.cs + \sum_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1}} SP_{i,j} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1 \\ SN_{i,j}=1}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \tag{3.91}$$

$$\sigma_{i,j} - M\,(1 - \psi_{j,i}) \leq I_{i,j}; I_{i,j} \leq \sigma_{i,j}; I_{i,j} \leq M.\psi_{j,i} \tag{3.92}$$

Nous constatons que dans les deux formulations apportées aux deux étapes, la variable $I$ a deux dimensions au lieu de trois. Ce même principe s'appliquant à tout le reste des formules résulte en une formulation simplifiée pour chacune des étapes. Plus de détails sont fournis dans l'annexe B.

#### 3.4.3.3 Complexité théorique

Suivant le même principe que dans la partie 3.4.2.8, nous présentons les tableaux 3.9 et 3.10 permettant de récapituler la complexité théorique en termes de la taille du problème pour la formulation PLMNE de la première et de la seconde étape de l'approche de déploiement.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires $(d)$ | $\|F\|^2\|C\| + \|\Phi\|^2\|\beta\| + \|F\|\|C\| + \|\Phi\|(1+\|\beta\|)$ | $\|F\|^2\|C\| + \|\Phi\|^2\|\beta\| + \|F\|\|C\| + \|\Phi\|(1+\|\beta\|)$ | $\|F\|^2\|C\| + \|\Phi\|^2\|\beta\| + \|F\|\|C\| + \|\Phi\|(1+\|\beta\|)$ | $2\|F\|^2\|C\| + \|\Phi\|^2\|\beta\| + 5\|\Phi\| + \|F\|\|C\| + \|\Phi\|\|\beta\| + \|F\|^2\|\Phi\| + \|F\|\|\Phi\|$ |
| #var. totales $(n)$ | $\|F\|^2(2\|C\|+2)+\|\Phi\|^2(2\|\beta\| + 2)+\|\Phi\|(2\|\beta\|+6) + \|F\|(\|C\|+4)+\|\gamma\|$ | $\|F\|^2(2\|C\|+2)+\|\Phi\|^2(2\|\beta\| + 2)+\|\Phi\|(2\|\beta\|+6) + \|F\|(\|C\|+4)+\|\gamma\|$ | $\|F\|^2(2\|C\|+2)+\|\Phi\|^2(2\|\beta\| + 2)+\|\Phi\|(2\|\beta\|+6) + \|F\|(\|C\|+3)+\|\gamma\|$ | $\|F\|^2(3\|C\| + 1 + \|\Phi\|)+\|\Phi\|^2(2\|\beta\|+1)+\|F\|(2+\|C\|+\|\Phi\|) + \|\Phi\|(9+\|\beta\|) + \|\gamma\| + \|C\|(\|\Phi\|+1)$ |
| #contraintes $(m)$ | $\|F\|^2(4\|C\|+1)+\|\Phi\|^2(5\|\beta\| + 6) + \|F\|(\|\Phi\|\|\beta\| - 3\|C\| - 2) + \|\Phi\|(4-3\|\beta\|)+\|C\| + \|\beta\| + 2\|\gamma\| + 7\|F\|\|\Phi\|$ | $\|F\|^2(4\|C\|+1)+\|\Phi\|^2(5\|\beta\| + 6) + \|F\|(\|\Phi\|\|\beta\| - 3\|C\| - 2) + \|\Phi\|(4-3\|\beta\|)+\|C\| + \|\beta\| + 2\|\gamma\| + 7\|F\|\|\Phi\|$ | $\|F\|^2(4\|C\|+1)+\|\Phi\|^2(5\|\beta\| + 6) + \|F\|(\|\Phi\|\|\beta\| - 3\|C\| - 6) + \|\Phi\|(4-3\|\beta\|)+\|C\| + \|\beta\| + 2\|\gamma\| + 3\|F\|\|\Phi\|$ | $\|F\|^2(4\|C\|+2\|\Phi\|+1) + \|\Phi\|^2(5\|\beta\| + 2)+\|F\|(4-3\|C\|)+\|F\|\|\Phi\|(4\|C\| + \|\beta\|+3)+\|\Phi\|(7-3\|\beta\|) + \|\beta\| + 3\|C\|+2\|\gamma\|$ |

Tableau 3.9 – Complexité théorique de la formulation PLMNE pour la phase $PP$ en termes de la taille du problème

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires ($d$) | $2\|F\|^2+2\|\Phi\|^2$ | $3\|F\|^2+3\|\Phi\|^2$ | $2\|F\|^2+2\|\Phi\|^2$ | $5(\|F\|^2+\|\Phi\|)+2\|\Phi\|^2+5\|\Phi\|+\|F\|\|\Phi\|$ |
| #var. totales ($n$) | $5\|F\|^2+5\|\Phi\|^2+4\|F\|+6\|\Phi\|+\|\gamma\|$ | $6\|F\|^2+6\|\Phi\|^2+4\|F\|+6\|\Phi\|+\|\gamma\|$ | $5\|F\|^2+5\|\Phi\|^2+3\|F\|+6\|\Phi\|+\|\gamma\|$ | $\|F\|^2(7+\|\Phi\|)+4\|\Phi\|^2+2(\|F\|+\|\Phi\|)+9\|\Phi\|+\|\gamma\|+\|C\|(\|\Phi\|+1)$ |
| #contraintes ($m$) | $5\|F\|^3+5\|\Phi\|^3+\|F\|^2(\|\Phi\|-7)-\|\Phi\|^2+\|F\|+6\|\Phi\|+2\|\gamma\|+10\|F\|\|\Phi\|$ | $6\|F\|^3+5\|\Phi\|^3+\|F\|^2(\|\Phi\|-7)-\|\Phi\|^2+6\|\Phi\|+2\|\gamma\|+10\|F\|\|\Phi\|$ | $5\|F\|^3+5\|\Phi\|^3+\|F\|^2(\|\Phi\|-7)-\|\Phi\|^2+6\|\Phi\|+2\|\gamma\|+6\|F\|\|\Phi\|$ | $6\|F\|^3+5\|\Phi\|^3+\|F\|^2(3\|\Phi\|-7)-6\|\Phi\|^2+3\|F\|+\|\Phi\|\|F\|(3\|C\|+2)+10\|\Phi\|+2\|\gamma\|+2\|C\|$ |

Tableau 3.10 – Complexité théorique de la formulation PLMNE pour la phase PO en termes de la taille du problème

En termes d'ordre de grandeur, la complexité théorique et moyenne en nombre d'opérations arithmétiques pour chacune des étapes $PP$ et $PO$ sont respectivement mentionnées dans les tableaux 3.11 et 3.12.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| $O_{PLMNE_{Moy}}$ = $O(m).O(m.n).O(d^2)$ | $O(\|F\|^2+\|\Phi\|^2).O(\|F\|^4+\|\Phi\|^4).O(\|F\|^4+\|\Phi\|^4)$ | $O(\|F\|^2+\|\Phi\|^2).O(\|F\|^4+\|\Phi\|^4).O(\|F\|^4+\|\Phi\|^4)$ | $O(\|F\|^2+\|\Phi\|^2).O(\|F\|^4+\|\Phi\|^4).O(\|F\|^4+\|\Phi\|^4)$ | $O(\|F\|^2+\|\Phi\|^2).O(\|F\|^4+\|\Phi\|^4).O(\|F\|^4+\|\Phi\|^4)$ |
| $O_{PLMNE_{Pire}}$ = $O(2^m).O(2^d)$ | $O(2^{2\|F\|^2+2\|\Phi\|^2})$ | $O(2^{2\|F\|^2+2\|\Phi\|^2})$ | $O(2^{2\|F\|^2+2\|\Phi\|^2})$ | $O(2^{2\|F\|^2+2\|\Phi\|^2})$ |

Tableau 3.11 – Complexité moyenne ($O_{PLMNE_{Moy}}$) et théorique ($O_{PLMNE_{Pire}}$) en termes du nombre d'opérations arithmétiques pour la formulation PLMNE de la phase $PP$

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| $O_{PLMNE_{Moy}} = O(m).O(m.n).O(d^2)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).O((\lvert F\rvert^3+\lvert\Phi\rvert^3).(\lvert F\rvert^2+\lvert\Phi\rvert^2)).O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).O((\lvert F\rvert^3+\lvert\Phi\rvert^3).(\lvert F\rvert^2+\lvert\Phi\rvert^2)).O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).O((\lvert F\rvert^3+\lvert\Phi\rvert^3).(\lvert F\rvert^2+\lvert\Phi\rvert^2)).O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ | $O(\lvert F\rvert^3+\lvert\Phi\rvert^3).O((\lvert F\rvert^3+\lvert\Phi\rvert^3).(\lvert F\rvert^2+\lvert\Phi\rvert^2)).O(\lvert F\rvert^4+\lvert\Phi\rvert^4)$ |
| $O_{PLMNE_{Pire}} = O(2^m).O(2^d)$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ | $O(2^{\lvert F\rvert^3+\lvert\Phi\rvert^3}).O(2^{\lvert F\rvert^2+\lvert\Phi\rvert^2})$ |

Tableau 3.12 – Complexité moyenne ($O_{PLMNE_{Moy}}$) et théorique ($O_{PLMNE_{Pire}}$) en termes du nombre d'opérations arithmétiques pour la formulation PLMNE de la phase $PO$

Afin de confronter l'approche PLMNE intégrale à l'approche en deux étapes, nous considérons le même exemple que dans la partie 3.4.2.8 : un système avec $16$ fonctions, $15$ signaux, $10$ processeurs, $1$ bus de communication et $4$ chemins. Les tableaux 3.13 et 3.14 présentent la taille du problème ainsi que la complexité théorique et moyenne en termes du nombre d'opérations arithmétiques, pour chacune des étapes de l'approche en deux étapes. Celles-ci sont mentionnées pour les différents modèles d'activation et sémantiques de synchronisation. Les valeurs entre parenthèses rappellent les résultats de l'approche PLMNE intégrale. Nous constatons clairement que le nombre de variables et de contraintes dans chacune des deux formulations de l'approche en deux étapes est largement inférieur à celui de l'approche intégrale montrée précédemment.

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires | 2975 (11811) | 2975 (14627) | 2975 (11811) | 9675 (21327) |
| #var. totales | 6880 (15716) | 6880 (18532) | 6864 (15444) | 13170 (24849) |
| #contraintes | 14413 (56878) | 14413 (62034) | 13389 (59650) | 29994 (91680) |
| $O_{PLMNE_{Moy}}$ | $1,264e^{19}$ $(7,092e^{21})$ | $1,264e^{19}$ $(1,525e^{22})$ | $1,089e^{19}$ $(7,665e^{21})$ | $1,109e^{21}$ $(9,499e^{22})$ |
| $O_{PLMNE_{Pire}}$ | $1,344e^{1791}$ $(2^{56878}.2,919e^{3555})$ | $1,344e^{1791}$ $(2^{62034}.1,464e^{4403})$ | $1,134e^{4926}$ $(2^{59650}.2,919e^{3555})$ | $1,24e^{9029}.2,918e^{2912}$ $(2^{91680}.1,166e^{6420})$ |

Tableau 3.13 – Complexité moyenne et théorique de la formulation PLMNE de la phase $PP$ pour un système de $16$ fonctions, $15$ signaux, $10$ processeurs, $1$ bus de communication et $4$ chemins

| Modèles d'activation | Activations pilotées par les données | | | Activations périodiques |
|---|---|---|---|---|
| Sémantiques de synchronisation | $'InputN-of-N/outputN-of-N'$ | $'InputN-of-N/output1-of-N'$ | $'Input1-of-N/output1-of-N'$ | / |
| #var. binaires | 962 (11811) | 1443 (14627) | 962 (11811) | 2120 (21327) |
| #var. totales | 2563 (15716) | 3044 (18532) | 2547 (15444) | 6893 (24849) |
| #contraintes | 42142 (56878) | 46222 (62034) | 41166 (59650) | 57735 (91680) |
| $O_{PLMNE_{Moy}}$ | $4,212e^{18}$ ($7,092e^{21}$) | $1,354e^{19}$ ($1,525e^{22}$) | $3,994e^{18}$ ($7,665e^{21}$) | $1,032e^{20}$ ($9,499e^{22}$) |
| $O_{PLMNE_{Pire}}$ | $2^{42142}.3,898e^{289}$ ($2^{56878}.2,919e^{3555}$) | $2^{46222}.2,433e^{434}$ ($2^{62034}.1,464e^{4403}$) | $2^{41166}.3,898e^{289}$ ($2^{59650}.2,919e^{3555}$) | $2^{57735}.1,526e^{638}$ ($2^{91680}.1,166e^{6420}$) |

Tableau 3.14 – Complexité moyenne et théorique de la formulation PLMNE de la phase $PO$ pour un système de 16 fonctions, 15 signaux, 10 processeurs, 1 bus de communication et 4 chemins

Ce qui a été présenté jusqu'à présent par rapport à la complexité reste théorique. Une étude de la consommation des ressources en pratique est donnée dans le chapitre évaluation 4.

## 3.5 Conclusion

Dans ce chapitre, nous avons commencé par définir le modèle des systèmes visés par le travail de cette thèse, puis nous avons présenté la méthode d'analyse des temps de réponse proposée pour ce modèle. La particularité de cette analyse réside dans sa possibilité de considérer le modèle fonctionnel. Contrairement aux analyses des graphes fonctionnels existantes, l'analyse proposée ne se limite pas à un modèle de tâches particulier tel que le modèle avec des tâches indépendantes strictement périodiques. Ensuite, nous avons utilisé cette analyse pour prendre en compte les contraintes temps réel durant le processus de déploiement. Nous avons proposé deux approches de déploiement visant à optimiser : le placement des fonctions et des signaux respectivement sur les processeurs et les bus, leur partitionnement sur des tâches et des messages, l'ordonnancement de ces derniers et l'attribution du mécanisme de protection pour les données partagées, et ce vis-à-vis de la moyenne des latences et de la consommation mémoire. Les approches proposées sont basées sur la programmation mathématique et elles considèrent deux modèles d'activations différents, à savoir le modèle d'activations périodiques et celui d'activations pilotées par les données. L'intérêt de la formulation PLMNE intégrale proposée réside dans l'expression linéaire du calcul des temps de réponse en particulier et du problème de déploiement en général, tout en considérant un placement, un partitionnement, des priorités et un mécanisme de protection de données partagées inconnus a priori. Nous avons montré que la résolution de la formulation PLMNE intégrale du problème de déploiement est d'une complexité élevée. Cependant, ceci ne remet pas en cause l'intérêt de cette approche. Elle permet notamment d'aider le concepteur lors du paramétrage des approches à base de méta-heuristiques [81, 82] passant mieux à l'échelle. Enfin, l'alternative en deux étapes proposée permet de réduire la complexité de la résolution du problème de déploiement. L'originalité de la formulation PLMNE en deux étapes, demeure dans le fait que le déploiement est abordé d'une façon intégrale dans le sens où les décisions faites durant une étape sont liées

aux décisions faites à l'autre étape et vice versa. S'ajoutant à cela, la fonction objective est la même pour chacune des étapes. Dans le chapitre 4 qui suit, nous présentons l'évaluation quantitative et qualitative des solutions proposées ainsi qu'une étude comparative avec celles de l'état de l'art.

CHAPITRE 4

# Expérimentation et évaluation

## 4.1 Introduction

Pour l'évaluation expérimentale des différentes approches proposées, nous nous sommes basés sur un ensemble de systèmes synthétiques. Ces systèmes, utilisés pour étudier les performances et le passage à l'échelle de chaque approche, ont été générés aléatoirement à l'aide d'un générateur que nous avons implémenté. Dans ce chapitre, nous présentons, d'une part, le contexte des expérimentations (partie 4.2) qui détaille le générateur aléatoire et explicite les instances de système considérées lors de l'évaluation ainsi que les critères d'évaluation. D'autre part, nous exposons dans la partie 4.3 les résultats des différentes expérimentations qui, d'une part montrent l'intérêt de nos approches comparées à celles existantes et qui, d'autre part, font apparaître la pertinence de l'approche de déploiement en deux étapes par rapport à l'approche intégrale. Les approches proposées sont évaluées par rapport aux ressources de calcul et de stockage requises pour la résolution du problème et à la qualité de la solution renvoyée vis-à-vis du critère de temps de réponse. Elles sont également illustrées à l'aide de systèmes industriels de l'automobile.

Il est à noter que les expérimentations ont été effectuées sur un processeur AMD Opteron (tm) 6164 HE cadencé à 1.7 Ghz avec 46Go de mémoire et possédant vingt-quatre cœurs et que CPLEX [164] de la communauté IBM ILOG a été utilisé comme solveur de PLMNE pour l'ensemble des expériences. CPLEX offre l'avantage de paralléliser systématiquement l'exécution de l'algorithme de recherche en utilisant tous les cœurs de la machine cible.

## 4.2 Contexte des expérimentations

Le contexte des expérimentations englobe la description du générateur aléatoire (partie 4.2.1), la description des instances de tests et le paramétrage du solveur CPLEX (partie 4.2.2) et, enfin, les critères d'évaluation considérés (partie 4.2.3).

### 4.2.1 Description du générateur aléatoire

Le générateur aléatoire de systèmes prend comme entrée le nombre de fonctions, le nombre de processeurs et la topologie du réseau et renvoie en sortie un modèle fonctionnel muni de paramètres temporels et un modèle de plateforme d'exécution enrichi par l'utilisation maximale de ressources. Notre générateur est composé de deux parties, une partie liée à la génération du modèle de la plateforme d'exécution et une autre à la génération du modèle fonctionnel.

1. **Génération du modèle de plateforme d'exécution :** cette partie dépend de la topologie du réseau spécifiée. L'ensemble des topologies possibles comprend trois topologies classiques : topologie en bus, topologie en anneau et topologie en maille. Lorsque la topologie est en bus, un seul bus liant tous les processeurs est généré. Par conséquent, chaque processeur a la possibilité de communiquer avec tout autre processeur via le même bus. Si la topologie choisie est en anneau, le nombre de bus est égal au nombre de processeurs moins un : chaque processeur communique uniquement avec deux autres processeurs, son prédécesseur et son successeur dans l'anneau. En ce qui concerne la topologie en maille, le nombre de bus généré est égal à $\sum_{i=1}^{n} i$ où $n$ est le nombre de processeurs. Chaque processeur peut communiquer avec tout autre processeur au travers d'un bus dédié.
2. **Génération du modèle fonctionnel :** cette partie concerne d'une part la génération de la structure du modèle fonctionnel, et d'autre part, la génération des paramètres temporels.

- Du point de vue structurel, nous commençons par déterminer le nombre de transactions pour ensuite décider la taille de chaque transaction. Pour cela, le nombre de transactions est généré aléatoirement entre 1 et le nombre de fonctions du système selon une loi uniforme discrète. De même, le nombre de fonctions au sein de chaque transaction est généré aléatoirement dans un intervalle $[1, ..., n]$ où $n$ est le nombre de fonctions dans le système, diminué du nombre de fonctions qui ont déjà été considérées dans les transactions précédentes et du nombre de transactions pour lesquelles la taille n'a pas encore été spécifiée. Afin de déterminer l'organisation des fonctions au sein d'une transaction, nous avons généré une relation d'ordre partiel entre ces fonctions de telle sorte que l'Ordre soit égal à 1 pour exactement une fonction (celle-ci représente la fonction source de la transaction) et qu'il soit choisi aléatoirement entre 2 et le nombre des fonctions de la transaction (selon une loi uniforme discrète) pour le reste des fonctions. La détermination des relations de précédence entre les fonctions d'une transaction est faite à l'aide de l'algorithme suivant : soit $X$ l'ensemble des fonctions ayant un Ordre "i" et $Y$ l'ensemble de fonctions ayant un Ordre "i-1", si la taille de $X$ ($\|X\|$) est supérieure à la taille de $Y$ ($\|Y\|$), alors chacune des $\|Y\| - 1$ premières fonctions dans $X$ a comme prédécesseur une fonction différente appartenant à $Y$, le reste des fonctions dans $X$ ont comme prédécesseur la dernière fonction dans $Y$. Lorsque $\|X\|$ est inférieur à $\|Y\|$ alors chacune des $\|X\| - 1$ première fonctions dans $X$ a comme prédécesseur une fonction distincte appartenant à $Y$, la dernière fonction dans $X$ a comme prédécesseurs le reste des fonctions dans $Y$. Sinon, si $\|X\|$ est égal à $\|Y\|$ alors chaque fonction dans $X$ a un prédécesseur différent dans $Y$. L'exemple de la figure 4.1 illustre le déroulement de cet algorithme.

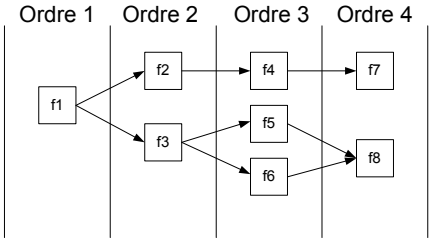



FIGURE 4.1 – Exemple de génération des relations de précédence

Pour compléter la structure, il reste à générer les signaux à partir des relations de précédence. Pour cela, il est nécessaire de préciser la sémantique considérée à la sortie d'une fonction : soit la fonction a un seul port de sortie et elle envoie donc un seul signal à une destination (ou à plusieurs destinations dans le cas d'un "Fork"), soit la fonction possède plusieurs ports en sortie et envoie par conséquent plusieurs signaux de telle sorte que chaque signal soit envoyé à une destination distincte. Le choix entre les deux sémantiques pour une fonction donnée est généré aléatoirement selon une loi de distribution uniforme dans $\{0, 1\}$. Par exemple dans la figure 4.1, si la sémantique générée pour $f_1$ est 1 alors $f_1$ envoie deux signaux : un pour $f_2$ et un pour $f_3$. Sinon elle envoie un seul signal simultanément à $f_2$ et $f_3$.

- La génération des paramètres temporels vise tout d'abord à générer des valeurs pour les WCETs des fonctions et pour les WCTTs des signaux. Nous générons ces valeurs suivant une loi uniforme continue dans un intervalle $[a, ..., b]$ où $a > 0$. $a$ et $b$ peuvent avoir une valeur quelconque. Ensuite, la charge d'utilisation maximale pour chaque processeur et chaque bus ainsi que leur facteur de vitesse sont choisis dans un intervalle $[a', ..., b']$ selon une loi uniforme continue. Pour le facteur de vitesse, $a'$ doit être supérieure à 0 et la valeur maximale pour $b'$ est de 1. En effet, le processeur

(bus) avec un facteur de vitesse égal à 1 représente le processeur (bus) le plus rapide. Enfin, la génération de la période de l'évènement externe d'une transaction est basée sur : le rapport entre le nombre de processeurs et le nombre de transactions, la valeur minimale que peut avoir la capacité d'utilisation maximale d'un processeur et la somme des WCETs des fonctions constituant la transaction. En effet, si le nombre de transactions est $x$ fois plus grand que le nombre de processeurs, cela signifie qu'il faut être capable de placer au moins $x$ transactions sur un processeur. Par conséquent, la somme des charges des transactions au sein d'un processeur doit être au maximum égale à la capacité minimale d'utilisation maximale d'un processeur. Par exemple, si le nombre de transactions est égal à 2 fois le nombre de processeurs alors il faut pouvoir placer deux transactions dans un processeur. De plus, si la valeur minimale de la capacité d'utilisation maximale d'un processeur est de $a' = 0.5$ et en supposant que la somme des WCET des fonctions d'une transaction est égale à 12 alors la valeur minimale que peut avoir la période $P$ de cette transaction est égale à $\frac{12}{\frac{0.5}{2}} = 48 \Rightarrow \left(P \geq \frac{12}{\frac{0.5}{2}}\right)$. Ce raisonnement a été établi afin d'éviter la génération de systèmes qui violent la contrainte d'utilisation des processeurs.

### 4.2.2 Description des instances de tests et paramétrage du solveur CPLEX

Afin d'évaluer les approches présentées dans le chapitre précédent, nous avons considéré plusieurs instances de système. Ces instances, englobant des systèmes de différentes tailles, ont été obtenues à l'aide du générateur aléatoire. Nous avons généré des modèles fonctionnels de 5, 10, 15, 20, 25, 30 et 35 fonctions et des modèles de plateformes d'exécution de 2, 4, 6 et 8 processeurs. La topologie du réseau de communication varie entre les trois topologies (topologie en bus, topologie en anneau et topologie en maille). La combinaison de ces différents paramètres (taille du modèle fonctionnel, taille de la plateforme d'exécution et type de la topologie) aboutit à 84 instances de dimensions distinctes. Nous avons généré 10 instances pour chaque dimension, obtenant ainsi un total de 840 systèmes. Nous rappelons que les 10 instances de même dimension diffèrent par la structure du modèle fonctionnel et par leurs paramètres temporels (WCETs des fonctions, WCTTs des signaux et périodes des transactions). Notons que lorsque la plateforme d'exécution comprend deux processeurs, la topologie en bus et la topologie en maille sont équivalentes, tandis que la topologie en anneau se caractérise par le nombre de bus liant les deux processeurs qui est égal à deux au lieu d'un. Du point de vue des paramètres temporels, nous avons fixé l'intervalle des WCETs pour les fonctions à [0.1,...,10] et celui des WCTTs pour les signaux à [0.2,...,20]. Les valeurs des WCTTs représentent le double de celles des WCETs car en réalité le coût de transmission sur le réseau est plus élevé que le temps de traitement au sein d'un processeur. Dans nos expérimentations, nous avons considéré pour toutes les ressources (processeurs+bus) générées une capacité d'utilisation maximale égale à 0.8 et un facteur de vitesse de 1. Par rapport à l'analyse des temps de réponse nous avons fixé le coût de changement de contexte à 0.5.

D'après [164], l'établissement de paramètres par défaut pour CPLEX atteint de bonnes performances pour une grande variété de problèmes en nombres entiers, nous avons donc choisi de garder ce paramétrage pour toutes les expériences. CPLEX met à la disposition de l'utilisateur certains paramètres lui permettant d'augmenter les performances du solveur lors de la résolution de PLNE difficiles. D'autres paramètres permettent l'ajustement de CPLEX par rapport aux besoins de l'utilisateur. L'annexe C donne plus de détails sur les options offertes par CPLEX ainsi que leur paramétrage par défaut.

### 4.2.3 Critères d'évaluation

Les critères sur lesquels l'évaluation est fondée sont présentés ci-après.

1. **Qualité des solutions :** ce critère concerne la qualité de la solution trouvée par une approche donnée. Celle-ci est quantifiée à l'aide de la moyenne des latences. Elle peut aussi être formulée par le taux de pessimisme ou le taux d'amélioration. Le taux de pessimisme (%) consiste à mesurer l'écart entre la moyenne des latences fournie par une approche "i" et la valeur optimale de la moyenne des latences. Ce taux est déterminé par la formule $\frac{(O_i - O_{opt})}{O_{opt}}$ où $O$ est la moyenne des latences. Notons que plus le taux de pessimisme est faible, meilleure est l'approche. Le taux d'amélioration (%) détermine de combien une approche "i" améliore les résultats obtenus par une approche "j" tout en considérant la solution optimale. Ce taux est calculé par la formule $100 - \frac{(O_i - O_{opt}) * 100}{O_j - O_{opt}}$. Cependant, lorsque l'approche "j" atteint déjà la solution optimale le taux d'amélioration est de $0\%$. Ceci signifie que l'approche "i" n'a rien à améliorer. Lors de la présentation des résultats, nous distinguerons ce dernier cas du $0\%$ qui indique que l'approche "i" n'a pas apporté d'amélioration.
2. **Performance en termes de ressources :** ce critère permet d'étudier le passage à l'échelle d'une approche donnée. Il représente le temps de calcul (ou d'exécution) indiquant la durée de temps nécessaire à une approche afin de résoudre complètement le problème de déploiement. Ce critère représente aussi la quantité de mémoire maximale en kilo-octets (KO) nécessaire à la résolution du problème.

## 4.3 Évaluation des approches proposées

Cette partie est consacrée à la présentation des résultats expérimentaux obtenus pour l'évaluation des approches proposées. Ces derniers sont organisés en deux parties principales. La première est dédiée aux systèmes dont les activations sont pilotées par les données (partie 4.3.1), tandis que la seconde adresse les systèmes ayant un modèle d'activation périodique (partie 4.3.2).

### 4.3.1 Évaluation pour les systèmes avec activations pilotées par les données

Dans un premier temps, nous évaluons notre contribution pour les systèmes dont les activations sont pilotées par les données. Pour cela, nous commençons par étudier dans la partie 4.3.1.1 l'influence de chaque sémantique de synchronisation apportée pour les tâches sur le résultat de déploiement. Ensuite, afin de montrer les améliorations apportées par notre solution de déploiement adressant à la fois le placement, le partitionnement et l'ordonnancement vis-à-vis des approches existantes, nous nous comparons aux approches faisant des hypothèses sur le partitionnement et à celles faisant des hypothèses sur le placement (dans les parties 4.3.1.2 et 4.3.1.3). Notre objectif est ici de montrer la nécessité de considérer le partitionnement **en même temps** que le placement et l'ordonnancement. Puis, nous évaluons dans la partie 4.3.1.4 nos approches de déploiement, à savoir l'approche PLMNE intégrale et l'approche en deux étapes tout en soulignant l'avantage et l'inconvénient de chacune. L'expérimentation de la partie 4.3.1.5 concerne l'évaluation de la stratégie d'améliorations itératives utilisée dans notre approche en deux étapes. Cette évaluation vise à montrer l'intérêt de chaque boucle dans l'approche. Enfin, dans la partie 4.3.1.6, nous illustrons chacune des approches de déploiement proposées dans cette

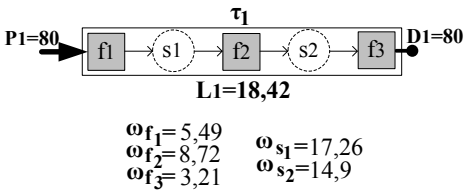
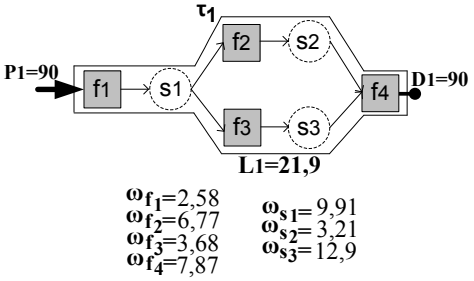
thèse et nous montrerons leur applicabilité en se basant sur un cas d'étude industriel du domaine de l'automobile.

#### 4.3.1.1 Évaluation des sémantiques de synchronisation de tâches

Avant d'évaluer les méthodes de déploiement proposées, il est nécessaire de clarifier l'impact de chaque sémantique de synchronisation des tâches et donc l'impact de l'analyse des temps de réponse appropriée sur la solution du déploiement. Pour mesurer cet impact, nous avons considéré trois versions de l'approche PLMNE intégrale. Chacune considère une sémantique parmi les sémantiques input N-of-N / output N-of-N, input N-of-N / output 1-of-N et input 1-of-N / output 1-of-N. Ensuite, nous nous sommes intéressés à la comparaison de la qualité des solutions obtenues par les différentes versions par rapport à la moyenne des latences, puis à la détermination de l'espace mémoire consommé par chaque version. Suite à la complexité de l'approche PLMNE intégrale, nous avons considéré uniquement $360$ instances de système parmi les $840$ instances décrites dans la partie 4.2.2. Ces $360$ instances se caractérisent par un modèle fonctionnel dont le nombre de fonctions ne dépasse pas $15$. Après avoir analysé les solutions pour chacune des $360$ instances, nous avons constaté que celles-ci peuvent être organisées principalement en quatre catégories présentées dans le tableau 4.1.

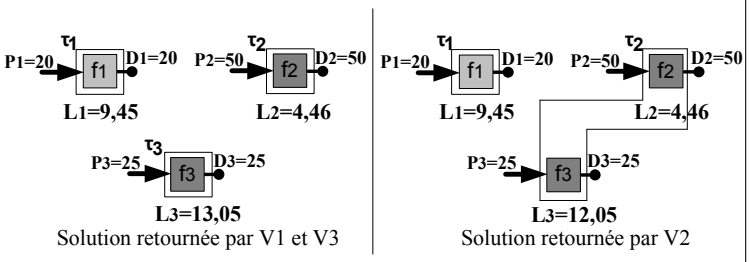
Tableau 4.1 – Impact des différentes sémantiques de synchronisation sur la qualité de la solution de déploiement renvoyée par l'approche PLMNE intégrale

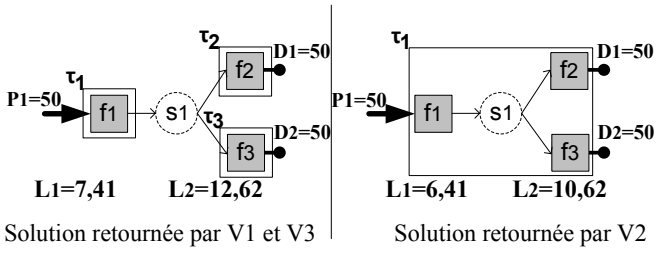
V1 = l'approche PLMNE intégrale considérant la sémantique input N-of-N / output N-of-N
V2 = l'approche PLMNE intégrale considérant la sémantique input N-of-N / output 1-of-N
V3 = l'approche PLMNE intégrale considérant la sémantique input 1-of-N / output 1-of-N

| Situations possibles | Circonstances | Discussion et illustration par exemple |
|---|---|---|
| **Cas 1 :** le même résultat est renvoyé par toutes les versions de l'approche PLMNE intégrale | Transaction linéaire. | Pour V2 et V3, bien que le résultat d'une fonction intermédiaire [a] peut être disponible avant le début d'exécution des fonctions qui la succèdent sur le même chemin, le seul résultat qui compte pour la fonction objectif est celui de la dernière fonction activée dans le chemin. Par conséquent, V2 et V3 renvoient le même résultat que celui de V1.   |
| | Transaction non-linéaire synchronisée [b]. | Comme dans le cas précédent, même si V2 et V3 permettent à toutes les fonctions intermédiaires de retourner leur résultat sans avoir à attendre l'exécution de leurs successeurs, c'est seulement le temps de réponse de la fonction puits qui est intéressant à la fonction objectif.   |
| | | |

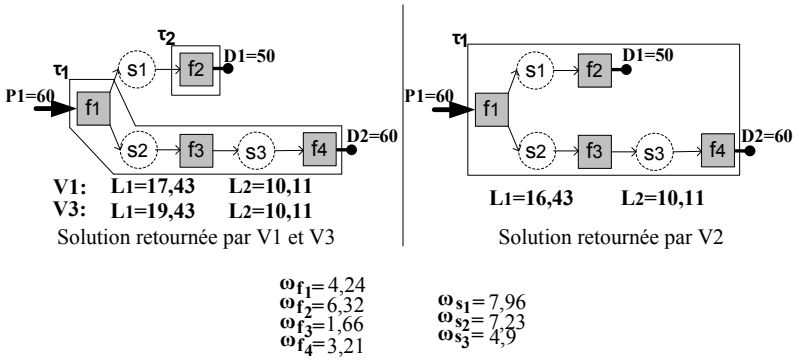
*a*. Une fonction intermédiaire est une fonction différente de la fonction puits.

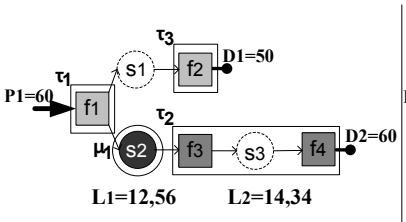
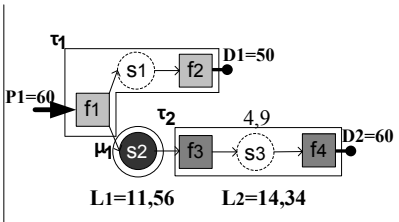
*b*. Une transaction non-linéaire synchronisée est une transaction non-linéaire avec exactement une fonction puits. Elle est aussi appelée graphe polaire.

