THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Réseaux, Télécommunications, Systèmes et Architecture

Présentée et soutenue par :

Mme EMILIE BERARD-DEROCHE

le mardi 12 décembre 2017

Titre :

Distribution d'une architecture modulaire intégrée dans un contexte hélicoptère

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. CHRISTIAN FRABOUL

M. JEAN-LUC SCHARBARG

Rapporteurs :

M. LAURENT GEORGE, ESIEE NOISY LE GRAND

M. YE-QIONG SONG, UNIVERSITÉ LORRAINE

Membre(s) du jury :

M. YE-QIONG SONG, UNIVERSITÉ LORRAINE, Président

M. CEDRIC MAUCLAIR, AIRBUS HELICOPTERS, Membre

M. CHRISTIAN FRABOUL, INP TOULOUSE, Membre

M. HENRI BAUER, ENSMA POITIERS, Membre

M. JEAN-LUC SCHARBARG, INP TOULOUSE, Membre

Mme KATIA JAFFRES-RUNSER, INP TOULOUSE, Membre

En plein cœur de toute difficulté se cache une possibilité.
Albert Einstein

Remerciements

J'ouvre ces remerciements en exprimant ma profonde gratitude à mon directeur de thèse et mes encadrants, Christian FRABOUL, Jean-Luc SCHARBARG et Fabrice LE SERGENT. Je les remercie de leur confiance en me proposant ce sujet de thèse, de m'avoir soutenue et épaulée tout au long de celle-ci en me donnant leurs bons conseils. Leur expertise, dans les différents domaines couverts par cette thèse, m'a permis d'offrir le travail présenté dans ce manuscrit.

Je suis très reconnaissante envers Ye-Qiong SONG d'avoir accepté la double fonction de président du jury et rapporteur ainsi que Laurent GEORGE pour celle de rapporteur de mon manuscrit. Je remercie également Henri BAUER d'avoir accepté de faire partie du jury. Je leur transmets toute ma gratitude pour leurs commentaires constructifs et les perspectives ouvertes lors des discussions qui ont suivi la soutenance de la thèse.

Je remercie Bruno PASQUIER de m'avoir accueilli dans son équipe au sein d'Airbus Group et d'avoir donné la ligne directrice de ce travail. Je remercie également chaleureusement Airbus Helicopters. Leur contribution financière a été déterminante dans l'aboutissement des travaux. Mes remerciements s'adressent tout particulièrement à Cédric MAUCLAIR qui a suivi d'un œil attentif ce travail : merci pour tous les échanges constructifs pour comprendre et avancer toujours plus loin sur ce sujet de recherche. Je remercie également mes collègues du D42 avec qui j'ai eu plaisir à partager des moments de discussion, des repas et des sorties. Je pense à Floriane, Soukayna, aux Alex, Mathilde, Andy, Gregor, Cécile, Olivier, Ivan, Christian, Agnès, Gilles, Jean, ... Sans oublier Benoît qui m'a conseillé et permis de trouver ce sujet de recherche dans un contexte industriel.

Je remercie l'ensemble du personnel de l'IRIT, avec une attention particulière pour SAM, Annabelle et Zazou qui ont toujours été disponibles pour m'aider à gérer les formalités administratives mais aussi pour leur énergie et leur bonne humeur. J'ai une pensée pour tous les membres de l'équipe IRT ainsi que les professeurs et les thésards qui ont partagé ces cinq années de thèse, en particulier à Katia, Juju, Riadh, Manu, Béa, Gentian, Carlos, Jérôme, André-Luc, Martial, Florence, Aziz, J.G., Nesrine, Guigui, Eric, Oana, Ma, Eli, Mehdi, Cédric, Dorin, Yoann, ...

Je remercie tous mes amis qui ont suivi de près ou de loin cette période et/ou qui se sont investis à divers niveaux. Je pense aux toulousains Chloé, Lisa, Gaëlle, Estelle, Flo, Pierre (merci pour la relecture), Romain, Nico, et aux bourguignons (lala lala lalalalalère....) Marine, Grand Steak, Choup, Thidi, Angie, Cécile et Stephen. Je les remercie pour leur amitié qui dure depuis tant d'années. Je remercie également Annie pour ses encouragements et son investissement qui m'a permis de mener sereinement cette thèse à son terme.

Je m'adresse enfin à ma famille et belle-famille en les remerciant pour tout leur amour et leurs encouragements. Merci à mes parents, Sylvie et Daniel, ainsi qu'à Julie, Pierre et Lysbeth, qui ont toujours cru en moi et ont manifesté la plus grande des fiertés. Merci à mes beaux-parents, Rita et Séraphin, ainsi qu'à tous mes beaux-frères et belles-sœurs pour l'accueil qu'ils m'ont réservé dans leur famille ainsi que pour toute l'huile de coude fournie depuis le début pour offrir un nid douillet à la famille construite au fur et à mesure de ces années.

Mes derniers remerciements vont pour mes amours, Eva et Yannick. Travailler, tout en ayant un bébé, n'est souvent pas chose facile, mais ma fille Eva a été extraordinaire pendant ces années de thèse. Je remercie très profondément l'homme qui est devenu mon mari pendant ces années de doctorat, Yannick, pour son soutien tout au long de cette thèse et de l'amour qu'il me porte depuis que nous nous sommes rencontrés. Il a su trouver les mots justes pour me remonter le moral quand je n'en pouvais plus, me redonner le sourire quand je ne l'avais plus. Pour tout cela et bien plus encore, merci !

Résumé

Les architectures modulaires intégrées (IMA) sont une évolution majeure de l'architecture des systèmes avioniques. Elles permettent à plusieurs systèmes de se partager des ressources matérielles sans interférer dans leur fonctionnement grâce à un partitionnement spatial (zones mémoires prédéfinies) et temporel (ordonnancement statique) dans les processeurs ainsi qu'une réservation des ressources sur les réseaux empruntés. Ces allocations statiques permettent de vérifier le déterminisme général des différents systèmes : chaque système doit respecter des exigences de bout-en-bout dans une architecture asynchrone. Une étude pire cas permet d'évaluer les situations amenant aux limites du système et de vérifier que les exigences de bout-en-bout sont satisfaites dans tous les cas.

Les architectures IMA utilisés dans les avions centralisent physiquement des modules de calcul puissants dans des baies avioniques. Dans le cadre d'une étude de cas hélicoptère, ces baies ne sont pas envisageables pour des raisons d'encombrement : des processeurs moins puissants, utilisés à plus de 80%, composent ces architectures. Pour ajouter de nouvelles fonctionnalités ainsi que de nouveaux équipements, le souhait est de distribuer la puissance de traitement sur un plus grand nombre de processeurs dans le cadre d'une architecture globale asynchrone.

Deux problématiques fortes ont été mises en avant tout au long de cette thèse. La première est la répartition des fonctions avioniques associée à une contrainte d'ordonnancement hors-ligne sur les différents processeurs. La deuxième est la satisfaction des exigences de communication de bout-en-bout, dépendantes de l'allocation et l'ordonnancement des fonctions ainsi que des latences de communication sur les réseaux.

La contribution majeure de cette thèse est la recherche d'un compromis entre la distribution des architectures IMA sur un plus grand nombre de processeurs et la satisfaction des exigences de communication de bout-en-bout. Nous répondons à cet enjeu de la manière suivante :

1. Nous formalisons dans un premier temps un modèle de partitions communicantes tenant en compte des contraintes d'allocation et d'ordonnancement des partitions d'une part et des contraintes de communication de bout-en-bout entre partitions

d'autre part.

2. Nous présentons dans un deuxième temps une recherche exhaustive des architectures valides. Nous proposons l'allocation successive des fonctions avioniques en considérant au même niveau la problématique d'ordonnancement et la satisfaction des exigences de bout-en-bout avec des latences de communication figées. Cette méthode itérative permet de construire des allocations de partitions partiellement valides. La construction des ordonnancements dans chacun des processeurs est cependant une démarche coûteuse dans le cadre d'une recherche exhaustive.
3. Nous avons conçu dans un troisième temps une heuristique gloutonne pour réduire l'espace de recherche associé aux ordonnancements. Elle permet de répondre aux enjeux de distribution d'une architecture IMA dans un contexte hélicoptère.
4. Nous nous intéressons dans un quatrième temps à l'impact des latences de communication de bout-en-bout sur des architectures distribuées données. Nous proposons pour celles-ci les choix de réseaux basés sur les latences de communication admissibles entre les différentes fonctions avioniques.

Les méthodes que nous proposons répondent au besoin industriel de l'étude de cas hélicoptère, ainsi qu'à celui de systèmes de plus grande taille.

Abstract

Integrated Modular Architectures (IMA) is a major evolution of avionics systems. A spatial (predefined memory zones) and temporal (off-line scheduling) partitioning as well as communication resources reservation permit several systems not to interfere in this architecture. The determinism of systems is proved thanks to these static allocations : each system must respect end-to-end requirements in an asynchronous architecture. A worst-case study permits to assess the bounds of systems in order to verify that end-to-end requirements are satisfied in all the cases.

IMA architectures physically centralize powerful computing resources in avionics bays in aircraft. These aren't feasible in helicopters due to size reasons : powerless processors, at least 80% used, set these architectures. In order to add new functionalities and equipment, the aim is to distribute processing power over a larger number of processors in the context of a globally asynchronous architecture.

Two strong issues have been advanced throughout this thesis. The first one is the distribution of avionics functions with an off-line scheduling constraint on the different processors. The second one is the satisfaction of end-to-end requirements, depending on allocation and scheduling of functions as well as communication latencies over the networks.

This thesis proposes a trade-off between the distribution of IMA architectures on a larger number of processors and the satisfaction of end-to-end communication requirements. We answer at this topic as follows :

1. First, we formalize a communicating partitions model based on the partitions allocation and scheduling constraints on the one hand and end-to-end communication constraints on the other hand.
2. Second, we present an exhaustive search of valid architectures. We introduce a successive allocation of avionics functions considering altogether the scheduling and the satisfaction of end-to-end constraints with fixed communication latencies. This iterative method allows the building of partially valid allocations schemes, but the scheduling search is expensive here.

3. Third, we create a greedy heuristic to reduce the scheduling search space. It permits to meet the challenges of the distribution of IMA architecture in a helicopter context.
4. Finally, we focus on the impact of end-to-end communication latencies on given distributed architectures. We define for them the networks based on eligible communication latencies between the different avionics functions.

Our methods answer the industrial case study needs as well as bigger size systems needs.

Table des matières

Table des figures

Liste des tableaux

Acronymes

AFDX *Avionics Full DupleX switched Ethernet*. 37, 38, 124, 149

AMC *Aircraft Management Computer*. 79, 81, 105, 131, 163–166, 168

APEX *APplication/EXecution*. 13–15, 123

BAG *Bandwidth Allocation Gap*. 37

BCTT *Best-Case Traversal Time*. 39

CAN *Controller Area Network*. 35, 124, 148

CEV Centre d'Essais en Vol. 6

COM/MON *COMmand-MONitor*. 36

COTS *Commercial Off-The-Shelf*. 8

CSMA-CA *Carrier Sense Multiple Access with Collision Avoidance*. 35, 36

CSMA-CR/BA *Carrier Sense Multiple Access with Collision Resolution and Bitwise Arbitration*. 35

DIMA *Distributed Integrated Modular Avionics*. 16, 40, 43, 46

DM *Deadline-Monotonic*. 23

DP priorité dynamique. 23

E/S Entrées/Sorties. 20, 21, 46, 79, 81, 112, 123–126, 128, 133, 135, 166

EDF *Earliest Deadline First*. 23, 24, 28

FIFO *First In - First Out*. 33, 42

FILO *First In - Last Out*. 42

FP priorité fixe. 23

IMA *Integrated Modular Avionics*. 5, 7, 8, 10, 12, 13, 16, 17, 19–21, 40, 42, 45, 47, 50, 52, 54, 86, 145, 147

IOM *Input/Output module*. 21, 124

LIFO *Last In - First Out*. 41

LILO *Last In - Last Out*. 41

LLF *Least Laxity First*. 24

LRM *Line Replaceable Module*. 21

MAF *MAjor Frame*. 14, 52–55, 61–64, 66, 68, 70, 71, 84, 86, 87, 90, 98, 100, 105, 108, 115, 149, 171, 172, 174, 175

MFD *Multi-Function Display*. 79, 81, 163

NP *Non-preemtive*. 23, 24

NoC *Network on Chip*. 149

OLNE Optimisation Linéaire en Nombres Entiers. 45

PPCM plus petit commun multiple. 43

RC *Rate-Constrained*. 38

RDC *Remote Data Concentrator*. 21, 124

RIU *Remote I/O Unit*. 21

RM *Rate-Monotonic*. 23

TDMA *Time Division Multiple Access*. 36

TT *Time-Triggered*. 38

VL *Virtual Link*. 15, 32, 37, 56

VMSA *Vehicle Monitoring System DAL A*. 20, 79, 81, 84, 87, 90, 105, 108, 116, 123, 131, 133, 134, 163–165, 168

WCET *Worst-Case Execution Time*. 12, 50, 55, 59, 81, 84, 86, 87, 106, 115

WCTT *Worst-Case Traversal Time*. 39, 45, 56, 57, 83, 113–115, 117, 125, 128, 130, 135

Introduction

Contexte

Dans cette thèse, nous nous intéressons aux systèmes embarqués temps-réel dans des aéronefs tels que les avions ou les hélicoptères. Ces systèmes ont évolué depuis un siècle : d'abord complètement mécaniques, ces systèmes, grâce à l'arrivée de l'électronique à la fin des années 1950, puis de l'informatique à la fin des années 1970, ont été petit à petit remplacés par ces derniers. Le terme avionique, mélange d'avion et d'électronique, est un des termes les définissant. Dans un premier temps ségrégué physiquement avec des équipements associés à chaque système avionique pour des raisons de sûreté de fonctionnement, ces systèmes ont partagé dans un premier temps les moyens de communication, puis dans un second temps les processeurs pour réduire le poids. Ceci s'illustre par des architectures distribuées, puis fédérées pour terminer par des architectures dites modulaires intégrées.