<table>
<tr>
<td rowspan="2">Cas 2 : V1 et V3 retournent le même résultat et V2 retourne un résultat de meilleure qualité</td>
<td>Des fonctions indépendantes.</td>
<td>Grâce à l'ordre d'exécution défini au sein des tâches, V2 privilégie le partitionnement des fonctions indépendantes sur la même tâche puisque ceci résulte en un effet des changements de contexte plus réduit et ne mène pas à une attente bilatérale entre les fonctions de la tâche. À l'opposé, les autres versions empêchent ce type de partitionnement en raison de la présence d'une attente bilatérale, et ce malgré la présence des périodes harmoniques pour V3. V2 retourne donc des solutions avec moins de tâches en comparaison avec les solutions renvoyées par V1 et V3.<br> 

$\omega_{f_1}$= 8,45<br>$\omega_{f_2}$= 3,46<br>$\omega_{f_3}$= 7,59</td>
</tr>
<tr>
<td>Un même signal est envoyé à plusieurs fonctions avec un coût de communica-tion élevé.</td>
<td>La nature d'un signal local exigeant qu'il soit transmis en intra-tâche ou en inter-tâches, conduit à seulement deux solutions possibles : mettre toutes les fonctions (la fonction émettrice et les fonctions réceptrices) du signal sur la même tâche ou mettre chaque fonction sur une tâche différente. V1 et V3 optent pour la seconde solution car la première mène à une attente bilatérale entre les fonctions de la tâche et donc à une solution moins performante. V2 choisit la première solution qui aboutit à un coût des changements de contexte plus petit en comparaison à celui des solutions obtenues par V1 et V3.<br> 

$\omega_{f_1}$= 2,8 $\omega_{f_3}$= 4,21<br>$\omega_{f_2}$= 2,61 $\omega_{s_1}$=14,6</td>
</tr>
</table>

| | | |
|---|---|---|
| **Cas 3 :** V2 retourne le résultat le plus performant, V3 renvoie le moins performant et V1 produit un résultat d'une qualité intermédiaire. | Une fonction envoie plusieurs signaux où chacun est destiné à une fonction différente. De plus, le coût de communication est considérable par rapport aux durées d'exécution. | Dans la solution renvoyée par V2, tous les signaux sont transmis en intra-tâche ce qui réduit le coût des changements de contexte sans avoir en contrepartie l'effet d'une attente bilatérale. V1 et V3 sont forcées à retourner une solution avec plus de tâches afin d'éviter cet effet d'attente bilatérale. V1 et V3 retournent la même solution mais de différentes qualités. Ceci est dû au fait que V1 résulte en des changements de contexte plus réduits que V3 puisque le retard provenant des fonctions réceptrices partitionnées avec la fonction émettrice est inclus dans la gigue. V3 considère ce retard dans le terme d'interférences ce qui augmente le coût des changements de contexte. Par exemple, dans V3, $f_3$ et $f_4$ peuvent préempter l'exécution de $f_2$ puisque $f_2$ et $f_3$ peuvent s'activer au même temps, tandis que dans V1, $f_2$ ne peut pas commencer son exécution avant la fin d'exécution de $f_3$ et $f_4$. Ce qui implique que le retard en provenance de $f_3$ et $f_4$ est inclus dans la gigue de $f_2$ puisque $f_3$ et $f_4$ appartiennent à la même tâche que $f_1$. V3 résulte en deux changements de contexte supplémentaires par rapport à V1. V2 revoie une solution mono-tâche mono-processeur.   |

| | | |
|---|---|---|
| **Cas 4 :** V2 et V3 retournent le même résultat qui est plus performant que celui renvoyé par V1 | Une fonction envoie plusieurs signaux où chacun est destiné à une fonction différente de celle des autres. De plus, le coût de communication est réduit par rapport aux durées d'exécution. | Contrairement au cas précédent, lorsque le coût de communication sur le réseau résulte en un retard moins important que celui dû aux interférences au sein d'un processeur, toutes les versions renvoient une solution distribuée. Cependant, V2 et V3 produisent la même solution qui consiste à mettre la fonction émettrice et l'une des fonctions réceptrices sur le même processeur et sur la même tâche et de placer le reste des fonctions réceptrices sur d'autres processeurs. Cette solution évite les interférences entre les fonctions réceptrices. D'autre part, V1 donne la même solution de placement que celle de V2 et V3 mais la solution de partitionnement est différente. Elle met la fonction émettrice et la fonction réceptrice sur différentes tâches afin d'empêcher une propagation de la gigue sur les autres processeurs. Celle-ci conduit à un nombre de tâches plus grand et donc à un coût de changement de contexte plus élevé.<br>  <br><br>Solution retournée par V1<br>  <br><br>Solution retournée par V2 et V3<br>$\omega_{f_1}$= 4,24<br>$\omega_{f_2}$= 6,32<br>$\omega_{f_3}$= 1,66<br>$\omega_{f_4}$= 3,21<br>$\omega_{s_1}$= 6,96<br>$\omega_{s_2}$= 3,23<br>$\omega_{s_3}$= 4,9 |

À partir de cette première expérimentation, nous concluons que pour les $360$ instances considérées, la sémantique de synchronisation input N-of-N / output 1-of-N (V2) a toujours prouvé sa pertinence vis-à-vis du critère de qualité. D'autre part, $78\%$ des instances pour lesquelles les sémantiques input N-of-N / output N-of-N (V1) et input 1-of-N / output 1-of-N (V3) ont renvoyé des solutions de qualités différentes représentent le cas 3 et seulement $22\%$ représentent le cas 4. Autrement dit, la sémantique input N-of-N / output N-of-N (V1) a montré sa pertinence sur plus d'instances en comparaison avec la sémantique input 1-of-N / output 1-of-N (V3). De plus, l'écart entre la qualité des solutions obtenues par V1 et V3 dans le cas 3 est au minimum deux fois plus que l'écart dans le cas 4.

Dans un deuxième temps, nous nous sommes intéressés à déterminer l'espace mémoire demandé par le processus d'exploration (solveur) pour chaque version de l'approche PLMNE intégrale. Pour une raison de lisibilité, la figure 4.2 montre la mémoire maximale utilisée par chaque version pour seulement $58$ applications parmi les $360$. À partir de la figure, nous constatons clairement que la sémantique input N-of-N / output 1-of-N (V2) conduit à une consommation mémoire plus élevée en comparaison avec les autres sémantiques. Ceci est dû au fait que V2 considère l'ordre d'exécution pour les fonctions au sein des tâches comme un degré de liberté supplémentaire par rapport aux autres versions. Par conséquent, le nombre de variables de décisions est plus important dans V2 que dans V1 et V3. De leur côté, les versions V1 et V3 consomment plus ou moins la même quantité de mémoire. Par ailleurs, pour un système ayant la dimension 15/Maille/4, la version V2 retourne un *statut* de mémoire insuffisante ($OOM$ pour Out-Of-Memory) après $3200064$ KO de mémoire maximale utilisée, tandis que V1 et V3 retournent la solution optimale avec $2973504$ KO et $2896980$ KO respectivement de mémoire maximale utilisée. De ce fait, pour une instance donnée, il y a plus de chance d'obtenir une solution optimale avec V1 et V3 qu'avec V2. Ceci reste aussi vrai par rapport au temps de calcul du solveur qui est assez élevée pour V2 en comparaison avec V1 et V3.

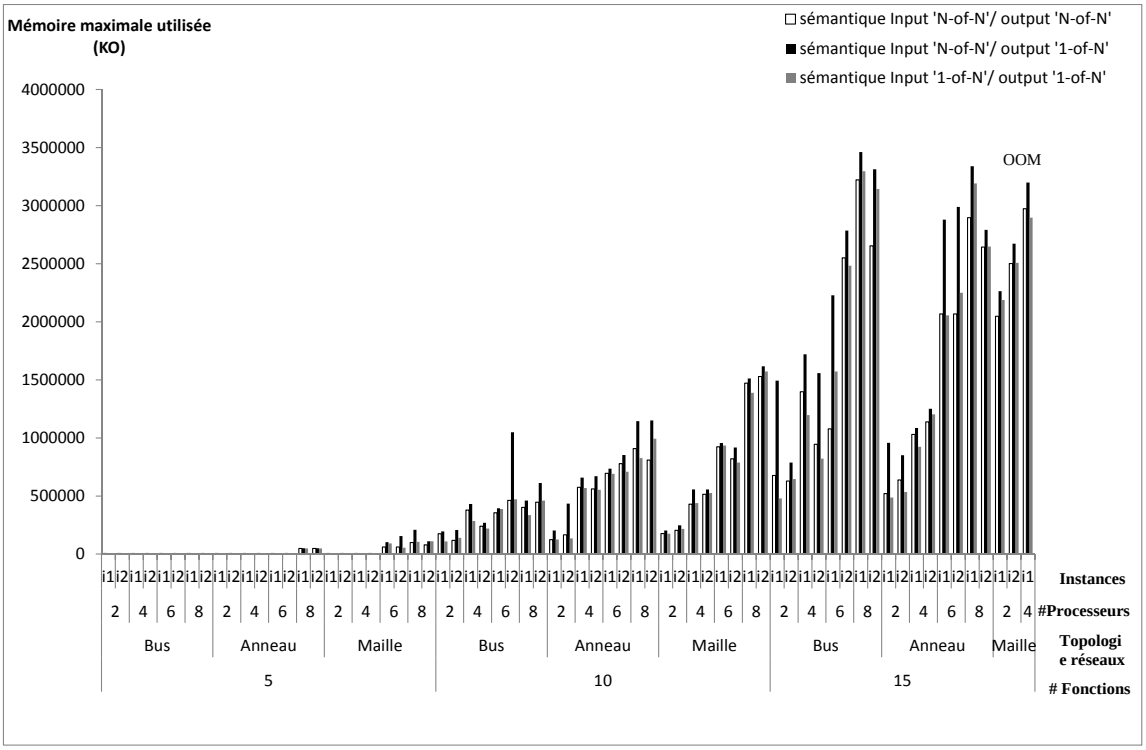



FIGURE 4.2 – Espace mémoire demandé par l'approche PLMNE intégrale pour les différentes sémantiques de synchronisations de tâches

À partir des deux études comparatives présentées ci-dessus, nous déduisons que les évaluations peuvent être multiples. De plus, parfois le choix de la sémantique de synchronisation est imposé par l'application. Par la suite, nous considérons la situation où la sémantique est input N-of-N / output N-of-N. Les études proposées peuvent éventuellement être étendues aux autres sémantiques.

#### 4.3.1.2 Comparaison avec les approches faisant des hypothèses sur le partitionnement

Dans cette expérimentation, nous nous intéressons à la comparaison de nos approches de déploiement (l'approche PLMNE intégrale et l'approche en deux étapes présentées dans les parties 3.4.2 et 3.4.3) avec celles de la littérature considérant un partitionnement fixe de fonctions sur les tâches. Nous considérons que le partitionnement est fixé selon les stratégies les plus populaires, à savoir :

1. **Le partitionnement à base de fonctions** qui consiste à partitionner chaque fonction sur une tâche différente [16]. Pour imposer ce type de partitionnement, il suffit d'ajouter à notre formulation PLMNE intégrale une contrainte empêchant l'attribution de la même priorité à plus de deux fonctions. La contrainte (3.32) devient donc $\psi_{i,j}+\psi_{j,i} = 1$ et les contraintes (3.33) sont remplacées par $\psi_{i,j} + \psi_{j,k} - 1 \leq \psi_{i,k}$.
2. **Le partitionnement à base de transactions** dans lequel toutes les fonctions appartenant à la même transaction sont partitionnées sur la même tâche [11, 12, 18]. Cette stratégie peut être réalisée en ajoutant à notre formulation PLMNE intégrale deux contraintes : la première oblige à attribuer la même priorité pour toutes les fonctions d'une transaction ($\psi_{i,j} = \psi_{j,i}$ si transaction de $f_i$ = transaction de $f_j$) et la seconde

impose le placement de ces fonctions sur le même processeur ($\sum_{k \in C} X_{i,j,k} = 1$ si transaction de $f_i$ = transaction de $f_j$).

Dans la suite de cette partie, nous faisons référence à l'approche de déploiement soumise à un partitionnement à base de fonctions et à un partitionnement à base de transactions par PLMNE intégrale/H1 et PLMNE intégrale/H2, respectivement. D'autre part, l'approche de déploiement en deux étapes proposée dans cette thèse est référencée par PLMNE 2-étapes.

- **Objectifs :** notre but à travers cette expérimentation se décline en trois points :
  - l'évaluation de l'impact des contraintes de partitionnement sur la performance de l'algorithme d'exploration intégré au solveur ;
  - la mise en évidence des limites des approches de partitionnement existantes par rapport au critère du temps de réponse ;
  - l'estimation des améliorations que peut apporter l'approche PLMNE 2-étapes aux approches de déploiement fixant le partitionnement.

- **Comparaison des performances :** afin d'illustrer le premier point cité ci-dessus, nous nous sommes intéressés à l'estimation du temps d'exécution du solveur et de la quantité de mémoire consommée par ce dernier pour chacune des approches PLMNE intégrale, PLMNE intégrale/H1 et PLMNE intégrale/H2. Pour cela, nous avons considéré $360$ instances parmi les $840$ instances décrites dans la partie 4.2.2. Ces $360$ instances se caractérisent par un modèle fonctionnel dont le nombre de fonctions ne dépasse pas $15$.

  Le temps d'exécution du solveur ainsi que sa consommation mémoire dépendent de la forme de l'espace de recherche considéré par l'algorithme d'exploration. Celle-ci est fortement liée à l'instance du problème à résoudre. Elle dépend des différents paramètres du système tels que ses paramètres temporels et la taille et la structure de ses modèles d'entrée. Étant donné que plusieurs paramètres peuvent influencer le comportement de l'algorithme d'exploration au sein du solveur, nous avons choisi d'étudier trois paramètres principaux liés à la taille du système, à savoir le nombre de fonctions, le nombre de processeurs et la topologie du réseau. Pour le critère de temps d'exécution, les résultats par rapport à chacun de ces paramètres sont représentés sous forme d'un diagramme en boîte et montrés par les figures 4.3, 4.4 et 4.5. Sur chaque graphe la croix désigne la valeur moyenne et les extrémités des lignes verticales représentent les valeurs maximale et minimale. Le reste des valeurs sont organisées en trois parties. La première partie est indiquée par la distance entre la valeur minimale et le trait inférieur du rectangle. Elle représente l'intervalle des valeurs pour $25\%$ des instances pour lequel le solveur est rapide. La deuxième partie correspond à l'intervalle des valeurs pour $50\%$ des instances pour lequel le solveur est de vitesse moyenne, elle est marquée par le rectangle. Enfin, la troisième partie est délimitée par le trait supérieur du rectangle et le trait de la valeur maximale. Elle décrit l'intervalle des valeurs pour $25\%$ des instances pour lequel le solveur est lent.

  Parmi les $360$ instances que nous avons considérées, le solveur a échoué à résoudre la formulation PLMNE intégrale pour $10$ instances. Cet échec est lié à une insuffisance dans la mémoire de la machine d'exécution. Principalement, les $10$ instances se caractérisent par un modèle fonctionnel de $15$ fonctions et une plateforme d'exécution composée de $8$ processeurs liés par un réseau en maille.

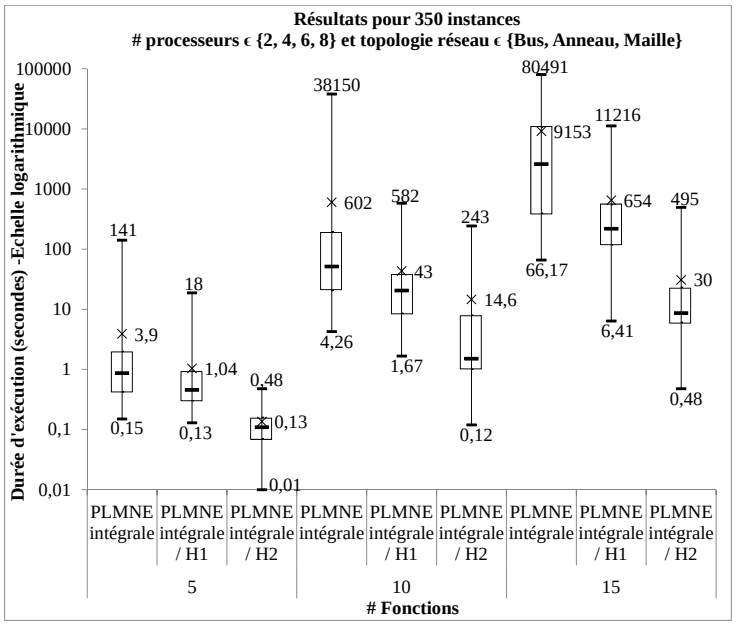



FIGURE 4.3 – Influence du nombre de fonctions sur le temps de calcul des approches PLMNE intégrale, PLMNE intégrale/H1, PLMNE intégrale/H2

La figure 4.3 montre l'évolution du temps d'exécution du solveur pour les différentes approches par rapport au nombre de fonctions. Nous constatons d'une part qu'en moyenne le temps d'exécution de chacune des approches croît proportionnellement au nombre de fonctions. D'autre part, les différentes hypothèses de partitionnement (H1 : partitionnement à base de fonctions et H2 : partitionnement à base de transactions) ont diminué le temps d'exécution de l'approche PLMNE intégrale. Par rapport à l'approche PLMNE intégrale/H2, ceci peut être expliqué en partie par le fait que la contrainte de placement ajoutée à la formulation PLMNE intégrale pour exprimer l'hypothèse de partitionnement à base de transactions a permis d'appliquer la séparation dans l'arbre de recherche seulement pour une fonction de chaque transaction, puisque pour le reste des fonctions appartenant à la même transaction le choix de placement est le même. Il convient par conséquent d'explorer une seule branche de placement pour ces dernières fonctions. De plus, pour tous les signaux du système, le choix de placement sur les bus de communication est exclu par l'algorithme d'exploration car ils sont tous transmis localement : leur placement ne dépend donc qu'uniquement de placement des fonctions. Une autre raison qui permettrait d'expliquer la diminution du temps d'exécution de l'approche PLMNE intégrale/H2 est que, pour les choix d'affectation de priorités, la relation d'ordre de priorité entre toutes les fonctions d'une même transaction est fixée. Ceci implique un élagage d'une partie de l'arbre de recherche et donc une accélération de l'algorithme d'exploration. Du point de vue de l'approche PLMNE intégrale/H1, nous remarquons clairement que l'amélioration de temps apportée par cette approche est moins importante que celle obtenue par l'approche PLMNE intégrale/H2. Ceci est dû au fait que les choix de placement dans l'approche PLMNE intégrale/H1 sont les mêmes que ceux dans l'approche PLMNE intégrale, dépendant dans les deux cas du nombre de fonctions. Néanmoins, les contraintes exprimant le partitionnement à base de fonctions ont également contribué à l'amélioration des temps d'exécution de l'approche PLMNE intégrale. En effet, l'imposition d'un ordre total pour les priorités des fonctions a permis d'exclure les branches de l'arbre de recherche exprimant la

possibilité d'avoir le même ordre de priorité pour un sous ensemble de fonctions.

Nous remarquons aussi un chevauchement des valeurs d'une même approche pour différent nombre de fonctions ainsi qu'une grande distance entre les temps d'exécution minimaux et maximaux. Ceci est dû en partie à la taille des instances incluses qui diffère par rapport au nombre de processeur et à la topologie du réseau.

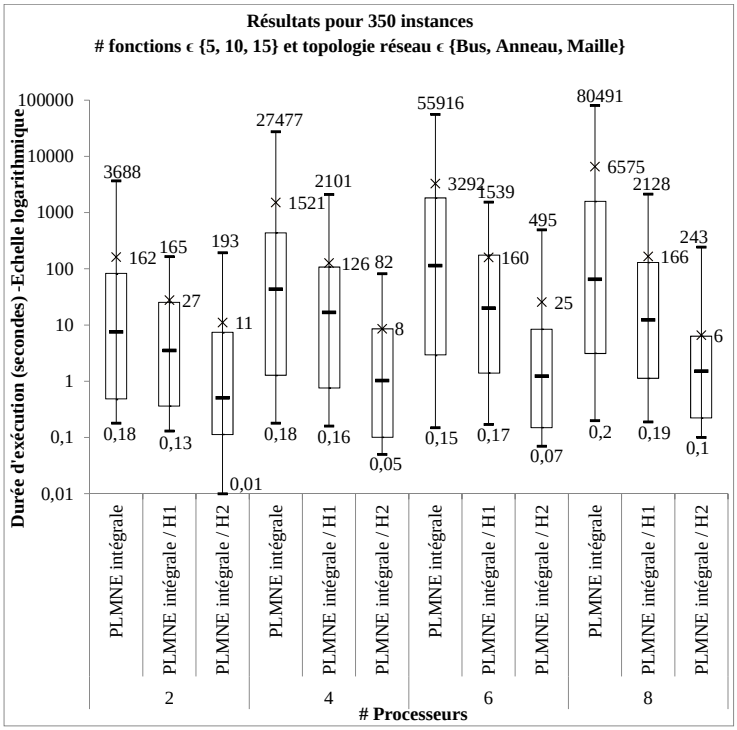



FIGURE 4.4 – Influence du nombre de processeurs sur le temps de calcul des approches PLMNE intégrale, PLMNE intégrale/H1, PLMNE intégrale/H2

L'évolution du temps d'exécution du solveur pour les différentes approches par rapport au nombre de processeurs est représentée sur la figure 4.4. À partir de celle-ci, nous tirons les mêmes observations mentionnées précédemment pour le paramètre du nombre de fonctions, à l'exception que le temps d'exécution moyen de l'approche PLMNE intégrale/H2 décroit pour le cas de quatre processeurs en comparaison avec le cas de deux processeurs. Il croit ensuite pour six processeurs puis il décroit de nouveau pour le cas de huit processeurs. Ce comportement n'est pas inattendu puisque le temps d'exécution de l'algorithme d'exploration dépend également des paramètres temporels de l'instance abordée.

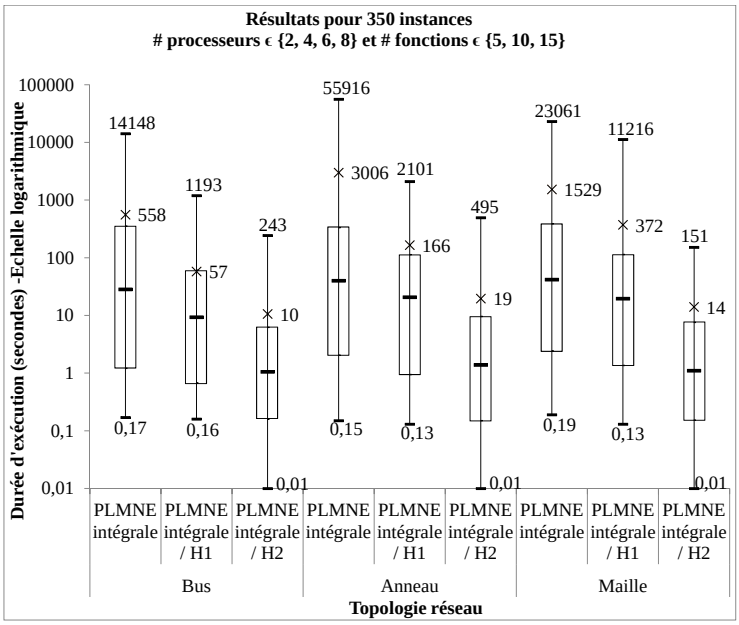



FIGURE 4.5 – Influence de la topologie du réseau sur le temps de calcul des approches PLMNE intégrale, PLMNE intégrale/H1, PLMNE intégrale/H2

En ce qui concerne le paramètre de la topologie du réseau, nous remarquons sur la figure 4.5 que le temps d'exécution moyen de l'ensemble des approches est plus important pour la topologie en anneau, comprenant plusieurs bus, que pour la topologie en bus. Cependant, bien que la topologie en maille soit munie de plus de bus que la topologie en anneau, le temps d'exécution moyen pour cette dernière reste plus important que celui pour la topologie en maille. Par conséquent, l'évolution du temps d'exécution du solveur est faiblement liée à la topologie du réseau. Par rapport à l'impact des hypothèses de partitionnement sur le temps d'exécution de l'approche intégrale, les observations apportées pour les autres paramètres, à savoir le nombre de fonctions et le nombre de processeurs, sont identiques pour le paramètre de la topologie du réseau.

À partir des trois figures 4.3, 4.4 et 4.5, nous déduisons que **le nombre de fonctions et le nombre de processeurs constituent les facteurs dominant** concernant l'influence sur le temps d'exécution du solveur. Par ailleurs, les deux hypothèses de partitionnement (à base de fonctions et à base de transactions) considérées ont largement augmenté le passage à l'échelle de l'approche PLMNE intégrale. Nous concluons donc que la résolution globale du problème de déploiement est plus complexe que sa résolution pour un partitionnement fixé.

Enfin, étant donné que l'allure du diagramme en boîte pour la mémoire maximale consommée est la même que celle obtenue pour le temps d'exécution, les résultats pour la performance en mémoire ne sont donc pas présentés.

- **Comparaison de la qualité des solutions :** nous allons maintenant évaluer la qualité des solutions retournées par les approches PLMNE intégrale, PLMNE intégrale/H1 et PLMNE intégrale/H2 par rapport à la moyenne des latences afin de montrer l'impact négatif d'un partitionnement fixé selon la stratégie à base de fonctions (H1) ou celle à base de transactions (H2) sur la qualité de la solution de déploiement. L'évaluation

concerne 350 instances parmi les 360 considérées. Ces 350 instances représentent les situations où le résultat optimal est obtenu. L'interprétation des résultats est organisée en quatre cas comme suit :

- **Cas 1 :** les approches PLMNE intégrale/H1 et PLMNE intégrale/H2 ont toutes les deux réussi à atteindre le résultat optimal pour 19 instances parmi les 350, ce qui implique un taux de $5,42\%$. Ces 19 instances se caractérisent par un modèle fonctionnel où toutes les transactions sont de taille égale à 1. Autrement dit, ce sont des instances de système composées uniquement de fonctions indépendantes. Dans cette situation, les deux hypothèses de partitionnement (à base de fonctions et à base de transactions) sont similaires. Par conséquent, le partitionnement optimal consiste à mettre chaque fonction (ou transaction) dans une tâche différente. Néanmoins, le regroupement de fonctions indépendantes sur la même tâche augmente la valeur des latences. Cette augmentation est due au fait que chaque fonction de la tâche doit attendre l'exécution de l'autre. Ceci crée une attente bilatérale, alors que l'attente est unidirectionnelle dans le cas de partitionnement sur différentes tâches. En se basant sur l'expérimentation présentée dans la partie 4.3.1.1, nous pouvons également conclure que dans le cas où la sémantique de synchronisation considérée est input N-of-N / output 1-of-N, alors aucune hypothèse de partitionnement ne permettra d'atteindre la solution optimale.
- **Cas 2 :** en plus des 19 instances, l'approche PLMNE intégrale/H2 a aussi réussi à trouver la solution optimale pour 111 autres instances ce qui lui fait un taux de réussite global de $37,14\%$. Ces 111 instances se définissent par un modèle fonctionnel dont les transactions sont soit linéaires soit non-linéaires synchronisées. Une transaction non-linéaire synchronisée est une transaction comportant plus d'un chemin de telle sorte que tous ses chemins se rejoignent à la fin. En d'autres termes, la dernière fonction activée est la même pour tous les chemins de la transaction (voir le cas 1 du tableau 4.1). Pour le cas des transactions linéaires, le partitionnement à base de transactions est pertinent car il permet d'éviter la propagation de la gigue entre les fonctions s'exécutant séquentiellement au sein de la transaction. De plus, l'effet du changement de contexte est réduit, puisqu'une seule tâche est dédiée à chaque transaction linéaire. Dans certaines situations où l'utilisation d'une transaction linéaire dépasse la capacité d'utilisation maximale de tous les processeurs du système, le partitionnement à base de transactions résulte en des solutions non réalisables par rapport à la contrainte d'utilisation de ressources alors que l'approche PLMNE intégrale est apte à trouver des solutions réalisables. Lorsque les transactions sont non-linéaires synchronisées, l'approche PLMNE intégrale/H2 renvoie une solution dans laquelle l'exécution séquentielle des fonctions concurrentes au sein de la même tâche ne dégrade pas la valeur de la latence. En effet, dans tous les cas, la dernière fonction de la transaction doit attendre l'exécution de toutes les fonctions appartenant à la transaction. Cependant, il peut exister des exceptions où le partitionnement à base de transactions ne permet pas d'atteindre un déploiement optimal pour le cas de transactions non-linéaires synchronisées. Ces exceptions concernent la situation où le coût de communication sur le réseau et la propagation de la gigue sont moins importants que le coût d'attente bilatérale liée à l'exécution séquentielle des fonctions concurrentes. Afin d'illustrer cette situation, nous nous sommes basés sur l'exemple fourni par la figure 4.6. Les valeurs situées au-dessus des fonctions et des signaux représentent respectivement les WCETs et les WCTTs. La plateforme d'exécution considérée est composée de deux processeurs et d'un bus de communication. La solution visualisée sur la figure 4.6 représente la solution optimale retournée par notre approche PLMNE intégrale. Celle-ci consiste à placer les fonctions $f_3$ et $f_4$ sur le

premier processeur et le reste des fonctions sur le deuxième, le tout résultant en deux communications sur le réseau. La priorité des tâches et des messages est inversement proportionnelle à leur indice. La latence pour cette solution est de $39,56$ unité de temps pendant que le partitionnement de toutes les fonctions de la transaction sur la même tâche résulte en une latence égale à $42,01$. La solution distribuée trouvée par notre approche a permis de réduire la gigue de la fonction $f_7$ en éliminant la concurrence entre les fonctions parallèles, $f_3, f_4$ d'une part et $f_5, f_6$ d'autre part.

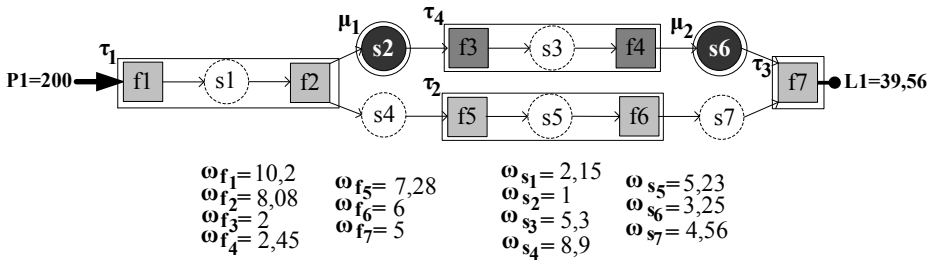



FIGURE 4.6 – Un exemple de transaction non-linéaire synchronisée pour laquelle l'approche PLMNE intégrale/H2 n'a pas trouvé la solution optimale

Un autre type de systèmes pour lequel l'approche PLMNE intégrale/H2 n'est pas pertinente est le cas de systèmes avec transactions non-linéaires non-synchronisées. Une transaction non-linéaire non-synchronisée est une transaction constituée de plusieurs chemins dont les dernières fonctions activées sont différentes (voir le cas 3 du tableau 4.1). Pour ce type de transaction, le regroupement de fonctions parallèles sur la même tâche empire forcément la latence de l'un des chemins. Ceci est dû à l'attente bilatérale entre les fonctions appartenant à des chemins différents. Cependant, pour ce cas de transactions non-linéaires non-synchronisées, il est possible que le partitionnement à base de transactions aboutisse à la solution optimale lorsque celle-ci n'est pas distribuée et lorsque la sémantique de synchronisation considérée est de type input N-of-N / output 1-of-N.

– **Cas 3 :** pour le cas de systèmes de transactions linéaires de taille supérieure à $1$ ou de transactions non-linéaires quelle que soit leur forme, l'approche PLMNE intégrale/H1 fournit des solutions moins performantes par rapport au critère de moyenne des latences puisque la propagation de la gigue et les changements de contexte sont fortement présents. Pourtant, hormis les systèmes de fonctions indépendantes, nous avons constaté que l'approche PLMNE intégrale/H1 permet pour d'autres cas particuliers de trouver la solution optimale. Reprenons par exemple le système de la figure 4.6. Si l'on considère que les fonctions $f_1$ et $f_2$ sont représentées par une seule fonction qui a pour temps d'exécution la somme des WCETs de $f_1$ et $f_2$ et pareillement pour $f_3$ et $f_4$ et $f_5$ et $f_6$ alors dans ce cas, la solution apportée par l'approche PLMNE intégrale/H1 est la solution optimale. Prenons un autre exemple composé de trois fonctions où la première fonction envoie le même signal à deux autre fonctions (voir le deuxième exemple pour le cas 2 du tableau 4.1). Cet exemple représente un autre cas particulier pour lequel PLMNE intégrale/H1 trouve la solution optimale : le regroupement de n'importe quel couple de fonctions dépendantes sur la même tâche n'est en effet pas autorisé par la nature du signal qui doit être soit local à une tâche soit échangé entre deux tâches. Par ailleurs, le partitionnement des deux fonctions indépendantes sur la même tâche ne représente pas la solution optimale. Dans notre expérimentation, ces derniers cas se sont produits pour $12$ instances parmi les $350$, soit un taux de $3,42\%$.
– **Cas 4 :** les deux approches (PLMNE intégrale/H1 et PLMNE intégrale/H2) ont échoué dans l'obtention de la solution optimale pour $208$ instances parmi les $350$, soit un taux de $59,42\%$. Généralement, les $208$ instances représentent des systèmes

constitués de transactions non-linéaires non-synchronisées ou d'un mélange de transactions linéaires, non-linéaires synchronisées, non-linéaires non-synchronisées et|ou linéaires de taille égale à 1. Principalement, lorsque le système est composé de transactions non-linéaires non-synchronisées, l'approche PLMNE intégrale/H1 est plus pessimiste que PLMNE intégrale/H2. Ceci est dû au coût élevé de la propagation de la gigue en comparaison avec le coût de l'attente bilatérale.

Afin de montrer l'écart entre la solution retournée par chaque approche fixant le partitionnement et la solution optimale, nous avons représenté les taux de pessimisme sur la figure 4.7 (sur la figure, la solution optimale est atteinte au point zéro de l'axe des ordonnées). Pour la lisibilité de la figure, nous représentons graphiquement le résultat pour uniquement cinq instances de chaque dimension considérée. Nous constatons à partir de cette figure que l'approche PLMNE intégrale/H1 est jusqu'à $81,35\%$ plus loin de la solution optimale, tandis que l'approche PLMNE intégrale/H2 l'est de $66,11\%$ au pire des cas. L'écart peut être plus important pour l'approche PLMNE intégrale/H1 comme il peut l'être pour l'approche PLMNE intégrale/H2. Il est indépendant de la taille du système. Comme dit précédemment, les deux approches arrivent parfois à atteindre la solution optimale. Nous rappelons que ces résultats sont fortement liés aux systèmes considérés.