Une architecture modulaire intégrée permet à plusieurs systèmes de se partager des ressources matérielles sans interférer dans leur fonctionnement. Chaque système peut être modifié sans interférer sur les autres systèmes. Le partage des ressources de calcul dans l'avionique est spécifié dans la norme ARINC 653 [1] : chaque fonction assignée à une partition est exécutée indépendamment des autres grâce à une isolation temporelle et accède à une zone mémoire prédéfinie grâce à un isolement spatial pour éviter les interférences entre les applications et par conséquent pour limiter la propagation des fautes. Un motif d'exécution est un ordonnancement statique défini à l'avance dans chaque processeur appelé module selon la norme ARINC 651 [2]. Le partage des ressources de communication est effectué par des canaux virtuels entre des partitions communicantes. Pour chaque canal, une bande passante maximale est prédéfinie, évitant une surconsommation des moyens de communication. Le déterminisme du réseau doit être certifié : la latence de communication doit être vérifiée pour chaque canal.

Un système doit respecter des exigences de bout-en-bout : par exemple, une donnée

de pression doit être affichée en moins de 400 ms sur l'écran du pilote [3]. Cette donnée est traitée par différentes partitions situées dans différents modules. Ces modules sont asynchrones, i.e. il n'y a pas d'horloge commune : un motif d'exécution d'un processeur n'est pas forcément synchrone avec le motif d'un autre processeur. Par ailleurs, une donnée peut être reçue à n'importe quel moment dans un motif d'exécution : elle est traitée dès que la partition destinatrice démarre. Un système avionique est par conséquent sensible aux diverses latences induites par l'exécutif. Le système doit être prédictif : une étude pire cas permet d'évaluer les situations amenant aux limites du système et de vérifier que les exigences de bout-en-bout sont satisfaites dans tous les cas.

L'intégration des fonctions avioniques sur les ressources de calcul doit vérifier deux critères. Le premier consiste à trouver un ordonnancement statique des fonctions assignées à des partitions sur chaque processeur. Le second critère doit valider que les exigences de bout-en-bout sont satisfaites dans le pire des cas.

Les architectures modulaires intégrées (IMA) contiennent des équipements centralisateurs dans des gros porteurs : les modules sont alloués dans des cabinets eux-mêmes intégrés dans des baies avioniques. Ces baies permettent de fournir des ressources de calcul puissantes refroidies par des systèmes d'air conditionné. Dans le cadre de plus petits aéronefs tels que les hélicoptères, ces baies avioniques ne sont pas envisageables pour des raisons d'encombrement : des processeurs moins puissants composent ces architectures. Par ailleurs, de nouveaux besoins apparaissent dans ce contexte : l'ajout de nouvelles fonctionnalités dans des processeurs utilisés à plus de 80%, ainsi qu'un souhait d'y connecter de nouveaux équipements alors que toutes les interfaces sont prises. Ceci nous mène à une distribution des architectures IMA pour offrir une plus grande possibilité de calcul ainsi qu'un plus grand nombre d'interfaces pour connecter de nouveaux équipements.

Des architectures IMA distribuées existent : ce sont des architectures complètement synchrones, que ce soit au niveau des processeurs ou bien des réseaux utilisés. Dans le cadre de ce manuscrit, nous visons des architectures IMA distribuées asynchrones. Les réseaux interconnectant les éléments de chaque système ont un impact sur les latences de bout-en-bout : les délais de communication sur les réseaux traversés sont contraintes par les exigences de bout-en-bout des systèmes.

Motivations

Dans cette thèse, nous nous focalisons sur la distribution des fonctions avioniques assignées à des partitions sur le plus grand nombre de processeurs possibles. Une architecture distribuée valide existe si toutes les partitions sont ordonnançables sur les

processeurs tout en respectant les exigences de bout-en-bout des différents systèmes. Nous considérons une architecture complètement asynchrone.

Dans l'avionique, différentes méthodes ont été considérées par le passé. La première consiste à allouer les partitions puis à rechercher les ordonnancements satisfaisant les exigences de bout-en-bout. La deuxième détermine les partitions à regrouper ensemble afin de réduire les communications. Ces deux méthodes ne répondent pas à notre problème de recherche d'architectures distribuées pour des raisons de coût. Nous nous appuyons sur une troisième méthode : considérer l'allocation, l'ordonnancement et la satisfaction des exigences de bout-en-bout au même niveau. Cette dernière méthode est utilisée dans différents travaux considérant des architectures complètement synchrones ou des éléments des systèmes s'activant dès qu'une donnée est reçue. Il n'y a à notre connaissance aucun travail empruntant cette méthode dans le cadre d'architecture asynchrone à motifs prédéfinis.

Pour cette raison, l'objectif de la thèse est de proposer des solutions pour trouver des architectures distribuées respectant les exigences de bout-en-bout. Cette recherche a amené à la création d'un outil permettant d'énumérer, en fonction de latences de communication fixes, les allocations dont les ordonnancements valident les exigences des systèmes. Or, une architecture peut contenir différents réseaux de communication : nous recherchons à quel point chaque réseau est contraint, i.e. nous déterminons les latences de communication maximales que les réseaux choisis ne doivent pas excéder. Ce dernier point permet à un intégrateur système de choisir les réseaux de communication adaptés à une architecture distribuée donnée.

Plan du mémoire

Ce manuscrit est découpé en quatre chapitres qui abordent, dans un premier temps, le contexte scientifique et industriel de cette thèse, pour ensuite détailler les contributions proposées.

Le **Chapitre 1** présente l'évolution des architectures avioniques et leurs limites par rapport aux nouveaux besoins de distribution des systèmes sur un plus grand nombre de processeurs répartis dans tout l'aéronef (type hélicoptère). Cette distribution visant à ajouter de nouvelles fonctionnalités ainsi que de nouveaux équipements doit s'effectuer dans le contexte d'une architecture modulaire intégrée (IMA).

Le **Chapitre 2** rappelle les principaux problèmes de distribution des partitions avioniques dans un contexte IMA :

— allocation des partitions sur les processeurs,

— ordonnancement des partitions sur un processeur donné,

— prise en compte des contraintes de communication de bout-en-bout entre partitions.

Nous présentons les stratégies de recherche d'allocations valides.

Dans le **Chapitre 3**, nous nous focalisons sur le problème de distribution des partitions sur un plus grand nombre de processeurs tout en respectant les contraintes de communication de bout-en-bout. Nous formalisons un modèle de partitions communicantes et proposons des algorithmes de recherche d'allocations valides en tenant en compte des contraintes d'allocation et d'ordonnancement des partitions d'une part et des contraintes de communication de bout-en-bout entre partitions d'autre part. Un premier algorithme basé sur une recherche exhaustive des ordonnancements est proposé.

Nous avons été amenés dans une démarche incrémentale à améliorer celui-ci dans le **Chapitre 4** afin de réduire l'espace de recherche associé aux ordonnancements. Un deuxième algorithme a été implémenté : il s'agit d'une heuristique s'appuyant sur la méthode dite de « séparation et évaluation » sélectionnant l'ordonnancement le plus intéressant localement pour la suite de la recherche.

Le **Chapitre 5** complète la recherche d'une architecture distribuée en spécifiant les contraintes sur les moyens de communication. Plus une architecture est distribuée, plus les délais dans une chaîne de partitions dépendantes augmentent. En effet, ces délais dépendent des latences de communication entre des partitions distribuées sur différents processeurs. Par conséquent, le choix de la technologie de communication est primordial pour le respect des exigences temporelles et pour la validité d'une architecture distribuée. Nous proposons une méthode pour spécifier les contraintes sur les moyens de communication.

Nous concluons ce manuscrit en résumant les principales contributions de cette thèse et en proposant différentes perspectives liées à ces travaux.

Chapitre 1

Évolution des architectures avioniques

Au XV^e siècle, l'ingénieur et savant italien Léonard de Vinci dessina le premier hélicoptère basé sur le principe de la vis d'Archimède, mais son absence de connaissances en aérodynamique n'a pas permis de mener l'idée à terme. Ce n'est qu'en 1907 que le premier vol en hélicoptère a été effectué par le Français Paul Cornu [4].

Nous allons nous intéresser dans cette thèse à l'évolution des architectures de l'aviation moderne, avion et hélicoptère, et plus particulièrement à leurs architectures avioniques. Une rétrospective de ces derniers est traité dans la partie 1.1. Celle-ci nous amène à définir un système avionique et l'architecture la plus en vogue actuellement dans la partie 1.2 : l'architecture modulaire intégrée nommée *Integrated Modular Avionics* (IMA). Nous y abordons ses limites et son évolution vers des architectures IMA distribuées. Ce dernier point nous amène à l'objectif de cette thèse en 1.3.

1.1 Vers des architectures avioniques modulaires intégrées

La création et le développement des avions et des hélicoptères n'ont pas eu le même essor, mais chacun a suscité un fort intérêt et connu un fort développement lors des guerres mondiales. Depuis l'irruption de l'électronique à la fin des années 1950 et surtout de l'informatique à la fin des années 1970, l'architecture des systèmes avioniques (mélange des termes avion et électronique) est devenue fondamentale.

1.1.1 Les premiers pas dans l'aéronautique : complètement mécanique

Les premières versions des avions et des hélicoptères sont complètement mécaniques. D'abord utilisés à des fins militaires lors de la première guerre mondiale, les avions ont

commencé à transporter des passagers dès 1919. Différentes évolutions technologiques sont apparues : les avions à réaction en 1936, puis les avions à turbo-réacteurs en 1952. Le premier hélicoptère complètement fonctionnel date de 1936 : le Focke Wulf Fw61. Ce n'est qu'après la seconde guerre mondiale que le potentiel des hélicoptères a été déterminé : ils peuvent atteindre des zones difficiles d'accès. Leur développement débuta.

Ce n'est qu'à partir des années 1955-1960 que l'électronique a pris place dans les aéronefs et a commencé à remplacer la mécanique dans de nombreux équipements en commençant par des architectures avioniques distribuées. [5]

1.1.2 Les architectures distribuées

L'arrivée de l'électronique dès les années 1960 a permis de remplacer des éléments des architectures mécaniques par des équipements analogiques. Chaque unité analogique effectue une tâche précise en fonction des entrées reçues. L'architecture implémentée est une architecture analogique distribuée : chaque unité est reliée aux autres unités d'un même système par des câbles dédiés. Elle s'oppose au concept d'architecture centralisée où le système dépend d'une entité centrale. Ces architectures analogiques sont caractérisées par un ensemble de signaux discrets pour les sous-systèmes avioniques individuels. Ils sont conçus et implémentés les uns par rapport aux autres de manière isolée, et sont certifiés séparément. Les différents équipements sont connectés point-à-point, impliquant une grande quantité de câbles et des systèmes difficilement modifiables. Nous pouvons retrouver cette architecture dans les avions Boeing 707, VC10, BAC 1-11, DC-9.

Avec l'avènement de l'informatique dans les années 1970, les architectures analogiques ont été remplacées par les architectures numériques distribuées. Les calculateurs numériques pour le pilotage automatique ont été utilisés pour la première fois en 1975 sur un ≪ Mystère 20 ≫ du Centre d'Essais en Vol (CEV). Les hélicoptères Super Puma et Dauphin ont été les premiers à implémenter un calculateur numérique pour le coupleur de vol au début des années 1980. Nous pouvons retrouver cette architecture dans les avions tels que Boeing B737, B757, B767 et Airbus A300, A320 et A330. [5, 6]

1.1.3 Les architectures fédérées

A partir des années 1980, les architectures fédérées sont implantées. Une architecture dite fédérée est équivalente à une architecture décentralisée ou distribuée : les ressources ne se situent pas au même endroit ou sur la même machine. Dans l'avionique, chaque fonction dispose de ses propres ressources matérielles pour son exécution : elle est ainsi isolée des autres, ce qui assure une barrière naturelle à la propagation des fautes. En effet,

en cas de dysfonctionnement de l'une d'entre elles, la faute n'influence pas les autres fonctions à part celle à qui elle envoie l'information. Pour se parer à cette propagation, de la redondance active et des voteurs sont utilisés : ceci implique un coût, du poids et de la puissance supplémentaires. À la différence d'un système décentralisé où une fonction ne va communiquer qu'avec d'autres fonctions spécifiques, un système d'une architecture fédérée va interagir avec d'autres systèmes. Nous retrouvons cette architecture dans l'Airbus A340. [7–9]

Un des avantages majeurs d'une architecture fédérée est de permettre l'hétérogénéité des calculateurs utilisés pour s'adapter aux fonctions avioniques. Son inconvénient est le manque de flexibilité et de souplesse : le manque d'interopérabilité entre les différentes fonctions de l'aéronef réduit son bon fonctionnement et ne permet pas d'obtenir des nouvelles fonctionnalités et de meilleurs diagnostics grâce au croisement des données. Par ailleurs, la moindre modification d'une fonction implique une nouvelle certification. [8]

1.1.4 Les architectures modulaires intégrées

Une architecture modulaire est la conception d'un système formé d'éléments qui peuvent être assemblés puis séparément modifiés, retirés ou ajoutés sans interférer avec le fonctionnement des autres éléments. Une architecture intégrée est la conception d'un système formé d'éléments qui ne peuvent pas fonctionner les uns sans les autres. Les architectures modulaires intégrées, IMA, reposent sur ces deux définitions : un ensemble de fonctions avioniques de différents systèmes se partage les ressources matérielles sans interférer entre elles.

Cette architecture, et plus particulièrement l'architecture logicielle, est décrite dans la norme ARINC 653 [1]. L'architecture physique est décrite dans la norme ARINC 651 [2]. Cette dernière est composée d'un ensemble de modules, les processeurs, placés dans des cabinets ou « *racks* ». Un ensemble de fonctions avioniques appelées partitions est hébergé dans les différents modules : les partitions se partagent leurs ressources. L'indépendance dans l'exécution par un isolement temporel et la gestion de la mémoire avec un isolement spatial permet d'éviter les interférences entre les applications et par conséquent de limiter la propagation des fautes. Par ailleurs, cette indépendance permet d'héberger des applications avec différentes criticités et d'offrir les garanties nécessaires au bon fonctionnement de l'aéronef. Plusieurs avantages sont à souligner : la réduction du poids, du volume et la quantité d'énergie utilisée, la portabilité et la modularité, et la réduction des coûts de maintenance évolutive. [7–11]

Dès 1994, l'architecture avionique modulaire intégrée a été retenue pour réaliser les nouveaux hélicoptères de la marque Eurocopter, devenue Airbus Helicopters. Le NH90

a été le premier hélicoptère à bénéficier de cette nouvelle architecture : les calculateurs sous forme de modules sont insérés dans les cabinets et sont connectés ensemble grâce à des liaisons numériques [5].