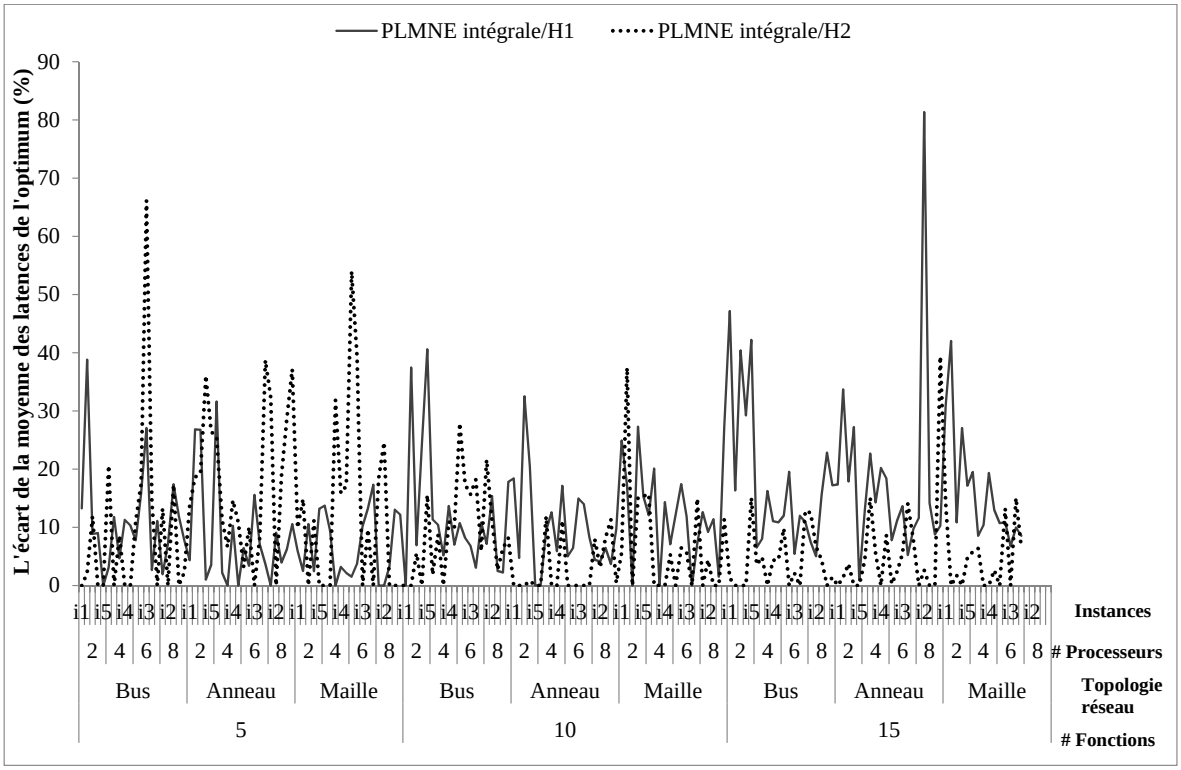



FIGURE 4.7 – Taux de pessimisme des approches faisant des hypothèses sur le partitionnement

Après avoir étudié les taux de pessimisme par rapport à la solution optimale, nous nous sommes intéressés à l'évaluation de l'approche en deux étapes (PLMNE 2-étapes). Pour cela, nous comparons, d'une part, les résultats des approches PLMNE intégrale, PLMNE 2-étapes et PLMNE intégrale/H1, et d'autre part, ceux des approches PLMNE intégrale, PLMNE 2-étapes et PLMNE intégrale/H2. Dans la première comparaison, l'approche PLMNE 2-étapes considère comme configuration initiale la solution retournée par l'approche PLMNE intégrale/H1. Elle est notée dans ce cas par "PLMNE 2-étapes|PLMNE intégrale/H1". Il en est de même pour "PLMNE 2-étapes|PLMNE intégrale/H2" qui signifie que l'approche PLMNE 2-étapes est basée sur la solution renvoyée par PLMNE intégrale/H2 dans la deuxième étude comparative. Ici l'approche en deux étapes est évaluée uniquement pour une seule itération de la boucle externe, pour plusieurs itérations les évaluations sont données plus loin dans ce chapitre (voir les parties 4.3.1.4 et 4.3.1.5).

À partir de cette expérimentation, nous avons déduit que sur les $350$ instances, l'approche PLMNE 2-étapes|PLMNE intégrale/H1 atteint la moyenne des latences optimale pour $279$ instances, soit un taux de $79,71\%$, tandis que l'approche PLMNE intégrale/H1 atteint l'optimal uniquement pour $31$ instances c'est-à-dire $8,85\%$. Similairement, pour l'approche PLMNE 2-étapes|PLMNE intégrale/H2 nous déduisons que parmi les $350$ instances celle-ci est optimale sur $231$ d'entre eux soit $65,62\%$, alors que l'approche PLMNE intégrale/H2 l'est seulement pour $130$ instances, soit $37,14\%$. De ce fait, l'approche PLMNE 2-étapes a amélioré d'une façon entière, c'est-à-dire jusqu'à l'obtention de la valeur optimale, les résultats de $248$ instances pour PLMNE intégrale/H1 et de $101$ pour PLMNE intégrale/H2.

L'écart avec la solution optimale pour cette expérimentation est donné par les figures 4.8 et 4.9. Au point zéro des figures, l'approche atteint la solution optimale.

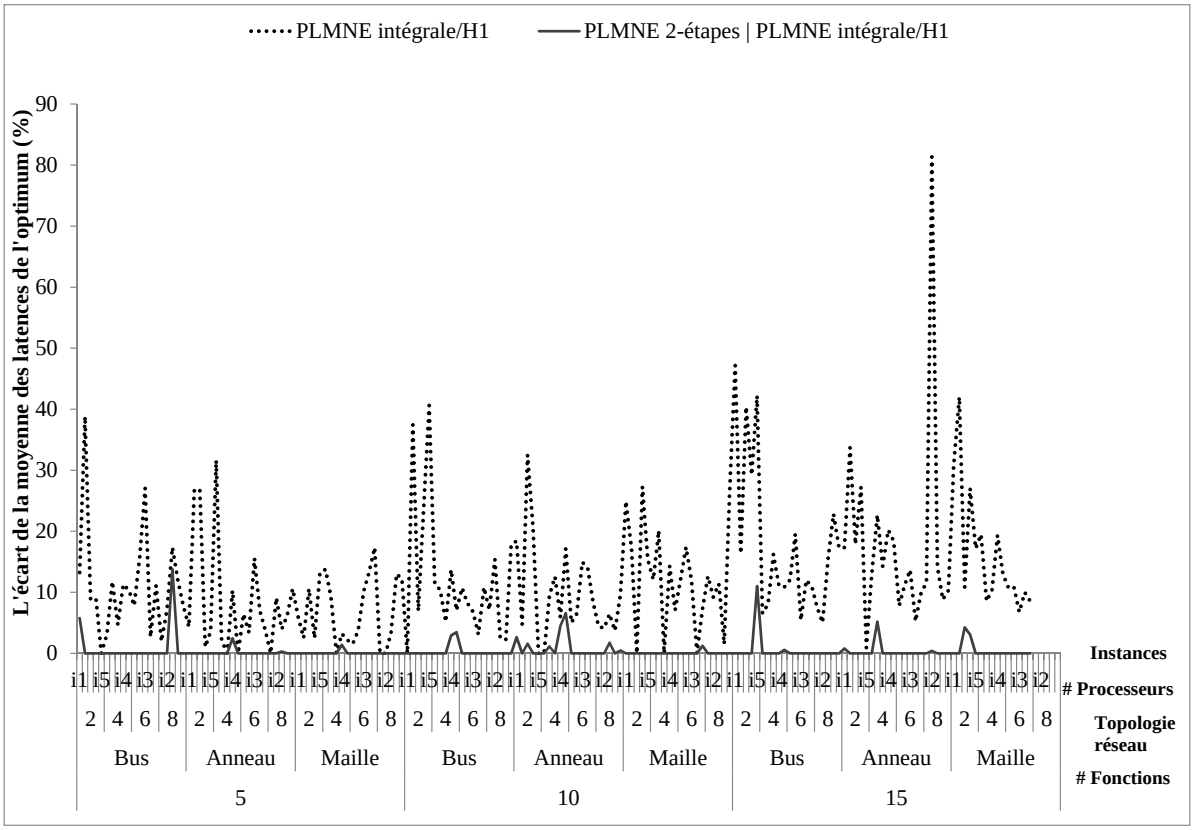



FIGURE 4.8 – Taux de pessimisme des approches PLMNE intégrale/H1 et PLMNE 2-étapes|PLMNE intégrale/H1

Nous constatons donc clairement que le taux de pessimisme de l'approche PLMNE intégrale/H1 est important pour la majorité des instances, il atteint jusqu'à $81,35\%$. Par ailleurs, l'approche PLMNE 2-étapes|PLMNE intégrale/H1 atteint un taux maximal de $13,77\%$ (voir la figure 4.8), elle est donc meilleure.

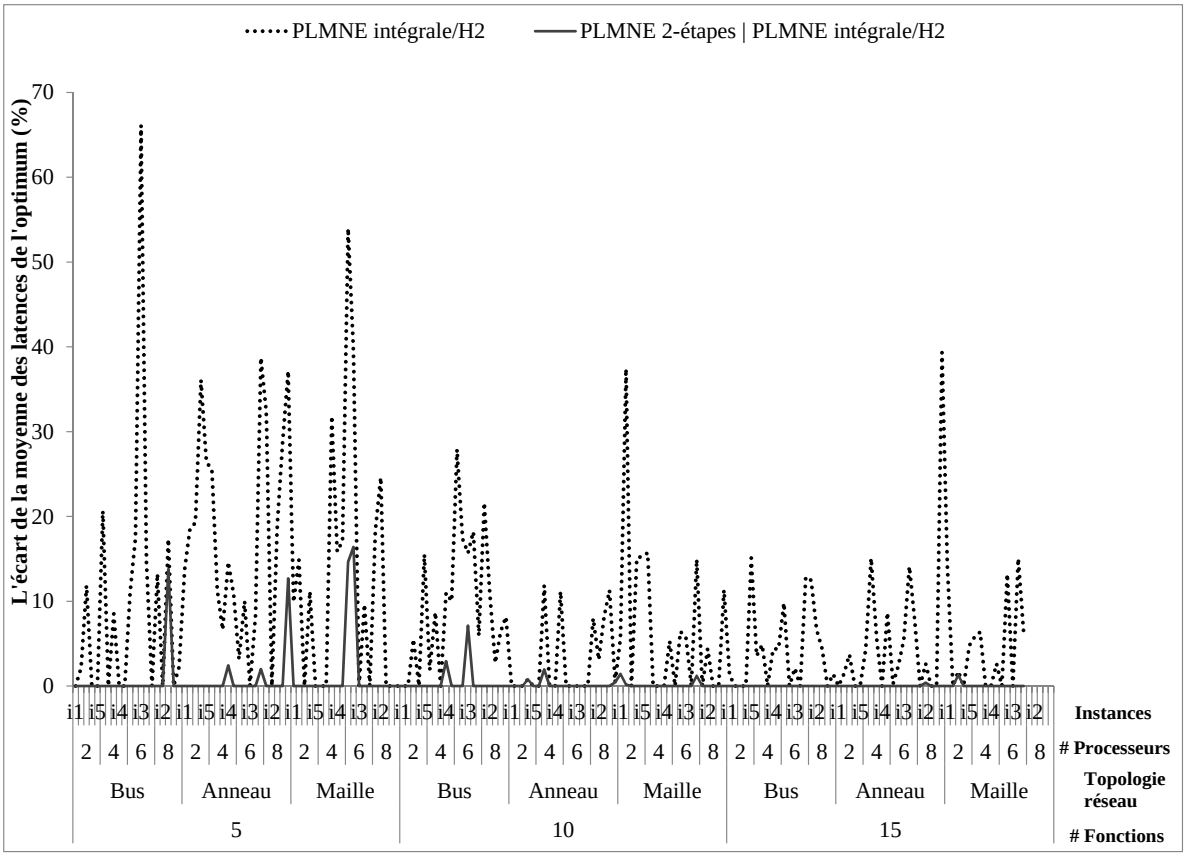



FIGURE 4.9 – Taux de pessimisme des approches PLMNE intégrale/H2 et PLMNE 2-étapes|PLMNE intégrale/H2

Les résultats de la figure 4.9 montrent que le taux de pessimisme de l'approche PLMNE intégrale/H2 arrive jusqu'à $66,11\%$ alors que celui de l'approche PLMNE 2-étapes|PLMNE intégrale/H2 ne dépasse pas $16,41\%$.

En conclusion, grâce à sa boucle interne, l'approche en deux étapes améliore considérablement la solution produite par les approches existantes fixant le partitionnement selon les stratégies à base de fonctions (H1) et à base de transactions (H2). Les taux d'amélioration fournis par l'approche PLMNE 2-étapes pour chacune des solutions apportées par PLMNE intégrale/H1 et PLMNE intégrale/H2 sont représentés respectivement sur les figures 4.10 et 4.11. Un taux d'amélioration de $100\%$ indique que l'approche PLMNE 2-étapes a réussi, à partir d'une solution non optimale produite par PLMNE intégrale/H1 ou PLMNE intégrale/H2, à trouver une solution dont la moyenne des latences est optimale. Un taux d'amélioration de $0\%$ implique que soit aucune amélioration n'est fournie par l'approche PLMNE 2-étapes, soit que la configuration initiale présente déjà une solution optimale. Prenons le cas de $50\%$, l'approche PLMNE 2-étapes améliore la solution fournie par l'approche PLMNE intégrale/H1 de $50\%$ : ceci implique que la distance entre la moyenne des latences de la solution renvoyée par PLMNE 2-étapes|PLMNE intégrale/H1 et la valeur optimale représente la moitié de la distance entre la moyenne des latences de la solution renvoyée par PLMNE intégrale/H1 et la valeur optimale.

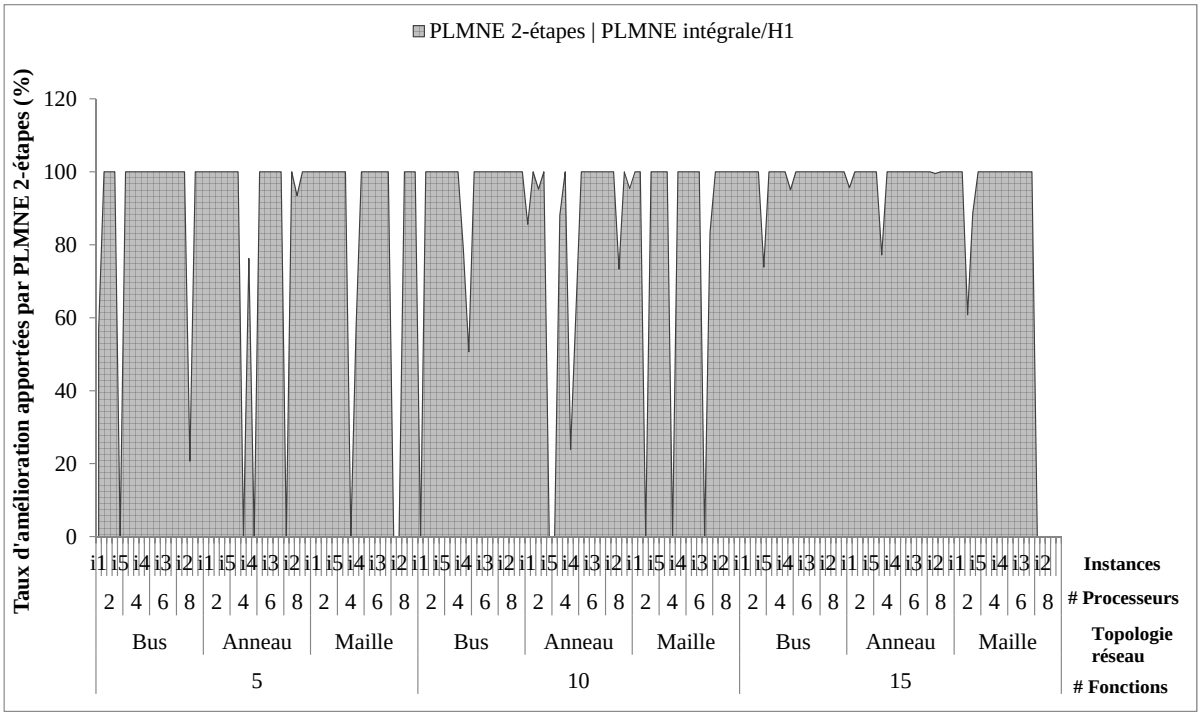



FIGURE 4.10 – Taux d'amélioration fournis par l'approche PLMNE 2-étapes|PLMNE intégrale/H1

À partir des résultats de cette expérimentation (voir la figure 4.10), nous déduisons que l'approche PLMNE 2-étapes a pu augmenter la qualité de la solution produite par PLMNE intégrale/H1 pour 319 instances parmi les 350 soit $91,14\%$ des cas. Il n'y a pas eu d'amélioration pour 31 instances. Ces 31 instances représentent la situation pour laquelle l'approche PLMNE intégrale/H1 a déjà retourné la solution optimale. De même, l'approche PLMNE 2-étapes a amélioré le résultat obtenu par PLMNE intégrale/H2 pour 212 instances soit un taux de $60,57\%$ (voir la figure 4.11). Parmi les 138 instances dont l'amélioration n'a pas eu lieu, 130 instances sont déjà optimale depuis le départ. Par conséquent, l'approche PLMNE 2-étapes s'est limitée à retourner un optimum local pour 8 instances des 350.

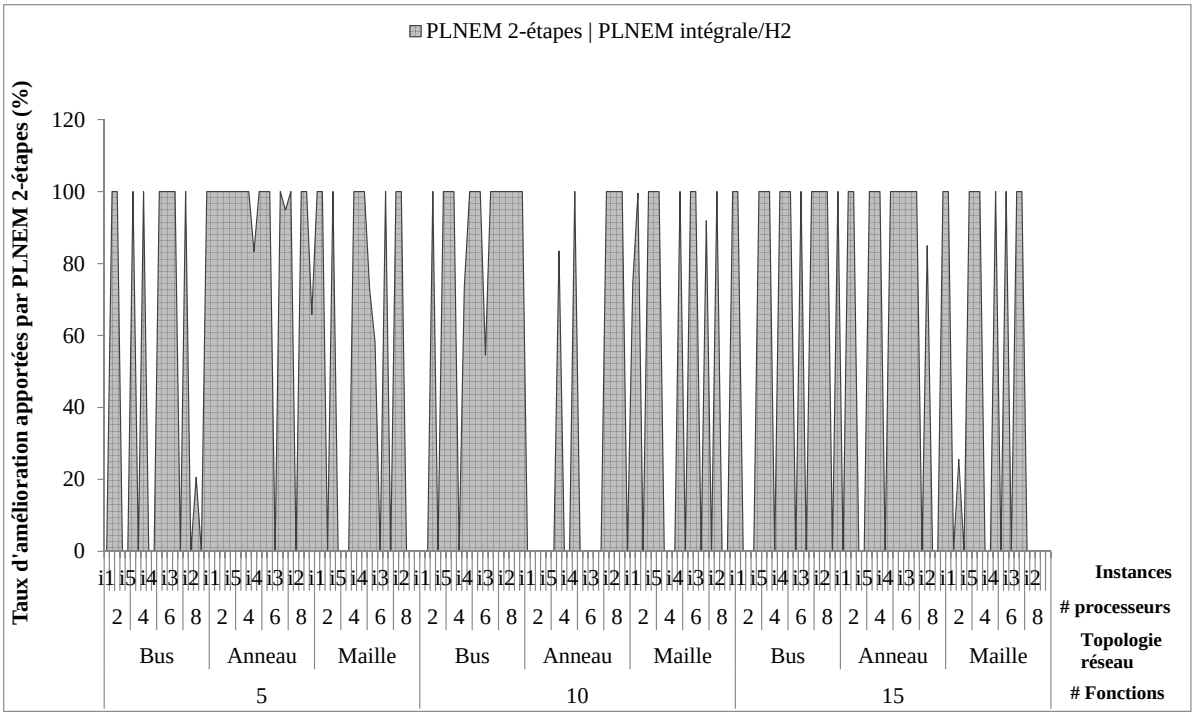



FIGURE 4.11 – Taux d'amélioration fournis par l'approche PLMNE 2-étapes|PLMNE intégrale/H2

Le tableau 4.2 récapitule et détaille les taux d'améliorations menées par l'approche en deux étapes. Sur $20,28\%$ des cas, le taux d'amélioration pour la solution retournée par PLMNE intégrale/H1 est compris entre $1\%$ et $99\%$. Pour PLMNE intégrale/H2, le taux d'amélioration appartenant à cet intervalle est obtenu sur $31,71\%$ des instances considérées.

| | $=0\%$ | | $\in [1\%, 100\%[$ | $=100\%$ |
|---|---|---|---|---|
| | Déjà optimale | Pas d'amélioration | | |
| PLMNE intégrale/H1 | $\frac{31}{350} = 8,85\%$ | $\frac{0}{350} = 0\%$ | $\frac{71}{350} = 20,28\%$ | $\frac{248}{350} = 70,85\%$ |
| PLMNE intégrale/H2 | $\frac{130}{350} = 37,14\%$ | $\frac{8}{350} = 2,28\%$ | $\frac{111}{350} = 31,71\%$ | $\frac{101}{350} = 28,85\%$ |

Tableau 4.2 – Résumé des taux d'améliorations apportées par l'approche PLMNE 2-étapes pour PLMNE intégrale/H1 et PLMNE intégrale/H2

#### 4.3.1.3 Comparaison avec les approches faisant des hypothèses sur le placement

Après notre comparaison avec les approches faisant des hypothèses sur le partitionnement, nous sommes intéressés à la comparaison de nos approches de déploiement avec les approches cherchant à trouver un partitionnement de fonctions/signaux sur les tâches/messages ainsi que leur ordonnancement sous l'hypothèse que le placement des fonctions/signaux sur les processeurs/bus est donné. Pour cela, nous avons considéré une approche en deux étapes dont la première étape est chargée de fixer le placement des fonctions sur les processeurs et celui des signaux sur les bus de communication. Ensuite, la deuxième étape consiste à utiliser le placement trouvé dans l'étape précédente pour optimiser le partitionnement et l'ordonnancement des fonctions et des signaux au sein de la même ressource et ceci par rapport au temps de réponse. Dans cette approche à deux étapes, au moment du placement, ni le concept de tâche ni celui de priorité est présent. Par conséquent, il n'est pas possible d'optimiser le placement des fonctions par rapport au temps de réponse. De ce fait, les approches existantes [15, 13, 167, 168, 158] se basent sur d'autres métriques d'optimisation que celle du temps de réponse. Généralement ces dernières représentent des métriques simples qui sont intermédiaires au temps de réponse et qui n'ont pas besoin de manipuler les tâches et les priorités. Dans cette expérimentation, nous considérons que le placement dans la première étape est optimisé vis-à-vis des métriques intermédiaires suivantes :

- **La contention au sein des ressources de calcul et de communication :** c'est une métrique qui vise à diminuer la concurrence à l'intérieur des ressources. Son principe consiste à minimiser la durée qu'une fonction doit attendre en raison de l'exécution d'une autre fonction parallèle pour accéder à la ressource. Deux fonctions sont parallèles si elles ne présentent pas une relation de précédence dans le modèle fonctionnel. La minimisation de cette métrique permet de baisser le coût d'interférences entre les fonctions concurrentes et donc de réduire la valeur des latences. La contention pour les ressources de calcul peut être obtenue par :

$$\forall c_c \in C : cont_{c_c} = \max_{f_i \in F} \sum_{f_j \in Parallel(f_i)} \left\lceil \frac{P_{f_i}}{P_{f_j}} \right\rceil \omega_{f_j,c_c}, \text{ tel que } f_i \text{ et } f_j \text{ sont placées sur } c_c \tag{4.1}$$

  $Parallel(f_i)$ indique l'ensemble des fonctions concurrentes. Pour les ressources de communication, la formule de contention est similaire à celle des ressources de calcul.
- **La charge sur le réseau de communication :** elle consiste à réduire le coût de communication inter processeurs. En raison de la propagation de la gigue qui est fortement présente dans la communication à distance et qui a un poids important sur le calcul de latence, la minimisation de cette métrique favorise un modèle de déploiement optimisé pour cette dernière. La charge sur le réseau est estimée par la formule :

$$\sum_{\beta_i \in \beta} \sum_{s_i \in \Phi} \frac{\omega_{s_i,\beta_i}}{P_{s_i}}, \text{ tel que } s_i \text{ est transmis sur le bus } \beta_i \tag{4.2}$$

- **L'extensibilité des fonctions [16, 137] :** elle représente l'augmentation maximale des WCETs des fonctions pour laquelle la capacité maximale d'utilisation des ressources de calcul n'est pas dépassée. La maximisation de cette métrique aide à équilibrer la charge des processeurs et par conséquent à réduire la concurrence au sein de chaque ressource. Ceci permet donc de guider la solution vers un déploiement optimisé par rapport au temps de réponse. L'extensibilité ($\lambda_{f_i}$) d'une fonction $f_i$ est calculée par la

formule ci-après :

$$\lambda_{f_i} = \eta_{c_c} - \sum_{f_j \in F} \frac{\omega_{f_j,c_c}}{P_{f_j}}, \text{ tel que } f_i \text{ et } f_j \text{ sont placées sur le processeur } c_c \tag{4.3}$$

L'approche en deux étapes adressant le placement en se basant sur des métriques intermédiaires est notre référence pour cette expérimentation. Afin de réaliser cette étude comparative, nous avons formulé chaque étape de cette approche par un PLMNE (Voir la formulation complète dans l'annexe A.1). Dans la première étape, la formulation est multiobjectif. Nous avons donc fixé le poids d'optimisation pour chaque métrique intermédiaire à $0,5$. Dans la suite de cette partie, nous faisons référence à cette approche par PLMNE 2-étapes/H3.

- **Objectifs :** nos objectifs via cette expérimentation consistent à :
  - évaluer l'impact d'un placement dirigé par des métriques intermédiaires sur la performance de l'algorithme d'exploration ;
  - étudier l'influence des métriques intermédiaires sur la qualité du déploiement vis-à-vis de la minimisation de la moyenne des latences ;
  - montrer l'intérêt de considérer la métrique de la moyenne des latences dès la phase de placement. Cet objectif est atteint par l'étude des améliorations que l'approche PLMNE 2-étapes peut apporter à l'approche PLMNE 2-étapes/H3.

- **Comparaison des performances :** cette partie vise à répondre au premier objectif mentionné ci-dessus, c'est-à-dire l'estimation du temps d'exécution du solveur et de la quantité de mémoire consommée par ce dernier pour chacune des approches PLMNE intégrale et PLMNE 2-étapes/H3. De même que pour l'expérimentation précédente, nous avons considéré l'étude du temps d'exécution du solveur par rapport aux trois paramètres principaux qui sont le nombre de fonctions, le nombre de processeurs et la topologie du réseau. Les figures 4.12, 4.13 et 4.14 présentent les résultats par rapport à chacun de ces paramètres sous forme d'un diagramme en boîte. La description des éléments de ce graphe est la même que celle présentée dans la partie précédente (partie 4.3.1.2).

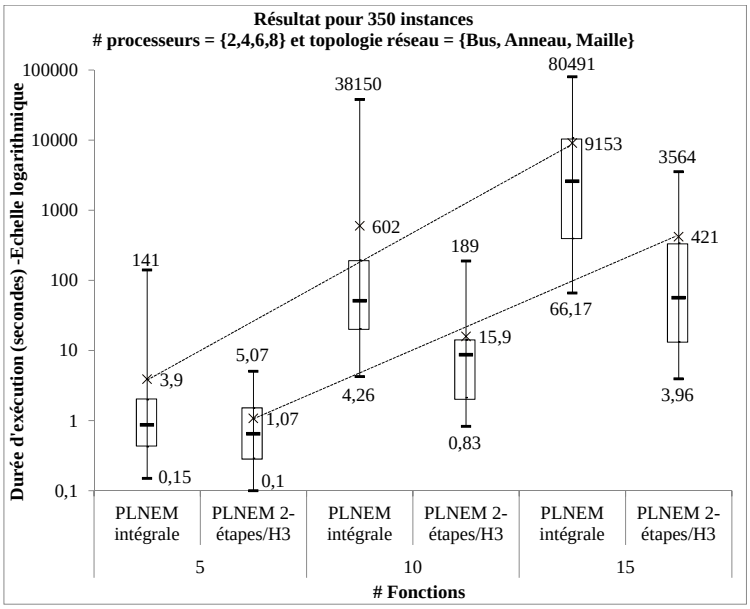



FIGURE 4.12 – Influence du nombre de fonctions sur le temps de calcul des approches PLMNE intégrale et PLMNE intégrale/H3

La figure 4.12 montre que pour l'ensemble des instances considérées, le temps d'exécution moyen du solveur pour chacune des deux approches PLMNE intégrale et PLMNE 2-étapes/H3, est en moyenne proche d'une relation proportionnelle par rapport au nombre de fonctions.

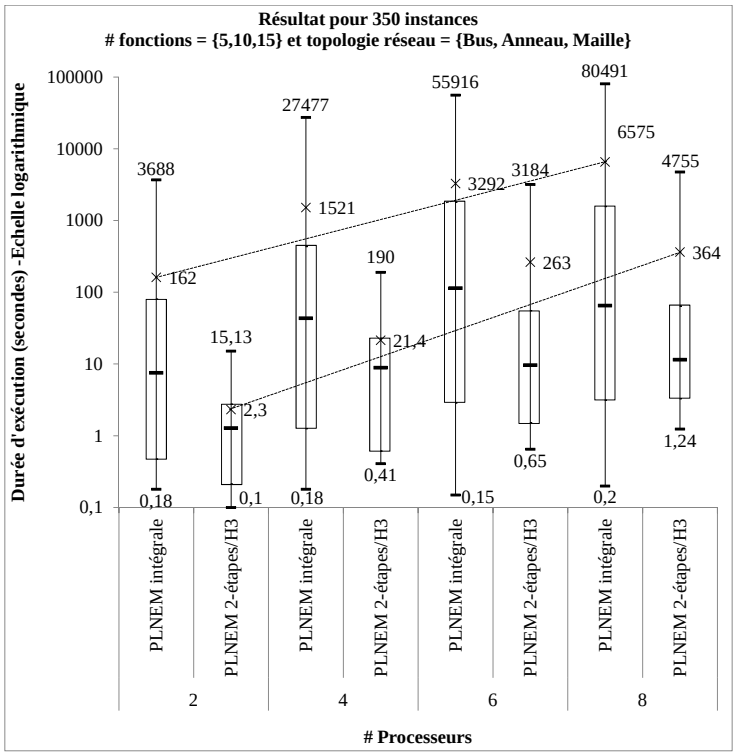



FIGURE 4.13 – Influence du nombre de processeurs sur le temps de calcul des approches PLMNE intégrale et PLMNE intégrale/H3

De même sur la figure 4.13, nous observons qu'au fur et à mesure que le nombre de processeurs augmente, le temps d'exécution moyen du solveur pour les deux approches PLMNE intégrale et PLMNE 2-étapes/H3 s'accroît.

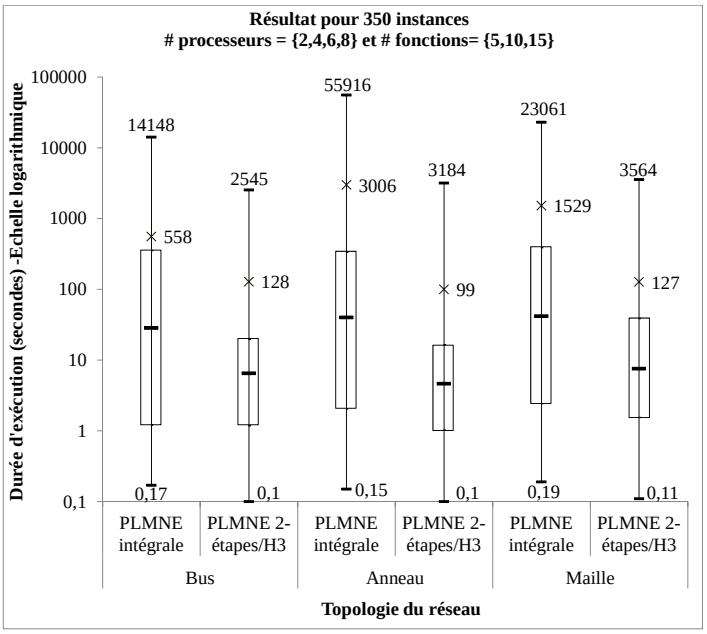



FIGURE 4.14 – Influence de la topologie du réseau sur le temps de calcul des approches PLMNE intégrale et PLMNE intégrale/H3

Toutefois, sur la figure 4.14, la différence entre les temps d'exécution du solveur par rapport aux différentes topologies considérées pour l'approche PLMNE 2-étapes/H3 est négligeable. **Nous pouvons donc en déduire que la topologie du réseau n'a pas une influence majeure sur le temps d'exécution du solveur en comparaison avec le nombre de fonctions et de processeurs.**

Sur les trois figures 4.12, 4.13 et 4.14, nous constatons clairement que le temps d'exécution du solveur pour l'approche PLMNE 2-étapes/H3 où le placement est optimisé par rapport aux critères intermédiaires au temps de réponse est beaucoup plus intéressant que celui pour l'approche PLMNE intégrale. Ce résultat provient, d'une part, du fait que la phase de placement (généralement la phase la plus coûteuse) dans l'approche PLMNE 2-étapes/H3 est beaucoup plus rapide puisqu'elle représente une simple formulation dans laquelle le calcul compliqué des temps de réponse n'est pas présent. Par ailleurs, l'espace de recherche pour chaque étape de l'approche PLMNE 2-étapes/H3 est largement réduit par rapport à celui considéré par le solveur dans l'approche PLMNE intégrale. Cependant, l'approche PLMNE 2-étapes/H3 montre ses limites quant à la qualité du déploiement obtenu. Ceci est illustré ci-après dans la partie suivante.

Enfin, pour les différents paramètres considérés, les graphes représentant la performance en mémoire ont la même forme que ceux des figures 4.12, 4.13 et 4.14.

- **Comparaison de la qualité des solutions :** cette partie répond aux deux autres objectifs mentionnés, à savoir l'estimation de la qualité de déploiement lorsque le placement est optimisé par rapport à des critères intermédiaires au temps et l'évaluation de l'apport de l'approche en deux étapes. Par rapport à ce dernier objectif, l'expérimentation se focalise sur l'évaluation de l'intérêt de la boucle interne de l'approche (comme c'est le cas pour l'expérimentation de la partie 4.3.1.2) en même temps que celui de la stratégie de décomposition adoptée par l'approche. La figure 4.15 présente une

étude comparative des approches PLMNE intégrale, PLMNE 2-étapes/H3 et PLMNE 2-étapes par rapport à la moyenne des latences. Rappelons que plus la valeur de cette moyenne est petite meilleur est le résultat. "PLMNE 2-étapes|PLMNE 2-étapes/H3" exprime que l'approche PLMNE 2-étapes considère comme configuration initiale la solution retournée par l'approche PLMNE 2-étapes/H3. Là aussi, nous évaluons l'approche en deux étapes uniquement pour une seule configuration initiale (voir les parties 4.3.1.4 et 4.3.1.5 pour son évaluation par rapport à plusieurs configurations initiales) .

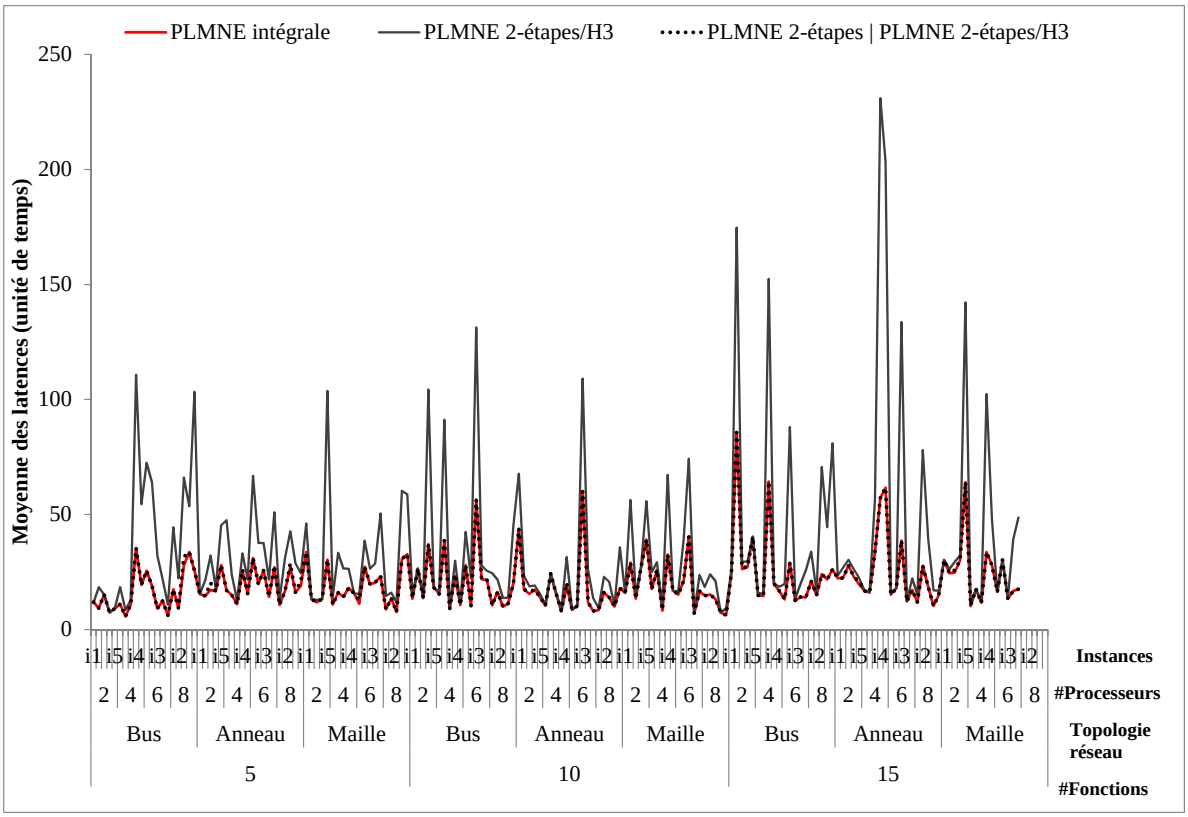



FIGURE 4.15 – Qualité des solutions de déploiement obtenues par l'approche PLMNE 2-étapes/H3 et leurs comparaisons avec les solutions renvoyées par l'approche en deux étapes

D'un côté, nous remarquons que l'écart entre la solution obtenue par l'approche PLMNE 2-étapes/H3 et la solution optimale est considérable pour la plupart des instances. Cette dernière a atteint l'optimal uniquement pour 32 instances sur les 350, soit un taux de $9,14\%$ tandis que notre approche a atteint la solution optimale pour 316 instances. Après avoir analysé, une à une, les solutions obtenues, nous avons pu déduire que, bien que pour certaines situations chacune des métriques intermédiaires, indépendamment des autres, puisse diriger l'approche PLMNE 2-étapes/H3 vers la solution optimale de déploiement, la combinaison des trois métriques intermédiaires réduit souvent la qualité de la solution. Par exemple, supposons qu'un système soit composé uniquement de transactions linéaires où le nombre de transactions est égal au nombre de processeurs, alors la solution optimale qui consiste à placer chaque transaction sur un processeur peut être obtenue en tenant compte des deux métriques intermédiaires qui sont la charge sur le réseau et la contention des ressources. Cependant, l'ajout du poids de l'extensibilité peut guider le placement vers une solution distribuée d'où la dégradation de la qualité de déploiement durant

la deuxième étape. Parfois la considération d'une seule métrique intermédiaire peut être mauvaise pour la solution du déploiement. Considérons un autre exemple pour lequel le système est constitué d'une transaction non linéaire non synchronisée et deux processeurs reliés par un bus. L'optimisation du placement dans la première étape de l'approche PLMNE 2-étapes/H3 par rapport à la charge sur le réseau permet de trouver une solution non distribuée tandis que l'optimisation par rapport à la contention des ressources permet d'obtenir une solution distribuée. Pour cet exemple, dans la deuxième étape, selon l'importance du coût de la propagation des gigues lié à la communication sur le réseau par rapport au coût d'interférences dues à la contention au sein des processeurs, uniquement une des métriques est capable de diriger le déploiement vers la solution optimale et pas les deux. D'un autre côté, nous constatons que l'approche PLMNE 2-étapes est capable d'augmenter la qualité de la solution retournée par l'approche PLMNE 2-étapes/H3 jusqu'à l'obtention d'une solution optimale.