Trois générations de l'IMA se sont succédées depuis les années 1990 dans les avions. La première intègre des éléments de système conçus par des fournisseurs spécifiques utilisant leurs propres standards et modules propriétaires. Cette première génération d'architecture se retrouve dans les Boeing B777. La seconde génération propose une architecture plus ouverte utilisant des composants génériques dits composants pris sur étagère ou *Commercial Off-The-Shelf* (COTS). La troisième et dernière génération se démarque par l'indépendance entre modules et applications. Les applications fonctionnent sous le standard APEX défini par l'ARINC 653 [1] : ces applications peuvent partager les ressources d'un même équipement. L'intégration logicielle sur les modules et la certification sont sous la responsabilité de l'intégrateur. Cette dernière génération se retrouve dans les Airbus A380 et A350. [6]

Certifier les différents systèmes indépendamment des uns des autres ainsi que l'interopérabilité des fonctions avioniques sont les avantages majeurs de cette architecture modulaire intégrée. Son inconvénient provient de son actuelle implémentation physique dans les différents aéronefs : cette architecture centralise les modules dans des baies avioniques afin de regrouper les ressources au même endroit. Cette centralisation n'est pas possible dans des avions de petite taille ou des hélicoptères puisqu'ils n'ont pas d'espaces dédiés. Cette problématique nous mène à des architectures modulaires intégrées distribuées.

1.2 Vers des architectures IMA distribuées

Dans cette partie, nous nous intéressons aux propriétés d'une architecture IMA et à ses limites quand nous cherchons à la distribuer.

1.2.1 Les propriétés de l'IMA

1.2.1.1 Les systèmes avioniques : des systèmes embarqués temps-réel stricts

Un système temps-réel est un système réactif, i.e. un système en interaction permanente avec son environnement physique, qui doit respecter des contraintes de cadence et de latence. La cadence est représentée par la durée s'écoulant entre deux acquisitions/traitements des événements en entrée d'une fonction. Celle-ci peut être périodique, sporadique ou apériodique. Les actions du système face aux événements d'entrées doivent

être effectuées en un temps inférieur à une borne maximale nommée latence. Le non-respect des contraintes temporelles peut causer un mauvais comportement du système qui peut entraîner une faute système critique [12, 13].

Un système est dit embarqué lorsque le matériel et le logiciel dédiés à une tâche précise fonctionnent de façon autonome avec des ressources limitées telles que des contraintes spatiales, i.e. faible encombrement, poids, partage de ressources, ou énergétiques, i.e. faible consommation, même dans des conditions de pression, température ou humidité inhabituelles pour des systèmes classiques. Ces systèmes embarqués sont omniprésents et se retrouvent dans nos produits de consommation courants tels que les téléphones portables, les télévisions, la domotique, les jouets, etc..., dans le médical, dans l'automobile, dans le spatial et dans l'aéronautique. Par exemple, les systèmes spatiaux doivent embarquer des processeurs permettant de faire le plus d'opérations possibles en utilisant le moins de puissance possible. En effet, les satellites embarquent des batteries se rechargeant grâce au soleil : afin d'offrir une durée de vie assez longue à celui-ci, les ressources utilisées doivent être les plus faibles possibles.

En fonction de la criticité des contraintes temporelles, nous distinguons deux types de systèmes temps-réel : souple et dur. Un système temps-réel souple ou mou accepte un certain pourcentage de non respect des contraintes temporelles. Ceci peut entraîner une dégradation de ses performances sans mettre en danger le comportement de celui-ci. À l'inverse, un système temps-réel strict, critique, ou dur doit respecter toutes les contraintes temps-réel pour éviter des comportements du système ayant des conséquences catastrophiques telles que des pertes humaines, écologique, etc. Ainsi, la propriété la plus importante d'un système temps-réel strict est d'être prévisible plutôt que rapide. En effet, un système prévisible permet de déterminer en avance tous ses comportements et de vérifier que toutes les contraintes temporelles sont respectées.

L'étude effectuée dans cette thèse traite des systèmes embarqués temps-réel stricts. Ces derniers doivent respecter des standards de certification décrits ci-dessous.

1.2.1.2 Certification des systèmes avioniques

Les systèmes avioniques sont soumis à de nombreuses exigences spécifiées par les autorités de certification. Les autorités les plus connues sont la FAA (*Federal Aviation Administration*) aux États-Unis et l'EASA (*European Aviation Safety Agency*) en Europe.

La recommandation ARP 4754 A [14] définit les processus de développement des avions civils et des systèmes critiques. Elle intègre notamment : la détermination et la validation des exigences, le processus de certification, la gestion de la configuration et la vérification de l'implémentation. L'ARP 4754 A permet de valider que les exigences sont

correctes et complètes avec des méthodes de haut niveau sans faire de différences entre le logiciel (recommandation DO-178C) et le matériel (recommandation DO-254) comme l'illustre la Figure 1.1.

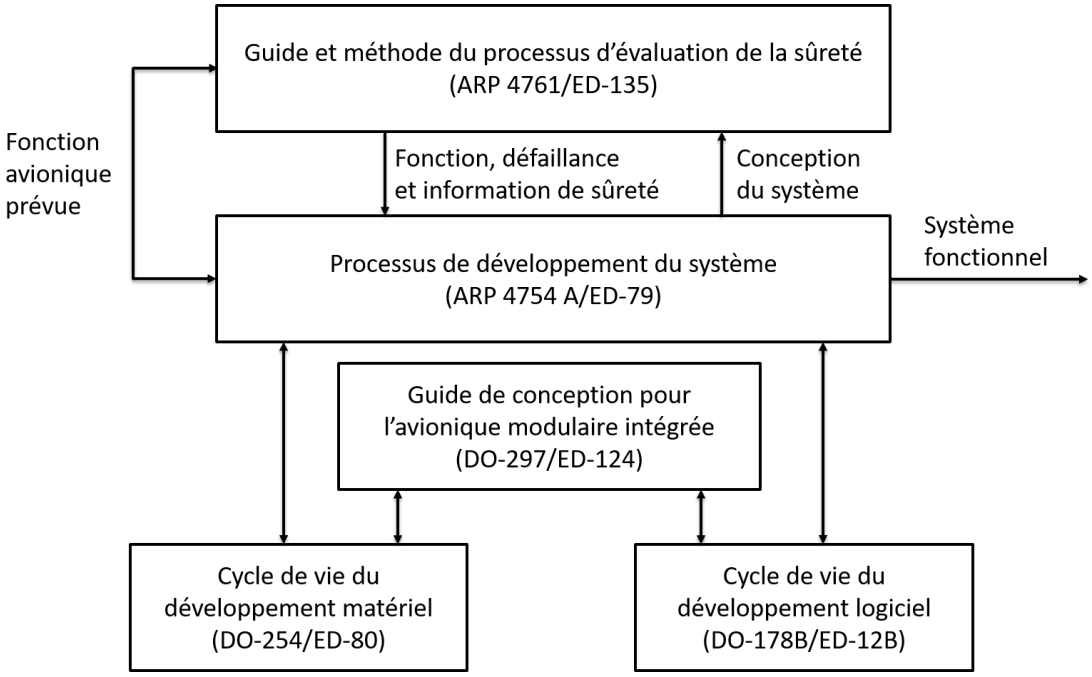


FIGURE 1.1 Guide de conception et techniques de certification en phase de développement (ARP 4754 A)

Nous nous intéressons dans cette thèse à l'allocation des différentes fonctions avioniques assignées dans des partitions dans une architecture distribuée. Cette architecture doit vérifier les exigences associées à l'IMA (recommandation DO-297) [15]. Pour être certifiée, toute l'architecture doit être prédictible, i.e. l'état d'un système doit être connu à tout instant, et respecter les contraintes temporelles de bout-en-bout des différents systèmes avioniques.

1.2.1.3 Architecture distribuée

Une architecture distribuée ou multiprocesseur va connecter un ensemble de processeurs ayant chacun sa propre mémoire. Les différents équipements peuvent être indépendants entre eux. L'échange d'information entre eux est fait par des échanges de messages explicites. Ce type d'architecture est peu adapté au partage de petites quantités d'information. La performance d'une telle architecture va dépendre des communications : le débit du lien ou le temps de propagation maximal.

À l'inverse, la mémoire est partagée par plusieurs processeurs dans une architecture parallèle. Les processeurs communiquent ensemble en écrivant et lisant les données dans une seule et même mémoire. Nous retrouvons ce type d'architecture dans les processeurs multicœurs. Un inconvénient majeur est l'estimation de la durée d'exécution d'une fonction : si plusieurs fonctions veulent accéder à la mémoire au même moment, il existe un conflit d'accès mémoire qui va mettre en attente de données les différentes fonctions [16].

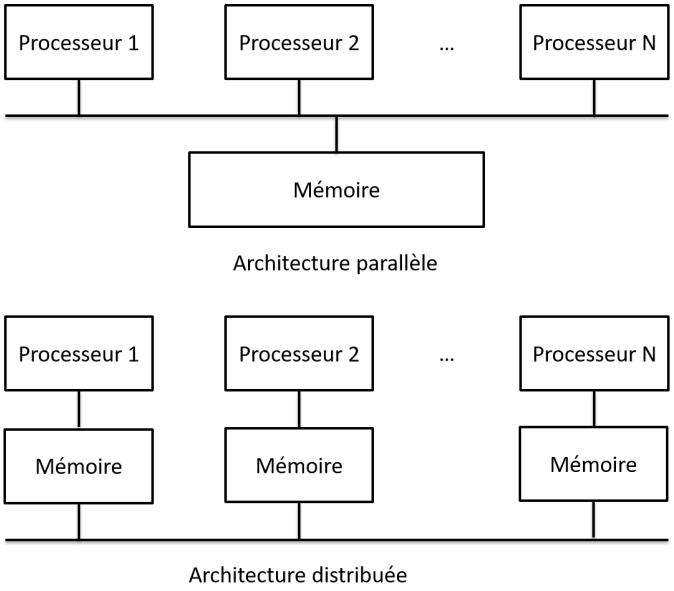


FIGURE 1.2 Architecture parallèle vs. architecture distribuée

Dans cette thèse, nous traitons des architectures distribuées dites multiprocesseurs. Ces processeurs sont asynchrones entre eux. Nous considérons que les processeurs utilisés sont identiques.

1.2.1.4 Les partitions avioniques

Un système temps-réel doit effectuer un certain nombre de tâches, i.e. récupérer les données d'un capteur, définir une commande, etc. appelées fonctions avioniques dans l'aéronautique. Cet ensemble de fonctions doit être alloué sur cette architecture multiprocesseur. Une fonction avionique est caractérisée par un ensemble de paramètres décrit ci-dessous :

— **Temps d'exécution** : Dans les systèmes temps-réel, le temps d'exécution d'une fonction peut varier entre une borne minimale et une borne maximale. Dans le

cadre d'une étude pire-cas, le pire temps d'exécution ou *Worst-Case Execution Time* (WCET) est utilisé : il consiste à prendre la borne maximale de la durée d'exécution d'une fonction.

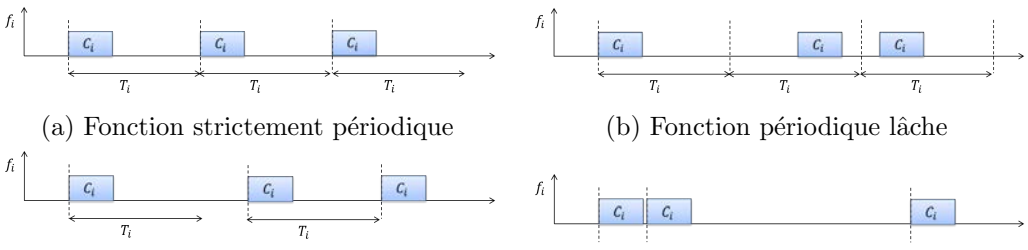
— **Périodicité** : Une fonction peut s'exécuter à des intervalles réguliers, nous parlons alors de fonction périodique, ou à des intervalles aléatoires ou semi-aléatoires, nous parlons alors de fonction respectivement apériodique ou sporadique [17].

— Une fonction f_i **périodique** s'exécute à des intervalles réguliers. Un premier modèle de fonction défini par le triplet (C_i, D_i, T_i) a été proposé par Liu et Layland [18] : le WCET d'une fonction est donné par C_i, D_i est son échéance relative dans sa période T_i. Par la suite, un paramètre supplémentaire a été ajouté : la date de réveil r_i de la fonction dans sa période, créant ainsi le quadruplet (r_i, C_i, D_i, T_i).

Une fonction périodique peut avoir une **périodicité stricte ou lâche**. Dans le premier cas illustré Figure 1.3a, la durée entre le début de deux exécutions consécutives de la fonction f_i est identique quel que soit l'instant choisi. Dans le second cas illustré Figure 1.3b, f_i peut s'exécuter quand elle le souhaite dans sa période.

— À la différence d'une fonction périodique où T_i représente une période fixe, T_i correspond à la durée minimale entre deux exécutions consécutives d'une fonction **sporadique** comme illustré Figure 1.3c. Ce type de fonction est alors modélisé par le triplet (C_i, D_i, T_i).

— Une fonction **apériodique** n'a pas de période propre comme le montre la Figure 1.3d. Elle peut s'activer à n'importe quel instant.



(a) Fonction strictement périodique (b) Fonction périodique lâche

(c) Fonction sporadique (d) Fonction apériodique

FIGURE 1.3 Spécificités des fonctions

Dans le cadre de l'IMA, nous travaillons sur des fonctions strictement périodiques. Ces fonctions sont assignées dans des partitions définies dans l'ARINC 653 [1].