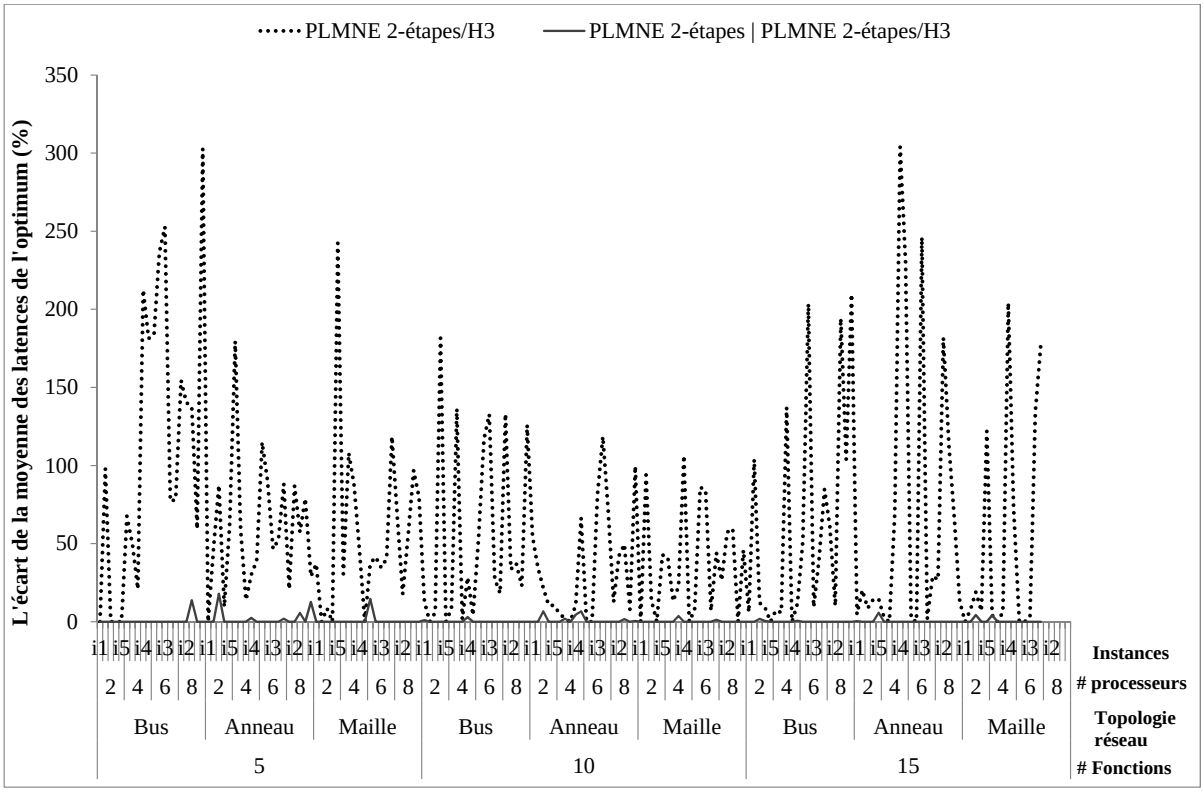



FIGURE 4.16 – Taux de pessimisme des approches PLMNE 2-étapes/H3 et PLMNE 2-étapes|PLMNE 2-étapes/H3

À partir de la figure 4.16, nous estimons le taux de pessimisme des approches PLMNE 2-étapes/H3 et PLMNE 2-étapes|PLMNE 2-étapes/H3. Nous déduisons que le taux de pessimisme pour PLMNE 2-étapes/H3 arrive jusqu'à $303,95\%$ pendant que le taux de pessimisme de l'approche en deux étapes n'excède pas les $17,9\%$.

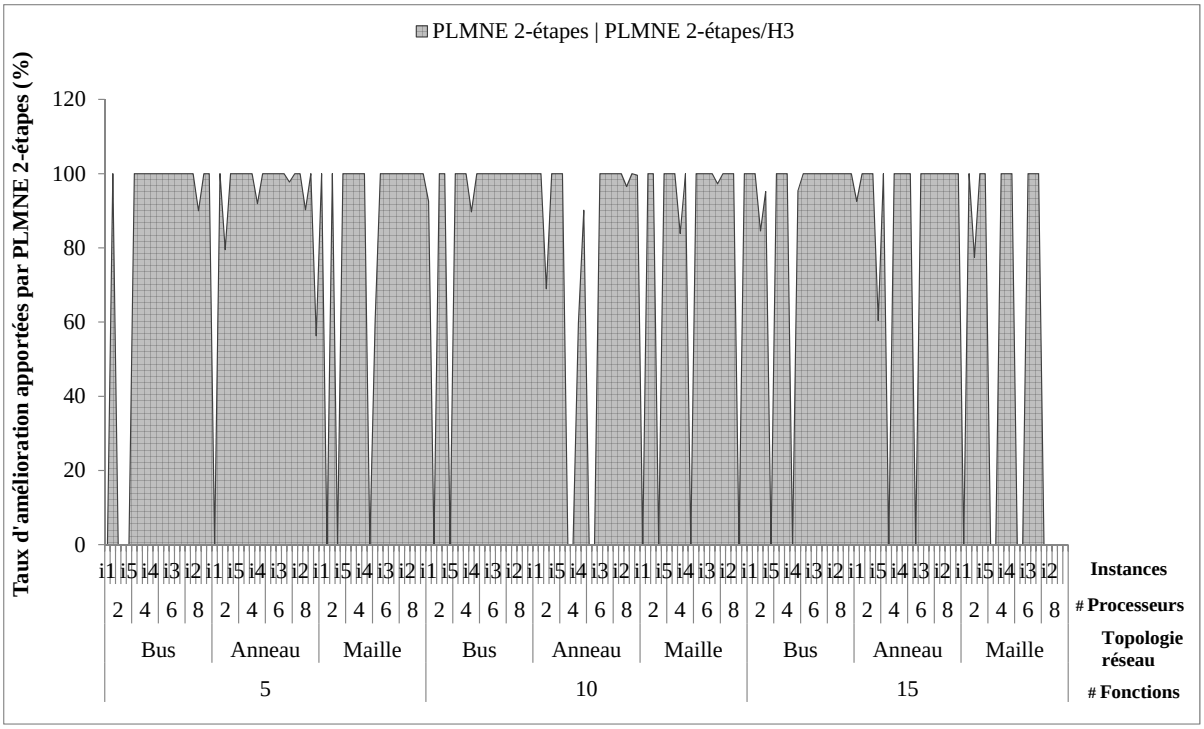



FIGURE 4.17 – Taux d'amélioration fournis par l'approche PLMNE 2-étapes|PLMNE 2-étapes/H3

Nous présentons la figure 4.17 et le tableau 4.3 pour montrer les améliorations amenées par l'approche en deux étapes par rapport à l'approche PLMNE 2-étapes/H3. À partir de la figure, nous constatons que l'approche PLMNE 2-étapes a apporté une amélioration pour la majorité des solutions produite par PLMNE 2-étapes/H3. D'après notre expérimentation, cette amélioration concerne $312$ instances parmi les $350$ soit un taux de $89,14\%$. Néanmoins, pour les $38$ instances restantes il n'y a pas eu d'amélioration. Il est à noter que $32$ instances parmi les $38$ présentaient déjà la solution optimale.

| | $= 0\%$ | | $\in [1\%, 100\%[$ | $= 100\%$ |
|---|---|---|---|---|
| | Déjà optimale | Pas d'amélioration | | |
| PLMNE 2-étapes/H3 | $\frac{32}{350} = 8,85\%$ | $\frac{6}{350} = 1,71\%$ | $\frac{28}{350} = 8\%$ | $\frac{284}{350} = 81,14\%$ |

Tableau 4.3 – Résumé des taux d'améliorations apportées par l'approche PLMNE 2-étapes pour PLMNE 2-étapes/H3

**Bilan :** au final, nous concluons que les approches existantes, c'est-à-dire celles faisant des hypothèses sur le placement ou sur le partitionnement, adressant partiellement le problème de déploiement, sont plus rapides que l'approche PLMNE intégrale. Cependant, notre approche fournit en contrepartie de meilleures solutions de déploiement par rapport au

critère du temps de réponse. En effet, nos approches ont prouvé leur efficacité par rapport à la moyenne des latences. Ceci est dû au fait qu'elles considèrent le problème de placement en même temps que celui du partitionnement et de l'ordonnancement. En outre, notre solution de partitionnement est un compromis entre les deux solutions extrêmes, à savoir le partitionnement à base de fonctions et le partitionnement à base de transactions. Concernant l'approche en deux étapes, son intérêt réside dans le fait qu'elle considère l'optimisation du temps de réponse dès la phase de placement ce qui lui permet de s'orienter vers de bonnes solutions. De plus, la présence de la boucle interne a montré sa contribution à la qualité de la solution finale. Elle a permis d'augmenter les chances de trouver un déploiement de bonne qualité qui peut même être optimal. Dans les deux expérimentations qui suivent, nous nous intéressons à l'estimation globale de l'approche en deux étapes, incluant ainsi la boucle externe.

#### 4.3.1.4 PLMNE intégrale versus déploiement en deux étapes

En raison de sa complexité, l'approche PLMNE intégrale reste limitée à la résolution d'instances de systèmes de taille faible. Nous avons vu précédemment que cette approche n'a pas donné de solution optimale pour un système composé de $15$ fonctions et de $8$ processeurs lorsqu'une topologie en maille est considérée. Cependant, celle-ci reste utile pour évaluer la qualité de la solution de déploiement produite par l'approche en deux étapes sur des instances de taille limitée. L'étude comparative pour cette expérimentation consiste à confronter les deux approches de déploiement, à savoir l'approche PLMNE intégrale et l'approche en deux étapes. Contrairement aux expérimentations précédentes où une seule configuration initiale de partitionnement et d'affectation de priorités est considérée pour la résolution du problème de déploiement par l'approche en deux étapes, dans cette expérimentation, nous avons choisi de considérer cinq configurations initiales. Chacune est générée aléatoirement en tenant compte des relations de précédence. Initialement, pour chaque fonction source dans le modèle fonctionnel, une priorité est générée aléatoirement selon une loi de distribution uniforme discrète. Ensuite, selon cette même loi, une priorité est attribuée à toutes les fonctions qui succèdent à la fonction source de telle sorte que cette priorité soit inférieure ou égale à celle générée pour toutes les fonctions qui précèdent dans la même transaction. Enfin, toutes les fonctions ayant la même priorité sont considérées comme étant une seule tâche.

- **Objectif :** notre but via cette expérimentation est d'étudier l'intérêt qu'apporte l'approche de déploiement en deux étapes par rapport à l'approche PLMNE intégrale.

- **Comparaison des performances :** à travers cette partie nous étudions les performances en termes de ressources consommées par chacune des approches de déploiement proposées dans la présente thèse. Pour cette expérimentation, nous présentons une étude comparative seulement par rapport à la mémoire maximale consommée. Une comparaison vis-à-vis du temps de calcul n'a pas de sens, le temps d'exécution du solveur pour l'approche en deux étapes étant fortement lié au nombre de configuration initiales considérées. La figure 4.18 affiche la moyenne de la mémoire maximale consommée par l'approche PLMNE intégrale et par l'approche en deux étapes (PLMNE 2-étapes avec boucle externe (5) [1]), et ceci pour chaque dimension de système considérée parmi les $84$ décrites dans la partie 4.2.2. La moyenne affichée pour chaque dimension inclut les $10$ instances générées. Comme déjà mentionné

1. Ici le 5 indique le nombre de configurations initiales de partitionnement et d'ordonnancement considérées, et donc le nombre de boucles externes.

précédemment, pour les systèmes de taille moyenne, l'approche PLMNE intégrale n'a pas eu suffisamment de ressource mémoire pour parcourir tout l'espace de recherche lui permettant ainsi de renvoyer la solution optimale. L'approche en deux étapes est capable d'adresser des systèmes de 30 fonctions et de 8 processeurs alors que l'approche PLMNE intégrale n'a pas réussi à traiter un système ayant 15 fonctions et 8 processeurs lorsque la topologie du réseau de communication est en maille. L'expérimentation montre également que l'approche en deux étapes a traité des systèmes de 35 fonctions, mais uniquement lorsque la plateforme d'exécution est composée de deux processeurs. Pour conclure, la consommation mémoire par l'approche en deux étapes est largement réduite. Ceci a permis de considérer des instances de taille plus importante que celle des instances traitées par l'approche PLMNE intégrale.

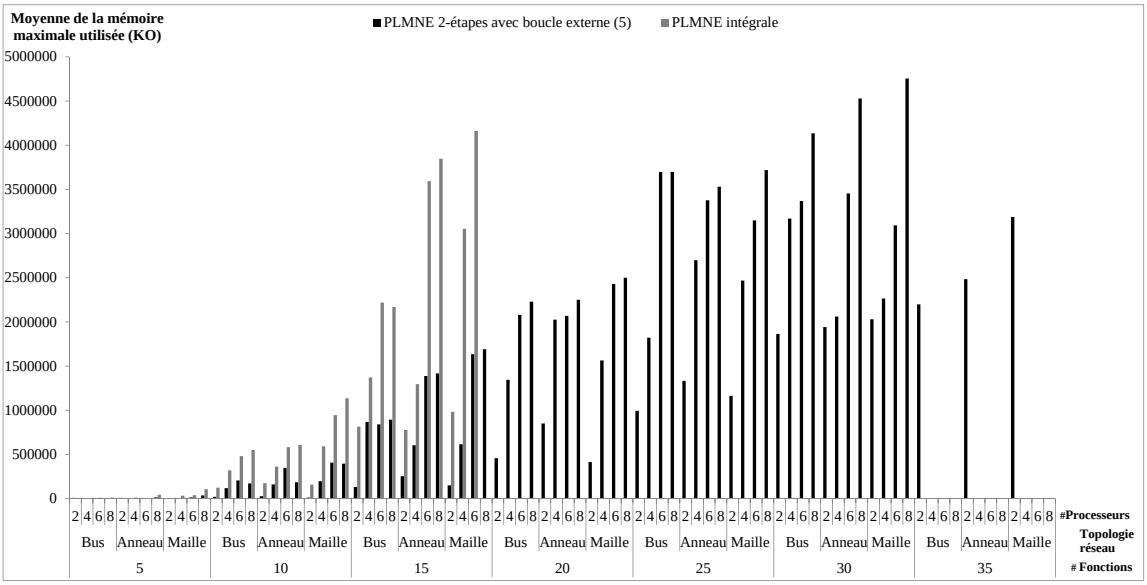



FIGURE 4.18 – Espace mémoire demandé par l'approche PLMNE intégrale et par l'approche en deux étapes

À partir de la figure 4.18, nous constatons également qu'en moyenne la mémoire maximale nécessaire à l'approche PLMNE intégrale pour traiter un système composé de 15 fonctions et de 6 processeurs est plus importante que celle demandée par l'approche PLMNE 2-étapes avec boucle externe (5) pour aborder un système de 25 fonctions et de 8 processeurs. Par ailleurs, nous remarquons que la consommation mémoire est fortement dépendante de l'instance traitée. Par exemple, quelle que soit l'approche considérée, pour deux systèmes ayant la même taille en termes du nombre de processeurs, de bus, de fonctions et de signaux, alors les quantités maximales de mémoire consommée sont différentes. Ceci peut être dû à plusieurs facteurs tels que les paramètres temporels, la structure des transactions et le rapport entre le nombre de transactions et le nombre de processeurs. Néanmoins, comme le montre la figure 4.18, le nombre de fonctions et le nombre de processeurs restent les paramètres dominants influençant la performance en mémoire de chacune des approches.

Jusqu'à présent, l'expérimentation montre que l'approche en deux étapes ne peut pas traiter des systèmes ayant plus de 30 fonctions et 8 processeurs et que l'approche PLMNE intégrale est limitée aux systèmes avec 15 fonctions et 8 processeurs. Or, la présence de contraintes d'allocation permet généralement d'augmenter le passage à

l'échelle de l'approche. Pour illustrer cette situation, nous avons choisi d'appliquer les deux approches à un système tiré de la littérature [142]. La figure 4.19 présente le modèle fonctionnel du système. Pour une raison de simplicité, les signaux sont représentés sur les arcs. Le modèle de la plateforme d'exécution ciblée se compose de 9 processeurs communiquant par un seul bus. La capacité d'utilisation maximale est fixée à 1 pour le bus et pour tous les processeurs. Les processeurs ont une vitesse de calcul différente et sont donc hétérogènes. Les WCETs ainsi que les contraintes d'allocation sont exprimés dans le tableau 4.4. Les valeurs pour chaque vecteur des WCTTs pour tous les signaux sont égales à 10.

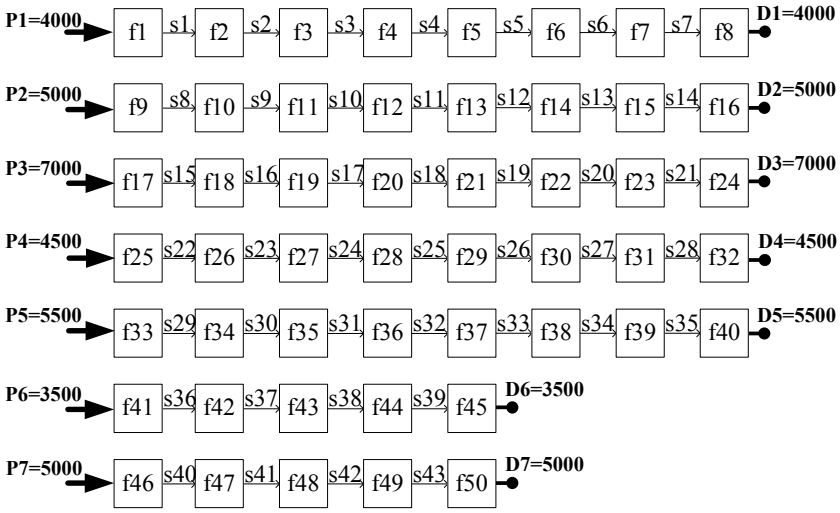



FIGURE 4.19 – Modèle fonctionnel d'un système industriel

| | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ | $c_8$ | $c_9$ | | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ | $c_8$ | $c_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | 48 | 26 | 35 | 50 | 36 | 40 | 45 | 27 | 28 | $f_{26}$ | | | | | | 48 | 36 | | |
| $f_2$ | 36 | 52 | 59 | 34 | 26 | 43 | 50 | 48 | 49 | $f_{27}$ | | | 26 | | 50 | | | | |
| $f_3$ | | | 49 | | | | | | | $f_{28}$ | | 51 | | 30 | | | | | |
| $f_4$ | | | | 50 | | | | | | $f_{29}$ | | | | | | 32 | 39 | | |
| $f_5$ | | | | | 46 | | | | | $f_{30}$ | 50 | 48 | 36 | 26 | 35 | 26 | 35 | 50 | 51 |
| $f_6$ | | | | | | 50 | | | | $f_{31}$ | 31 | 39 | | 44 | | | | | |
| $f_7$ | | | | | | | 51 | | | $f_{32}$ | | | 52 | | 27 | | | | |
| $f_8$ | 26 | 35 | 29 | 42 | 50 | 37 | 20 | 52 | 53 | $f_{33}$ | 44 | 26 | 35 | 26 | 35 | 30 | 28 | 49 | 50 |
| $f_9$ | 50 | 40 | 31 | 29 | 33 | 46 | 37 | 29 | 30 | $f_{34}$ | | | | | 30 | 40 | 50 | | |
| $f_{10}$ | | | 52 | | | | | | | $f_{35}$ | | | | | 49 | 39 | 29 | | |
| $f_{11}$ | 40 | 39 | 50 | 33 | 36 | 39 | 43 | 41 | 42 | $f_{36}$ | | | | | 53 | 25 | 40 | | |
| $f_{12}$ | | | | | 50 | | | | | $f_{37}$ | | 32 | 40 | 48 | | | | | |
| $f_{13}$ | | | | 39 | | | | | | $f_{38}$ | | 30 | 50 | 45 | | | | | |
| $f_{14}$ | | | | | | | 38 | | | $f_{39}$ | | 54 | 28 | 36 | | | | | |
| $f_{15}$ | | | | | | 52 | | | | $f_{40}$ | 50 | 28 | 39 | 47 | 44 | 35 | 26 | 35 | 36 |
| $f_{16}$ | 33 | 46 | 37 | 29 | 35 | 29 | 42 | 50 | 51 | $f_{41}$ | 52 | 28 | 35 | 26 | 35 | 50 | 45 | 33 | 34 |
| $f_{17}$ | | 32 | | 55 | | 29 | | | | $f_{42}$ | | | | | 48 | 38 | 28 | | |
| $f_{18}$ | | | 51 | | 26 | | | | | $f_{43}$ | | | | | 30 | 50 | 40 | | |
| $f_{19}$ | 51 | 42 | 26 | 35 | 50 | 36 | 33 | 29 | 30 | $f_{44}$ | | | | | 26 | 38 | 48 | | |
| $f_{20}$ | | 50 | | 28 | | | 36 | | | $f_{45}$ | | 51 | 30 | 42 | | | | | |
| $f_{21}$ | 35 | 29 | 42 | 42 | 26 | 35 | 50 | 54 | 55 | $f_{46}$ | | 34 | 49 | 30 | | | | | |
| $f_{22}$ | | | | | | 29 | 50 | | | $f_{47}$ | 28 | 39 | 47 | 50 | 26 | 35 | 50 | 51 | 52 |
| $f_{23}$ | | | | | | 48 | 32 | | | $f_{48}$ | | 36 | 29 | 53 | | | | | |
| $f_{24}$ | | | 31 | | 45 | | | | | $f_{49}$ | 51 | 44 | 33 | 28 | 50 | 26 | 39 | 33 | 34 |
| $f_{25}$ | 53 | 26 | 35 | 50 | 36 | 26 | 35 | 50 | 51 | $f_{50}$ | 37 | 50 | 52 | 32 | 29 | 27 | 36 | 40 | 41 |

Les cases contenant un chiffre désignent le WCET des fonctions sur le processeur sous-jacent et les cases vides indiquent que les fonctions ne peuvent pas être allouées sur le processeur sous-jacent

Tableau 4.4 – Vecteur des WCETs et contraintes d'allocation

Pour cet exemple, **malgré la présence des contraintes d'allocation, l'approche PLMNE intégrale n'a retourné aucune solution** après $86450$ secondes. Cependant, pour une résolution en deux étapes, nous avons considéré cinq configurations initiales générées de la façon décrite au début de cette partie. L'approche en deux étapes a utilisé $2472444$ KO de mémoire maximale lui permettant ainsi de retourner une solution avec une moyenne des latences égale à $757,714$ unité de temps. Cette solution est montrée sur la figure 4.20. Nous rappelons que les nuances de gris représentent la solution de placement pour les fonctions et que les solutions de partitionnement pour les fonctions et pour les signaux sont respectivement indiquées par les noms des tâches et ceux des messages mentionnés au-dessus d'eux. La tâche (le message) avec le plus important indice indique la tâche (le message) ayant la plus faible priorité. En analysant la solution obtenue et en se basant sur notre expérience (indiquant qu'indépendamment des contraintes d'allocation, la solution optimale pour cet exemple consiste à placer chaque transaction sur un processeur différent puis à partitionner toute la transaction en une seule tâche), nous déduisons que celle-ci est de bonne qualité pour deux raisons. Tout d'abord, l'approche a essayé de trouver un placement des fonctions de telle sorte que la communication sur le réseau

est minimale. La seconde raison est que le partitionnement de toutes les fonctions dépendantes appartenant au même processeur se fait sur la même tâche. Ceci permet de réduire de coût de la propagation des gigues. **Nous pouvons noter que l'approche en deux étapes a atteint l'objectif de résoudre des systèmes ayant la même taille que celle des cas d'études utilisés dans la littérature**.

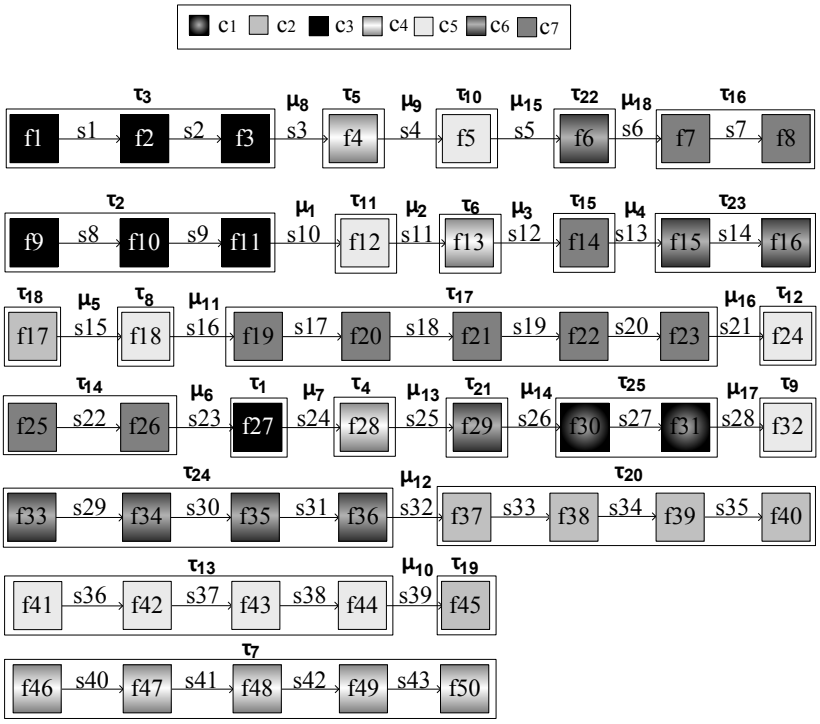



FIGURE 4.20 – Solution de déploiement pour le système décrit dans la figure 4.19

Après avoir étudié la performance de l'approche en deux étapes et illustré son passage à l'échelle lorsque des contraintes d'allocation sont présentes, nous nous sommes intéressés par la suite à l'évaluation de la qualité de la solution de déploiement renvoyée par cette approche.

- **Comparaison de la qualité des solutions :** pour évaluer la qualité de la solution retournée par l'approche en deux étapes, nous comparons les solutions de déploiement obtenues avec les solutions optimales retournées par l'approche PLMNE intégrale. Cette étude comparative peut être réalisée uniquement pour les $350$ instances où la solution optimale est trouvée. L'approche en deux étapes a atteint la solution optimale pour $324$ instances parmi les $350$, soit un taux de $92,57\%$. Une partie des résultats est présentée sur la figure 4.21. Nous constatons que le taux maximal de pessimisme atteint par l'approche PLMNE 2-étapes avec boucle externe (5) est de $13,77\%$, ce qui reste toutefois un résultat acceptable.

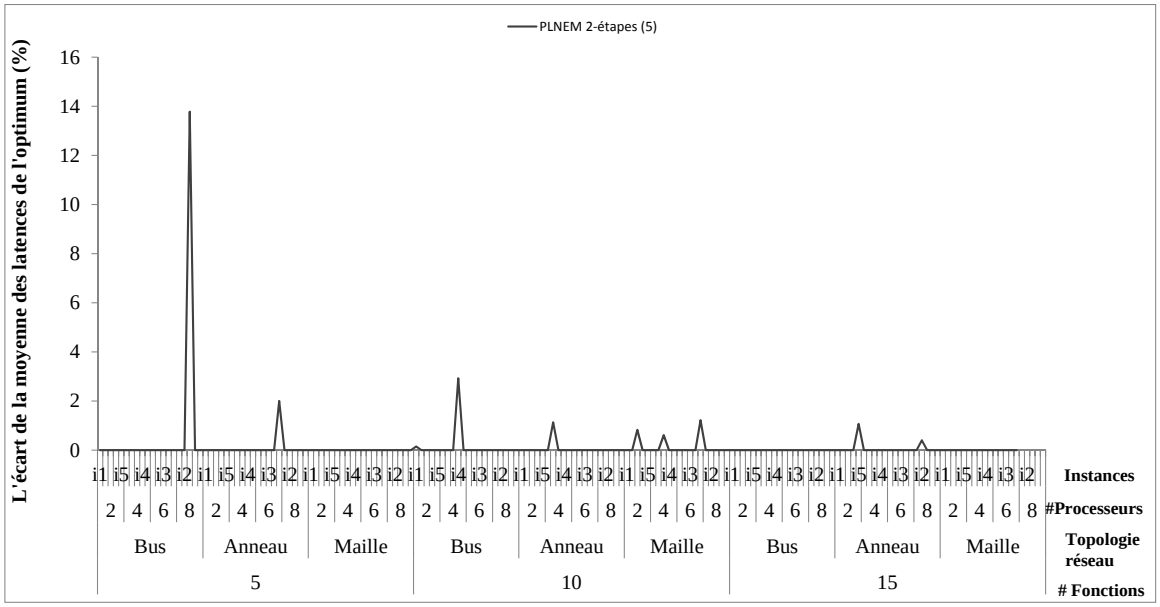



FIGURE 4.21 – Taux de pessimisme de l'approche en deux étapes

#### 4.3.1.5 Évaluation des améliorations itératives

Dans la partie précédente, nous avons vu que l'approche en deux étapes a permis de trouver des solutions de déploiement intéressantes. Dans cette partie, nous étudions de près cette approche afin de déterminer sa spécificité. Pour cela, nous nous intéressons à l'évaluation des améliorations itératives de l'approche en deux étapes en estimant l'apport de chaque boucle de l'approche à la solution de déploiement retournée. D'un autre côté, cette expérimentation permet de se positionner par rapport aux approches d'optimisation existantes basées sur les stratégies de décomposition et d'améliorations itératives. Nous considérons ici l'ensemble des instances générées conformément à ce qui est présenté dans la partie 4.2.2. Nous rappelons que nous avons $840$ instances. Pour chaque instance, nous générons cinq configurations de partitionnement et d'ordonnancement selon la démarche décrite dans la partie 4.3.1.4.

- **Objectif :** l'idée derrière cette expérimentation est de montrer la valeur ajoutée des boucles, interne et externe, dans l'approche en deux étapes. L'objectif principal est donc de mesurer les améliorations itératives apportées par chacune des boucles.

  Du point de vue du critère de performance en termes de ressources, il est évident que les boucles de l'approche en deux étapes ainsi que le nombre d'itérations n'influencent pas sa consommation mémoire. D'autre part, le temps d'exécution du solveur pour l'approche en deux étapes est fortement lié au nombre d'itérations considérées. En d'autres termes, plus nous considérons de configurations initiales, plus le temps d'exécution est grand. En contrepartie, comme le montre la partie suivante, les chances pour que la qualité de la solution soit meilleure sont plus élevées.

- **Comparaison de la qualité des solutions :** cette partie permet de mettre en évidence les caractéristiques de l'approche en deux étapes et de répondre à l'objectif mentionné ci-dessus. La figure 4.22 représente une comparaison entre la qualité des solutions renvoyées par l'approche en deux étapes (PLMNE 2-étapes avec boucle externe (5)) et celle des solutions obtenues par cette même approche dans les cas où : (i) une seule itération de la boucle interne est considérée (PLMNE 2-étapes sans boucle interne),

puis (ii) une seule configuration initiale est considérée (PLMNE 2-étapes sans boucle externe). Sur l'axe des ordonnées, nous représentons la moyenne de la moyenne des latences pour chaque dimension considérée. Rappelons que nous avons 84 dimensions et que pour chacune la moyenne est faite par rapport aux 10 instances générées.

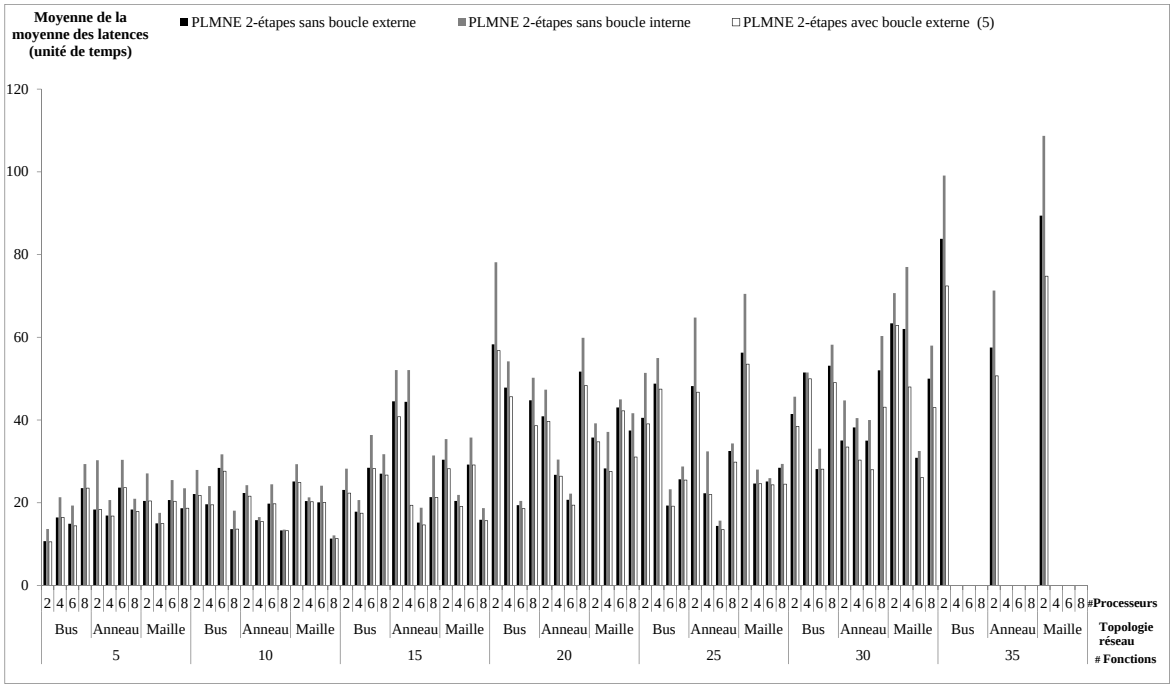



FIGURE 4.22 – L'impact des boucles, interne et externe, sur la qualité des solutions de déploiement obtenue par l'approche en deux étapes

À partir de la figure 4.22, nous constatons qu'en moyenne PLMNE 2-étapes sans boucle interne renvoie les solutions les moins performantes en comparaison avec les autres solutions obtenues par PLMNE 2-étapes avec boucle externe (5) et PLMNE 2-étapes sans boucle externe, et ceci pour toutes les dimensions considérées. Nous constatons également que pour certains cas, les solutions retournées par PLMNE 2-étapes sans boucle externe sont d'une qualité identique à celle des solutions renvoyées par PLMNE 2-étapes avec boucle externe (5). Ce résultat est tout à fait attendu car la boucle externe de l'approche en deux étapes, contrairement à la boucle interne, ne garantit pas une orientation vers de meilleures solutions, le choix des configurations initiales étant aléatoire. Autrement dit, à une itération donnée de la boucle externe, nous pouvons choisir une configuration initiale qui nous conduit vers une solution de qualité identique ou moins performante que celle trouvée à l'itération précédente. Ceci n'empêche pas que pour d'autres cas l'approche PLMNE 2-étapes avec boucle externe (5) a apporté de meilleures solutions comparées aux solutions obtenues avec PLMNE 2-étapes sans boucle externe. Par ailleurs, nous remarquons que les améliorations apportées par chaque boucle sont différentes d'une instance à une autre et sont totalement aléatoires. Ceci est dû au fait qu'elles dépendent non seulement de l'instance considérée en termes de taille, structure et paramètres temporels, mais aussi des configurations initiales générées.