Une application est décomposée en partitions. Ces partitions sont définies par un temps d'exécution et une période stricte respectivement définis par le pire temps d'exécution et

la période de la fonction associée illustrés Figure 1.4. Une fonction s'exécute alors dans le temps que lui offre sa partition. Les partitions d'un système avionique ont des périodes harmoniques entre elles. D'autres attributs existent tels que son identifiant, son nom et ses exigences de communication inter-partition traitées en 2.2.

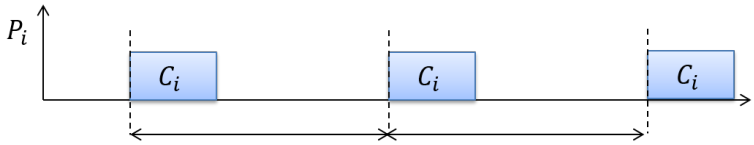


FIGURE 1.4 Paramètres d'une partition

Chaque partition s'exécute selon une sémantique lecture-calcul-écriture. À l'invocation de la partition, les ports d'entrée, représentés par le port destination Figure 1.5, sont lus avant d'effectuer tout calcul, puis à la fin de celui-ci, les données générées sont écrites dans les ports de sortie, i.e le port source sur la Figure 1.5. Cette sémantique assure l'indépendance de la fonction de n'importe quelle autre activité concurrente dès lors qu'elle a commencé son exécution.

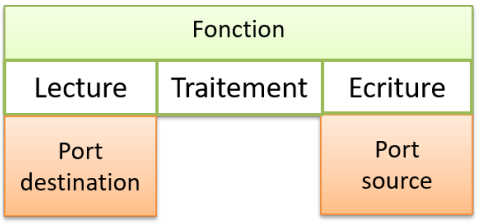


FIGURE 1.5 Sémantique d'exécution d'une partition

Une fonction est préemptée si elle peut s'interrompre pendant sa tâche pour laisser passer une autre fonction plus prioritaire. Dans le cadre de l'IMA, les partitions ne peuvent pas être préemptées [1].

Chaque processeur héberge de multiples partitions dans lesquelles les applications peuvent être exécutées en utilisant les ressources affectées. Ces partitions sont allouées et ordonnancées sur une architecture matérielle, grâce à la création d'un ordonnancement hors-ligne décrit ci-dessous.

1.2.1.5 Un partitionnement spatial et temporel robuste des partitions

Le standard ARINC 653 [1] propose de décorréler la partie applicative des communications grâce à l'interface *APplication/EXecution* (APEX). Cette interface permet d'homogénéiser la couche applicative alors que les équipements matériels cibles sont

hétérogènes. De la même façon, les réseaux interconnectant tous les équipements peuvent être différents, mais les interfaces sont identiques du point de vue de la partition grâce aux ports de communication associés à l'interface APEX.

L'ARINC 653 [1] définit un partitionnement spatial robuste : chaque partition se voit allouer statiquement des zones mémoires spécifiques, c'est à dire qu'aucune autre partition ne pourra accéder à ces zones mémoires.

De la même façon, L'ARINC 653 [1] définit un partitionnement temporel robuste grâce à l'interface APEX. Chaque partition se voit assigner une fenêtre d'exécution strictement périodique dans un motif cyclique appelé *MAjor Frame* (MAF). Les partitions sont isolées les unes des autres temporellement grâce à un ordonnancement hors-ligne illustré Figure 1.6. À aucun moment une partition peut interrompre la partition qui s'exécute.

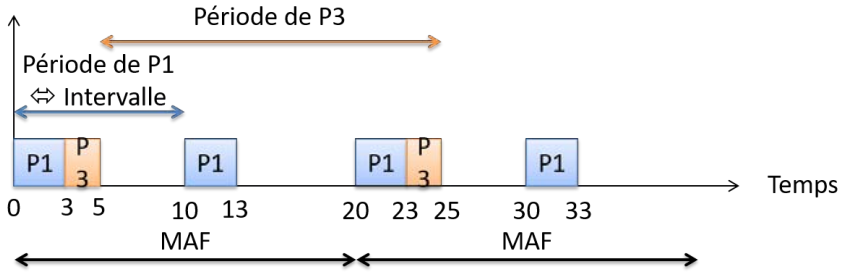


FIGURE 1.6 Définition d'une MAF

Chaque processeur héberge de multiples partitions dans lesquelles les applications peuvent être exécutées en utilisant les ressources affectées. Ces applications n'interfèrent pas entre elles étant donné qu'elles sont affectées à une partition. Le choix d'allouer une partition sur un processeur particulier est actuellement effectué par l'intégrateur système. Ce problème d'allocation et d'ordonnancement multiprocesseur revient *infine* à un problème d'ordonnancement monoprocesseur.

Dans le cadre des systèmes temps-réel stricts, les propriétés du système doivent être garanties. Ceci passe aussi par le déterminisme du réseau.

1.2.1.6 Déterminisme du réseau

Il existe différentes façons de relier les processeurs et d'autres équipements tels que des routeurs, des passerelles, etc. comme le montre la Figure 1.7 : topologies anneau, bus, étoile, hiérarchique, linéaire ou maillée.

L'échange de données entre deux partitions distinctes est effectué via la transmission de messages. Ces messages sont transmis par une partition source vers une (transmission

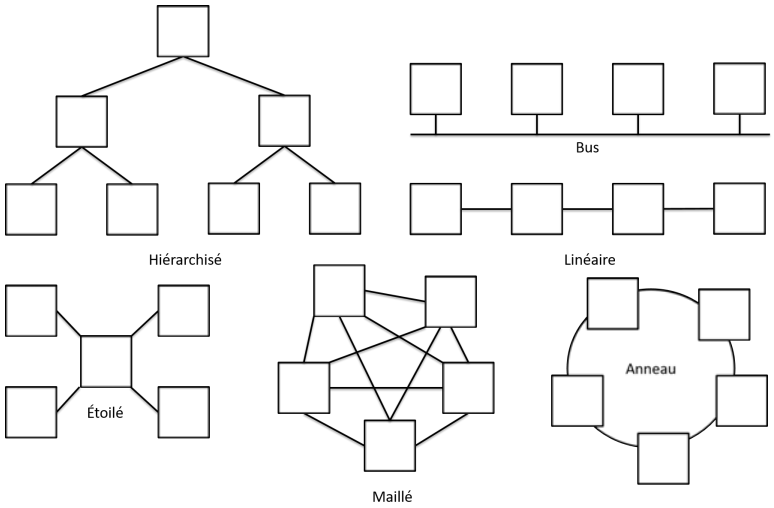


FIGURE 1.7 Les topologies d'interconnexion

unicast) ou plusieurs (transmission multicast ou broadcast) partitions destinatrices. Le mécanisme utilisé pour associer les dépendances entre partitions est le canal virtuel dit canal APEX dans le cadre de l'ARINC 653 [1] ou *Virtual Link* (VL). Un canal, ou VL, définit un lien logique unidirectionnel entre une partition source et la-les partition-s destinatrice-s. Les partitions ont accès à ces canaux via des points d'accès définis appelés « ports ». Ainsi, un canal décrit une route connectée entre un port source et un-des port-s destination-s comme nous l'illustrons Figure 1.8.

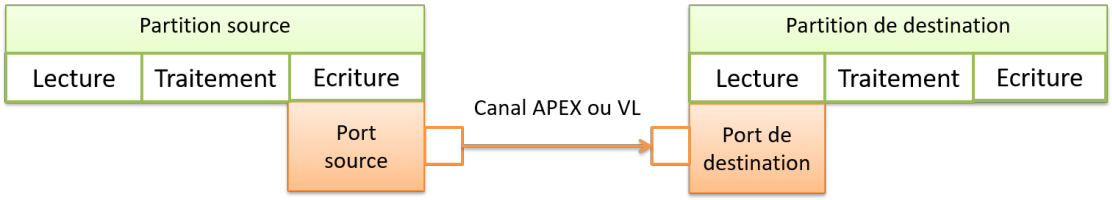


FIGURE 1.8 Communication entre deux partitions

Les communications ARINC 653 sont basées sur le principe de l'indépendance des mécanismes de transport au niveau partition. Ce mécanisme assure que les messages partant d'un port source atteignent dans le même ordre les ports destinations. Quelle que soit la partition, elle n'a pas de connaissance du mécanisme de transport : les mécanismes tels que la fragmentation, le routage, etc. sont transparents au niveau des partitions. Chaque VL est mappé sur un ou plusieurs réseaux : la latence de communication doit être déterminée par des méthodes d'évaluation pire cas développées en 2.2.4.

1.2.1.7 Exigences de communication de bout-en-bout

Les systèmes temps réel stricts imposent des contraintes de communication de bout-en-bout à respecter. Il s'agit de vérifier qu'une donnée générée dans une première partition est reçue et traitée par la dernière partition d'une chaîne de partitions communicantes en un temps inférieur à une borne donnée. À partir d'ici, l'expression « de bout-en-bout », précédée des noms « contrainte », « délai » ou « exigence », signifie la distance entre le début de l'exécution de la première partition et la fin de l'exécution de la dernière partition d'une chaîne de données. Ainsi, pour chaque chaîne de données traitée, le délai de bout-en-bout calculé doit être inférieur à une borne maximale. Ceci doit être vrai pour toutes les allocations dites valides, i.e. une allocation où un ordonnancement existe. L'allocation et l'ordonnancement des partitions sur les processeurs, ainsi que leur communication, doivent être pris en compte dans l'évaluation des contraintes de bout-en-bout et ce quelle que soit l'architecture.

1.2.2 Évolution des besoins et limites de l'IMA

Nos travaux s'appuient sur une demande industrielle spécifique : distribuer sur un plus grand nombre de processeurs des applications avioniques existantes dans un contexte hélicoptère. En effet, les systèmes avioniques sont composés d'un nombre croissant de fonctions de plus en plus complexes. Ceci amène à des architectures avioniques composées de processeurs de plus en plus puissants et complexes. De tels processeurs ne peuvent pas être utilisés dans le cadre de petits aéronefs tels que les hélicoptères, où des contraintes de coût ou physiques empêchent d'avoir des baies avioniques dédiées comme dans les Airbus A350, A380, les Boeing B777, B787, etc. pour placer l'ensemble des processeurs.

Une solution classique pour résoudre ce problème est d'avoir un plus grand nombre de processeurs, qui peuvent être moins puissants, et de les distribuer sur l'ensemble de l'hélicoptère. Ces éléments peuvent être rapprochés des éléments d'entrées et de sorties tels que les capteurs, les actionneurs, les écrans, etc. Le problème qui s'ensuit est la distribution des fonctions avioniques composant les différents systèmes sur ces processeurs, leur ordonnancement statique selon l'IMA [1], ainsi que le respect des contraintes de bout-en-bout des différents systèmes. Tout au long de cette thèse, nous illustrons notre démarche dans un exemple d'étude de cas détaillé en Annexe A.

Différents travaux traitent de la distribution de l'architecture modulaire intégrée dite *Distributed Integrated Modular Avionics* (DIMA) [8]. Cette architecture combine les avantages de l'IMA et de l'architecture fédérée. Les modules sont physiquement distribués et sont connectés via un système de communication tolérant aux fautes [19]. La

séparation physique des modules permet d'obtenir une barrière naturelle à la propagation des fautes. Cette séparation se situe au niveau des interfaces des modules. Les modules sont toujours partitionnés, ce qui permet par rapport à une architecture fédérée d'avoir une réduction de câblage et ainsi de réduire la complexité du système : les données d'une fonction sont ainsi partagées avec d'autres fonctions. Un avantage de cette architecture est de réduire le nombre d'équipements tout en offrant une puissance de calcul suffisante. Un autre avantage est de pouvoir se rapprocher des équipements périphériques tels que les capteurs, les actionneurs, etc. qui sont répartis sur l'ensemble de l'aéronef [20]. Ce type d'architecture peut être ainsi plus adapté à des petits porteurs tels que les hélicoptères.

Le problème qui s'ensuit est l'allocation et l'ordonnancement des partitions sur un plus grand nombre de processeurs satisfaisant les contraintes de bout-en-bout. L'augmentation du nombre de processeurs ainsi que du nombre de partitions amène à une explosion combinatoire, que ce soit au niveau des allocations (problème NP-Complet [21]) ou des ordonnancements hors-ligne de partitions non préemptées (problème NP-complet dans le sens fort dans le cas d'un unique processeur [22]). Afin de réduire le coût de recherche d'allocations valides, de nouvelles stratégies doivent être mises en place pour obtenir des architectures distribuées.

1.3 Objectif général

Cette thèse s'intéresse au problème de distribution des fonctions avioniques assignées à des partitions sur un plus grand nombre de processeurs, dans le but de définir les architectures matérielles répondant à tous les critères : ordonnancement hors-ligne et satisfaction des exigences de bout-en-bout dans un milieu asynchrone. Dans le prochain chapitre, nous présentons le contexte global concernant la distribution d'architecture IMA.

Chapitre 2

Distribution des architectures IMA

Ce chapitre propose d'identifier les différentes méthodes employées pour distribuer des systèmes temps-réel embarqués sur un plus grand nombre de processeurs. Une architecture IMA regroupe une partie matérielle et une partie logicielle qui doivent être associées : différentes méthodes d'assignation et d'ordonnancement sont présentées dans la partie 2.1. La partie logicielle, décomposée en fonctions, peut être distribuée dans un certain nombre d'équipements. Des dépendances entre ces fonctions existent : par exemple, une fonction a besoin des informations d'un capteur pour envoyer une commande. Un ensemble de communications doit être mis en place afin de transmettre toutes les informations nécessaires au bon fonctionnement de l'aéronef. Ces techniques sont détaillées dans la partie 2.2. Mais il ne suffit pas que ces informations arrivent au(x) bon(s) destinataire(s), il faut aussi qu'elles arrivent dans les temps. Par exemple, le délai entre l'acquisition de la donnée d'un capteur et son enregistrement dans les enregistreurs de vol doit être garanti : il ne doit pas dépasser les 500 millisecondes [23]. Ces contraintes de latence de bout-en-bout sont décrites dans la partie 2.3. La recherche d'une architecture IMA distribuée valide doit prendre en considération simultanément l'allocation, l'ordonnancement des différentes fonctions, ainsi que la satisfaction des contraintes de bout-en-bout. Différentes stratégies prenant ces trois paramètres sont détaillées dans la partie 2.4. Ce chapitre se termine par le positionnement de cette thèse dans la partie 2.5.

2.1 Distribution des partitions

Une architecture IMA est une architecture hétérogène par la nature des processeurs. Certains processeurs ne vont s'occuper que des capteurs, d'autres ne vont gérer que les calculs des systèmes avions, etc. Nous considérons que les processeurs de même

type sont homogènes tout au long de cette thèse. Nous présentons en 2.1.1 l'allocation des partitions en fonction de leur rôle sur ces derniers. D'autre part, il ne suffit pas d'allouer ces partitions sur un ensemble de processeurs, il faut aussi les ordonnancer. Nous détaillons en 2.1.2 différents algorithmes d'allocation et d'ordonnancement sur des architectures multiprocesseurs.

2.1.1 Allocation des fonctions avioniques

Une architecture avionique comprend un certain nombre d'équipements qui n'ont pas les mêmes rôles. À titre d'exemple, le système *Vehicle Monitoring System DAL A* (VMSA) comprend des capteurs, des actionneurs, des calculateurs, des écrans et des enregistreurs. De plus amples détails sont donnés en Annexe A. Chaque partition est associée à l'un de ces équipements.

2.1.1.1 Association entre architectures matérielle et fonctionnelle

L'architecture IMA peut être décomposée en 5 classes d'équipements de nature homogène décrites ci-dessous. Les partitions avioniques, selon leur rôle, sont associées à une de ces classes représentées Figure 2.1 :

1. Les **Entrées/Sorties (E/S)** : il s'agit des capteurs et actionneurs du système. Un capteur va récupérer l'état d'une grandeur physique et le transformer en une valeur utilisable, e.g. donner la température extérieure. Un actionneur va transformer l'énergie qui lui est fournie en un phénomène physique, e.g. le déplacement d'une gouverne, suite à une commande reçue. Il existe une fonction spécifique pour chaque capteur et chaque actionneur, le premier envoyant périodiquement les données acquises, le second réceptionnant les commandes à effectuer.
2. Les **écrans** affichent au(x) pilote(s) l'état courant du système. Différentes partitions peuvent être associées à un écran telles que la réception des données, l'affichage de celles-ci, etc.
3. On dénombre deux **enregistreurs** obligatoires dans les aéronefs et ils peuvent être au nombre de quatre pour déterminer les causes d'un accident [23]. Deux enregistreurs supplémentaires détaillés en Annexe A enregistrent la santé générale du véhicule et l'état des équipements. Ces informations permettent d'évaluer les besoins de maintenance de l'hélicoptère. Les données à enregistrer doivent respecter des intervalles de temps spécifiés dans [24]. Un ensemble de partitions exécutées cycliquement peut être associé à chaque enregistreur.

4. Les **passerelles** sont des systèmes matériels et logiciels permettant de faire une liaison entre deux réseaux qui peuvent être différents. En avionique, trois types de passerelles existent :
   (a) Un *Input/Output module* (IOM) est un équipement qui va transformer une donnée analogique ou discrète d'un capteur en une donnée numérique afin de l'envoyer au calculateur. Il fait le traitement inverse lorsqu'une donnée numérique doit être envoyée à un actionneur [2].
   (b) Un *Remote I/O Unit* (RIU) est équivalent à un IOM. La seule différence réside dans le fait qu'il est éloigné physiquement des calculateurs [15].
   (c) Un *Remote Data Concentrator* (RDC) est un équipement spécifié dans l'ARINC 655 [25] qui interconnecte un ensemble d'E/S avec une plateforme IMA. Un RDC convertit les protocoles et les formats de données tels que les données ARINC 429 ou 629, le réseau commuté AFDX, les signaux analogiques, numériques et discrets.

   Nous nous intéressons plus particulièrement ici aux RDCs. Ce sont des éléments qui doivent être prédictibles, interopérables et adaptables selon le standard ARINC 655 [25].
5. Les **calculateurs** sont au cœur de l'architecture IMA. Chaque processeur est placé dans un module interchangeable ou *Line Replaceable Module* (LRM). L'ensemble des LRMs est alloué physiquement dans les cabinets. Ils restent interchangeables à tout moment. Pour des raisons de sûreté de fonctionnement, ces modules seront différents pour éviter qu'une faute matérielle soit répliquée. Les applications, décomposées en partitions, sont allouées dans ces modules.

2.1.1.2 Redondance

En cas de défaillance matérielle ou logicielle, l'aéronef doit être tolérant aux fautes même s'il fonctionne en mode dégradé. Un mécanisme utilisé est le mécanisme de redondance : une même fonction est effectuée par des moyens différents. Cette dissimilarité, obtenue soit par des équipements différents, soit par différentes conceptions d'un système, permet d'éviter qu'une même erreur se reproduise [26].

2.1.2 Algorithmes d'allocation et d'ordonnancement multiprocesseur

L'ordonnancement des partitions décrites en 1.2.1.4 dépend des spécificités du système. Un ordonnancement peut se faire en-ligne (*in-line*) ou hors-ligne (*off-line*). Dans le premier cas, les fonctions sont ordonnancées à la volée. Dans le second cas, une séquence fixe est

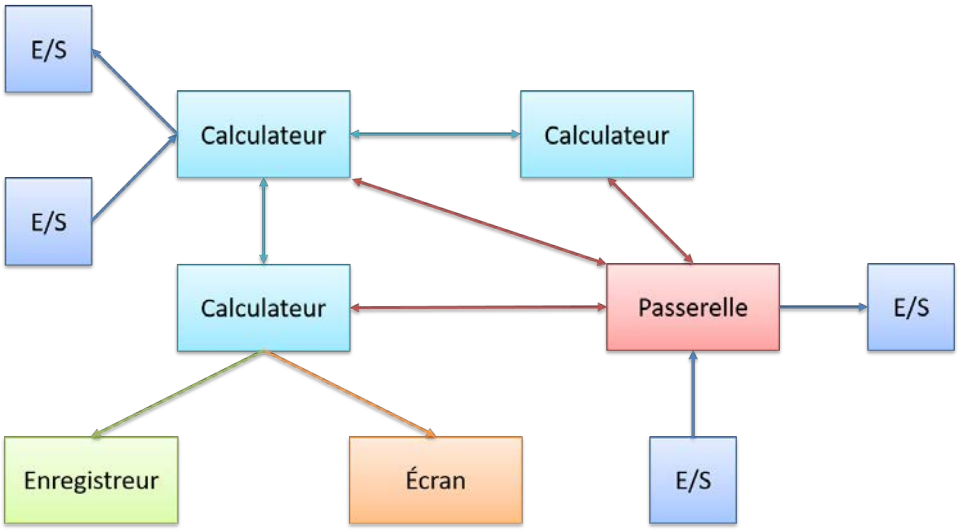


FIGURE 2.1 Ensemble des équipements présents dans une architecture avionique

déterminée. Nous présentons ci-après les principales stratégies d'ordonnancement en-ligne puis hors-ligne multiprocesseur.

2.1.2.1 Terminologie sur les ordonnanceurs

Dans cette partie, nous donnons un ensemble de définitions sur les ordonnanceurs et les ordonnancements.

Un ensemble de fonctions est **ordonnançable** selon un algorithme d'ordonnancement si la séquence générée n'engendre pas de fautes temporelles (séquence valide).

Un ensemble de fonctions est **faisable** s'il existe un algorithme d'ordonnancement avec lequel il est ordonnançable.

Une politique d'ordonnancement est **optimale** si elle peut ordonnancer n'importe quel ensemble de fonctions faisable.

Un ordonnanceur est **non-oisif** (*work-conserving* ou *non-idling*) si la-les ressource-s est-sont toujours utilisée-s dès qu'il existe une fonction prête à être exécutée. À l'inverse, un ordonnanceur est **oisif** (*non-work-conserving* ou *idling*) si une ressource est mise en attente alors qu'il existe au moins une fonction prête à être exécutée.

Un ordonnancement est **monoprocesseur** si toutes les fonctions ne peuvent s'exécuter que sur un seul et même processeur : une allocation temporelle est recherchée. Un ordonnancement est **multiprocesseur** si plusieurs processeurs sont disponibles : dans ce cas, une allocation temporelle et une allocation spatiale, i.e. le choix du processeur à utiliser, sont recherchées. Nous résumons ci-dessous les algorithmes en-ligne et hors-ligne

dans le cas monoprocesseur et dans le cas multiprocesseur.

2.1.2.2 Ordonnancement en-ligne

Un ordonnancement en-ligne repose sur la notion de priorité : selon l'algorithme choisi, différentes priorités sont affectées aux fonctions. Lors du déroulement de l'algorithme, l'ordonnanceur prend chaque décision d'ordonnancement selon les fonctions présentes dans la file d'ordonnancement.

Ordonnancement en-ligne monoprocesseur

Dans les ordonnancements en-ligne monoprocesseurs, deux types de politique d'ordonnancement existent : ceux à priorité fixe (FP) et ceux à priorité dynamique (DP). Dans le premier cas (FP), une priorité est assignée au début de l'algorithme à chaque fonction ; dans le second cas (DP), la priorité de chaque fonction varie durant l'exécution.

Quel que soit la politique d'ordonnancement, des fonctions préemptives ou non peuvent être ordonnancées. Dans le second cas, les algorithmes d'ordonnancements sont spécifiées avec l'entête NP pour *Non-preemtive* (NP). L'ordonnancement de fonctions non-préemptives ont reçu peu d'attention avant l'avènement des systèmes embarqués.

Liu et Layland [18] ont proposé le premier algorithme d'ordonnancement dit ***Rate-Monotonic* (RM)**. Ils considèrent un ensemble de n fonctions périodiques à échéance sur requête ($D = T$). C'est un ordonnancement à priorité fixe puisqu'il attribue la priorité la plus forte à la partition ayant la plus petite période. Sur la Figure 2.2, trois fonctions f_1, f_2 et f_3 de période respective 4, 6 et 8 et de pire temps d'exécution 1, 2 et 3 doivent être ordonnancées sur le même processeur. Quoiqu'il se passe, f_1 est prioritaire par rapport à f_2, qui est elle-même plus prioritaire que f_3. Ainsi, dès que f_1 est activée, elle doit s'activer tout de suite, obligeant les fonctions f_2 et f_3 à s'interrompre pendant son exécution. Liu et Layland [18] ont prouvé que RM est optimal dans le cas de systèmes préemptifs avec échéance sur requête. Park [27] a montré que NP-RM est aussi optimal si les fonctions sont non-préemptées.

Proposé par Leung et Whitehead [28], ***Deadline-Monotonic* (DM)** est aussi un ordonnancement à priorité fixe. Il attribue la plus haute priorité à la fonction ayant le délai le plus petit. DM est un algorithme optimal dans le cas de fonctions périodiques à départ simultané et à échéance contrainte ($D_i \leq T_i$) [28]. George et al. [29] ont prouvé qu'il est aussi optimal lorsque les fonctions sont sporadiques. Il n'y a pas à notre connaissance de preuve d'optimalité pour l'algorithme NP-DM.

***Earliest Deadline First* (EDF)** ou ordonnancement de la première échéance la plus proche en français est un algorithme à priorité dynamique : il attribue la priorité la plus grande à la fonction qui a son échéance la plus proche. Comme le montre la

Figure 2.2, lors de l'exécution de f_3 dans sa deuxième période, f_3 est préemptée par f_1 puisque l'échéance de cette dernière est la plus proche, tandis que f_2 se met en attente d'être exécutée puisque f_3, après cette préemption, est la tâche ayant l'échéance la plus proche. L'algorithme NP-EDF est optimal lorsque l'ordonnanceur est oisif et les fonctions sont périodiques [30]. Par contre, Nasri et al. [31] ont montré l'inverse lorsque NP-EDF considère un ordonnanceur non-oisif.

Proposé par Mok [32], l'**algorithme *Least Laxity First* (LLF)** est un algorithme à priorité dynamique. Il attribue la plus haute priorité à la fonction qui à la marge la plus petite de son échéance. Si nous prenons le dernier diagramme temporel de la Figure 2.2, nous constatons que f_1 ne démarre pas de suite dans sa deuxième période : comme la marge de f_3 est plus petite d'une unité de temps, f_3 continue son exécution dans sa première période. Ensuite, les marges entre f_1 et f_3 avant que f_1 démarre son exécution dans sa deuxième période sont les mêmes : f_1 débute son exécution, puis après une unité de temps, f_3 termine son exécution. Mok [32] a prouvé que LLF est optimal tandis que NP-LLF ne l'est pas.

Ces ordonnancements sont à la base de quantités de variantes telles que group-EDF (gEDF) [33], Precautious-RM [34, 35], etc.

Ordonnancement en-ligne multiprocesseur

Nous nous intéressons ici aux ordonnancements en-ligne multiprocesseurs. Les différents algorithmes doivent prendre en compte l'assignation des fonctions à de multiples processeurs. Nous pouvons nous limiter au cas de processeurs identiques puisque nous considérons une architecture matérielle avec des classes d'équipements homogènes (cf. 2.1.1).

Il existe trois classes d'algorithmes d'ordonnancement multiprocesseur : les ordonnancements par partitionnement, les ordonnancements globaux et les ordonnancements hybrides.

1. Un **ordonnancement par partitionnement** répartit dans un premier temps un ensemble de n fonctions dans les m processeurs disponibles : ce choix est un problème de Bin Packing, comment placer n objets dans m sacs, connu pour être NP-Complet. Cette allocation spatiale est en général effectuée hors-ligne. Dans un second temps, les fonctions sont ordonnancées processeur par processeur. Ceci revient à un problème d'ordonnancement à m processeurs. Comme les fonctions sont pré-allouées, elles ne peuvent pas migrer d'un processeur à l'autre.

   Le nombre de processeurs peut être fixé ou être minimisé. Deux types de méthodes sont utilisés pour résoudre ces problèmes : les méthodes dites exactes et les heuristiques classiques.