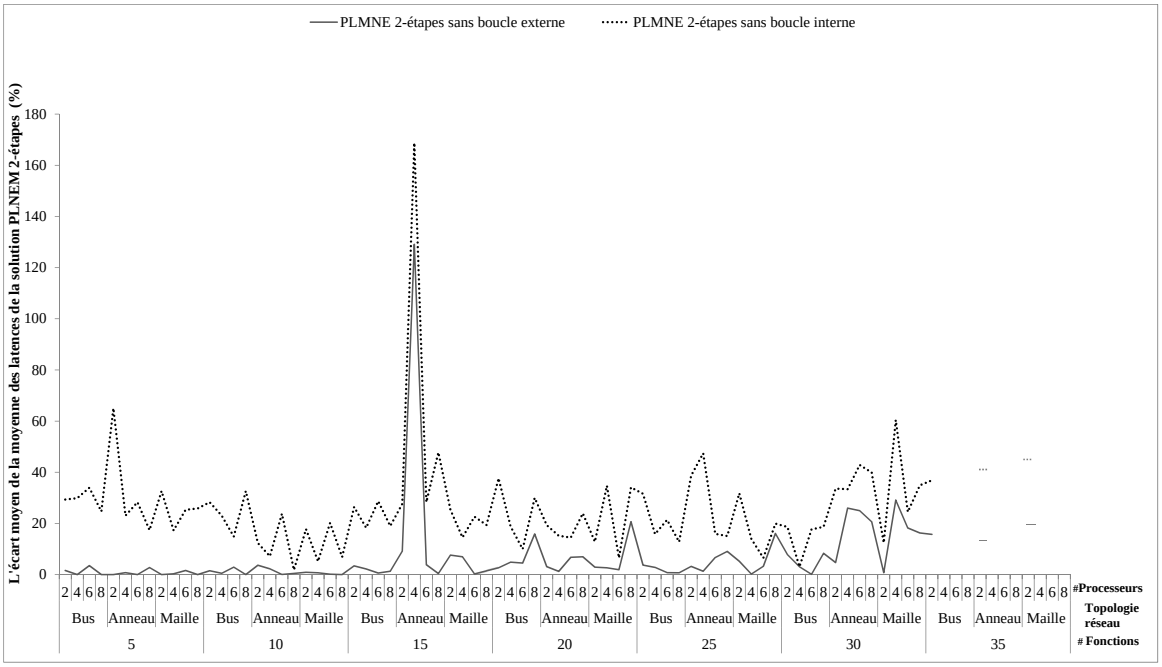



FIGURE 4.23 – Taux de pessimisme de approche PLMNE 2-étapes sans la considération des boucles

La figure 4.23 présente le taux de pessimisme moyen qui peut être évité par l'approche en deux étapes grâce à chacune des boucles. Nous pouvons noter qu'un taux de pessimisme égal à $0\%$ implique que la moyenne de la moyenne des latences de la solution adjacente est égale à celle de la solution renvoyée par l'approche en deux étapes (PLMNE 2-étapes avec boucle externe (5)). Nous en déduisons clairement que pour les instances considérées, le taux de pessimisme moyen de PLMNE 2-étapes sans boucle interne atteint jusqu'à $168,72\%$, alors que celui de PLMNE 2-étapes sans boucle externe arrive à $129,1\%$. S'ajoutant à cela, une seule itération de la boucle externe suffit parfois à atteindre une solution de bonne qualité. Le tableau 4.5 présente en pourcentage la fréquence des améliorations apportées par chacune des boucles dans l'approche PLMNE 2-étapes, et ceci en considérant les $750$ instances des $840$ où l'approche a retourné une solution. Cette fréquence est spécifiée pour différents taux d'amélioration. Sur $16\%$ des cas une seule itération de la boucle interne a suffi pour obtenir une solution de bonne qualité. Cette fréquence est plus élevée pour la boucle externe où elle atteint $38,53\%$. Par ailleurs, la boucle interne a apporté des améliorations pour $81,6\% + 2,4\%$ des instances, tandis que la boucle externe l'a réalisé pour $60,53\% + 0,93\%$.

| | $= 0\%$ | $\in [1\%, 100\%]$ | $> 100\%$ |
|---|---|---|---|
| Boucle interne | $\frac{120}{750} = 16\%$ | $\frac{612}{750} = 81,6\%$ | $\frac{18}{750} = 2,4\%$ |
| Boucle externe | $\frac{289}{750} = 38,53\%$ | $\frac{454}{750} = 60,53\%$ | $\frac{7}{750} = 0,93\%$ |

Tableau 4.5 – Fréquence des améliorations apportées par chacune des boucles dans l'approche PLMNE 2-étapes

En conclusion, la boucle interne influence considérablement la qualité des solutions renvoyées par l'approche en deux étapes en comparaison avec la boucle externe. D'autre part, la boucle externe permet d'augmenter les chances de trouver un déploiement de bonne qualité, parfois optimal, et de contourner les optimums locaux.

Rappelons que dans la littérature, les approches d'optimisation basées sur les stratégies de décomposition et d'améliorations itératives ne bénéficient pas des avantages offerts par celle proposée dans cette thèse. Par exemple dans [16], le problème d'optimisation considéré est traité en deux étapes avec une boucle externe, mais la boucle interne n'est pas présente. Par ailleurs, le problème adressé dans [10] est résolu en plusieurs étapes de telle sorte que chaque étape est dédiée à un sous problème. Néanmoins, l'approche de résolution ne contient aucune boucle permettant une amélioration itérative.

Dans les parties précédentes, nous avons présenté une étude statistique permettant d'évaluer les approches proposées dans le cas où les activations du système sont pilotées par les données. Nous avons montré leur intérêt en comparaison avec les approches de la littérature et nous avons étudié leurs caractéristiques. Dans la partie suivante, nous nous intéressons à appliquer ces approches à un cas réel de l'automobile.

#### 4.3.1.6 Cas d'étude industriel : système automobile BBW

L'objectif de cette partie est de montrer l'applicabilité des approches proposées dans le domaine industriel de l'automobile.

**Description :** dans le domaine de l'automobile, la technologie "Drive-By-Wire" ou "X-By-Wire" [169] remplace les pièces mécaniques et hydrauliques traditionnelles dans un système de contrôle par un système électronique composé de contrôleurs (ECU), capteurs et actionneurs. Ceci permet d'améliorer la sécurité et de fournir des fonctionnalités avancées dans les véhicules. Dans cette partie, afin d'illustrer notre approche de déploiement, nous nous basons sur l'exemple présenté dans le cadre du projet MAENAD [170], à savoir le système "Brake-By-Wire" (BBW). La fonctionnalité de ce système est la suivante (voir la figure 4.24) : lorsque le conducteur appuie sur la pédale, les actionneurs de frein doivent appliquer une force de freinage sur la roue qui est en relation avec l'angle de la pédale. En plus de cette fonctionnalité basique de freinage, le système est muni de la fonctionnalité ABS dans le sens où si la vitesse d'une roue est significativement plus petite que la vitesse estimée du véhicule, la force de freinage est réduite sur cette roue jusqu'à ce que sa vitesse soit comparable à la vitesse estimée du véhicule [171]. Un tel système est considéré comme critique et les propriétés temporelles ont un impact direct sur la sécurité de la voiture. Par exemple, la durée entre la demande du conducteur et l'activation des actionneurs de roues peut être considérée comme un indicateur de performance du système. Si celle-ci dépasse une certaine limite, le conducteur pourrait perdre totalement le contrôle de la voiture.

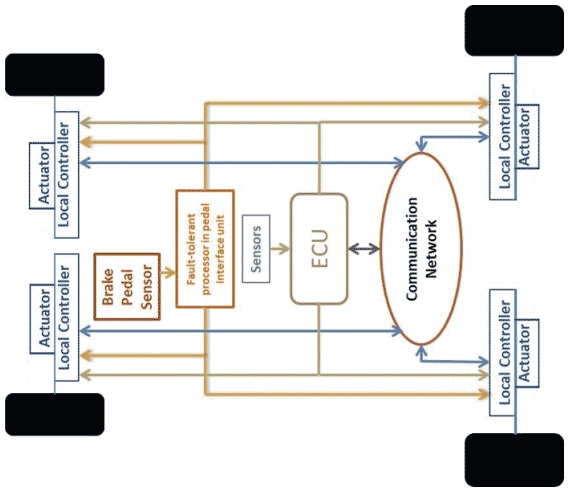



FIGURE 4.24 – Une représentation schématique d'un système "Brake-By-Wire" [3].

Le modèle fonctionnel de ce système, présenté par la figure 4.25, peut visuellement être divisé en trois parties. Dans la première partie, un pourcentage de la force de freinage maximale demandée par le conducteur est calculé à partir du voltage lié à l'angle de la pédale (les fonctions "Brake pedal sensor" et "Brake pedal LDM[2]"). Ensuite, une force de freinage basique est calculée sur chaque roue sans incorporer la fonctionnalité ABS. Celle-ci permet d'estimer la vitesse du véhicule en se basant sur la vitesse des roues (les fonctions "Brake torque calculator" et "Global brake controler"). Dans la deuxième partie, la fonctionnalité ABS est ajoutée sur chaque roue. Á cette étape, la force de freinage désirée dans l'étape précédente peut être ajustée en fonction de la différence entre la vitesse estimée du véhicule et la vitesse des roues (les fonctions "ABS"). Dans la dernière partie, la force de freinage demandée se traduit en un voltage électrique appliqué à chacune des roues (les fonctions "Brake actuator LDM" et "Brake actuator"). La période d'activation, les WCETs, les WCTTs, ainsi que les échéances de bout-en-bout sont indiquées sur la figure 4.25.

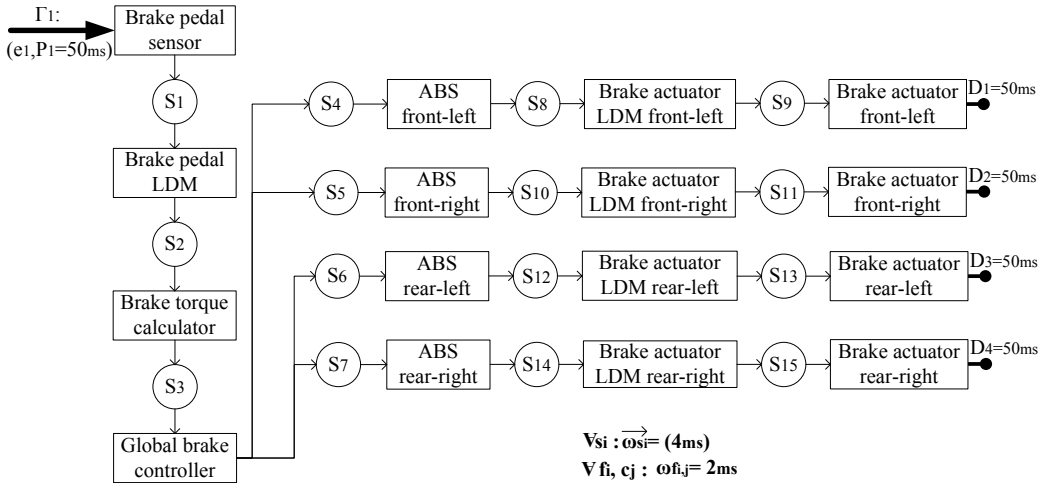



FIGURE 4.25 – Modèle fonctionnel du système BBW.

L'architecture matérielle du système BBW (figure 4.26) consiste à employer un actionneur près de chaque roue pour réaliser la pression de freinage. Chaque actionneur est gouverné

2. Local Device Manager

par une unité de contrôle électronique connectée à la pédale de frein et à l'unité centrale. L'interaction du conducteur via la pédale de frein est capturée par un capteur. L'ensemble des éléments du système communiquent par l'intermédiaire d'un CAN bus. La capacité d'utilisation maximale est fixée à 1 pour le bus de communication et tous les éléments de la plateforme.

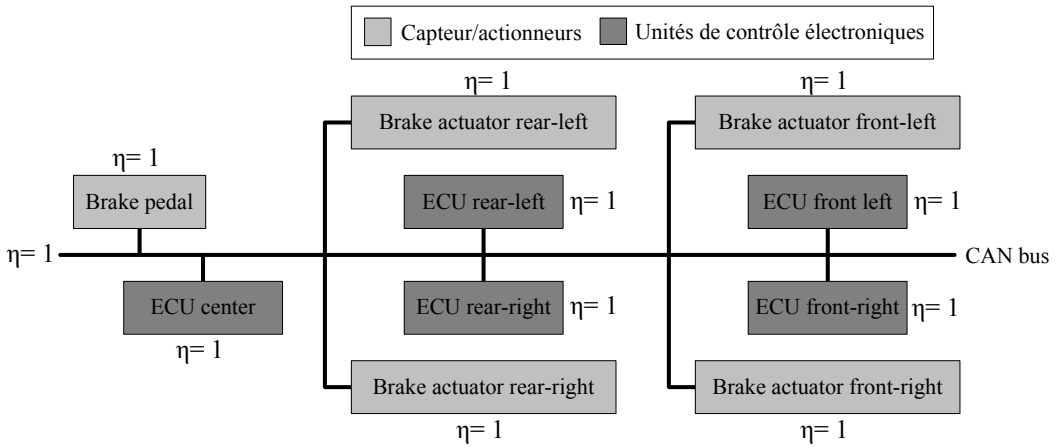



FIGURE 4.26 – Modèle de plateforme d'exécution pour le système BBW.

Face à ce système, le problème pour les concepteurs, est de trouver une association des fonctionnalités du système à la plateforme d'exécution, de prédire les performances du système par rapport au temps de réaction des actionneurs aux changements des positions de la pédale et de prendre en considération toutes les contraintes d'allocations fixées qui consistent à placer la fonction "Brake pedal sensor" sur le capteur et toutes les fonctions "Brake actuator" sur les actionneurs sous-jacents. Ici le modèle d'entrée est bien un modèle fonctionnel et non pas un modèle de tâches. Pour répondre à cette question, nous avons appliqué l'approche de déploiement présentée ci-avant. Les activations sont ici pilotées par les données. À titre illustratif, nous avons choisi de considérer la sémantique de synchronisation 'Input N-of-N/output N-of-N'.

**Solution :** après 89800 secondes, l'approche PLMNE intégrale a retourné la solution d'un déploiement optimal par rapport à la moyenne des latences pour le système BBW décrit précédemment. Pour arriver à ce résultat l'approche a dû utiliser au maximum 2999492 KO de mémoire. La figure 4.27 montre la solution de déploiement obtenue. Le placement des fonctions et des signaux est illustré par différentes nuances de gris. Les signaux transmis sur le réseau ont tous la même nuance puisqu'ils appartiennent au même bus. Nous pouvons constater que pour cet exemple, les contraintes d'allocation ont augmenté le passage à l'échelle de l'approche intégrale, elles lui ont permis d'aborder un système avec 16 fonctions et 10 processeurs.

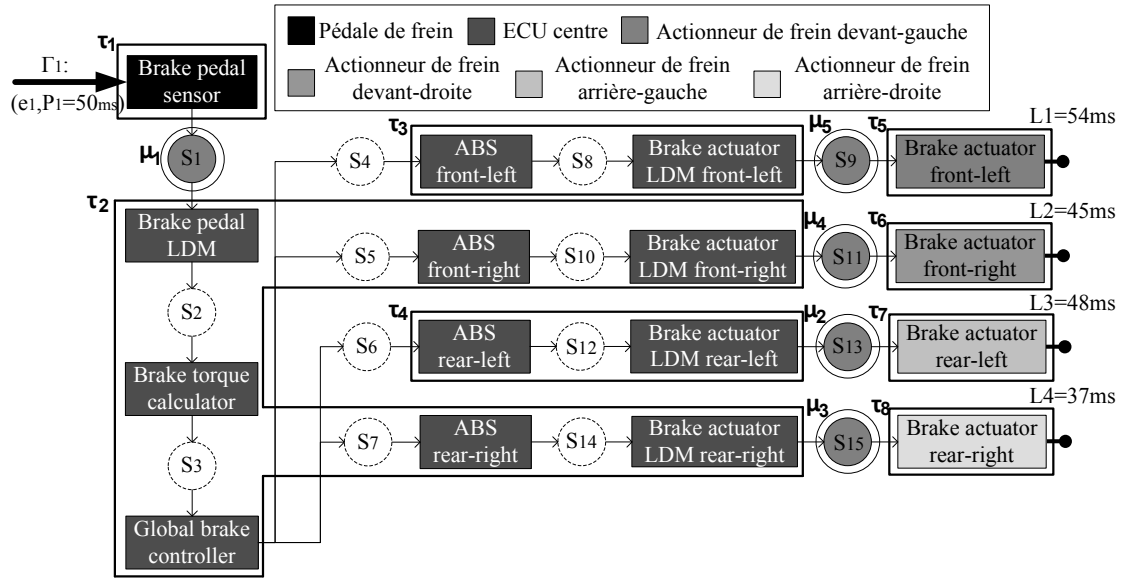



FIGURE 4.27 – Solution optimale pour le déploiement du système BBW

Afin d'appliquer l'approche de déploiement en deux étape au système BBW présenté, nous avons considéré deux configurations initiales pour l'étape $PO$. Dans la première configuration chaque fonction est partitionnée sur une tâche différente et chaque signal est transmis par un message distinct. Par la suite, les priorités des tâches et des messages sont générées aléatoirement selon une loi uniforme discrète tout en considérant les relations de précédence. Tandis que dans la deuxième configuration toutes les fonctions appartenant à la transaction sont partitionnées sur la même tâche. La résolution du système BBW par l'approche de déploiement en deux étapes a conduit à la même solution apportée par l'approche intégrale (voir la figure 4.27) mais après seulement $528$ secondes et $487716$ KO de mémoire maximale consommée. Nous constatons ainsi que la performance en temps et en mémoire de l'approche en deux étapes est largement réduite en comparaison avec l'approche intégrale ($89800$ secondes et $2999492$ KO de mémoire pour résoudre le même problème).

En se basant sur l'intuition du métier, les auteurs du système BBW ont proposé un déploiement dans lequel chaque fonction est partitionnée sur une tâche différente. Le placement proposé consiste à affecter les couples de fonctions ("ABS front-left", "Brake actuator LDM front-left"), ("ABS front-right", "Brake actuator LDM front-right"), ("ABS rear-left", "Brake actuator LDM rear-left") et ("ABS rear-right", "Brake actuator LDM rear-right") respectivement sur les unités de contrôle "ECU front-left", "ECU front-right", "ECU rear-left" et "ECU rear-right". Le reste des fonctions est placé sur l'ECU centre. Cette solution conduit aux latences $42$, $61$, $77$ et $93$. On s'aperçoit que la solution obtenue est moins pertinente par rapport à la métrique du temps de réponse en comparaison avec la solution retournée par les approches proposées dans cette thèse.

Étant donné que la sécurité des systèmes de freinage "Brake-By-Wire" est de nature critique et que l'incorporation d'un degré de tolérance aux pannes est souvent menée par la redondance du système qui permet aux répliques du système de prendre la main dans le cas où le système initial subit une défaillance, nous avons répliqué le système BBW pour étudier le passage à l'échelle de l'approche intégrale par rapport à ce problème. Malheureusement, pour le système répliqué, **l'approche intégrale n'est pas en mesure d'établir une solution**

**quelconque**. En d'autres termes, après $86450$ secondes, le premier nœud de l'arbre de recherche n'est pas encore traité. En appliquant l'approche en deux étapes au système BBW dupliqué, nous avons obtenu une solution dans $19534$ secondes et après $3567816$ KO de consommation mémoire maximale. Cette solution parait optimale car, d'une part, la solution pour la première partie représentant le système BBW est identique à celle obtenue par l'approche PLMNE intégrale, et d'autre part, une solution équivalente est fournie pour la partie répliquée du système.

### 4.3.2 Évaluation pour les systèmes avec activations périodiques

Après avoir présenté les différentes expérimentations dédiées aux systèmes avec activations pilotées par les données, nous montrons dans cette partie comment notre contribution peut s'adapter aux systèmes avec un modèle d'activation périodique. La différence avec l'activation pilotée par les données vis-à-vis de la formulation du problème de déploiement réside sur deux points centraux. Le premier concerne l'analyse des temps de réponse qui est plus complexe pour le cas d'activations pilotées par les données en comparaison avec le cas d'activations périodiques. Cette complexité est due à l'existence d'un lien entre les calculs des temps de réponse qui se manifeste via la présence des gigues. Cependant, dans le cas d'activations périodiques, les gigues s'annulent et donc le lien disparait résultant ainsi en un nombre de contraintes et de variables réduit. De plus, une seule sémantique de synchronisation peut être considérée dans le cas d'activations périodiques. Rappelons que cette dernière nécessite la considération d'un degré de liberté supplémentaire qui est la détermination de l'ordre de séquence des fonctions appartenant à la même tâche. Pour le cas d'activations périodiques, les fonctions communicantes via des partages de données ne sont pas synchronisées, d'où l'obligation de déterminer un mécanisme de protection pour les données partagées. Le modèle d'activation périodique exige au total la résolution de cinq sous problèmes, à savoir le placement, le partitionnement, l'ordonnancement, l'ordre de séquence et le mécanisme de protection, contre seulement trois pour les activations pilotées par les données.

Les conclusions apportées par rapport à la qualité des solutions obtenues dans les études comparatives avec les approches faisant des hypothèses sur le partitionnement (partie 4.3.1.2) et celles faisant des hypothèses sur le placement (partie 4.3.1.3), ainsi que dans l'évaluation de l'approche en deux étapes (parties 4.3.1.4 et 4.3.1.5) sont les mêmes pour le cas d'activations périodiques. Les résultats ne sont donc pas présentés, et nous nous focalisons principalement dans les expérimentations suivantes sur l'évaluation de la performance en termes de ressources :

- **Objectif :** l'objectif des expérimentations est de déterminer l'impact d'un modèle d'activation périodique sur la performance de la formulation PLMNE apportée au problème de déploiement et d'évaluer cet impact par rapport à celui issue du modèle d'activations pilotées par les données.

Ces expérimentations sont fondées sur $840$ instances de système générées aléatoirement comme décrit dans la partie 4.2.2. Le temps d'accès à une donnée partagée (ou la durée d'une section critique) est fixé à $0,8$. Au sein de chaque processeur, la mémoire utilisée est nulle et la capacité maximale en mémoire est définie à $50$. Par ailleurs, suite à la complexité de l'approche PLMNE intégrale établie auparavant, nous nous intéressons dans ces expérimentations uniquement à l'approche en deux étapes. Pour cette dernière, nous avons considéré cinq configurations initiales produites de manière aléatoire (voir la

partie 4.3.1.4).

Dans les parties suivantes, nous étudions tout d'abord l'impact de l'analyse des temps de réponse sur la performance du solveur (partie 4.3.2.1). Ensuite, nous évaluons l'impact d'un degré de liberté supplémentaire (l'affectation du mécanisme de protection pour les données partagées) (partie 4.3.2.2). Pour ces deux dernières études, nous tenons compte seulement de la minimisation de la moyenne des latences comme métrique d'optimisation. Enfin, nous présentons dans la partie 4.3.2.3 un cas d'étude illustratif de notre contribution adressant à la fois la minimisation de la moyenne des latences et de la consommation mémoire.

#### 4.3.2.1 Évaluation de l'effet de l'analyse des temps de réponse

Afin de déterminer l'influence de l'analyse des temps de réponse impliquée dans le modèle d'activation périodique par rapport à celle utilisée dans le cas d'activations pilotées par les données, nous comparons l'approche en deux étapes appliquée à un système ayant des activations périodiques avec cette même approche adressant cette fois ci le même système sous l'hypothèse que les activations sont pilotées par les données. Pour cela, nous considérons la sémantique de synchronisation 'input N-of-N / output 1-of-N' dans le cas d'activations pilotées par les données car celle-ci représente l'équivalent de la sémantique considérée dans le cas d'activations périodiques. Les mêmes configurations initiales sont considérées dans chacun des cas. L'idée de cette comparaison est d'identifier, dans le cadre du problème de déploiement, la différence entre une analyse des temps de réponse avec et sans propagation des gigues. Pour la cohérence de l'étude comparative, nous ne considérons pas l'affectation du mécanisme de protection pour les données partagées dans le cas d'activations périodiques. Par conséquent, nous abordons pour chaque modèle d'activation le placement, le partitionnement, l'ordonnancement et l'ordre de séquence.

- **Comparaison des performances :** afin de répondre à l'objectif de cette expérimentation, nous présentons la figure 4.28 dans laquelle la performance de l'approche de déploiement en deux étapes considérant un modèle d'activation périodique (PLMNE 2-étapes (5) "activations périodiques sans mécanisme") est confrontée à celle de cette même approche lorsque les activations sont pilotées par les données (PLMNE 2-étapes (5) "activations pilotées par données"). Ici, la performance est mesurée en termes de la moyenne des durées d'exécution du solveur. Néanmoins, la performance en mémoire maximale consommée conduit à la même allure graphique que celle de la figure 4.28. La performance dans la figure 4.28 est donnée pour les $84$ dimensions de système considérées. Chaque point représente la moyenne pour $10$ instances ayant la même dimension.

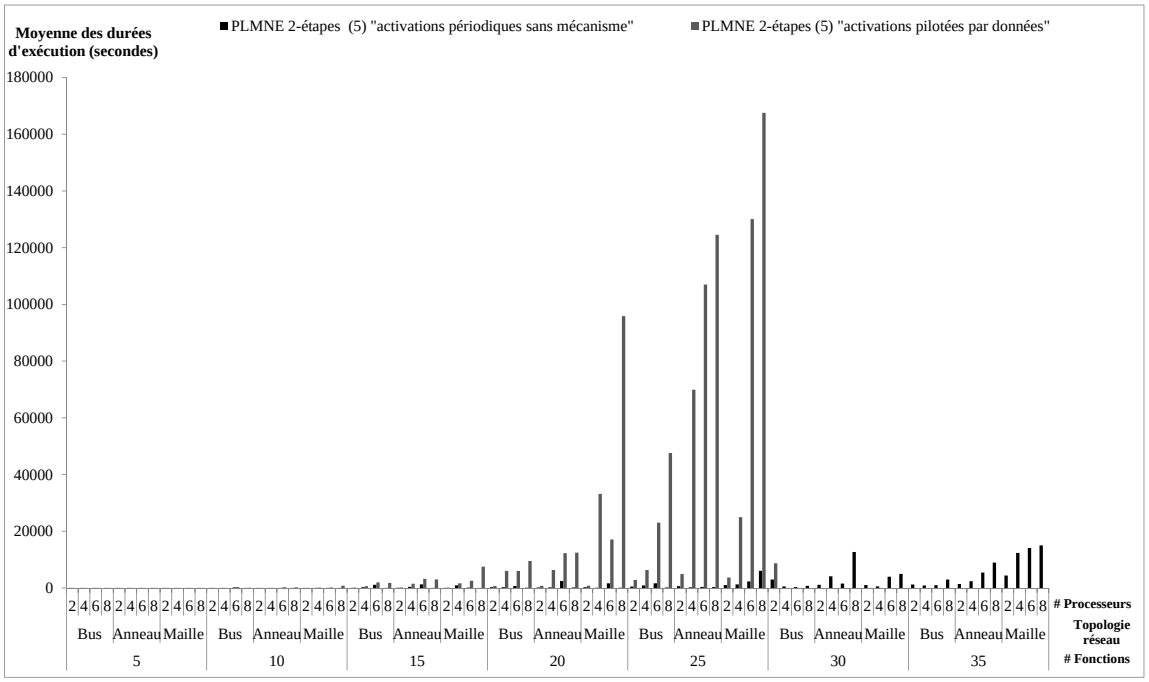



FIGURE 4.28 – L'impact de l'analyse des temps de réponse sur la performance en temps d'exécution de l'approche en deux étapes considérant les deux différents modèles d'activation

Clairement, nous constatons à partir de la figure qu'en moyenne le temps d'exécution de l'approche en deux étapes lorsque les activations du modèle fonctionnel sont périodiques est largement réduit par rapport au cas où les activations sont pilotées par les données. Nous constatons également que l'approche en deux étapes passe mieux à l'échelle pour un système ayant des activations périodiques en comparaison avec le cas d'activations pilotées par les données. Pour les instances considérées, l'approche est capable d'apporter une solution pour des systèmes de 35 fonctions et 8 processeurs lorsque les activations sont périodiques. Cependant, elle n'a pas résolu ces systèmes pour le cas d'activations pilotées par les données. Après avoir analysé ces résultats, nous avons conclu deux raisons expliquant cette différence entre les temps d'exécution dans les deux situations. Dans la première, l'analyse des temps de réponse pour le modèle d'activation périodique introduit moins de variables et de contraintes dans la formulation PLMNE par rapport à celles considérées lors de la formulation de l'analyse apportée au modèle d'activations pilotées par les données. La seconde raison est que l'analyse des temps de réponse pour le cas d'activations périodiques a permis à l'algorithme d'exploration d'élaguer considérablement l'espace de recherche en excluant des parties de taille importante. Ceci est dû au fait que le calcul des latences pour ce cas d'activations est trop pessimiste et donc énormément de solutions sont invalides vis-à-vis des contraintes temporelles notamment celles qui représentent un déploiement distribué.

Du point de vue de la qualité des solutions, à partir des résultats des instances considérées, nous avons constaté que deux situations se sont produites. La première est que l'approche en deux étapes fournit des solutions d'une même qualité pour les deux modèles d'activations. Ce cas survient lorsque le système est composé uniquement de fonctions indépendantes. La deuxième situation correspond à tous les autres types de

systèmes où il existe des fonctions dépendantes. Dans ce cas, les activations pilotées par les données nous permettent d'obtenir de meilleures solutions en comparaison avec les solutions retournées dans le cas d'activations périodiques. Ce résultat est dû aux activations non synchronisées conduisant à un calcul pessimisme des latences. Rappelons que pour les activations périodiques, la latence inclut, d'une part, les temps de réponse de toutes les fonctions et signaux appartenant au chemin sous-jacent. D'autre part, pour chaque communication sur le réseau, elle ajoute les périodes aux latences. Ceci est plus coûteux en comparaison avec le cas d'activations pilotées par les données où la latence représente tout simplement le temps de réponse de la dernière fonction activée sur le chemin.

#### 4.3.2.2 Évaluation de l'effet de protection des données partagées

Dans les systèmes avec activations périodiques, il est indispensable de protéger les données partagées via les échanges de signaux, car les activations des fonctions dépendantes sont asynchrones. Nous avons vu précédemment que l'existence de différents mécanismes de protection, ayant chacun un impact sur un paramètre donné parmi la mémoire et le temps, explique l'intérêt de considérer le mécanisme de protection comme variable conduisant ainsi à une solution de compromis souvent souhaitable. Dans cette expérimentation, nous nous intéressons à mesurer l'impact du choix optimisé du mécanisme de protection sur la performance de l'approche. En d'autres termes, notre but est de déterminer si la considération du mécanisme de protection comme un degré de liberté supplémentaire dans le cas d'activations périodiques diminue la performance de l'approche de déploiement en comparaison avec le cas des systèmes avec activations pilotées par les données. Pour cela, nous comparons les performances de l'approche en deux étapes dans le cas d'activations périodiques où le placement, le partitionnement, l'ordonnancement, l'ordre de séquence et le mécanisme de protection sont adressés et dans le cas d'activation pilotées par les données où uniquement les quatre premiers degrés de liberté sont pris en compte.

- **Comparaison par rapport à la complexité algorithmique :** similairement à l'expérimentation précédente, nous présentons dans la figure 4.29 la complexité en termes de la moyenne des durées d'exécution du solveur pour l'approche en deux étapes considérant deux situations : le cas d'activations périodiques (PLMNE 2-étapes (5) "activations périodiques") et le cas des activations pilotées par les données (PLMNE 2-étapes (5) "activations pilotées par données"). La différence avec l'expérimentation précédente est que cette fois-ci l'approche en deux étape considère également l'optimisation du choix de mécanisme de protection pour le cas d'activations périodiques.

  Nous remarquons que, par rapport à l'expérimentation précédente, aucun changement n'a eu lieu vis-à-vis de la différence entre la complexité de l'approche en deux étapes pour un modèle d'activation périodique et celle pour le cas d'activations pilotées par les données. De ce fait, nous pouvons conclure que pour les instances traitées, la considération du mécanisme de protection comme étant un degré de liberté supplémentaire a un coût négligeable en termes de complexité en comparaison avec l'avantage qu'apporte une analyse simple et un calcul de latence pessimiste dans le cas d'activations périodiques. Par conséquent, l'impact de l'analyse des temps de réponse des systèmes avec activations pilotées par les données sur la complexité de l'approche en deux étapes reste considérable.

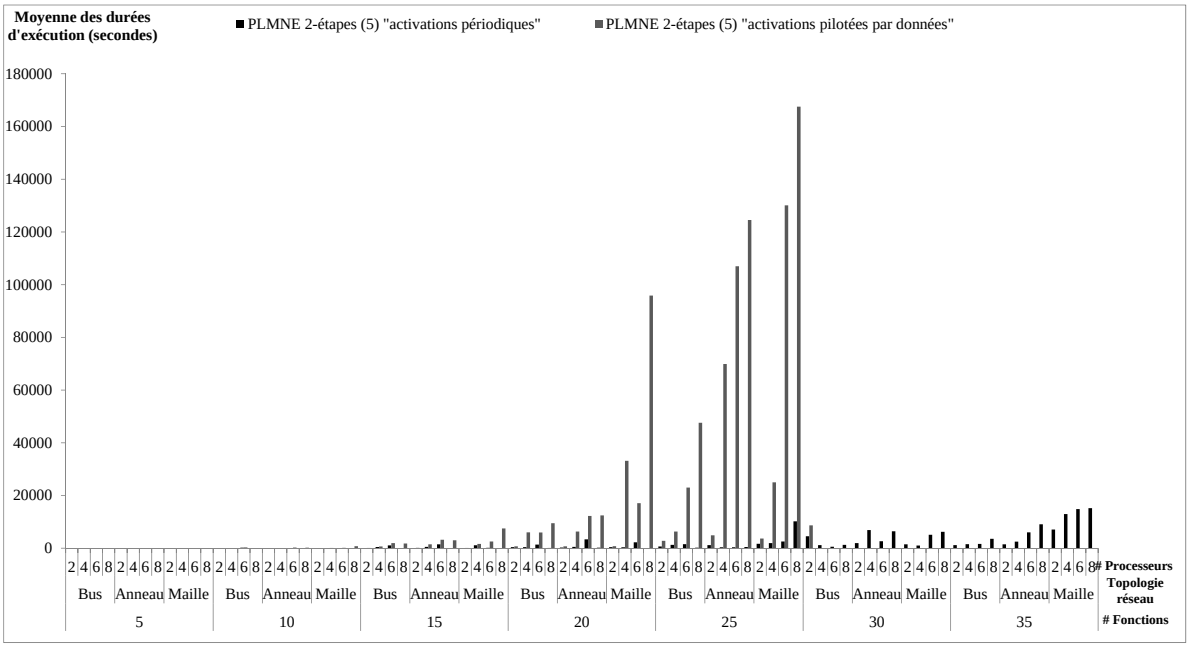



FIGURE 4.29 – L'impact de l'optimisation du choix de mécanisme de protection sur la complexité en temps d'exécution de l'approche en deux étapes

**Remarque :** à partir des deux expérimentations précédentes, nous avons remarqué que pour un modèle d'activation périodique, lorsque la mémoire disponible au sein des processeurs est largement suffisante, alors les deux versions (avec et sans la considération de l'optimisation du mécanisme de protection) de l'approche en deux étapes retournent des solutions de qualités identiques. Ceci est dû au fait que nous avons visé uniquement l'optimisation de la moyenne des latences. La version considérant l'optimisation du mécanisme de protection renvoie des solutions ou le mécanisme de protection choisi est "Rate transition", n'ayant ainsi pas d'impact sur les valeurs des latences. Par ailleurs, lorsque la quantité de mémoire n'est pas suffisante, il est évident que la version considérant l'optimisation du mécanisme de protection peut renvoyer des solutions moins performantes que celles obtenues par la version n'adressant pas l'optimisation du mécanisme de protection, puisqu'elle est obligée de choisir les verrous sémaphores comme mécanisme. Par conséquent, ce choix influence les valeurs des latences.