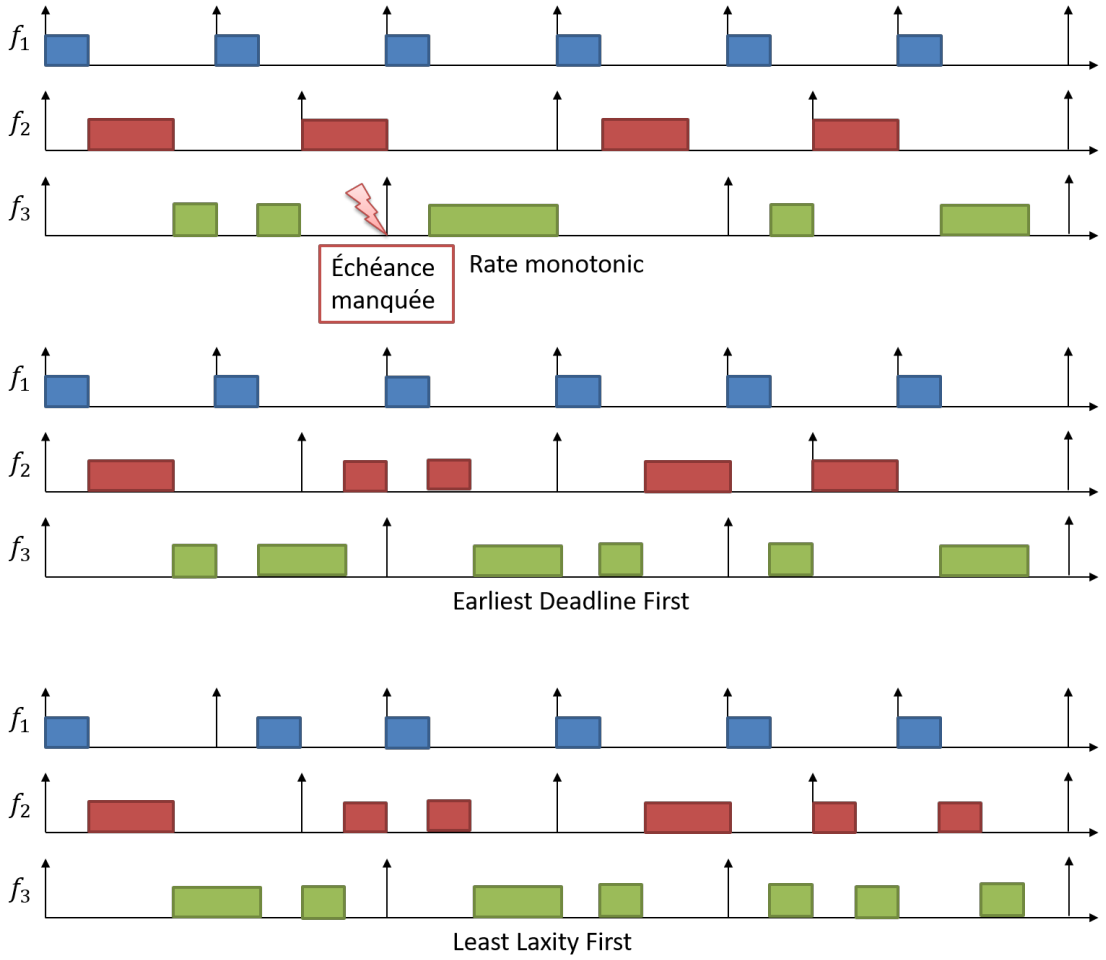


FIGURE 2.2 Ordonnancement en-ligne monoprocesseur de 3 fonctions préemptées périodiques à départ simultané et à échéance sur requête. $f_1 = (1, 4)$, $f_2 = (2, 6)$, $f_3 = (3, 8)$

Korf [21] propose un **algorithme exact** de partitionnement basé sur la méthode de séparation et évaluation (*Branch and Bound*). C'est une méthode générique de résolution de problème combinatoire. Elle consiste à diviser le problème en plusieurs sous-problèmes (séparation), puis à ne traiter que les sous-problèmes intéressants, ou à supprimer ceux qui ne donneront aucune solution (évaluation). Cette technique est répétée jusqu'à ce qu'au moins une solution ait été trouvée ou qu'aucune solution ne fonctionne. Grâce à cet algorithme, Korf est capable de déterminer le nombre de containers nécessaires pour ranger n éléments. L'algorithme présenté passe à l'échelle jusqu'à une centaine d'éléments.

Garey et al. [36] ont proposé les premières **heuristiques classiques** pour gérer l'allocation en mémoire de n fonctions sur m processeurs. Ces heuristiques sont au nombre de quatre représentées Figure 2.3 et sont décrites ci-dessous :

— L'algorithme **First-Fit** choisit le premier processeur où l'espace mémoire est assez grand pour contenir la fonction. Les fonctions sont allouées du numéro 1 au numéro 7 dans les différents processeurs Figure 2.3. f_1 est la première fonction allouée et ordonnancée. f_2 est aussi ordonnancée dans le premier processeur car il reste assez de place pour cette dernière. Étant donné que f_3 utilise 85% du processeur, elle est allouée sur le processeur suivant. Comme il n'y a pas assez de place pour f_4 dans le premier ou le deuxième processeur, f_4 est allouée sur le troisième processeur. C'est la même chose pour f_5. f_6 et f_7 sont allouées et ordonnancées dans le premier processeur qui est capable de les accueillir. L'avantage de cet algorithme est d'être très rapide car il s'arrête dès qu'il a trouvé le premier processeur répondant à tous les critères. Son inconvénient majeur est le manque d'optimisation de l'utilisation de la mémoire : suite à une allocation, l'espace mémoire restant peut être trop petit pour pouvoir allouer d'autres fonctions, cet espace devenant inutilisable par la suite.

— L'algorithme **Next-Fit** est une version modifiée de l'algorithme First-Fit. Au lieu de commencer sa recherche à partir du premier processeur, il la recommence pour allouer une nouvelle fonction à partir du processeur courant. L'algorithme se comporte de la même façon que l'algorithme First-Fit jusqu'à la cinquième fonction Figure 2.3. Lorsque cet algorithme cherche à allouer f_6, il part du processeur où il est actuellement, sans revenir en arrière, ici le processeur où est f_5. f_6 puis f_7 sont allouées dans ce quatrième processeur puisqu'il reste assez de place pour elles. Cet algorithme a l'avantage d'être très rapide mais ses principaux inconvénients sont la perte d'espaces mémoires et le coût en nombre de processeurs.

— L'algorithme **Best-Fit** choisit le processeur où l'espace mémoire requis pour la fonction est le plus petit. Sur la Figure 2.3 partie Best-Fit, f_4, au lieu d'être allouée dans le troisième processeur, est allouée sur le premier : la mémoire restante dans le premier processeur est la mémoire nécessaire pour l'exécution de f_4. Comme aucun processeur utilisé a d'espace suffisant pour f_5, celle-ci est allouée sur un nouveau processeur. L'espace mémoire restant dans le deuxième processeur étant insuffisant pour f_6, celle-ci est allouée dans le même processeur que f_5. f_7 peut être allouée dans le deuxième ou le troisième processeur : le

choix se porte sur le troisième processeur puisque l'espace restant est plus petit que sur le deuxième processeur. Son avantage est d'optimiser l'utilisation de la mémoire en cherchant à remplir tous les processeurs déjà utilisés. Deux inconvénients sont soulevés : être plus lent que les algorithmes précédents à cause d'une scrutation de tous les processeurs et le risque de laisser des espaces mémoires minuscules.

— L'algorithme **Worst-Fit** équilibre la charge de tous les processeurs en choisissant le processeur le plus vide : c'est son principal avantage. L'inconvénient apparaît dès qu'une fonction a besoin d'un espace mémoire plus grand que l'espace mémoire qu'offrent tous les processeurs déjà utilisés : dans ce cas, la fonction ne peut pas être allouée. Cette fois-ci, dans la Figure 2.3 partie Worst-Fit, les différentes fonctions vont être allouées dans chaque processeur ayant le plus d'espace mémoire disponible.

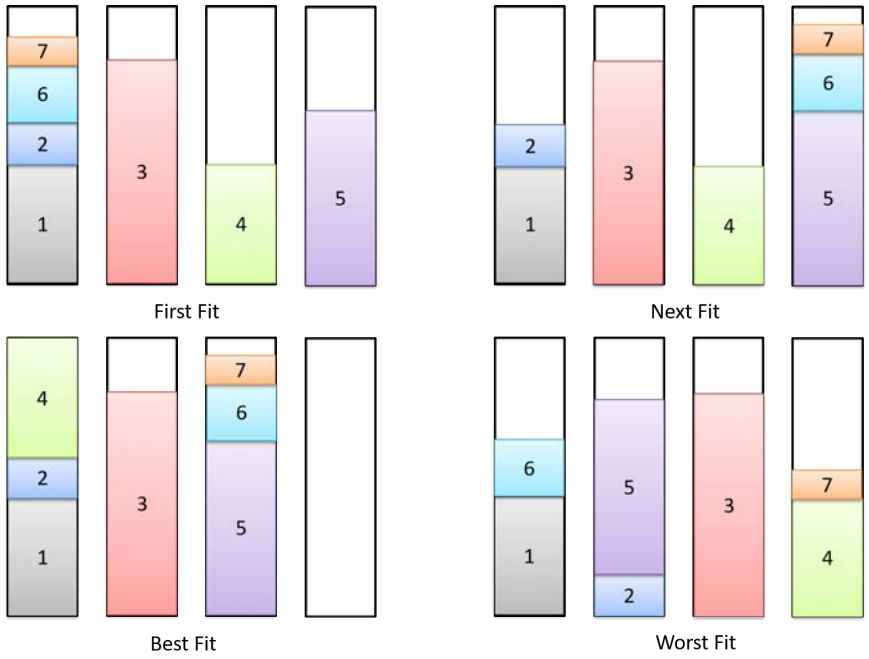


FIGURE 2.3 Comparaison des algorithmes First Fit, Next Fit, Best Fit, Worst Fit

Quelle que soit l'heuristique utilisée, sa complexité est de l'ordre de $\mathcal{O}(n \times m)$ [37].

Ces heuristiques peuvent être adaptées à une allocation temporelle en recherchant les blocs de temps libre sur un intervalle donné, e.g. sur la plus petite des périodes d'un ensemble de partitions. Elles ne semblent pas s'adapter au contexte de

l'avionique, car il paraît difficile de trouver un critère d'adaptation simple, les partitions devant s'activer de façon strictement périodique.

2. Un **ordonnancement global** alloue les fonctions au fur et à mesure sur les processeurs. Dans ce cas, une fonction a la possibilité de migrer d'un processeur à l'autre. Parmi les ordonnancements globaux, nous pouvons citer deux courants : PFair et DP-Fair. Ces deux courants proposent des algorithmes optimaux. PFair pour *Proportionate Fair* [38] offre pour chaque fonction une durée d'exécution proportionnelle à son utilisation du processeur sur un intervalle T. Une fonction peut se retrouver préemptée. Nous retrouvons des algorithmes basés sur ce principe tels que EPDF [39], PD [38] et PD^2 [40]. DP-Fair pour *Deadline Partitioning Fair* [41] propose de réduire le coût de la préemption de PFair en diminuant le nombre d'événements d'ordonnancement : le temps est découpé en intervalles tels que sur chaque intervalle les fonctions partagent la même date d'échéance [42]. Nous pouvons citer des algorithmes tels que BF (*Boundary Fair*) [41], DP-WRAP [42], LLREF (*Largest Local Execution Time First*) [43] s'appuyant sur ce principe.

3. Dans le but de réduire les coûts de préemption et de migration des ordonnanceurs équitables, des politiques d'**ordonnancements hybrides** optimales telles que QPS (*Quasi-Partitioning Scheduler*) ou RUN (*Reduction to UNiprocessor*) permettent d'ordonnancer des tâches, pour la première sporadiques, pour la deuxième périodiques, indépendantes et à échéances implicites [44]. Ces deux algorithmes encapsulent les tâches dans des serveurs où un ordonnancement monoprocesseur EDF à budget limité est effectué. L'algorithme optimal U-EDF (*Unfair-EDF*) propose de réserver du temps pour des tâches sporadiques à échéances implicites sur les processeurs en fonction des taux d'utilisation des travaux. Un nouveau processeur est utilisé si les processeurs déjà utilisés sont remplis [45].

   Les ordonnancements globaux ou hybrides ne s'adaptent pas à notre étude car les partitions n'ont pas le droit de migrer d'un processeur à l'autre pour des raisons de sûreté de fonctionnement et ne peuvent pas être interrompues.

2.1.2.3 Ordonnancement hors-ligne

À la différence d'un ordonnancement en-ligne qui produit un schéma d'exécution à la volée, un ordonnancement hors-ligne utilise une séquence d'ordonnancement prédéterminée à l'avance. Cette séquence est répétée cycliquement. Ce type d'ordonnancement permet d'être déterministe : à tout moment, nous connaissons l'état du système. Il manque néanmoins de flexibilité dès que des fonctions sporadiques ou apériodiques doivent être

prises en compte.

L'approche hors-ligne multiprocesseur s'appuie aussi sur les deux classes d'ordonnancement multiprocesseur en-ligne : un ordonnancement par partitionnement ou un ordonnancement global. Dans le premier cas, les algorithmes monoprocesseurs hors-ligne sont utilisés et décrits ci-dessous.

Ordonnancement hors-ligne monoprocesseur

Les ordonnancements hors-ligne monoprocesseurs consistent à définir une séquence d'exécution pour un ensemble de fonctions allouées sur celui-ci. Cette séquence peut être trouvée à partir de deux grandes catégories de techniques [46] : les techniques exactes et les techniques approchées.

1. Les **techniques exactes** utilisent des techniques d'énumération telles que la méthode de séparation et évaluation [47, 48], la programmation linéaire en nombres entiers [49], à base de modèles tels que les réseaux de Pétri [46, 50], les automates finis [51] ou les chaînes Markoviennes [52]. La complexité de ces techniques est exponentielle car elles sont exhaustives.

   Baker et Su [47] proposent d'énumérer toutes les permutations des fonctions possibles grâce à la génération d'un arbre de recherche. À chaque nouveau niveau N, la fonction ajoutée dans le processeur est exécutée à la N^e position. Ceci revient à énumérer $n!$ permutations possibles, n étant le nombre de fonctions. Pour réduire l'espace de recherche, leur idée est de développer la branche où les fonctions ne dépassent pas ou dépassent le moins possible leur échéance : le nœud le plus prometteur est à chaque fois développé.