D'un autre point de vue, nous concluons que l'estimation empirique de la performance de l'approche de déploiement en deux étapes pour ces deux expérimentations ne correspond pas à la complexité théorique présentée dans la partie 3.4.3.3. Autrement dit, bien que pour le cas d'activations pilotées par les données, le nombre de contraintes et de variables est largement plus petit que celui introduit dans le cas d'activations périodiques, la performance en pratique pour ce dernier cas est moins importante. Ceci confirme l'idée qu'il est impossible de prédire la performance d'une formulation PLMNE en se basant sur la taille du problème.

Dans la partie suivante, nous étudions via un cas d'étude le compromis entre l'optimisation de la mémoire et celle de la moyenne des latences dans le cas d'activations périodiques.

#### 4.3.2.3 Cas d'étude industriel : système automobile ABS et CCS

L'intérêt de cette partie est d'illustrer l'applicabilité des approches proposées dans le contexte industriel de l'automobile, et ce lorsque le modèle d'activation considéré est le modèle périodique. Elle vise également à étudier l'optimisation multiobjectif qui se produit dans ce cas.

**Description :** le système automobile considéré dans cette partie est composé de deux sous systèmes, à savoir "Anti-Lock Brake Sub system" (ABS) [173] et "Cruise Control Sub system" (CCS) [174]. L'objectif du sous système ABS est de s'opposer au blocage des roues lorsqu'un freinage violent se produit. Il assure ainsi un freinage adéquat permettant au conducteur de conserver la maîtrise du véhicule. L'architecture fonctionnel de l'ABS est représentée sur l'image de droite de la figure 4.30. Elle est composée de la fonction "Data Processing" qui traite les données en provenance du capteur, de la fonction "Anti-locking brake" calculant la commande à envoyer à l'actionneur et de "Diagnosis" qui désactive la fonction "Anti-locking brake" dans le cas où une erreur est détectée. La fonction "Anti-locking brake" est activée selon la sémantique "OU" d'où sa duplication sur les deux transactions qui deviennent par la suite linéaires et indépendantes. Par ailleurs, la mission du sous système CCS est de maintenir la vitesse d'un véhicule indépendamment du fait qu'il se déplace sur une surface plane ou sur une surface en pente. L'architecture fonctionnel du CCS (voir l'image de gauche de la figure 4.30), est composée de huit fonctions organisées en deux transactions linéaires : "Input Acquisition" responsable de l'acquisition des données du capteur, "Input Interpretation" chargée de l'interprétation des données de capteur acquises pour déterminer la demande du conducteur, "Diagnosis" détecte les erreurs ou les incohérences des données acquises, "Limp Home" décide l'action à prendre en cas d'erreur détectée, "Speed Setpoint" calcule la vitesse désirée par le conducteur, "Application Condition" et "Basic Function" sont responsables du le calcul des états et des transitions du sous système afin de décider si des activités spécifiques de contrôle de vitesse sont menées et "Controller" maintient la vitesse du véhicule. Sur la figure 4.30, le couple de valeurs au-dessus des fonctions représente respectivement la période et le WCET. La période des signaux est identique à celle de la fonction émettrice. En se basant sur [175, 176], nous avons fixé la taille des signaux de donnée à 8 octets. La valeur définie pour la durée d'une section critique est de 2 ms.

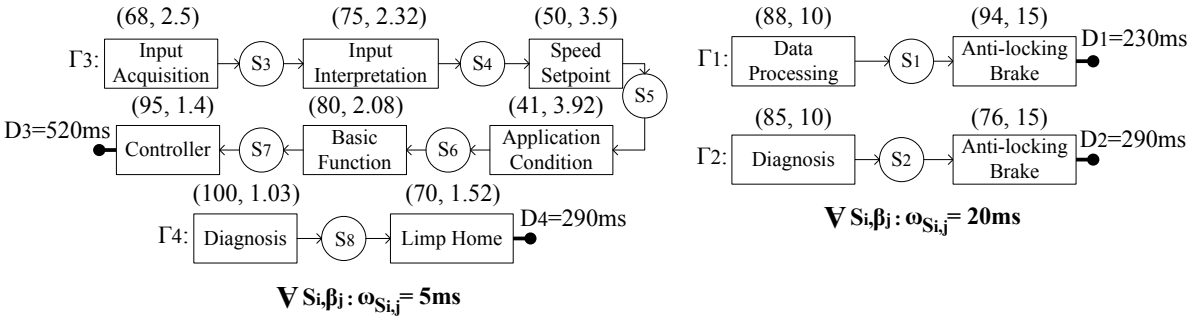



FIGURE 4.30 – Modèle fonctionnel du système ABS + CCS

L'architecture matérielle pour chacun des sous systèmes ABS et CCS est constituée de deux unités de contrôle électroniques (processeurs) liées par un bus CAN. La vitesse de calcul pour tous les processeurs est identique et il en est de même pour la vitesse de transmission des bus. Nous avons fixé la capacité d'utilisation maximale à $0,8$ pour tous les processeurs et les bus de communication. La capacité de mémoire maximale des

processeurs est définie à $4$ KO (l'équivalent de $4096$ octets) et la mémoire déjà occupée sur chaque processeur est supposée nulle.

**Solution :** le système (ABS + CCS) est abordé par l'approche PLMNE intégrale. Afin de répondre à l'objectif de cette expérimentation, nous avons employé la fonction objectif décrite par la formule 3.87. Celle-ci s'intéresse à la minimisation de la moyenne des latences en même temps qu'à la minimisation de la moyenne de la consommation mémoire. Pour chacune de ces métriques, nous avons fait varier les poids des coefficients d'optimisation par rapport à l'ensemble $\{0, 0.25, 0.5, 0.75, 1\}$. Les résultats obtenus sont exprimés par le graphe de la figure 4.31. Sur l'axe des abscisses, nous présentons les valeurs des poids de coefficient des métriques et sur l'axe des ordonnées nous déterminons la valeur de chaque métrique dans la solution trouvée.

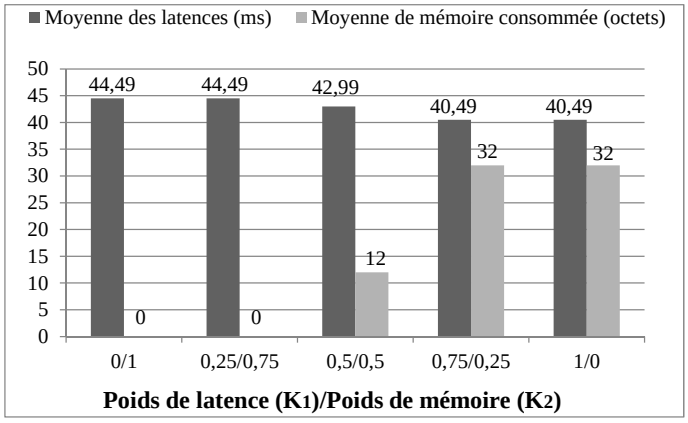



FIGURE 4.31 – Résultats d'une optimisation multiobjectif pour le système ABS + CCS

Dans toutes les solutions obtenues, les signaux sont transmis localement aux processeurs. De plus, les périodes non harmoniques des fonctions ont conduit à des solutions où tous les signaux locaux représentent une transmission inter-tâches. Dans les solutions correspondant aux poids $0/1$ et $0.25/0.75$, tous les signaux inter-tâches sont protégés par des verrous sémaphores tandis que dans les solutions correspondant aux poids $0.75/0.25$ et $1/0$, ils sont protégés par le mécanisme "Rate Transition". Pour les poids $0.5/0.5$, les signaux $s_3, s_4, s_5, s_6$ et $s_7$ sont protégés par des verrous sémaphores et le reste des signaux est protégé par le mécanisme "Rate Transition". Nous avons constaté que les cinq signaux $s_3, s_4, s_5, s_6$ et $s_7$ représentent le cas où le rapport entre la durée de la section critique et l'échéance de bout-en-bout du chemin du signal est inférieur au rapport entre la taille du signal multipliée par deux et la mémoire maximale du processeur auquel appartient le signal. Les autres signaux représentent le cas inverse.

En conclusion, dans le cas où des signaux inter-tâches existent, le compromis entre la latence et la mémoire dépend de la relation entre le rapport durée des sections critiques/échéances des chemins et le rapport taille des signaux/mémoire maximale des processeurs.

## 4.4 Conclusion

Dans ce chapitre, nous avons présenté un ensemble d'expérimentations afin d'évaluer les différentes approches de déploiement proposées. Dans chaque expérimentation,

nous nous sommes basés sur une série de tests générés d'une façon aléatoire. Pour les systèmes avec activations pilotées par les données, nous avons mesuré l'impact de chaque sémantique de synchronisation des tâches, d'une part, sur la qualité de la solution de déploiement obtenue par l'approche PLMNE intégrale, et d'autre part, sur la performance de l'algorithme d'exploration intégré au solveur. Ensuite, nous avons déterminé l'apport de notre proposition par rapport aux approches de la littérature. Pour cela, nous avons considéré les approches faisant des hypothèses de partitionnement des fonctions et des signaux, respectivement, sur les tâches et les messages d'une part [2, 112, 17, 142], et d'autre part, les approches faisant des hypothèses sur le placement des fonctions et des signaux, respectivement, sur les processeurs et les bus de communication [15, 13, 167, 168, 158]. Puis, nous avons estimé l'intérêt des stratégies de décomposition et d'améliorations itératives utilisées dans l'approche de déploiement en deux étapes. Enfin, nous avons approfondi l'étude évaluative de l'approche en deux étapes en comparant la solution renvoyée avec la solution optimale et en estimant les améliorations pouvant être apportées par chacune des boucles de l'approche. Nous avons également étudié la performance en mémoire des deux approches de déploiement présentées dans cette thèse et nous les avons illustré à l'aide du système automobile "Brake-By-Wire". Dans la seconde partie des expérimentations, nous nous sommes focalisés sur les systèmes ayant un modèle d'activation périodique. Nous avons étudié la performance de la formulation PLMNE pour ces derniers puis nous nous sommes basés sur les systèmes "Anti-lock Brake" et "Cruise Control" pour étudier l'optimisation multiobjectif survenant dans ce cas d'activation.

Nous pouvons résumer les résultats obtenus lors des évaluations effectuées selon les points suivants :

- Pour les systèmes avec activations pilotées par les données, la sémantique de synchronisation input N-of-N / output N-of-N apporte un compromis intéressant entre le critère de qualité et celui de performance.
- Le traitement des sous problèmes de déploiement (placement et partitionnement) d'une façon indépendante dégrade la qualité de l'architecture opérationnelle vis-à-vis de la métrique du temps de réponse.
- Le nombre de fonctions et le nombre de processeurs sont les paramètres ayant l'impact le plus important sur la performance de l'algorithme d'exploration.
- Malheureusement, l'approche intégrale ne permet pas de résoudre des systèmes ayant une taille similaire à ceux utilisés dans la littérature [133, 16, 142]. En revanche, l'approche en deux étapes a montré sa pertinence pour ces systèmes. De façon plus générale, celle-ci est capable de traiter des systèmes de taille plus importante que celle des systèmes abordés par l'approche intégrale.
- L'approche en deux étapes apporte des améliorations importantes pour les instances de systèmes considérées. Son point fort, par rapport aux approches existantes, réside principalement dans le fait qu'elle considère la même métrique d'optimisation durant la résolution de chacun des sous problèmes et dans sa boucle interne remettant en cause les décisions prises dans les étapes précédentes. Ceci lui permet de toujours s'orienter vers des solutions meilleures ou identiques.
- La performance de l'algorithme d'exploration dépend fortement de l'instance du système considérée. Autrement dit, pour des systèmes de même taille et de différents paramètres temporels, la performance du solveur en temps ou en mémoire peut fortement varier.
- Bien que la considération d'un modèle d'activation périodique nécessite la résolution d'un sous problème supplémentaire aux sous problèmes abordés dans le cas d'activations pilotées par les données, son impact sur la performance de l'approche de déploiement reste moins important en comparaison avec celui des activations pilotées

par les données. Ceci est dû à l'analyse des temps de réponse qui est plus simple (sans les gigues) dans le cas d'activations périodiques et qui conduit à un calcul pessimiste de la latence permettant ainsi de réduire significativement l'espace de recherche au moment de l'exploration. En contrepartie, ce dernier cas aboutit à des solutions de déploiement moins performantes comparées à celles retournées pour les activations pilotées par les données, à cause des activations des fonctions dépendantes qui sont asynchrones.

CHAPITRE 5

# Conclusion générale et perspectives

Cette dernière partie conclut l'étude par un rappel de la problématique abordée dans la présente thèse ainsi que les principales contributions apportées. Enfin, ce chapitre propose certaines perspectives pour la suite des recherches à mener dans le domaine de déploiement à base de modèles pour les systèmes temps réel critiques distribués.

## 5.1 Rappel de la problématique

Les applications temps réel critiques deviennent de plus en plus complexes. La gestion de cette complexité durant le développement logiciel et la réduction du coût de développement s'appuient sur la mise en place d'architectures logicielles adéquates. On essaie alors de garantir les performances du système et son exactitude par rapport aux propriétés temporelles, et ce dès la conception architecturale. De nombreuses méthodologies de développement à base de modèles ont été proposées en ce sens. Celles-ci introduisent un haut niveau d'abstraction et une validation précoce des propriétés temps réel du système. Cependant, bien que ces approches proposent des solutions attractives pour la description et la validation des architectures logicielles, elles restent incomplètes par rapport à la synthèse automatique d'architectures valides et optimisées du point de vue temps réel. Il existe certaines approches de la littérature abordant ce dernier point mais uniquement de façon partielle. Il nous est apparu nécessaire d'apporter de nouvelles méthodes permettant de résoudre de manière globale le problème de la synthèse automatique d'architectures opérationnelles valides et optimisées pour les systèmes temps réel distribués et multitâche.

## 5.2 Rappel de la contribution

Afin de répondre aux lacunes de la littérature, nous avons proposé dans cette thèse des techniques d'analyse et d'optimisation permettant la synthèse automatique d'architectures opérationnelles pour les systèmes temps réel distribués. Notre objectif principal est de prendre en compte le modèle fonctionnel durant cette phase de synthèse.

L'analyse des temps de réponse étant un point essentiel, nous avons commencé par étudier les différentes analyses existantes afin de déterminer celle appropriée à notre modèle de système. Cependant, la majorité des analyses considèrent un modèle de tâches comme étant le modèle d'analyse et supposent que le modèle fonctionnel est masqué. Nous avons donc été amenés à ajuster l'analyse holistique pour tenir compte de la répartition des éléments du modèle fonctionnel sur ceux du modèle de tâche. Initialement, nous avons montré que l'analyse des temps de réponse pour ce type de systèmes dépend de la sémantique de synchronisation considérée pour les tâches. Autrement dit, elle se définit par rapport aux instants auxquels une tâche est capable de traiter une donnée en entrée et de fournir le résultat correspondant. Puis, nous avons présenté l'analyse appropriée à chaque sémantique.

Par la suite, le travail de cette thèse se focalise principalement sur le déploiement d'un modèle fonctionnel, représentant le comportement attendu du système, sur un modèle de plateforme composée d'unités matérielles (des processeurs et des bus de communication) et logicielles (des tâches et des messages). Trois questions clés ont été identifiées et abordées : le placement des fonctions et des signaux respectivement sur les ressources de calcul et de communication, le partitionnement des fonctions et des signaux respectivement sur des tâches logicielles et des messages, et enfin, l'affectation des priorités aux tâches et messages permettant ainsi leur ordonnancement. Ces trois questions représentent les préoccupations majeures des concepteurs de systèmes pour les phases d'intégration et de configuration. Par conséquent, les travaux présentés dans cette thèse offrent des guides au concepteur dans

la prise de décision pour les différents choix répondant à ces questions. Pour guider les choix, nous avons considéré principalement les contraintes temporelles et les différentes contraintes liées aux capacités des ressources telles que la mémoire et la charge des processeurs. Du point de vue de l'optimisation, nous sommes intéressés à la minimisation de la moyenne des latences. Au début, nous avons adressé des systèmes dont les activations sont pilotées par les données, puis, nous avons étendu notre proposition afin d'aborder les systèmes basés sur des activations périodiques. Cette extension a nécessité la résolution d'un quatrième sous problème qui consiste en l'affectation du mécanisme de protection pour les données partagées. Nous avons considéré l'utilisation des verrous sémaphores et des blocs "Rate Transition" comme de possibles mécanismes de protection. Les verrous sémaphores ont un impact négatif sur le temps de réponse du système tandis que l'utilisation des blocs "Rate Transition" résulte en une consommation mémoire supplémentaire. Dans ce cas de systèmes périodiques, une optimisation multiobjectif s'intéressant à une solution de compromis entre la minimisation du temps de réponse et la minimisation de la consommation mémoire a été proposée.

Pour résoudre ces problèmes, une première approche s'appuie sur une formulation intégrale du problème de déploiement, vu comme une conjonction des sous problèmes de placement, de partitionnement et d'ordonnancement, dans le cadre d'un programme linéaire mixte en nombres entiers (PLMNE). Elle intègre toutes les contraintes du système considérées et vise à optimiser les métriques citées ci-dessus. Cette formulation exacte a permis d'assurer l'optimalité de la solution retournée. Elle s'est avérée inefficace pour les instances de taille importante, par exemple basées sur 6 processeurs et 20 fonctions. Néanmoins, cette solution est utile pour évaluer l'optimalité d'autres algorithmes sur des instances de taille limitée mais aussi pour guider la configuration des approches à base de méta-heuristiques.

Afin de fournir une solution de déploiement applicable pour les applications ayant une taille similaire à celle des cas d'étude de la littérature, une approche de synthèse en deux étapes permettant de réduire la complexité de l'approche intégrale a été proposée. L'application de cette approche en deux étapes à un ensemble de systèmes générés aléatoirement a montré que cette alternative est capable de retourner des solutions intéressantes par rapport à la minimisation de la moyenne des latences. Cette approche se base aussi sur la programmation linéaire mixte en nombres entiers. Elle consiste à utiliser en entrée une solution aléatoire de partitionnement et d'ordonnancement pour explorer l'espace des solutions de placement possibles à l'aide d'un PLMNE. Ensuite, dans la deuxième étape, la solution de placement trouvée précédemment est utilisée par un autre PLMNE pour chercher des alternatives de partitionnement et d'ordonnancement. Ce processus, représenté par les deux étapes, est répété jusqu'à un point de convergence où la solution de déploiement trouvée ne change pas d'une itération à l'autre. Cette propriété de convergence est garantie puisqu'à chaque itération le même objectif et les mêmes contraintes sont visés. À l'obtention d'un point de convergence, l'approche suggère de considérer en entrée d'autres solutions aléatoires de partitionnement et d'ordonnancement afin d'explorer différentes parties de l'espace de recherche et de s'échapper des optimaux locaux. Le but de l'approche étant qu'à chaque itération elle puisse retourner une solution de déploiement meilleure que celle renvoyée durant l'itération précédente.

Pour conclure, la proposition a permis de résoudre des lacunes dans le domaine du développement logiciel des systèmes temps réel critiques. L'une des lacunes réside dans l'analyse des temps de réponse appliquée au modèle fonctionnel. Elle a également considéré conjointement les problèmes de placement, de partitionnement et d'ordonnancement. Cette conjonction a conduit à une complexité beaucoup plus élevée en comparaison avec les approches existantes considérant ces problèmes d'une manière indépendante. En revanche les approches proposées fournissent de meilleures solutions de déploiement. Ceci

confirme l'intérêt d'adresser parallèlement les différentes questions permettant de résoudre le problème de déploiement. Par ailleurs, la stratégie employée par l'approche en deux étapes a permis d'améliorer la solution de déploiement en comparaison avec les stratégies d'optimisation existantes.

## 5.3 Perspectives

Bien que les approches proposées dans cette thèse aient conduit à des résultats intéressants, nous pensons qu'il est encore possible de les compléter et de leur apporter des améliorations. Par ailleurs, l'extension de leur applicabilité parait importante, permettant ainsi de répondre aux différentes questions ouvertes dans le domaine du développement logiciel d'applications critiques en général et de la synthèse architecturale plus particulièrement. Nous proposons donc de poursuivre la recherche en traitant les points suivants :

### 5.3.1 Améliorations de l'étude

- L'analyse des temps de réponse utilisée dans ce travail peut être affinée pour tenir compte des détails du protocole de communication utilisé. Par exemple pour CAN, il faudrait considérer les bits de surcharge. D'autre part, la taille maximale des données que peut contenir une trame ne doit pas dépasser $8$ octets. Il est donc nécessaire d'exprimer la taille des données d'un message par un nombre de $0$ à $8$ octets. Par exemple, si deux signaux sont partitionnés sur le même message et que la taille de chacun est de $10$ bits, alors la longueur des données du message est de $3$ octets ($24$ bits au lieu de $20$) [47, 112, 178].
- Une autre direction de recherche qui mériterait d'être poursuivie concerne la définition d'un ensemble de patrons guidant le concepteur vers un choix de configurations initiales adéquates et qui résultent en un nombre minimum d'itérations. Afin d'atteindre cet objectif, il faut tout d'abord classifier les systèmes par rapport à la structure des architectures fonctionnelle et matérielle et aux paramètres temporels, puis appliquer plusieurs formes de configurations initiales à chaque classe pour enfin sélectionner celle conduisant à de meilleures performances en qualité de la solution et en nombres d'itérations.
- Pour l'approche en deux étapes, la possibilité d'exclure les espaces de recherche déjà explorés auparavant par les itérations de la boucle externe qui précède une itération externe donnée représente une excellente idée pour la continuation des travaux de cette thèse. Ceci permet de réduire le temps dû à des explorations inutiles et d'offrir la possibilité de considérer d'autres espaces de recherche plus prometteurs. Le principe de la recherche tabou [180] qui consiste à utiliser une mémoire permettant de stocker les solutions visitées et d'éviter une recherche "aveugle", peut être appliqué pour aborder cette perspective.
- Nous avons vu dans ce mémoire que les formulations PLMNE sont d'une complexité importante. Nous pouvons appliquer certaines solutions apportées dans la littérature pour réduire cette complexité. Il s'agit, par exemple, d'employer la méthode de "local branching" présentée dans [179]. Celle-ci est initiée par une solution réalisable quelconque pour ensuite utiliser une contrainte de branchement local comme une règle de séparation sur l'arbre de recherche énumératif. L'idée est d'accélérer le processus d'exploration en réduisant la taille de l'espace de recherche représenté par l'arbre de gauche de chaque nœud de l'arbre global puis d'utiliser la solution trouvée pour construire la règle de séparation sur l'arbre de droite. Il est également possible d'augmenter le passage à l'échelle de l'approche en deux étapes en considérant

plusieurs étapes, chacune dédiée à un sous problème différent. Cela consiste par exemple à traiter le partitionnement et l'affectation du mécanisme de protection par des étapes différentes.

### 5.3.2 Travaux complémentaires pour étendre l'applicabilité de l'étude

- Dans cette étude, nous avons considéré des systèmes avec échéances contraintes. Une extension aux systèmes ayant des échéances arbitraires est à étudier. Étant donné que ceci revient à une formulation impossible de l'analyse des temps de réponse en PLMNE, une solution alternative serait d'utiliser d'autres techniques d'optimisation telles que les méta-heuristiques (les algorithmes génétiques et le recuit simulé).
- Des travaux futurs devraient aussi considérer d'autres modèles de systèmes comme les systèmes ayant des activations mixtes entre les activations périodiques et les activations pilotées par les données. Dans ce cas, la formule (2.13) est utilisée pour calculer les latences. Il devrait être possible de considérer les systèmes pilotés par le temps (Time triggered) dont la communication est réalisée via des bus de type FlexRay. Pour cela, l'analyse des temps de réponse doit être combinée à l'analyse d'ordonnancement statique.
- Par rapport au modèle fonctionnel, il est envisageable d'intégrer des fonctions ayant plusieurs données en sortie définies selon la sémantique d'exécution de type "OU", c'est-à-dire un modèle fonctionnel muni de communications conditionnées. La considération de ces systèmes implique une adaptation de l'analyse des temps de réponse. Celle-ci peut se traduire par la prise en compte unique de la séquence de fonctions suivant le nœud de condition et conduisant au scénario "pire cas".
- L'établissement de règles métiers pertinentes en fonction des paramètres temporels et de la structure des architectures fonctionnelle et matérielle du système, est un travail plus poussé qui mérite d'être abordé dans le futur afin de définir des heuristiques dédiées permettant de considérer des systèmes plus complexes tels que les systèmes mentionnés dans les trois perspectives précédentes.
- Les systèmes temps réel émergents ont tendance à être déployés sur des architectures multi-cœurs. Pour aborder la synthèse architecturale de tels systèmes, il devient nécessaire d'adapter l'analyse des temps de réponse au cas d'ordonnancement hiérarchique. Certains travaux ont abordé l'ordonnancement des systèmes multi-cœurs en proposant un ordonnancement statique cyclique [186]. D'autres travaux ont présenté une analyse pour les systèmes avec des tâches indépendantes ordonnancées par un exécutif non préemptif [184] [185].

ANNEXE A

# Formulation PLMNE pour un placement dirigé par des métriques intermédiaires au temps

Certaines approches de déploiement [168, 15], adressant l'optimisation du temps de réponse du système, considèrent que le placement des fonctions et des signaux sur les ressources de la plateforme d'exécution peut être déterminé sans avoir nécessité de construire les tâches et de leur attribuer des priorités d'exécution. Lors de cette phase de placement, ces approches se basent sur l'optimisation de simples métriques intermédiaires à la métrique du temps de réponse. Généralement ces métriques intermédiaires concernent la charge du réseau de communication, la contention (ou la concurrence) au sein des ressources de calcul et de communication et l'extensibilité des fonctions par rapport aux futurs changements dans leur WCET. La formulation en programme linéaire mixte en nombres entiers apportée au problème de placement dans ce type d'approches est donnée dans la suite.

## A.1 Formulation PLMNE

Le problème de placement ici consiste à trouver des valeurs pour les variables de décision $A_{i,j}$ et $AS_{i,j}$ optimisant la fonction multi-objectifs formulée par l'équation suivante et respectant toutes les contraintes citées juste après.

$$Maximiser : K_0 \left( \sum_{i \in F} \frac{ext_i}{P_i} \right) - K_1 \left( \sum_{l \in \beta} \sum_{i \in \Phi} AS_{i,l} \frac{\omega_{i,l}}{P_i} \right) - K_2 \left( \sum_{l \in C} \right) cont_l - K_3 \left( \sum_{l \in \beta} \right) contS_l$$

Où, $K_0, K_1, K_2$ et $K_3$ représentent le poids exprimant l'importance associée à chaque objectif. Le premier terme concerne la maximisation de l'extensibilité des fonctions, le second exprime la minimisation de la charge sur le réseau de communication et les deux derniers termes s'adressent respectivement à la minimisation des contentions au sein des processeurs

et des bus de communication.

1. **Placement des fonctions/signaux sur les processeurs/bus :**

$$\begin{cases} \forall i \in F : \sum_{j \in C(i)} A_{i,j} = 1 \\ \forall i \in F : \sum_{j \in C \setminus C(i)} A_{i,j} = 0 \\ \forall i \in \Phi : \sum_{j \in \beta} AS_{i,j} = G_i \\ \forall i \in \Phi, k \in rec(i) : 1 - \sum_{j \in C} X_{snd(i),k,j} = G_i \\ \forall i,k \in F, j \in C : 0 \leq A_{i,j} + A_{k,j} - (2.X_{i,k,j}) \leq 1 \\ \forall i,k \in \Phi, j \in \beta : 0 \leq AS_{i,j} + AS_{k,j} - (2.XS_{i,k,j}) \leq 1 \\ \forall i \in \Phi, l \in \beta, k \in rec(i) : 0 \leq \sum_{j \in PB(l)} A_{snd(i),j} + \sum_{j \in PB(l)} A_{k,j} + G_i - (3.AS_{i,l}) \leq 3 \end{cases}$$

2. **Définition de l'extensibilité et des contraintes d'utilisation de ressources :**

$$\begin{cases} \forall i \in F, j \in C : ext'_{i,j} + \sum_{k \in F} A_{k,j} \left( \frac{\omega_{k,j}}{P_k} \right) \leq \eta_j \\ \forall i \in F, j \in C : \frac{ext_i}{P_i} - M(1 - A_{i,j}) \leq ext'_{i,j} \\ \forall i \in F, j \in C : ext'_{i,j} \leq \frac{ext_i}{P_i} \\ \forall i \in F, j \in C : ext'_{i,j} \leq M.A_{i,j} \\ \forall j \in \beta : \sum_{i \in \Phi} AS_{i,j} \left( \frac{\omega_{i,j}}{P_i} \right) \leq \eta_j \end{cases}$$

3. **Calcul des contentions :**

$$
\begin{cases}
\forall i \in F : Interf_i = \sum_{k \in \{F \setminus i\}} \sum_{j \in C} I_{i,k,j} * \omega_{k,j} \\
\forall i,k \in F, \xi_{i,k} = 1 : 0 \leq \sigma_{i,k} - \left(\frac{P_i}{P_k}\right) < 1 \\
\forall i,k \in F, j \in C, \xi_{i,k} = 1 : \sigma_{i,k} - M(1 - X_{i,k,j}) \leq I_{i,k,j} \\
\forall i,k \in F, j \in C, \xi_{i,k} = 1 : I_{i,k,j} \leq \sigma_{i,k} \\
\forall i,k \in F, j \in C, \xi_{i,k} = 1 : I_{i,k,j} \leq M.X_{i,k,j} \\
\forall i \in F, j \in C : cont_j \geq maxInt_{i,j} \\
\forall i \in F, j \in C : Interf_i - M(1 - A_{i,j}) \leq maxInt_{i,j} \\
\forall i \in F, j \in C : maxInt_{i,j} \leq Interf_i \\
\forall i \in F, j \in C : maxInt_{i,j} \leq M.A_{i,j} \\
\forall i \in \Phi : InterfS_i = \sum_{k \in \{\Phi \setminus i\}} \sum_{j \in \beta} IS_{i,k,j} * \omega_{k,j} \\
\forall i,k \in \Phi, \xi_{i,k} = 1 : 0 \leq \sigma s_{i,k} - \left(\frac{P_i}{P_k}\right) < 1 \\
\forall i,k \in \Phi, j \in \beta, \xi S_{i,k} = 1 : \sigma s_{i,k} - M(1 - XS_{i,k,j}) \leq IS_{i,k,j} \\
\forall i,k \in \Phi, j \in \beta, \xi S_{i,k} = 1 : IS_{i,k,j} \leq \sigma s_{i,k} \\
\forall i,k \in \Phi, j \in \beta, \xi S_{i,k} = 1 : IS_{i,k,j} \leq M.XS_{i,k,j} \\
\forall i \in \Phi, j \in \beta : contS_j \geq maxIntS_{i,j} \\
\forall i \in \Phi, j \in \beta : InterfS_i - M(1 - AS_{i,j}) \leq maxIntS_{i,j} \\
\forall i \in \Phi, j \in \beta : maxIntS_{i,j} \leq InterfS_i \\
\forall i \in \Phi, j \in \beta : maxIntS_{i,j} \leq M.AS_{i,j}
\end{cases}
$$

## A.2 Complexité théorique

La complexité en termes de la taille du problème de la formulation ci-dessus ainsi que sa complexité moyenne ($O_{PLMNE_{Moy}}$) et théorique ($O_{PLMNE_{Pire}}$) sont données par le tableau A.1.

| | |
|---|---|
| #var. binaires ($d$) | $\|F\|^2\|C\| + \|\Phi\|^2\|\beta\| + \|F\|\|C\| + \|\Phi\|(1 + \|\beta\|)$ |
| #var. totales ($n$) | $\|F\|^2(2\|C\|+1) + \|\Phi\|^2(2\|\beta\|+1) + \|F\|(3\|C\|+1) + \|\Phi\|(2\|\beta\|+2) + \|\beta\| + \|C\|$ |
| #contraintes ($m$) | $\|F\|^2(4\|C\|+1) + \|\Phi\|^2(4\|\beta\|+1) + \|F\|(\|\Phi\|\|C\| + \|\Phi\|\|\beta\| + \|\Phi\| - 3\|C\| - 1) + \|\Phi\|(\|\beta\|+1) + \|\beta\|$ |
| $O_{PLMNE_{Moy}}$ = $O(m).O(m.n).O(d^2)$ | $O(\|F\|^2 + \|\Phi\|^2).\ O((\|F\|^2 + \|\Phi\|^2).(\|F\|^2 + \|\Phi\|^2)).\ O(\|F\|^4 + \|\Phi\|^4)$ |
| $O_{PLMNE_{Pire}}$ = $O(2^m).O(2^d)$ | $O(2^{\|F\|^2+\|\Phi\|^2}).\ O(2^{\|F\|^2+\|\Phi\|^2})$ |

Tableau A.1 – Taille du problème et complexité théorique et moyenne de la formulation PLMNE pour une phase de placement considérant des métriques intermédiaires au temps

Pour un système de 16 fonctions, 15 signaux, 10 processeurs, 1 bus de communication et 4 chemins, la complexité de la formulation PLMNE pour une phase de placement considérant des métriques intermédiaires au temps est donnée dans le tableau A.2. Les valeurs entre parenthèses indiquent les résultats pour la phase de placement de l'approche en deux étapes présentée dans cette thèse. À titre d'exemple, nous avons présenté les résultats pour le cas d'activations pilotées par les données lorsque la sémantique de synchronisation est 'Input N-of-N / output N-of-N'.

| | | | |
|---|---|---|---|
| #var. binaires | $d$ | 2975 | (2975) |
| #var. totales | $n$ | 6618 | (6880) |
| #contraintes | $m$ | 14036 | (14413) |
| $O_{PLMNE_{Moy}}$ | | $1,153e^{19}$ | $(1,264e^{19})$ |
| $O_{PLMNE_{Pire}}$ | | $6,625e^{5120}$ | $(1,344e^{1791})$ |

Tableau A.2 – Complexité théorique de la formulation PLMNE d'une phase de placement considérant des métriques intermédiaires au temps pour un système de 16 fonctions, 15 signaux, 10 processeurs, 1 bus de communication et 4 chemins

ANNEXE B

# Formulation PLMNE complète pour l'approche de déploiement en deux étapes

Cette partie décrit en détail la formulation en programmes linéaires mixtes en nombres entiers pour chaque sous problème de l'approche de déploiement en deux étapes. La fonction objectif à optimiser est exprimée de la même manière pour les deux sous problèmes. Si les activations du modèle fonctionnel sont pilotées par les données, elle est comme suit :

$$Minimiser : \frac{\sum_{i \in \gamma} L_i}{\#\gamma}$$

Lorsque le modèle d'activation considéré est périodique, la fonction objectif devient :

$$Minimiser : K_1 \frac{\sum_{\rho_i \in \gamma} \frac{L_i}{D_i}}{\sharp\gamma} + K_2 \frac{\sum_{c \in C} \frac{\delta_c + \epsilon_c}{Mem_c}}{\sharp C}$$