   Xu et Parnas [48] présentent un algorithme de séparation et évaluation qui recherche un ordonnancement de pré-exécution sur un unique processeur en prenant en compte l'instant de réveil, l'échéance et des relations de précédence entre les fonctions. Ceci est équivalent à la recherche d'un ordonnancement hors-ligne faisable.

   Korst et al. [22] ont montré que le problème d'ordonnancement de fonctions strictement périodiques non-préemptées est NP-complet au sens fort dans le cas monoprocesseur, mais il reste solvable en un temps polynomial si les périodes sont harmoniques. La durée de la séquence correspond à l'hyper-période des fonctions présentes dans le processeur, i.e. le plus petit commun multiple de toutes les périodes [53] ; dans le cadre d'un ensemble de fonctions harmoniques, la durée de cette séquence est donnée par la fonction qui a la période la plus élevée. Cette séquence est découpée en intervalle appelé « *bin* ». La durée d'un *bin* est définie par la plus petite des périodes des fonctions. Eisenbrand et al. [54, 55] proposent un

algorithme d'arbre de recherche utilisant les *bins* pour ordonnancer des fonctions de périodes harmoniques comme le montre la Figure 2.4. À chaque niveau de cet arbre, une nouvelle harmonique, supérieure à celle du niveau précédent, est traitée pour ordonnancer dans les intervalles disponibles une fonction moins fréquente.

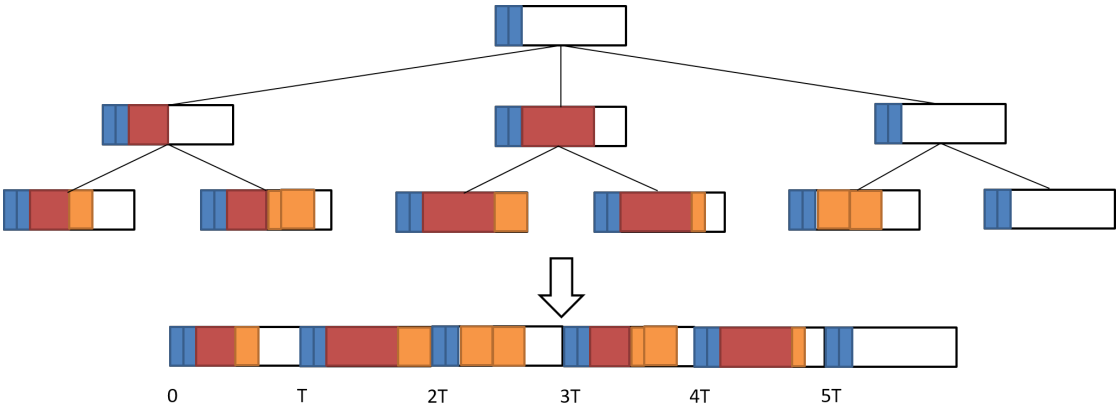


FIGURE 2.4 Ordonnancement par un arbre de recherche de bins

2. Les **techniques approchées** cherchent une solution valide tout en réduisant l'espace de recherche. À la différence des techniques exactes, elle sont non exhaustives. Nous pouvons citer, parmi ces techniques, les algorithmes de mise en sac ou de liste, les algorithmes stochastiques tels que les algorithmes génétiques ou le recuit simulé. Ramamritham et al. [56] proposent une heuristique où du *backtracking*, un retour arrière, est effectué si l'échéance d'une fonction est manquée : la position de la fonction est modifiée jusqu'à respecter toutes ses contraintes.

Ordonnancement hors-ligne multiprocesseur

L'ordonnancement hors-ligne multiprocesseur doit prendre en compte une contrainte supplémentaire : le choix du processeur où sera exécutée la fonction. Cucu et Sorel [57] ont démontré que la génération d'une séquence avec des fonctions strictement périodiques avec des périodes non harmoniques n'était pas possible : des collisions peuvent se produire entre les fonctions.

Nous retrouvons les deux classes d'ordonnancement multiprocesseur en-ligne dans le cadre hors-ligne.

1. **Ordonnancement par partitionnement** :

   Shepard et Gagne [58] étendent les travaux de Xu et Parnas [48] dans le cadre multiprocesseur : ils recherchent une répartition adéquate des fonctions sur les

différents processeurs et effectuent l'ordonnancement par la méthode de séparation et d'évaluation sur chaque processeur.

Kermia [59] construit quant à lui des *clusters*, des regroupements de fonctions basés sur des contraintes de précédences, pour allouer les fonctions, et il les ordonnance ensuite.

Eisenbrand et al. [55] utilisent l'algorithme First-Fit pour allouer les fonctions dans les différents processeurs disponibles. Elles sont allouées par période croissante pour pouvoir utiliser l'algorithme des *bins* dans chaque processeur.

Ces méthodes fixent le nombre de processeurs à la première allocation trouvée. Elles ne permettent pas de faire une énumération exhaustive de celles-ci.

2. **Ordonnancement global** :

   Bratley et al. [60] sont les seuls, à notre connaissance, à proposer un algorithme global dans le cadre d'un ordonnancement hors-ligne multiprocesseur. Cet algorithme consiste en un arbre de recherche bicolore représenté Figure 2.5 : un nœud au niveau inférieur est soit rond, la nouvelle tâche est alors placée sur le même processeur que la tâche du nœud supérieur, soit carré, un nouveau processeur est utilisé. L'ordonnancement dans chaque processeur est défini par l'ordre d'allocation des tâches dans ces derniers. Leur algorithme évite les symétries triviales, par exemple, nous ne pourrons pas avoir dans une branche la fonction f_1 allouée sur le premier processeur, la fonction f_2 sur le second et dans une autre branche, f_2 allouée sur le premier processeur et f_1 sur le second.

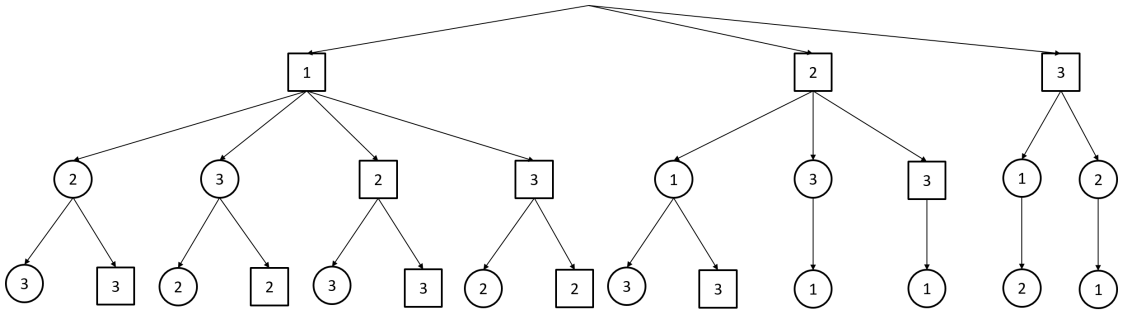


FIGURE 2.5 Arbre de recherche bicolore

Une solution classique pour distribuer un système temps-réel embarqué avec des contraintes dures est d'ordonnancer des fonctions non-préemptées avec des périodicités strictes [57]. Dans cette thèse, nous traitons de l'allocation et de l'ordonnancement hors-ligne multiprocesseur d'un ensemble de partitions, associées chacune à une fonction, strictement périodiques avec des périodes harmoniques. Ces partitions peuvent avoir des

dépendances entre elles, par exemple une partition attend les données d'un capteur pour calculer la commande à envoyer à un actionneur. Ces dépendances sont définies par les communications.

2.2 Communication entre partitions

Une partition interagit avec d'autres partitions afin de s'exécuter dans une architecture avionique. Nous parlons de « précédence » dès lors qu'il existe un ordre entre deux partitions sans qu'il y ait de transfert de données. Nous parlons de « dépendance » entre deux partitions si l'une a besoin du résultat d'exécution de l'autre pour être exécutée. Il y a dans ce cas une précédence entre ces partitions. Ces dépendances passent par deux niveaux dans les communications : les liens virtuels VL (cf. 1.2.1.6) et les ports de communication (2.2.1). Les dépendances n'existent pas seulement entre deux partitions, elles composent des chaînes de dépendances que nous appelons chaînes de communication (2.2.2). Nous décrivons ensuite les principaux réseaux utilisés dans l'avionique où sont alloués les liens virtuels (2.2.3). Pour conclure cette partie, nous donnons les principales techniques utilisées pour évaluer les pires délais de transmission dans les réseaux avioniques (2.2.4).

2.2.1 Ports de communication

Un port fournit les ressources nécessaires à une partition pour envoyer ou recevoir des messages dans un canal spécifique. Chaque port individuel peut être configuré pour fonctionner dans un mode spécifique : le mode « *queuing* » ou le mode « *sampling* ».

Le mode *sampling* représenté Figure 2.6 permet la transmission de messages successifs de même longueur mais avec des données mises à jour. Les messages sont écrits dans une file tampon ne pouvant contenir qu'un seul message de taille fixée. Un message reste dans le port source jusqu'à ce qu'il soit transmis par le canal ou qu'une nouvelle occurrence écrase la précédente, quel que soit ce qu'il y a eu avant. La partition source peut ainsi envoyer un message à n'importe quel moment. Chaque nouvelle instance du message écrase le message courant quand il atteint le port destination. Il n'est détruit qu'à partir du moment où une autre instance l'écrase. Les partitions destinatrices peuvent ainsi accéder au dernier message reçu.

Le mode *queuing* représenté Figure 2.7, permet la transmission de messages contenant des données différentes, et par conséquent ayant des tailles variables. Dans ce mode, aucun message ne doit être intentionnellement perdu : chaque nouvelle instance d'un message n'est pas autorisée à écraser les données précédentes pendant son transfert. Les messages écrits dans le port source par la partition sont stockés dans une file d'attente

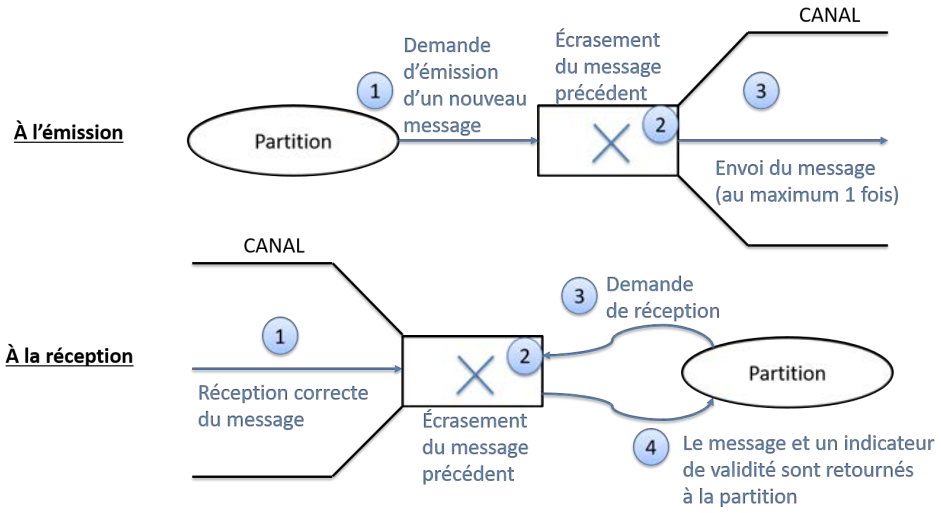


FIGURE 2.6 Transmission en mode sampling

jusqu'à ce qu'ils soient envoyés au-x port-s destination-s par le canal selon la politique de service premier arrivé, premier servi (*First In - First Out* (FIFO)). Les messages sont aussi stockés dans une file d'attente FIFO dans le port destination jusqu'à ce qu'ils soient lus par la partition destinatrice.

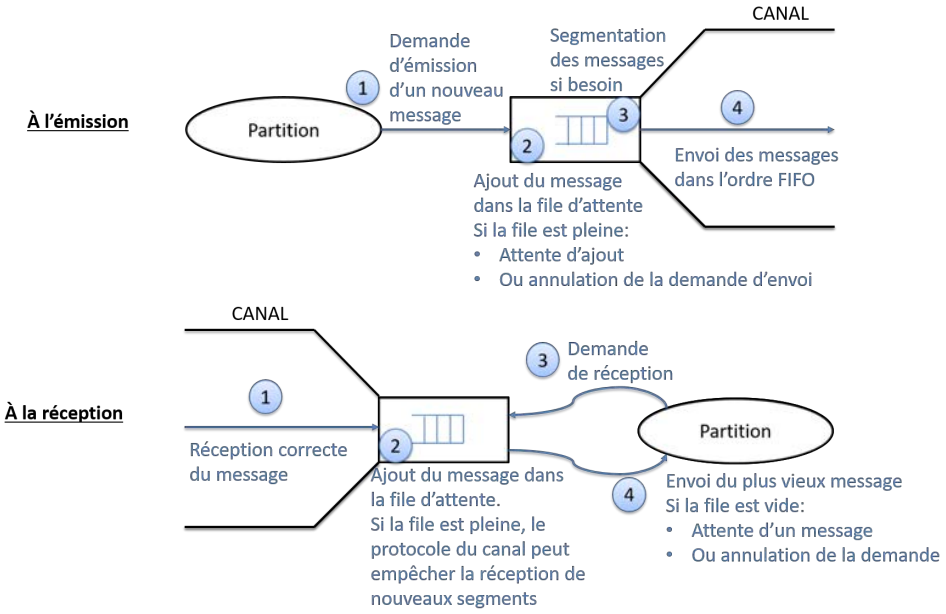


FIGURE 2.7 Transmission en mode queuing

Quel que soit le mode de transfert, un message reçu partiellement ne sera pas délivré au port destination.

2.2.2 Chaînes de communication

Deux types de chaîne peuvent être considérés dans les systèmes embarqués : les chaînes par événements et les chaînes de données [61].