## B.1 Formulation PLMNE pour l'étape de placement

1. **Placement des fonctions/signaux sur les processeurs/bus :**

$$
\begin{cases}
\forall i \in F : \sum_{j \in C(i)} A_{i,j} = 1 \\
\forall i \in F : \sum_{j \in C \setminus C(i)} A_{i,j} = 0 \\
\forall i,j \in F, k \in C : 0 \leq A_{i,k} + A_{j,k} - (2.X_{i,j,k}) \leq 1 \\
\forall j \in C : \sum_{i \in F} A_{i,j} \left( \frac{\omega_{i,j}}{P_i} \right) \leq \eta_j \\
\forall i \in \Phi : \sum_{j \in \beta} AS_{i,j} = G_i \\
\forall i \in \Phi, j \in rec(i) : 1 - \sum_{k \in C} X_{snd(i),j,k} = G_i \\
\forall i,j \in \Phi, k \in \beta : 0 \leq AS_{i,k} + AS_{j,k} - (2 \cdot XS_{i,j,k}) \leq 1 \\
\forall i \in \Phi, j \in rec(i), b \in \beta : 0 \leq G_i + \sum_{k \in PB(b)} A_{snd(i),k} + \sum_{k \in PB(b)} A_{j,k} - 3 \cdot AS_{i,b} \leq 3 \\
\forall j \in \beta : \sum_{i \in \Phi} AS_{i,j} \left( \frac{\omega_{i,j}}{P_i} \right) \leq \eta_j \\
\forall i,j \in \Phi, \pi_i = \pi_j : \sum_{k \in \beta} XS_{i,j,k} \leq \sum_{k \in C} X_{snd(i),snd(j),k}
\end{cases}
$$

2. **Calcul du WCRT pour les fonctions :** la formulation détaillée est donnée pour le cas d'une sémantique de synchronisation de type 'Input N-of-N / output N-of-N' et pour les deux autres sémantiques, nous nous contentons de mentionner la différence avec celle-ci.
   – ***Selon la sémantique de synchronisation 'Input N-of-N / output N-of-N' :***

$$
\begin{cases}
\forall i \in F : R_i = W_i + J_i \\
\forall i,j \in F : 0 \leq \sigma_{i,j} - \left(\frac{W_i + J_j}{P_j}\right) < 1 \\
\forall i,j \in F, \pi j > \pi_i : \sigma_{i,j} - M(1 - \sum_{k \in C} X_{i,j,k}) \leq I_{i,j} \\
\forall i,j \in F, \pi j > \pi_i : I_{i,j} \leq \sigma_{i,j} \\
\forall i,j \in F, \pi j > \pi_i : I_{i,j} \leq M. \sum_{k \in C} X_{i,j,k} \\
\forall i \in F : W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i\}: \\ \pi_j = \pi_i}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1 \\ \pi_j > \pi_i}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\
\forall i \in \gamma, j \in F, j = snk_{(i)} : L_i \geq R_j \\
\forall i \in \gamma : L_i \leq D_i \\
\forall j \in \Phi, i \in rec(j) : \phi_i \geq RS_j \\
\forall i,j \in F, \pi_i = \pi_j : \phi_j - M(1 - \sum_{k \in C} X_{i,j,k}) \leq \theta_{i,j} \\
\forall i,j \in F, \pi_i = \pi_j : \theta_{i,j} \leq \phi_j \\
\forall i,j \in F, \pi_i = \pi_j : \theta_{i,j} \leq M. \sum_{k \in C} X_{i,j,k} \\
\forall i,j \in F, \pi_j = \pi_i : J_i \geq \theta_{i,j}
\end{cases}
$$

   – ***Selon la sémantique de synchronisation 'Input N-of-N / output 1-of-N' :*** la différence avec la formulation précédente réside dans la formule de $W_i$. Celle-ci est remplacée par l'équation suivante :

$$
\begin{cases}
\forall i \in F : W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i\}: \\ \pi_j = \pi_i \\ SO_j > SO_i}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \xi_{i,j}=1 \\ \pi_j > \pi_i}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs)
\end{cases}
$$

   – ***Selon la sémantique de synchronisation 'Input 1-of-N / output 1-of-N' :*** la différence avec la formulation considérant la sémantique 'Input N-of-N / output N-of-N' concerne d'une part la formule de $W_i$ et d'autre part les formules de la gigue. Par rapport à ces dernières, il suffit de supprimer les quatre dernières formules et de mettre la gigue égale à $\phi_i$. Par rapport à $W_i$, celle-ci est remplacée par l'équation suivante :

$$
\begin{cases}
\forall i \in F : W_i = \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i\}: \\ \xi_{i,j}=1 \\ \pi_j = \pi_i}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in \{F \setminus i\}: \\ \pi_j = \pi_i \\ dp_{j,i}=1}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} \\
\qquad + \sum_{\substack{j \in F: \\ \xi_{i,j}=1 \\ \pi_j > \pi_i}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs)
\end{cases}
$$

3. **Calcul du WCRT pour les signaux :**

$$
\begin{cases}
\forall i \in \Phi : RS_i = WS_i + \sum\limits_{k \in \beta} AS_{i,k} * \omega_{i,k} + \sum\limits_{\substack{j \in (\Phi \setminus i): \\ \pi_i = \pi_j}} \sum\limits_{k \in \beta} XS_{i,j,k} * \omega_{j,k} + JS_i \\
\forall i,j \in \Phi : 0 \leq \sigma s_{i,j} - (\frac{WS_i + JS_j}{P_j}) < 1 \\
\forall i,j \in \Phi, \pi j > \pi_i : \sigma s_{i,j} - M \left( 1 - \sum\limits_{k \in \beta} XS_{i,j,k} \right) \leq IS_{i,j} \\
\forall i,j \in \Phi, \pi j > \pi_i : IS_{i,j} \leq \sigma s_{i,j} \\
\forall i,j \in \Phi, \pi j > \pi_i : IS_{i,j} \leq M. \sum\limits_{k \in \beta} XS_{i,j,k} \\
\forall i,j \in \Phi, \pi_j < \pi_i, k \in \beta : BS_{i,k} \geq XS_{i,j,k} * \omega_{j,k} + \sum\limits_{\substack{l \in \{\Phi \setminus (i,j)\}: \\ \pi_j = \pi_l}} XS_{j,l,k} * \omega_{l,k} \\
\forall i \in \Phi, k \in \beta : BS_{i,k} \geq \sum\limits_{\substack{j \in \Phi: \\ \pi_i = \pi_j}} XS_{i,j,k} * \omega_{j,k} \\
\forall i \in \Phi : WS_i = \sum\limits_{k \in \beta} BS_{i,k} + \sum\limits_{\substack{j \in \Phi: \\ \pi_j > \pi_i \\ \xi S_{i,j} = 1}} \sum\limits_{k \in \beta} IS_{i,j} * \omega_{j,k} \\
\forall i \in \Phi : R_{snd(i)} - M\left(1 - (1 - G_i)\right) \leq \alpha_i \\
\forall i \in \Phi : \alpha_i \leq R_{snd(i)} \\
\forall i \in \Phi : \alpha_i \leq M(1 - G_i) \\
\forall i \in \Phi : J_{snd(i)} - M\left(1 - (1 - G_i)\right) \leq \int_i \\
\forall i \in \Phi : \int_i \leq J_{snd(i)} \\
\forall i \in \Phi : \int_i \leq M(1 - G_i) \\
\forall i \in \Phi, j \in rec(i) : JS_i \geq \alpha_i(1 - SP_{snd(i),j}) + \int_i * SP_{snd(i),j} \\
\forall i,j \in \Phi : JS_i \geq \Lambda_{i,j} \\
\forall i,j \in \Phi, \pi_j = \pi_i : R_{snd(j)} - M(1 - \sum\limits_{k \in \beta} XS_{i,j,k}) \leq \Lambda_{i,j} \\
\forall i,j \in \Phi, \pi_j = \pi_i : \Lambda_{i,j} \leq R_{snd(j)} \\
\forall i,j \in \Phi, \pi_j = \pi_i : \Lambda_{i,j} \leq M. \sum\limits_{k \in C} XS_{i,j,k}
\end{cases}
$$

4. **Modèle d'activation périodique :** lorsque les activations sont périodiques, les contraintes de placement sont identiques à celles présentées pour le cas d'activations pilotées par les données. Par rapport aux contraintes de calcul des pires temps de réponse, tout d'abord, nous avons besoin d'ajouter les contraintes suivantes qui sont liées à l'attribution des mécanismes de protection pour les données partagées.

$$
\begin{cases}
\forall i \in \Phi : Y_{\zeta_i} \leq \sum\limits_{\substack{j \in rec(i): \\ \pi_{snd(i)} \neq \pi_j}} \sum\limits_{k \in C} X_{snd(i),j,k} \\
\forall i \in \Phi, j \in rec(i), \pi_{snd(i)} \neq \pi_j : Y_{\zeta_i} \geq \sum_{k \in C} X_{snd(i),j,k} \\
\forall i \in \Phi : Y_{\zeta_i} = mem_{\zeta_i} + lock_{\zeta_i} \\
\forall i \in \Phi : V_{\zeta_i} \leq \sum\limits_{\substack{j \in rec(i): \\ \pi_{snd(i)} < \pi_j}} \sum\limits_{k \in C} X_{snd(i),j,k} \\
\forall i \in \Phi, j \in rec(i), \pi_{snd(i)} < \pi_j : V_{\zeta_i} \geq \sum_{k \in C} X_{snd(i),j,k} \\
\forall i \in \Phi : \delta'_{\zeta_i} = DS(i) \cdot \left( \sum\limits_{\substack{j \in rec(i): \\ \pi_{snd(i)} > \pi_j}} Z_{j,snd(i)} + Z'_{\zeta_i} + mem_{\zeta_i} \right) \\
\forall i \in \Phi, j \in rec(k), \pi_j < \pi_{snd(i)} : 0 \leq \sum\limits_{k \in C} X_{j,snd(i),k} + mem_{\zeta_i} - (2.Z_{j,snd(i)}) \leq 1 \\
\forall i \in \Phi : 0 \leq V_{\zeta_i} + mem_{\zeta_i} - (2.Z'_{\zeta_i}) \leq 1 \\
\forall k \in C : \delta_k = \sum\limits_{i \in \Phi} \delta_{i,k}" \\
\forall i \in \Phi, k \in C, j \in rec(i), \pi_{snd(i)} \neq \pi_j : \delta'_{\zeta_i} - M(1 - X_{snd(i),j,k}) \leq \delta_{i,k}" \\
\forall i \in \Phi, k \in C, j \in rec(i), \pi_{snd(i)} \neq \pi_j : \delta_{i,k}" \leq \delta'_{\zeta_i} \\
\forall i \in \Phi, k \in C, j \in rec(i), \pi_{snd(i)} \neq \pi_j : \delta_{i,k}" \leq M.X_{snd(i),j,k} \\
\forall k \in C : \delta_k + \epsilon_k \leq Mem_k \\
\forall i, j \in F, k \in \Phi : B_i \geq cond_{i,j,\zeta_k} \cdot \omega'_{j,\zeta_k} \\
\forall i, j \in F, k \in \Phi, \pi_i > \pi_j : 0 \leq PC_{i,\zeta_k} + lock_{\zeta_k} + \sum\limits_{l \in C} X_{j,i,l} - 3 \cdot cond_{i,j,\zeta_k} \leq 2 \\
\forall k \in \Phi, i \in \{snd(k) \cup rec(k)\} : PC_{i,\zeta_k} = lock_{\zeta_k} \\
\forall k \in \Phi, i \in \{F \setminus (snd(k) \cup rec(k))\}, j \in (snd(k) \cup rec(k)), \pi_j \geq \pi_i : PC_{i,\zeta_k} \geq \sum\limits_{l \in C} X_{i,j,l} \\
\forall k \in \Phi, i \in \{F \setminus (snd(k) \cup rec(k))\} : PC_{i,\zeta_k} \leq \sum\limits_{\substack{j \in \{snd(k) \cup rec(k)\}: \\ \pi_j \geq \pi_i}} \sum\limits_{l \in C} X_{i,j,l}
\end{cases}
$$

Ensuite, toutes les formules permettant de calculer les WCRTs pour les fonctions et les signaux sont remplacées par les équations suivantes :

$$
\begin{cases}
\forall i,j \in F : 0 \leq \sigma_{i,j} - \left(\frac{R_i}{P_j}\right) < 1 \\
\forall i,j \in F, \pi j > \pi_i : \sigma_{i,j} - M\left(1 - \sum_{k \in C} X_{i,j,k}\right) \leq I_{i,j} \\
\forall i,j \in F, \pi j > \pi_i : I_{i,j} \leq \sigma_{i,j} \\
\forall i,j \in F, \pi j > \pi_i : I_{i,j} \leq M. \sum_{k \in C} X_{i,j,k} \\
\forall i \in F : R_i = B_i + \sum_{k \in C} A_{i,k}(\omega_{i,k} + 2.cs) + \sum_{\substack{j \in \{F \setminus i\}: \\ \pi_j = \pi_i \\ So_j > So_i}} \sum_{k \in C} X_{i,j,k} * \omega_{j,k} + \sum_{\substack{j \in F: \\ \pi_j > \pi_i}} \sum_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\
\forall i \in \gamma : L_i = \sum_{j \in \rho_i} R_j + \sum_{\substack{k \in \rho_i: \\ l \in rec(k) w.r.t\ \rho_i}} RS_k + (P_l + P_k)G_k \\
\forall i \in \gamma : L_i \leq D_i \\
\forall i \in F : R_i \leq P_i \\
\forall i \in \Phi : RS_i \leq P_i
\end{cases}
$$

$$
\begin{cases}
\forall i \in \Phi : RS_i = WS_i + \sum_{k \in \beta} AS_{i,k} * \omega_{i,k} + \sum_{\substack{j \in \{\Phi \setminus i\}: \\ \pi_i = \pi_j}} \sum_{k \in \beta} XS_{i,j,k} * \omega_{j,k} \\
\forall i,j \in \Phi : 0 \leq \sigma s_{i,j} - \left(\frac{WS_i}{P_j}\right) < 1 \\
\forall i,j \in \Phi, \pi j > \pi_i : \sigma s_{i,j} - M\left(1 - \sum_{k \in \beta} XS_{i,j,k}\right) \leq IS_{i,j} \\
\forall i,j \in \Phi, \pi j > \pi_i : IS_{i,j} \leq \sigma s_{i,j} \\
\forall i,j \in \Phi, \pi j > \pi_i : IS_{i,j} \leq M. \sum_{k \in \beta} XS_{i,j,k} \\
\forall i,j \in \Phi, \pi_j < \pi_i, k \in \beta : BS_{i,k} \geq XS_{i,j,k} * \omega_{j,k} + \sum_{\substack{l \in \{\Phi \setminus (i,j)\}: \\ \pi_j = \pi_l}} XS_{j,l,k} * \omega_{l,k} \\
\forall i \in \Phi, k \in \beta : BS_{i,k} \geq \sum_{\substack{j \in \Phi: \\ \pi_i = \pi_j}} XS_{i,j,k} * \omega_{j,k} \\
\forall i \in \Phi : WS_i = \sum_{k \in \beta} BS_{i,k} + \sum_{\substack{j \in \Phi: \\ \pi_j > \pi_i}} \sum_{k \in \beta} IS_{i,j} * \omega_{j,k}
\end{cases}
$$

## B.2 Formulation PLMNE pour l'étape de partitionnement et d'ordonnancement

1. **Partitionnement et ordonnancement des fonctions :**

$$\begin{cases} \forall i,j \in F : \psi_{i,j} + \psi_{j,i} \leq 1 \\ \forall i,j,k \in F : \psi_{i,j} + \psi_{j,k} - 1 \leq \psi_{i,k} \\ \forall i,j,k \in F : \psi_{i,j} - (\psi_{j,k} + \psi_{k,j}) \leq \psi_{i,k} \\ \forall i,j,k \in F : \psi_{j,k} - (\psi_{j,i} + \psi_{i,j}) \leq \psi_{i,k} \\ \forall i,j,k \in F : \psi_{i,j} + \psi_{j,i} + \psi_{j,k} + \psi_{k,j} \geq \psi_{i,k} \\ \forall i,j,k \in F : \psi_{i,j} + \psi_{j,i} + \psi_{j,k} + \psi_{k,j} \geq \psi_{k,i} \\ \forall i,j \in F : 1 = SP_{i,j} + \psi_{j,i} + \psi_{i,j} \end{cases}$$

2. **Partitionnement et ordonnancement des signaux :**

$$\begin{cases} \forall i,j \in \Phi : \psi S_{i,j} + \psi S_{j,i} \leq 1 \\ \forall i,j,k \in \Phi : \psi S_{i,j} + \psi S_{j,k} - 1 \leq \psi S_{i,k} \\ \forall i,j,k \in \Phi : \psi S_{i,j} - (\psi S_{j,k} + \psi_{k,j}) \leq \psi S_{i,k} \\ \forall i,j,k \in \Phi : \psi S_{j,k} - (\psi S_{j,i} + \psi_{i,j}) \leq \psi S_{i,k} \\ \forall i,j,k \in \Phi : \psi S_{i,j} + \psi S_{j,i} + \psi_{j,k} + \psi S_{k,j} \geq \psi S_{i,k} \\ \forall i,j,k \in \Phi : \psi S_{i,j} + \psi S_{j,i} + \psi S_{j,k} + \psi S_{k,j} \geq \psi S_{k,i} \\ \forall i,j \in \Phi : 1 = SPS_{i,j} + \psi S_{j,i} + \psi S_{i,j} \\ \forall i \in \Phi, j,k \in rec(i), SN_{j,k} = 1 : SP_{snd(i),j} - SP_{snd(i),k} = 0 \\ \forall i,j \in \Phi, \Omega_i = \Omega_j, SN_{snd(i),snd(j)} = 0 : SPS_{i,j} = 0 \end{cases}$$

3. **Calcul du WCRT pour les fonctions :** de même que pour la phase de placement, nous détaillons la formulation pour la sémantique de synchronisation 'Input N-of-N / output N-of-N', puis nous montrons uniquement la différence pour les autres sémantiques.

– ***Selon la sémantique de synchronisation 'Input N-of-N / output N-of-N' :***

$$\begin{cases} \forall i \in F : R_i = W_i + J_i \\ \forall i,j \in F : 0 \leq \sigma_{i,j} - \left(\frac{W_i + J_j}{P_j}\right) < 1 \\ \forall i,j \in F, SN_{i,j} = 1 : \sigma_{i,j} - M\,(1 - \psi_{j,i}) \leq I_{i,j} \\ \forall i,j \in F, SN_{i,j} = 1 : I_{i,j} \leq \sigma_{i,j} \\ \forall i,j \in F, SN_{i,j} = 1 : I_{i,j} \leq M.\psi_{j,i} \\ \forall i \in F, Proc_i = k : W_i = \omega_{i,k} + 2.cs + \sum\limits_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1}} SP_{i,j} * \omega_{j,k} + \sum\limits_{\substack{j \in F: \\ \xi_{i,j}=1 \\ SN_{i,j}=1}} \sum\limits_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\ \forall i \in \gamma, j \in F, j = snk_{(i)} : L_i \geq R_j \\ \forall i \in \gamma : L_i \leq D_i \\ \forall j \in \Phi, i \in rec(j) : \phi_i \geq RS_j \\ \forall i,j \in F, SN_{i,j} = 1 : \phi_j - M(1 - SP_{i,j}) \leq \theta_{i,j} \\ \forall i,j \in F, SN_{i,j} = 1 : \theta_{i,j} \leq \phi_j \\ \forall i,j \in F, SN_{i,j} = 1 : \theta_{i,j} \leq M.SP_{i,j} \\ \forall i,j \in F, SN_{i,j} = 1 : J_i \geq \theta_{i,j} \end{cases}$$

– ***Selon la sémantique de synchronisation 'Input N-of-N / output 1-of-N' :*** pour considérer cette sémantique dans notre formulation, il suffit de remplacer la formule du $W_i$ par les formules suivantes :

$$\begin{cases} \forall i \in F, Proc_i = k : W_i = \omega_{i,k} + 2.cs + \sum\limits_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1}} SPGO_{i,j} * \omega_{j,k} + \sum\limits_{\substack{j \in F: \\ \xi_{i,j}=1 \\ SN_{i,j}=1}} \sum\limits_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\ \forall i,j \in F : 0 \leq SP_{i,j} + So_{j,i} - (2.SPGO_{i,j}) \leq 1 \\ \forall i,j \in F : So_{i,j} + So_{j,i} = 1 \\ \forall i,j,k \in F : So_{i,j} + So_{j,k} - 1 \leq So_{i,k} \\ \forall i,j \in F : So_{i,j} = 1 \text{ Si } dp_{i,j} = 1 \end{cases}$$

– ***Selon la sémantique de synchronisation 'Input 1-of-N / output 1-of-N' :*** là aussi il faut supprimer les quatre dernières équations liées au calcul de la gigue des fonctions puis fixer la gigue à $\phi_i$. De plus, il faut remplacer la formule du $W_i$ par les formules suivantes :

$$\begin{cases} \forall i \in F, Proc_i = k : W_i = \omega_{i,k} + 2.cs + \sum\limits_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1 \\ \xi_{i,j}=1}} SP_{i,j} * \omega_{j,k} + \sum\limits_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1 \\ dp_{j,i}=1}} SP_{i,j} * \omega_{j,k} \\ \qquad\qquad + \sum\limits_{\substack{j \in F: \\ \xi_{i,j}=1 \\ SN_{i,j}=1}} \sum\limits_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\ \forall i,j \in S : 1 = \psi_{i,j} + \psi_{j,i} \text{ Si } (P_i \geq P_j) \text{ et } (P_i \text{ modulo } P_j) \neq 0 \end{cases}$$

4. **Calcul du WCRT pour les signaux :**

$$
\begin{cases}
\forall i \in \Phi, \Omega_i = k : RS_i = WS_i + \omega_{i,k} * G_i + \sum_{\substack{j \in \{\Phi \setminus i\}: \\ \Omega_j = k}} SPS_{i,j} * \omega_{j,k} + JS_i \\
\forall i,j \in \Phi : 0 \leq \sigma s_{i,j} - \left( \frac{WS_i + JS_j}{P_j} \right) < 1 \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : \sigma s_{i,j} - M\,(1 - \psi S_{j,i}) \leq IS_{i,j} \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : IS_{i,j} \leq \sigma s_{i,j} \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : IS_{i,j} \leq M.\psi S_{j,i} \\
\forall i \in \Phi, G_i = 0 : BS_i = 0 \\
\forall i,j \in \Phi, \Omega_i = \Omega_j = k : BS_i \geq \psi S_{i,j} * \omega_{j,k} + \sum_{\substack{l \in \{\Phi \setminus (i,j)\}: \\ \Omega_j = \Omega_l}} SPS_{j,l} * \omega_{l,k} \\
\forall i,j \in \Phi : BS_i \geq \sum_{\substack{j \in \Phi: \\ \Omega_i = \Omega_j = k}} SPS_{i,j} * \omega_{j,k} \\
\forall i \in \Phi : WS_i = BS_i + \sum_{\substack{j \in \Phi: \\ \Omega_j = \Omega_i = k \\ \xi S_{i,j} = 1}} IS_{i,j} * \omega_{j,k} \\
\forall i \in \Phi, j \in rec(i), G_i = 0 : R_{snd(i)} - M\left(1 - (1 - SP_{snd(i),j})\right) \leq \alpha_i \\
\forall i \in \Phi : \alpha_i \leq R_{snd(i)} \\
\forall i \in \Phi : \alpha_i \leq M(1 - SP_{snd(i),j}) \\
\forall i \in \Phi : J_{snd(i)} - M\left(1 - SP_{snd(i),j}\right) \leq \int_i \\
\forall i \in \Phi : \int_i \leq J_{snd(i)} \\
\forall i \in \Phi : \int_i \leq M.SP_{snd(i),j} \\
\forall i \in \Phi G_i = 0 : JS_i = \alpha_i + \int_i \\
\forall i,j \in \Phi, G_i = 1 : JS_i \geq \Lambda_{i,j} \\
\forall i,j \in \Phi, \Omega_j = \Omega_i : R_{snd(j)} - M\,(1 - SPS_{i,j}) \leq \Lambda_{i,j} \\
\forall i,j \in \Phi, \Omega_j = \Omega_i : \Lambda_{i,j} \leq R_{snd(j)} \\
\forall i,j \in \Phi, \Omega_j = \Omega_i : \Lambda_{i,j} \leq M.SPS_{i,j}
\end{cases}
$$

5. **Modèle d'activation périodique :** les contraintes de partitionnement et d'ordonnancement pour ce modèle d'activation sont identiques à celles présentées pour les activations pilotées par les données. Similairement à la phase précédente, pour la formulation du calcul des temps de réponse dans le cas d'activations périodiques, il faut tout d'abord ajouter les contraintes suivantes qui sont liées à l'attribution des mécanismes de protection pour les données partagées.

$$
\begin{cases}
\forall i \in \Phi : Y_{\zeta_i} \leq \sum\limits_{\substack{j \in rec(i): \\ SN_{snd(i),j}=1}} (1 - SP_{snd(i),j}) \\
\forall i \in \Phi, j \in rec(i), SN_{snd(i),j} = 1 : Y_{\zeta_i} \geq (1 - SP_{snd(i),j}) \\
\forall i \in \Phi : Y_{\zeta_i} = mem_{\zeta_i} + lock_{\zeta_i} \\
\forall i \in \Phi : V_{\zeta_i} \leq \sum\limits_{\substack{j \in rec(i): \\ SN_{snd(i),j}=1}} \psi_{j,snd(i)} \\
\forall i \in \Phi, j \in rec(i), SN_{snd(i),j} = 1 : V_{\zeta_i} \geq \psi_{j,snd(i)} \\
\forall i \in \Phi : \delta'_{\zeta_{s_i}} = DS(i) \cdot \left( \sum\limits_{\substack{j \in rec(i): \\ SN_{snd(i),j}=1}} Z_{j,snd(i)} + Z'_{\zeta_i} + mem_{\zeta_i} \right) \\
\forall i \in \Phi, j \in rec(k) : 0 \leq \psi_{snd(i),j} + mem_{\zeta_i} - (2.Z_{j,snd(i)}) \leq 1 \\
\forall i \in \Phi : 0 \leq V_{\zeta_i} + mem_{\zeta_i} - (2.Z'_{\zeta_i}) \leq 1 \\
\forall k \in C : \delta_k = \sum\limits_{i \in \Phi} \delta_{i,k}" \\
\forall i \in \Phi, k \in C, j \in rec(i), SN_{snd(i),j} = 1, Proc_j = k : \delta'_{\zeta_i} - M(1 - (1 - SP_{snd(i),j})) \leq \delta_{i,k}" \\
\forall i \in \Phi, k \in C, j \in rec(i), SN_{snd(i),j} = 1, Proc_j = k : \delta_{i,k}" \leq \delta'_{\zeta_i} \\
\forall i \in \Phi, k \in C, j \in rec(i), SN_{snd(i),j} = 1, Proc_j = k : \delta_{i,k}" \leq M(1 - SP_{snd(i),j}) \\
\forall k \in C : \delta_k + \epsilon_k \leq Mem_k \\
\forall i, j \in F, k \in \Phi : B_i \geq cond_{i,j,\zeta_k} \cdot \omega'_{j,\zeta_k} \\
\forall i, j \in F, k \in \Phi, SN_{i,j} = 1 : 0 \leq PC_{i,\zeta_k} + lock_{\zeta_k} + \psi_{i,j} - 3 \cdot cond_{i,j,\zeta_k} \leq 2 \\
\forall k \in \Phi, i \in \{snd(k) \cup rec(k)\} : PC_{i,\zeta_k} = lock_{\zeta_k} \\
\forall k \in \Phi, i \in \{F \setminus (snd(k) \cup rec(k))\}, j \in (snd(k) \cup rec(k)), SN_{i,j} = 1 : PC_{i,\zeta_k} \geq \psi_{j,i} + SP_{i,j} \\
\forall k \in \Phi, \forall i \in \{F \setminus (snd(k) \cup rec(k))\} : PC_{i,\zeta_k} \leq \sum\limits_{\substack{j \in \{snd(k) \cup rec(k)\}: \\ SN_{i,j}=1}} \psi_{j,i} + \sum\limits_{\substack{j \in \{snd(k) \cup rec(k)\}: \\ SN_{i,j}=1}} SP_{i,j}
\end{cases}
$$

Ensuite, les formules de calcul des WCRTs pour les fonctions et les signaux sont remplacées, respectivement, par les deux ensembles d'équations suivants :

$$
\begin{cases}
\forall i,j \in F : 0 \leq \sigma_{i,j} - \left(\frac{R_i}{P_j}\right) < 1 \\
\forall i,j \in F, SN_{i,j} = 1 : \sigma_{i,j} - M\left(1 - \psi_{j,i}\right) \leq I_{i,j} \\
\forall i,j \in F, SN_{i,j} = 1 : I_{i,j} \leq \sigma_{i,j} \\
\forall i,j \in F, SN_{i,j} = 1 : I_{i,j} \leq M\psi_{j,i} \\
\forall i \in F, Proc_i = k : R_i = B_i\omega_{i,k} + 2.cs + \sum\limits_{\substack{j \in \{F \setminus i\}: \\ SN_{i,j}=1}} SPGO_{i,j} * \omega_{j,k} + \sum\limits_{\substack{j \in F: \\ SN_{i,j}=1}} \sum\limits_{k \in C} I_{i,j}(\omega_{j,k} + 2.cs) \\
\forall i,j \in F : 0 \leq SP_{i,j} + So_{j,i} - (2.SPGO_{i,j}) \leq 1 \\
\forall i,j \in F : So_{i,j} + So_{j,i} = 1 \\
\forall i,j,k \in F : So_{i,j} + So_{j,k} - 1 \leq So_{i,k} \\
\forall i \in \gamma : L_i = \sum\limits_{j \in \rho_i} R_j + \sum\limits_{\substack{k \in \rho_i: \\ G_k = 1 \\ l \in rec(k) \text{vis-à-vis } \rho_i}} RS_k + P_l + P_k \\
\forall i \in \gamma : L_i \leq D_i \\
\forall i \in F : R_i \leq P_i \\
\forall i \in \Phi : RS_i \leq P_i \\
\forall i,j \in F : 1 = \psi_{i,j} + \psi_{j,i} \text{ Si } (P_i \geq P_j) \text{ et } (P_i \text{ modulo } P_j) \neq 0
\end{cases}
$$

$$
\begin{cases}
\forall i \in \Phi, \Omega_i = k : RS_i = WS_i + \omega_{i,k} * G_i + \sum\limits_{\substack{j \in \{\Phi \setminus i\}: \\ \Omega_j = k}} SPS_{i,j} * \omega_{j,k} \\
\forall i,j \in \Phi : 0 \leq \sigma s_{i,j} - \left(\frac{WS_i}{P_j}\right) < 1 \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : \sigma s_{i,j} - M\left(1 - \psi S_{j,i}\right) \leq IS_{i,j} \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : IS_{i,j} \leq \sigma s_{i,j} \\
\forall i,j \in \Phi, \Omega_i = \Omega_j : IS_{i,j} \leq M.\psi S_{j,i} \\
\forall i \in \Phi, G_i = 0 : BS_i = 0 \\
\forall i,j \in \Phi, \Omega_i = \Omega_j = k : BS_i \geq \psi_{i,j} * \omega_{j,k} + \sum\limits_{\substack{l \in \{\Phi \setminus (i,j)\}: \\ \Omega_j = \Omega_l}} SPS_{j,l} * \omega_{l,k} \\
\forall i,j \in \Phi : BS_i \geq \sum\limits_{\substack{j \in \Phi: \\ \Omega_i = \Omega_j = k}} SPS_{i,j} * \omega_{j,k} \\
\forall i \in \Phi : WS_i = BS_i + \sum\limits_{\substack{j \in \Phi: \\ \Omega_j = \Omega_i = k}} IS_{i,j} * \omega_{j,k} \\
\forall i,j \in \Phi : 1 = \psi S_{i,j} + \psi S_{j,i} \text{ Si } (P_i \geq P_j) \text{ et } (P_i \text{ modulo } P_j) \neq 0
\end{cases}
$$

ANNEXE C

# Documentation sur le paramétrage du solveur CPLEX

Dans ce chapitre, nous introduisons les principaux paramètres de CPLEX et nous mentionnons par la suite leur valeur par défaut.

Ces paramètres concernent par exemple **l'insistance sur la faisabilité et l'optimalité** qui représente le compromis entre une solution réalisable (respectant toutes les contraintes) obtenue dans un intervalle de temps raisonnable et une solution optimale retournée après une durée plus au moins longue. La valeur par défaut pour ce paramètre consiste à effectuer des analyses avant la phase de séparation de l'algorithme "Branch-and-Cut" afin d'espérer de trouver rapidement la solution optimale. Si le modèle est assez difficile pour prouver l'optimalité et que l'utilisateur se contente de trouver une bonne solution, ce paramètre peut être réglé de telle sorte que moins d'efforts de calcul soient attribués à la preuve de l'optimalité au profit de la recherche des solutions réalisables afin d'en trouver le plus tôt. Lorsque la faisabilité est privilégiée la preuve de l'optimalité peut prendre plus de temps par rapport au cas où l'on prend la valeur par défaut. Quelle que soit la valeur choisie pour ce paramètre, l'algorithme va soit trouver la solution optimale, soit prouver qu'il n'existe pas de solutions réalisables. Les différents choix permettent seulement de guider CPLEX à trouver des solutions réalisables par rapport aux besoins de l'utilisateur.

Un autre paramètre important est **la valeur de coupure**. Celui-ci détermine la valeur pour laquelle le solveur doit élaguer toute solution ayant une fonction objectif plus grande (si on considère un problème de minimisation), et inversement plus petite (si on considère un problème de maximisation), que la valeur de coupure. Il peut prendre n'importe quelle valeur mais la valeur par défaut signifie qu'aucune coupure n'est fournie. Cette valeur par défaut n'écarte pas les coupures faites naturellement par l'algorithme de "Branch-and-Cut" lorsque la fonction objectif de la solution courante est utilisée pour l'élagage de branches. L'utilisateur peut aussi manipuler CPLEX pour l'arrêter dès que son but est atteint. Pour cela, il modifie la condition d'arrêt qui peut être le temps, le nombre de nœud parcourus, la taille de l'arbre de recherche ou le nombre de solutions trouvées. Par défaut CPLEX s'arrête lorsque la solution optimale a été prouvée.

Dans le même but, CPLEX définit le paramètre **tolérance d'écart relatif ou absolue** qui

détermine en pourcentage la marge entre l'objectif de la meilleure solution entière trouvée à un instant donné et l'objectif de la meilleure solution du problème relâché parmi tous les nœud restant. La valeur par défaut pour ce paramètre est de $0.01\%$ et par conséquent CPLEX s'arrête lorsque cette valeur est obtenue. Cette valeur par défaut est due à l'imprécision des valeurs numériques.

CPLEX comprend également un ensemble de paramètres permettant de contrôler la stratégie de recherche qu'il va poursuivre. Ces paramètres permettent de contrôler le chemin que CPLEX traverse lors du parcours de l'arbre de recherche. Parmi ces paramètres nous avons la **stratégie de sélection de prochain nœud** à aborder dans l'arbre de recherche. Par défaut CPLEX choisit le nœud avec la meilleure valeur de la fonction objectif. Le parcours en profondeur est aussi une valeur possible pour ce paramètre. La fréquence à laquelle le nœud avec la meilleure fonction objectif est sélectionné peut être déterminée à l'aide du paramètre **intervalle des meilleures bornes** qui prend par défaut la valeur $7$ signifiant que cette stratégie est appliquée occasionnellement. Ensuite, nous avons la **stratégie de sélection de variable fractionnaire** sur laquelle le prochain branchement est fait. En se basant sur le problème et sur ses progrès, CPLEX sélectionne par défaut la meilleure règle pour décider de la prochaine variable à brancher. Parmi les règles connues à CPLEX, nous citons comme exemple la règle d'infaisabilité minimale qui consiste à choisir la variable fractionnaire la plus proche d'un nombre réel, et celle d'infaisabilité maximale qui revient à choisir la variable fractionnaire la plus loin d'un nombre réel. L'idée derrière la règle d'infaisabilité minimale est de converger rapidement vers une solution entière tandis que pour l'infaisabilité maximale, le principe est d'effectuer les grands changements le plus tôt possible pendant le parcours de l'arbre. Puis, la **direction de la séparation** consiste à déterminer quelle branche explorer en premier pour une variable donnée (celle de droite ou celle de gauche). Par défaut CPLEX décide automatiquement en fonction de l'état du problème. Enfin, nous avons la **tolérance de retour sur trace** déterminant la fréquence à laquelle le retour sur trace est effectué lors de la phase de séparation. La valeur par défaut pour ce paramètre est de $1$, ce qui implique que la probabilité pour qu'un retour en arrière soit fait avant l'aboutissement à une nouvelle solution courante, une solution infaisable ou une coupure est nulle. Ce paramètre se rapproche de la valeur $0$, plus la probabilité qu'un retour en arrière soit fait avant l'arrivée à un point d'arrêt sur la branche est élevée. Veuillez noter que ce paramètre permet simplement à CPLEX de faire des retours sur trace mais il ne le force pas.

CPLEX propose aussi d'accélérer le temps d'exécution en appliquant certaines heuristiques lors de la procédure de "Branch-and-cut". Avec le réglage par défaut, CPLEX invoque automatiquement des heuristiques lorsqu'il juge cela nécessaire. Par exemple, l'heuristique "**pompe de faisabilité**" consiste à trouver une solution réalisable pour le nœud racine de l'arbre. Avoir une solution réalisable dès le départ du "Branch-and-Cut" permet à CPLEX d'éliminer des portions de l'espace de recherche et donc de rétrécir l'arbre et réduire la taille globale du programme linéaire en nombres entiers. Par défaut CPLEX choisi automatiquement d'appliquer ou non cette heuristique. Dans le cas où elle est appliquée, il décide soit de trouver une bonne solution réalisable à la racine soit de se contenter que d'une solution réalisable.

# Bibliographie


[1] Bekooij, M., Hoes, R., Moreira, O., Poplavko, P., Pastrnak, M., Mesman, B., Mol, J. D., Stuijk, S., Gheorghita, V., and van Meerbergen, J. "Dataflow analysis for real-time embedded multiprocessor system design", In *Dynamic and robust streaming in and between connected consumer-electronic devices*, 81–108. Springer (2005).

[2] Bartolini, C., Lipari, G., and Di Natale, M. "From functional blocks to the synthesis of the architectural model in embedded real-time applications", In *Proc. 11th IEEE Real Time and Embedded Technology and Applications Symposium*, 458–467, (2005).

[3] http://www.cvel.clemson.edu/auto/AuE835_Projects_2011/Ramesh_project.html.

[4] Cottet, F. and Grolleau, E. "Systèmes temps réel de contrôle-commande-Conception et implémentation : Conception et implémentation", Dunod, (2005).

[5] "AUTOSAR 4.0 Specifications", http://www.autosar.org/.

[6] Consortium, A. et al. "East-ADL2 specification", Technical report, ITEA, (2008).

[7] OMG. "UML Profile for MARTE : Modeling and Analysis of Real-Time Embedded Systems, version 1.1", http ://www.omg.org/spec/MARTE/1.1, (2011).

[8] Feiler, P. H., Gluch, D. P., and Hudak, J. J. "The architecture analysis & design language (AADL) : An introduction", Technical report, DTIC Document, (2006).

[9] Perrotin, M., Conquet, E., Dissaux, P., Tsiodras, T., and Hugues, J. "The TASTE Toolset : turning human designed heterogeneous systems into computer built homogeneous software", In *Proceedings of the 5 th International Congress Embedded Real-Time Software and Systems (ERTS 2010). Toulouse, France*, 19–21. Citeseer, (2010).

[10] Pop, P., Eles, P., Peng, Z., and Pop, T. "Analysis and optimization of distributed real-time embedded systems", In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, volume 11, 593–625. ACM, (2004).

[11] Saksena, M., Karvelas, P., and Wang, Y. "Automatic synthesis of multi-tasking implementations from real-time object-oriented models", In *Proceedings of 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 360–367, (2000).

[12] Kodase, S., Wang, S., and Shin, K. "Transforming structural model to runtime model of embedded software with real-time constraints", In *Proceedings of the conference on Design, Automation and Test in Europe*, 170–175, (2003).

[13] Wang, S. and Shin, K. "Task construction for model-based design of embedded control software", In *IEEE Transactions on Software Engineering* **32**(4), 254–264 (2006).

[14] Zheng, W., Zhu, Q., Di Natale, M., and Sangiovanni-Vincentelli, A. "Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems", In *Proc. of the IEEE RTSS Conference*, (2007).

[15] Kugele, S., Haberl, W., Tautschnig, M., and Wechs, M. "Optimizing automatic deployment using non-functional requirement annotations", In *Leveraging Applications of Formal Methods, Verification and Validation*, 400–414. Springer (2009).

[16] Zhu, Q., Yang, Y., Scholte, E., Di Natale, M., and Sangiovanni-Vincentelli, A. "Optimizing extensibility in hard real-time distributed systems", In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, 275–284. IEEE, (2009).

[17] Zhu, Q., Yang, Y., Di Natale, M., Scholte, E., and Sangiovanni-Vincentelli, A. "Optimizing the software architecture for extensibility in hard real-time distributed systems", In *Industrial Informatics, IEEE Transactions on* **6**(4), 621–636 (2010).

[18] Zhang, M. and Gu, Z. "Optimization issues in mapping AUTOSAR components to distributed multithreaded implementations", In *22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, 23–29. IEEE, (2011).

[19] Etemaadi, R., Lind, K., Heldal, R., and Chaudron, M. R. "Quality-Driven Optimization of System Architecture : Industrial Case Study on an Automotive Sub-System", In *Journal of Systems and Software* (2013).

[20] Davis, R. I. and Burns, A. "A survey of hard real-time scheduling for multiprocessor systems", In *ACM Computing Surveys (CSUR)* **43**(4), 35 (2011).

[21] González Harbour, M., Gutiérrez García, J., Palencia Gutiérrez, J., and Drake Moyano, J. "Mast : Modeling and analysis suite for real time applications", In *the 13th Euromicro Conference on Real-Time Systems*, 125–134. IEEE, (2001).

[22] Singhoff, F., Legrand, J., Nana, L., and Marcé, L. "Cheddar : a flexible real time scheduling framework", In *ACM SIGAda Ada Letters*, volume 24, 1–8. ACM, (2004).

[23] Chetto, H. and Delacroix, J. "Minimisation des temps de réponse des tâches sporadiques en présence des tâches périodiques", In *Conf. on Real-Time Systems*, (1993).

[24] Decotigny, D. "Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps réel", Novembre (2002).

[25] Xu, J. and Parnas, D. L. "On satisfying timing constraints in hard-real-time systems", In*IEEE Transactions on Software Engineering* **19**(1), 70–84 (1993).

[26] Chevochot, P. and Puaut, I. "An approach for fault-tolerance in hard real-time distributed systems", In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 292–293. IEEE, (1999).

[27] Chevochot, P. and Puaut, I. "Tolérance aux fautes dans les systèmes répartis temps-réel strict", In *TSI. Technique et science informatiques* **18**(8), 837–870 (1999).

[28] Grolleau, E. "Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de pétri en environnement monoprocesseur et multiprocesseur", PhD thesis, université de Poitiers, (1999).

[29] Schwabl, W., Reisinger, J., and Gruensteidl, G. "A survey of MARS", Technical report, Citeseer, (1989).

[30] Choquet-Geniet, A. and Grolleau, E. "Minimal schedulability interval for real-time systems of periodic tasks with offsets", In *Theoretical computer science* **310**(1), 117–134 (2004).

[31] Fauberteau, F., Midonnet, S., and George, L. "Laxity-based restricted-migration scheduling", In *the 16th Conference on Emerging Technologies & Factory Automation (ETFA), 2011 IEEE*, 1–8, (2011).

[32] Richard, P. "Analyse des temps de réponse et de la demande processeur en ordonnancement temps réel de tâches périodiques", In *Ecole d'été Temps Réel (ETR-05)* (2005).

[33] Richard, P. "Analyse du temps de réponse des systèmes temps réel", In *Actes de l'Ecole d'été Temps Réel* , 241–262 (2003).

[34] Rahni, A. "Contributions á la validation d'ordonnancement temps reel en presence de transactions sous priorité s fixes et EDF", PhD thesis, université de Poitiers, (2008).

[35] Liu, C. L. and Layland, J. W. "Scheduling algorithms for multiprogramming in a hard-real-time environment", In *Journal of the ACM (JACM)* **20**(1), 46–61 (1973).

[36] Devillers, R. and Goossens, J. "Liu and Layland's schedulability test revisited", In *Information Processing Letters* **73**(5), 157–161 (2000).

[37] Leung, J. Y.-T. and Whitehead, J. "On the complexity of fixed-priority scheduling of periodic, real-time tasks", In *Performance evaluation* **2**(4), 237–250 (1982).

[38] Audsley, N. C. "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times",. Technical report, University of York, (1991).

[39] Detouzos, M. "Control Robotics : The Procedural Control of Physical Processors", In *Proc. IFIP Congress*, 807–813, (1974).

[40] Labetoulle, J. " Un algorithme optimal pour la gestion des processus en temps réel ", In *Revue Francaise d ?Automatique, Informatique et Recherche Opérationnelle*, 11–17, (1974).

[41] Lehoczky, J., Sha, L., and Ding, Y. "The rate monotonic scheduling algorithm : Exact characterization and average case behavior", In *Proceedings of Real Time Systems Symposium*, 166–171. IEEE, (1989).

[42] OSEK/VDX Group "Operating System Specification 2.2.3", http://www.osek-vdx.org/, Feb (2005).

[43] Feiler, P. H. "Real-time application development with OSEK : A review of the OSEK standards", Technical report, DTIC Document, (2003).

[44] Trinquet, Y. and Elloy, J.-P. "Systèmes d'exploitation temps réel-Exemples d'exécutifs industriels", Ed. Techniques Ingénieur, (2010).

[45] Sha, L., Rajkumar, R., and Lehoczky, J. P. "Priority inheritance protocols : An approach to real-time synchronization", In *Computers, IEEE Transactions on* **39**(9), 1175–1185 (1990).

[46] OSEK Group, O. et al. "OSEK/VDX Time-triggered Operating System Specification, Version 1.0", (2001).

[47] Davis, R. I., Burns, A., Bril, R. J., and Lukkien, J. J. "Controller Area Network (CAN) schedulability analysis : Refuted, revisited and revised", In *Real-Time Systems* **35**(3), 239–272 (2007).

[48] FlexRay Consortium. "FlexRay communications system-protocol specification", *Version* **2**(1), 198–207 (2005).

[49] Booch, G., Reese, F., and Reese, L. "Conception orientée objets et applications", Addison-Wesley, (1992).

[50] Davis, A. L. and Keller, R. M. "Data flow program graphs", In *IEEE Computer*, Vol 15, No. 2, (1982).

[51] BOUAKAZ, A. "Real-time scheduling of dataflow graphs", PhD thesis, Université de Rennes 1, (2013).

[52] Lee, E. A. and Messerschmitt, D. G. "Synchronous data flow", In *Proceedings of the IEEE* **75**(9), 1235–1245 (1987).

[53] Hatley, D. J. and Pirbhai, I. A. "Strategies for real-time system specification", volume 112, Dorset House New York, (1988).

[54] Ngo, K. H. and Grolleau, E. "La méthode DARTS et la programmation multitâche en LabVIEW", In *Proc. FurturVIEW*, 69–74 (2003).

[55] Hilliard, R. "IEEE-std-1471-2000 recommended practice for architectural description of software-intensive systems", *IEEE, http ://standards. ieee. org* (2000).

[56] Medvidovic, N. and Taylor, R. N. "A classification and comparison framework for software architecture description languages", In *IEEE Transactions on Software Engineering* **26**(1), 70–93 (2000).

[57] Magee, J., Dulay, N., Eisenbach, S., and Kramer, J. "Specifying distributed software architectures", In *Software Engineering-ESEC'95*, 137–153. Springer (1995).

[58] Bézivin, J. and Gerbé, O. "Towards a precise definition of the OMG/MDA framework", In *Proceedings 16th Annual International Conference on Automated Software Engineering, ASE 2001*, 273–280. IEEE, (2001).

[59] Camarinha-Matos, L. M. and Afsarmanesh, H. "Towards a reference model for collaborative networked organizations", In *Information technology for balanced manufacturing systems*, 193–202. Springer (2006).

[60] Serral, E., Valderas, P., and Pelechano, V. "Towards the model driven development of context-aware pervasive systems", In *Pervasive and Mobile Computing* **6**(2), 254–280 (2010).

[61] Blanc, X. and Salvatori, O. "MDA en action : Ingénierie logicielle guidée par les modèles", Editions Eyrolles, (2011).

[62] Boulet, P., Dekeyser, J.-L., Dumoulin, C., Marquet, P., Kajfasz, P., Ragot, D., et al. "Sophocles : Cyber-Enterprise for System-on-Chip Distributed Simulation–Model Unification", In *IFIP International Workshop On IP Based System-on-Chip Design*, (2003).

[63] Kadima, H. "MDA : Conception orientée objet guidée par les modèles", Dunod Paris, (2005).

[64] Belangour, A., Bézivin, J., and Fredj, M. "Towards a new software development process for MDA", In *Proceedings of the European Workshop on Milestones, Models and Mappings for Model-Driven Architecture* , 1 (2006).

[65] Estublier, J., Vega, G., and Ionita, A. D. "Composing domain-specific languages for wide-scope software engineering applications", In *Model Driven Engineering Languages and Systems*, 69–83. Springer (2005).

[66] Caceres, P., Marcos, E., and Vela, B. "A MDA-based approach for web information system development", In *Workshop in software model engineering*, (2003).

[67] DeAntoni, J. and Babau, J.-P. "A MDA-based approach for real time embedded systems simulation", In *Proceedings of Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT 2005*, 257–264. IEEE, (2005).

[68] Schmidt, D. C. "Guest editor's introduction : Model-driven engineering", In *Computer* **39**(2), 0025–31 (2006).

[69] Gérard, S., Terrier, F., and Tanguy, Y. "Using the model paradigm for real-time systems development : Accord/uml", In *Advances in Object-Oriented Information Systems*, 260–269. Springer (2002).

[70] Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., and Lange, K. "AUTOSAR–A Worldwide Standard is on the Road", In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, (2009).

[71] Cruz, F., Barreto, R., and Cordeiro, L. "Towards a model-driven engineering approach for developing embedded hard real-time software", In *Proceedings of the 2008 ACM symposium on Applied computing*, 308–314. ACM, (2008).

[72] Simonot-Lion, F. "Une contribution à la modélisation et à la validation d'architectures temps réel", *HDR Thesis, INPL, Nancy, France* (1999).

[73] Ramponi, G., Sicuranza, G., and Ukovich, W. "A computational method for the design of 2-D nonlinear Volterra filters", In *IEEE Transactions on Circuits and Systems*, **35**(9), 1095–1102 (1988).

[74] Nash, S. G. and Sofer, A. "Linear and nonlinear programming", volume 692, McGraw-Hill New York, (1996).

[75] Dantzig, G. B. "Linear programming and extensions", Princeton university press, (1998).

[76] Boyd, S. P. and Vandenberghe, L. "Convex optimization", Cambridge university press, (2004).

[77] Thoft-Christensen, P. and Murotsu, Y. "The Branch-and-Bound Method", Springer, (1986).

[78] Polisetty, P. K. and Gatzke, E. P. "Piecewise linear relaxation techniques for solution of nonconvex nonlinear programming problems", In *Journal of Global Optimization* (2005).

[79] Polisetty, P. K. and Gatzke, E. P. "A decomposition-based MINLP solution method using piecewise linear relaxations", In *International Transactions in Operations Research* (2005).

[80] de Werra, D. and Hertz, A. "Tabu search techniques", In *Operations-Research-Spektrum* **11**(3), 131–141 (1989).

[81] Bäck, T. "Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms", volume 996, Oxford university press Oxford, (1996).

[82] Van Laarhoven, P. J. and Aarts, E. H. "Simulated annealing", Springer, (1987).

[83] Resende, M. G. and Ribeiro, C. C. "Greedy randomized adaptive search procedures", In *Handbook of metaheuristics*, 219–249. Springer (2003).

[84] Steiglitz, K. and Papadimitriou, C. H. "Combinatorial optimization : Algorithms and complexity", In *Prentice Hall, New Jersey., UV Vazirani (1984). On two geometric problems related to the travelling salesman problem. J. Algorithms* **5**, 231–246 (1982).

[85] Wolsey, L. A. "Integer programming", In *IIE Transactions* **32**(273-285), 2–58 (2000).

[86] Till, J., Engell, S., Panek, S., and Stursberg, O. "Empirical complexity analysis of a milp-approach for optimization of hybrid systems", In *IFAC Conference on Analysis and Design of Hybrid Systems*, 129–134, (2003).

[87] Hocks, M. "Innere-punkt-methoden und automatische ergebnisverifikation in der linearen optimierung", PhD thesis, Karlsruhe, Univ., Diss., (1995).

[88] Boyd, S. and Mattingley, J. "Branch and bound methods", In *Notes*, Stanford University, Stanford, CA, (2003).

[89] Ka, A. and Mok, L. "Fundamental design problems of distributed systems for the hard-real-time environment", volume 1, MIT Thesis, May, (1983).

[90] Joseph, M. and Pandya, P. "Finding response times in a real-time system", In *the Computer Journal* **29**(5), 390–395 (1986).

[91] Lehoczky, J. P. "Fixed priority scheduling of periodic task sets with arbitrary deadlines", In *Proceedings of the 11th IEEE Real-Time Systems Symposium* , 201–209. IEEE, (1990).

[92] Tindell, K. "Using offset information to analyse static priority pre-emptively scheduled task sets", Technical report, University of York, Department of Computer Science, (1992).

[93] Tindell, K. "Adding time-offsets to schedulability analysis", Technical report, University of York, Department of Computer Science, (1994).

[94] Tindell, K. W., Burns, A., and Wellings, A. J. "An extendible approach for analyzing fixed priority hard real-time tasks", In *Real-Time Systems* **6**(2), 133–151 (1994).

[95] Palencia, J. C. and González Harbour, M. "Schedulability analysis for tasks with static and dynamic offsets", In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 26–37. IEEE, (1998).

[96] Mäki-Turja, J. and Nolin, M. "Tighter response time analysis of tasks with offsets", In *Proc. 10th International conference on Real-Time Computing and Applications (RTCSA-04)*, (2004).

[97] Maki-Turja, J. and Nolin, M. "Fast and tight response-times for tasks with offsets ", In *Proceedings of the 17th Euromicro Conference on Real-Time Systems, 2005.(ECRTS 2005)*, 127–136. IEEE, (2005).

[98] Mäki-Turja, J. and Nolin, M. "Efficient implementation of tight response-times for tasks with offsets ", In *Real-Time Systems* **40**(1), 77–116 (2008).

[99] Mäki-Turja, J. and Nolin, M. "Faster response time analysis of tasks with offsets ", In *Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS)*, (2004).

[100] Saksena, M. and Karvelas, P. "Designing for schedulability : integrating schedulability analysis with object-oriented design ", In *12th Euromicro Conference on Real-Time Systems, 2000. Euromicro RTS*, 101–108. IEEE, (2000).

[101] Tindell, K. and Clark, J. "Holistic schedulability analysis for distributed hard real-time systems ", In *Microprocessing and microprogramming* **40**(2), 117–134 (1994).

[102] Tindell, K. W. "Fixed priority scheduling of hard real-time systems", PhD thesis, University of York, (1994).

[103] Tindell, K., Burns, A., and Wellings, A. J. "Calculating controller area network (CAN) message response times ", In *Control Engineering Practice* **3**(8), 1163–1169 (1995).

[104] Tindell, K. and Burns, A. "Guaranteeing message latencies on control area network (CAN) ", In *Proceedings of the 1st International CAN Conference* (1994).

[105] Tindell, K., Hansson, H., and Wellings, A. J. "Analysing real-time communications : controller area network (CAN) ", In *Proceedings of Real-Time Systems Symposium*, 259–263. IEEE, (1994).

[106] Di Natale, M., and Zeng, H. "Practical Issues with the Timing Analysis of the Controller Area Network ", In *2013 18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*,1–8, (2013).

[107] Palencia Gutiérrez, J., Gutiérrez García, J., and González Harbour, M. "On the schedulability analysis for distributed hard real-time systems ", In *Ninth Euromicro Workshop on Real-Time Systems*, 136–143. IEEE, (1997).

[108] Palencia, J. C. and Harbour, M. G. "Exploiting precedence relations in the schedulability analysis of distributed real-time systems ", In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, 328–339. IEEE, (1999).

[109] Pop, T., Eles, P., and Peng, Z. "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems ", In *Proceedings of the tenth international symposium on Hardware/software codesign*, 187–192. ACM, (2002).

[110] Traore, K., Grolleau, E., Rahni, A., and Richard, M. "Response-time analysis of tasks with offsets ", In *IEEE Conference on Emerging Technologies and Factory Automation, ETFA'06* , 1–8. IEEE, (2006).

[111] Zheng, W., Di Natale, M., Pinello, C., Giusto, P., and Vincentelli, A. S. "Synthesis of task and message activation models in real-time distributed automotive systems ", In *Proceedings of the conference on Design, automation and test in Europe*, 93–98. EDA Consortium, (2007).

[112] Zheng, W., Zhu, Q., Di Natale, M., and Vincentelli, A. S. "Definition of task allocation and priority assignment in hard real-time distributed systems ", In *28th IEEE International Real-Time Systems Symposium, RTSS 2007*, 161–170. IEEE, (2007).

[113] Davare, A., Zhu, Q., Di Natale, M., Pinello, C., Kanajan, S., and Sangiovanni-Vincentelli, A. "Period optimization for hard real-time distributed automotive systems", In *Proceedings of the 44th annual Design Automation Conference*, 278–283. ACM, (2007).

[114] Di Natale, M., Zheng, W., Pinello, C., Giusto, P., and Vincentelli, A. "Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems ", In *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE*, 293–302. IEEE, (2007).

[115] García, J. G., Gutiérrez, J. C. P., and Harbour, M. G. "Schedulability analysis of distributed hard real-time systems with multiple-event synchronization", In *12th Euromicro Conference on Real-Time Systems, Euromicro RTS 2000*, 15–24. IEEE, (2000).

[116] Perathoner, S., Wandeler, E., Thiele, L., Hamann, A., Schliecker, S., Henia, R., Racu, R., Ernst, R., and Harbour, M. G. "Influence of different system abstractions on the performance analysis of distributed real-time systems", In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 193–202. ACM, (2007).

[117] Richter, K., Jersak, M., and Ernst, R. "A formal approach to MpSoC performance verification", In *Computer* **36**(4), 60–67 (2003).

[118] Richter, K., Racu, R., and Ernst, R. "Scheduling analysis integration for heterogeneous multiprocessor SoC", In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, 236–245. IEEE, (2003).

[119] Thiele, L., Chakraborty, S., and Naedele, M. "Real-time calculus for scheduling hard real-time systems", In *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems, ISCAS 2000 Geneva*, volume 4, 101–104. IEEE, (2000).

[120] Thiele, L., Chakraborty, S., Gries, M., and Künzli, S. "Design space exploration of network processor architectures", In *Network Processor Design : Issues and Practices* **1** (2002).

[121] Havender, J. W. "Avoiding deadlock in multitasking systems", In *IBM systems journal* **7**(2), 74–84 (1968).

[122] Locke, D., Sha, L., Rajikumar, R., Lehoczky, J., and Burns, G. "Priority inversion and its control : An experimental investigation", In *ACM SIGAda Ada Letters*, volume 8, 39–42. ACM, (1988).

[123] Chen, M.-I. and Lin, K.-J. "Dynamic priority ceilings : A concurrency control protocol for real-time systems", In *Real-Time Systems* **2**(4), 325–346 (1990).

[124] Rajkumar, R., Sha, L., and Lehoczky, J. "An experimental investigation of synchronization protocols", In*IEEE Real-Time Systems Newsletter* **5**(2-3), 11–17 (1989).

[125] Parks, T. M. and Lee, E. A. "Non-preemptive real-time scheduling of dataflow systems", In *International Conference on Acoustics, Speech, and Signal Processing, ICASSP-95.*, volume 5, 3235–3238. IEEE, (1995).

[126] Goddard, S. "On the Management of Latency in the synthesis of real-time signal processing systems from processing graphs", PhD thesis, Citeseer, (1998).

[127] Moreira, O., Valente, F., and Bekooij, M. "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor", In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 57–66. ACM, (2007).

[128] Bamakhrama, M. and Stefanov, T. "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications", In *Proceedings of the ninth ACM international conference on Embedded software*, 195–204. ACM, (2011).

[129] Bouakaz, A., Talpin, J.-P., and Vitek, J. "Affine data-flow graphs for the synthesis of hard real-time applications", In *12th International Conference on Application of Concurrency to System Design (ACSD)*, 183–192. IEEE, (2012).

[130] Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., and Meedeniya, I. "Software architecture optimization methods : A systematic literature review", In *IEEE Transactions on Software Engineering* **39**(5), 658–683. IEEE, (2013).

[131] Tindell, K. W., Burns, A., and Wellings, A. J. "Allocating hard real-time tasks : an NP-hard problem made easy", In *Real-Time Systems* **4**(2), 145–165 (1992).

[132] Bastarrica, M. C., Caballero, R. E., Demurjian, S. A., and Shvartsman, A. A. "Two optimization techniques for component-based systems deployment", In *Integration* **2**(2), 3 (2001).

[133] Richard, M., Richard, P., and Cottet, F. "Allocating and scheduling tasks in multiple fieldbus real-time systems", In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation* **1**, 137–144 (2003).

[134] Sorel, Y. "Syndex : System-level cad software for optimizing distributed real-time embedded systems", In *Journal ERCIM News* **59**(68-69), 31 (2004).

[135] Bate, I. and Emberson, P. "Incorporating Scenarios And Heuristics To Improve Flexibility In Real-Time Embedded Systems", In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 221–230, (2006).

[136] Lupu, I., Courbin, P., George, L., and Goossens, J. "Multi-criteria evaluation of partitioning schemes for real-time systems", In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8. IEEE, (2010).

[137] Al Sheikh, A., Brun, O., Hladik, P.-E., et al. "Partition scheduling on an IMA platform with strict periodicity and communication delays", In *Proceedings of the 18th International Conference on Real-Time and Network Systems*, 179–188, (2010).

[138] Peng, W., Li, H., Yao, M., and Sun, Z. "Deployment optimization for autosar system configuration", In *2nd International Conference on Computer Engineering and Technology (ICCET)*, volume 4, V4–189. IEEE, (2010).

[139] Azketa, E., Uribe, J., Gutiérrez, J., Marcos, M., and Almeida, L. "Permutational Genetic Algorithm for the Optimized Assignment of Priorities to Tasks and Messages in Distributed Real-Time Systems", In *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 958–965, (2011).

[140] García, J. G. and Harbour, M. G. "Optimized priority assignment for tasks and messages in distributed hard real-time systems", In *Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems* , 124–132. IEEE, (1995).

[141] Azketa, E., Uribe, J., Marcos, M., Almeida, L., and Gutiérrez, J. J. "Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation", In *Proceedings of the 1st Workshop on Synthesis and Optimization Methods for Real-time Embedded Systems (SOMRES)*, (2011).

[142] Azketa, E., Uribe, J. P., Gutiérrez, J. J., Marcos, M., and Almeida, L. "Software function allocation and configuration of an autosar-compliant system", In *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM*, (2011).

[143] Daghsen, A., Chaaban, K., Saudrais, S., Shawky, M., et al. "Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems", In *SAE 2012 World Congress & Exhibition*, (2012).

[144] Feljan, J., Carlson, J., and Seceleanu, T. "Towards a model-based approach for allocating tasks to multicore processors", In *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 117–124. IEEE, (2012).

[145] Messner, W. C. and Tilbury, D. M. "Control Tutorials for MATLAB and Simulink : A Web-Based Approach, CD ROM", Reading, MA : Addison-Wesley, (1998).

[146] Shoukry, Y., Kumar, A., El-Kharashi, M. W., Bahig, G., and Hammad, S. "Graph-based approach for software allocation in automotive networked embedded systems : A partition-and-map algorithm", In *Specification & Design Languages (FDL), 2013 Forum on*, 1–6. IEEE, (2013).

[147] Voss, S. and Schatz, B. "Deployment and scheduling synthesis for mixed-critical shared-memory applications", In *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, 100–109. IEEE, (2013).

[148] Dutertre, B. and De Moura, L. "The yices smt solver", *Tool paper at http ://yices. csl. sri. com/tool-paper. pdf* **2**, 2 (2006).

[149] Li, R., Etemaadi, R., Emmerich, M. T., and Chaudron, M. R. "An evolutionary multiobjective optimization approach to component-based software architecture design", In *IEEE Congress on Evolutionary Computation (CEC)*, 432–439. IEEE, (2011).

[150] Douglass, B. P. "Doing hard time : developing real-time systems with UML, objects, frameworks, and patterns", volume 1. Addison-Wesley Professional, (1999).

[151] Selic, B. "Tutorial : real-time object-oriented modeling (ROOM)", In *Proceedings of Real-Time Technology and Applications Symposium*, 214–217. IEEE, (1996).

[152] Harbour, M. G., Klein, M. H., and Lehoczky, J. P. "Fixed priority scheduling periodic tasks with varying execution priority", In *Proceedings Real-Time Systems Symposium, Twelfth*, 116–128. IEEE, (1991).

[153] Li, Y. and Wen, C. "Task Construction with Temporal Consistency for Embedded Control Software Design", In *International Symposium on Computer Science and Computational Technology, 2008. ISCSCT'08*, volume 1, 76–79. IEEE, (2008).

[154] Long, R., Li, H., Peng, W., Zhang, Y., and Zhao, M. "An approach to optimize Intra-ECU communication based on mapping of AUTOSAR runnable entities", In *International Conference on Embedded Software and Systems, ICESS'09*, 138–143. IEEE, (2009).

[155] Ferrari, A., Di Natale, M., Gentile, G., Reggiani, G., and Gai, P. "Time and memory tradeoffs in the implementation of AUTOSAR components", In *Proceedings of the Conference on Design, Automation and Test in Europe*, 864–869. European Design and Automation Association, (2009).

[156] Zeng, H. and Di Natale, M. "Efficient implementation of autosar components with minimal memory usage", In *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems*, 130–137, (2012).

[157] Zeng, H., Di Natale, M., and Zhu, Q. "Efficient implementation of autosar components with minimal memory usage", In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 1–8. IEEE, (2012).

[158] Daghsen, A. "Methodology of analysis and optimization of real-time embedded systems : application to automotive field", PhD thesis, Université de Compiègne, (2013).

[159] Hamann, A., Jersak, M., Richter, K., and Ernst, R. "A framework for modular analysis and exploration of heterogeneous embedded systems", In *Real-Time Systems* **33**(1-3), 101–137 (2006).

[160] Burns, A. and Wellings, A. "Real-time systems and programming languages", Addison Wesley (2001).

[161] Klein, M. H., Ralya, T., Pollak, B., and Obenza, R. *A Practitioner's Handbook for Real-Time Analysis : Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, (1993).

[162] Sofronis, C., Tripakis, S., and Caspi, P. "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling", In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, 21–33. ACM, (2006).

[163] Wang, G., Di Natale, M., and Sangiovanni-Vincentelli, A. "Improving the size of communication buffers in synchronous models with time constraints", In *IEEE Transactions on Industrial Informatics*, **5**(3), 229–240 (2009).

[164] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[165] Davis, R. I. and Burns, A. "Response time upper bounds for fixed priority real-time systems", In *Real-Time Systems Symposium, 2008*, 407–418. IEEE, (2008).

[166] Mehiaoui, A., Wozniak, E., Tucci-Piergiovanni, S., Mraidha, C., Di Natale, M., Zeng, H., Babau, J.-P., Lemarchand, L., and Gerard, S. "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems", In *Proceedings of the 14th ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, 121–132. ACM, (2013).

[167] Wang, S., Merrick, J. R., and Shin, K. G. "Component allocation with multiple resource constraints for large embedded real-time software design", In *Proceedings of the 10 th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2004*, 219–226. IEEE, (2004).

[168] Mehiaoui, A., Tucci-Piergiovanni, S., Babau, J., and Lemarchand, L. "Optimizing the Deployment of Distributed Real-Time Embedded Applications", In *IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 400–403. IEEE, (2012).

[169] Autosicura, P. "Drive by wire", *White Paper, Pi Technology, Milton Hall, Ely Road, Milton, Cambridge, CB4 6WZ, UK* (2004).

[170] http://www.maenad.eu/.

[171] Peraldi-Frati, M.-A., Karlsson, D., Hamann, A., Kuntz, S., Nordlander, J., et al. "The TIMMO-2-USE project : Time modeling and analysis to use", In *ERTS2012 International Congres on Embedded Real Time Software and Systems*, (2012).

[172] Audsley, N., Burns, A., Richardson, M., and Wellings, A. "Hard Real-Time Scheduling : The Deadline Monotonic Approach", In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, 127–132, (1991).

[173] Mraidha.C, Tucci Piergiovanni.S, and Gerard.S. "Optimum : a MARTE-based methodology for schedulability analysis at early design stages", In *SIGSOFT Softw. Eng. Notes, vol 36, issn 0163-5948, ACM, New York, USA* (Janvier 2011).

[174] Anssi, S., Tucci-Piergiovanni, S., Kuntz, S., Gerard, S., and Terrier, F. "Enabling Scheduling Analysis for AUTOSAR Systems", In *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, **0**, 152–159 (2011).

[175] Ralston, A., Reilly, E. D., and Hemmendinger, D. "Encyclopedia of computer science", volume 536, Van Nostrand Reinhold New York, (1993).

[176] Blaauw, G. A. and Brooks Jr, F. P. *Computer architecture : concepts and evolution*. Addison-Wesley Longman Publishing Co., Inc., (1997).

[177] *Papyrus*. http ://eclipse.org/papyrus/.

[178] Pölzlbauer, F., Bate, I., and Brenner, E. "On Extensible Networks for Embedded Systems", In *20th IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, 69–77, (2013).

[179] Fischetti, M. and Lodi, A. "Local branching", In *Mathematical Programming* **98**(1-3), 23–47 (2003).

[180] Glover, F., Laguna, M., et al. *Tabu search*, volume 22. Springer, (1997).

[181] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A. "Metropolis : An integrated electronic system design environment", In *Computer, IEEE* **36**(4), 45–52 (2003).

[182] Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Yang, G., Zeng, H., and Zhu, Q. "A next-generation design framework for platform-based design", In *Conference on using hardware design and verification languages (DVCon)*, volume 152, (2007).

[183] Davare, A., Densmore, D., Guo, L., Passerone, R., Sangiovanni-Vincentelli, A. L., Simalatsar, A., and Zhu, Q. "metro II : A design environment for cyber-physical systems", In *ACM Transactions on Embedded Computing Systems (TECS)* **12**(1s), 49 (2013).

[184] Schliecker, S., Rox, J., Negrean, M., Richter, K., Jersak, M. and Ernst, R. "System level performance analysis for real-time automotive multicore and network architectures", In *IEEE TransactionsComputer-Aided Design of Integrated Circuits and Systems*, 979–992 (2009).

[185] Guan, N., Stigge, M., Yi W. and Yu, Ge. "Cache-aware scheduling and analysis for multicores", In *Proceedings of the seventh ACM international conference on Embedded software*, 245–254 (2009).

[186] Guan, N., Stigge, M., Yi W. and Yu, Ge. Monot, A., Navet, N., Bavoux, B., Simonot-Lion, F. and others "Multicore scheduling in automotive ECUs", In *Embedded Real Time Software and Systems-ERTSS 2010*, (2010).

[187] Bini, E., Buttazzo, G.C. and Buttazzo, G.M. Monot, A., Navet, N., Bavoux, B., Simonot-Lion, F. and others "Rate monotonic analysis : the hyperbolic bound", In *IEEE Transactions on Computers, vol.52, no.7*, 933–942(2003).