Une **chaîne par événements** (*Event triggered*) est constituée d'un ensemble de fonctions qui vont s'activer dès la réception d'un événement sur un port d'entrée. Ce type de chaîne n'est pas adapté dans le cadre de cette étude car la périodicité stricte des partitions peut ne pas être respectée.

Une **chaîne de données** (*Time triggered*) est constituée de fonctions qui peuvent être activées à des fréquences indépendantes les unes des autres. À leur activation, les fonctions récupèrent les données reçues et s'exécutent pour envoyer leur commande à la fonction suivante. Ce type de chaîne est adapté à notre problème car une partition est exécutée strictement périodiquement, sans attendre un événement particulier en entrée.

Dans la suite de ce manuscrit, nous utilisons le terme chaîne pour désigner une chaîne de données.

2.2.3 Les principaux standards de communication dans l'avionique

Dans cette partie, nous proposons une brève synthèse des technologies de communication utilisées dans l'aéronautique. Pour des raisons de sûreté et de sécurité des appareils, les réseaux employés doivent être déterministes, i.e. le temps maximal pour transmettre un message doit être connu. Nous montrons ci-dessous comment chaque réseau répond à cette garantie.

2.2.3.1 ARINC 429

Le bus ARINC 429 [62] est bus point-multipoint utilisé dans les architectures avioniques depuis l'avènement des architectures fédérées. Il y a une source pour 20 destinataires au maximum comme le montre la Figure 2.8 : ce média est par conséquent déterministe puisque le temps maximum pour transmettre un message est connu. Un destinataire ne peut pas répondre sur le même média : il doit avoir son propre bus où il est l'émetteur pour pouvoir répondre, même pour un acquittement.

Un émetteur envoie un message de taille 32 bits dont 19 sont prévus pour les données. Un label de 8 bits identifie les données transmises sur le bus comme l'illustre la Figure

2.8. Un émetteur diffuse périodiquement les mises à jour des données produites. Les destinataires lisent celles-ci à leur rythme.

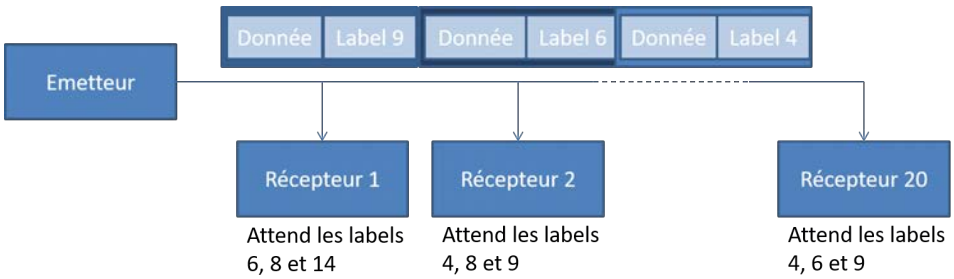


FIGURE 2.8 Transmission des données sur un bus ARINC 429

L'avantage de ce bus est son déterminisme. Cependant, la nature mono-émettrice de chaque équipement nécessite un nombre de câbles devenant rapidement grand pour connecter tous les équipements source de données vers leur-s destinataire-s : des problèmes d'intégration peuvent apparaître.

2.2.3.2 CAN

CAN, pour *Controller Area Network* (CAN), est un bus de communication sur priorité des événements standardisé dans l'ARINC 825 [63]. Développé dans un premier temps par l'industrie automobile pour les systèmes temps-réel embarqués nécessitant un haut niveau d'intégrité, il a été repris dans un second temps dans les industries aéronautique et spatiale. Il implémente un protocole de communication *Carrier Sense Multiple Access with Collision Resolution and Bitwise Arbitration* (CSMA-CR/BA). La résolution de collision (CR pour *Collision Resolution*) permet d'être non destructif : toutes les transmissions commencent au même instant. La couche physique implémente un « ET » logique pour effectuer un arbitrage bit par bit (BA pour *Bitwise Arbitration*) : dès qu'une station ne reconnaît pas le bit qu'elle a transmis, elle s'arrête d'émettre. Comme chaque flux a une priorité différente des autres, il restera à la fin un seul émetteur sur le bus. CAN est très souvent classé dans les protocoles de communication *Carrier Sense Multiple Access with Collision Avoidance* (CSMA-CA), mais il n'essaie pas d'éviter les collisions, il les résout en prenant en compte la priorité de chaque flux au début d'une transmission.

2.2.3.3 MIL-STD-1553

Le bus MIL-STD-1553 [64] a été en tout premier lieu utilisé dans l'avionique militaire. Il implémente un protocole de communication à accès contrôlé par un chef d'orchestre,

nommé « *polling* ». Ce chef d'orchestre scrute répétitivement les stations et les autorise à envoyer un message.

C'est un bus commande/réponse. Différents terminaux sont connectés au bus : le contrôleur du bus qui orchestre toutes les communications et les terminaux distants. Les terminaux distants peuvent être une station distante ou une interface vers un ou plusieurs sous-systèmes. Le contrôleur est le propriétaire du bus : les terminaux distants écoutent le bus en attendant les commandes envoyées par le chef d'orchestre pour envoyer ou recevoir des données. C'est un système centralisé hiérarchisé autour du contrôleur du bus.

L'architecture du MIL-STD-1553 est une architecture commande/surveillance (*COMmand-MONitor* (COM/MON)) : un nombre de surveillants écoute le bus afin de recueillir des données pour de futures analyses. Ces surveillants sont transparents pour tous les terminaux, même pour un terminal utilisé en mode normal alors qu'il est aussi surveillant.

2.2.3.4 ARINC 629

Avec l'augmentation du nombre de partitions communicantes et par conséquent du nombre de bus ARINC 429, Boeing a proposé en 1989 un nouveau bus multi-émetteur, multi-récepteur : l'ARINC 629. L'ARINC 629 [65] est utilisé dans les Boeing 777 et les Airbus 330 et 340 depuis 1995 [66, 67].

Bus de données bidirectionnel à 2 Mbps, l'ARINC 629 est capable de connecter jusqu'à 120 terminaux sur un même bus. Le protocole de communication associé à ce bus se décline en deux formes : le protocole basique (*Basic Protocol*) et le protocole combiné (*Combined Protocol*). Le protocole basique est basé sur du CSMA-CA avec une notion de *Time Division Multiple Access* (TDMA) dynamique. Il permet la transmission pour chaque station d'un message à chaque période, identique pour toutes les stations. Le protocole combiné de l'ARINC 629 associe quant à lui les modes périodiques et apériodiques. Dans ce protocole de communication, les trafics sont divisés en trois niveaux comme le montre la Figure 2.9 :

— Le niveau 1 contient tous les trafics périodiques.
— Le niveau 2 contient tous les trafics apériodiques dits urgents.
— Le niveau 3 contient tous les trafics apériodiques non urgents. Dans ce niveau, les messages générés lors du cycle précédent sont prioritaires par rapport à ceux générés dans le cycle courant.

Les trafics des niveaux 1 et 2 doivent être garantis, tandis que ceux du niveau 3 sont des trafics dits *best-effort* : ils sont envoyés s'il reste du temps.

L'ARINC 629, de par son partage des ressources de communication, a permis de

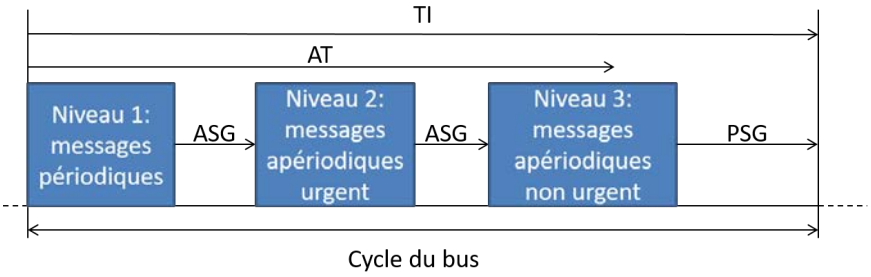


FIGURE 2.9 Cycle du bus dans le protocole combiné de l'ARINC 629

réduire le nombre de liens physiques entre les différentes stations. Il est aussi déterministe puisqu'il est possible de donner les délais de transmission maximum pour des flux garantis. Un de ses inconvénients est le coût élevé des composants, qui a amené à réfléchir à de nouveaux protocoles utilisant des composants sur étagères.

2.2.3.5 ARINC 664 part 7 ou AFDX

Avionics Full DupleX switched Ethernet (AFDX) est un réseau Ethernet commuté à 100 Mbps spécifié dans l'ARINC 664 partie 7 [68]. Il vise à réduire le volume et le poids dûs à la complexité croissante de l'ajout d'équipements supplémentaires.

Pour obtenir un réseau Ethernet commuté déterministe, le réseau AFDX inclut des mécanismes spécifiques tels qu'une configuration statique et le contrôle du trafic.

La configuration statique consiste à définir statiquement le réseau complet avant un décollage : la topologie du réseau, le nombre d'unités connectées, l'allocation des VLs donnant une bande passante pour chaque message transitant par le réseau AFDX, les paramètres et les tables des commutateurs. Chaque VL est défini par trois paramètres :

1. Une bande passante allouée (*Bandwidth Allocation Gap* (BAG)) correspondant à la durée minimale entre deux émissions d'une trame du même VL (flux sporadique). La valeur du BAG est égale à 2^k millisecondes, k étant choisi entre 0 et 7 (soit 1, 2, 4, ... ou 128 ms).
2. Une longueur maximale de la trame Ethernet L_{max} en octets,
3. Une gigue maximale.

Le contrôle du trafic est utilisé pour garantir une qualité de service donnée : un délai ou une gigue à ne pas dépasser, la probabilité qu'une trame soit perdue à cause d'une congestion. Des mécanismes de régulation de trafic (*traffic shaping*) régulent le trafic à la source afin d'assurer que chaque flux est dans son modèle donné. Des mécanismes de

politique de trafic (*traffic policing*) assurent au niveau de chaque commutateur qu'une source ne transmet pas plus qu'elle n'est autorisée.

2.2.3.6 TTEthernet

TTEthernet pour *Time-Triggered Ethernet* [69] propose une amélioration du standard Ethernet pour prendre en compte à la fois des trafics temps-réel et non temps-réel [70]. Pour cela, ce standard ajoute une sur-couche sur la pile Ethernet contenant les services *Time-Triggered* tels que la synchronisation et le maintien d'une horloge commune pour tous les équipements d'une architecture distribuée, et la transmission des messages selon des cycles temporels. Pour ce dernier cas, un ordonnancement de la transmission des messages dits *Time-Triggered* est effectué hors-ligne : ceci permet d'avoir le contrôle des délais de transmission et de la gigue pour la transmission de messages temps-réel critiques.

Trois classes de trafic peuvent être transmises sur un réseau TTEthernet : les messages *Time-Triggered*, à taux contraint et *Best-Effort*.

Les messages *Time-Triggered* (TT) sont transmis sur le réseau selon une séquence prédéfinie hors-ligne. Par conséquent, à aucun moment deux messages TT peuvent se retrouver en concurrence pour l'accès au média de communication : la latence et la gigue de communication sont connues à l'avance. La priorité de ces messages est supérieure aux deux autres classes de trafic définies ci-après. Si une station ne transmet pas un message TT, la bande passante réservée est alors libérée pour les autres classes de trafic.

Les messages à taux contraint (*Rate-Constrained* (RC)) sont utilisés par les applications ayant des exigences de déterminisme et temps réel moins strictes que les applications *time-triggered* strictes. Ce sont des messages émis sporadiquement conformément au standard ARINC 664 partie 7 (AFDX). Pour chaque application, une bande passante est prédéfinie et les délais sont dans les limites définies. De la gigue peut apparaître à cause des délais d'attente dans les commutateurs.

Les messages dits *Best-Effort* (BE) sont associés aux trafics qui ne sont pas temps-réel et implémentent l'approche Ethernet classique. Ces messages ont la priorité la plus faible et n'ont pas de garantie par rapport aux autres classes de trafic : ils sont envoyés sur la bande passante restante du réseau.

2.2.4 Évaluation des délais de transmission pire cas

Évaluer les délais de transmission pire cas consiste à certifier qu'un message arrivera quoiqu'il se passe en un temps maximal donné. La recherche d'une allocation distribuée,

sans connaissance *a priori* de l'architecture matérielle, amène à une complexité tant à la fois sur la recherche d'une allocation et d'un ordonnancement que sur l'allocation des routes sur le réseau. Déterminer le délai pire cas de bout-en-bout pour chaque message sur un média partagé est dépendant de tous les flux y passant. Pour vérifier les exigences de bout-en-bout des systèmes, Lauer [3] tout comme Badache [71] prennent en compte des délais de traversée du réseau variables mais bornés entre un meilleur (*Best-Case Traversal Time* (BCTT)) et un pire temps de traversée (*Worst-Case Traversal Time* (WCTT)).

Dans la suite de ce manuscrit, nous considérons une borne supérieure du délai, indépendante des réseaux de communication. Cette borne nous permet de rechercher une allocation distribuée sans connaissance *a priori* des réseaux utilisés. Cette borne peut être vérifiée par différentes méthodes dès lors que nous connaissons l'architecture matérielle et les routes sur lesquelles passent les messages. Les trois principales méthodes utilisées dans l'avionique sont le calcul réseau [72, 73], l'approche par trajectoire [74] et la vérification de modèles [75, 76].

2.3 Contraintes de latence de bout-en-bout

Dans les systèmes temps-réel stricts, des contraintes temporelles de bout-en-bout doivent être vérifiées. Par exemple, le délai entre l'acquisition de la donnée d'un capteur et son enregistrement dans les enregistreurs de vol doit être garanti : il ne doit pas dépasser les 500 millisecondes [23] comme nous l'illustrons Figure 2.10. Nous nous intéressons ici aux calculs des latences de bout-en-bout des systèmes. Elles ne doivent pas excéder des bornes maximales spécifiées par les systémiers. Dans le cadre d'une architecture distribuée, l'allocation et l'ordonnancement des partitions sur les processeurs, ainsi que leur communication doivent être pris en compte dans l'évaluation des contraintes de communication de bout-en-bout. Les systèmes peuvent être caractérisés par différents paramètres qui vont impacter le calcul de ces latences : tous les processeurs peuvent être synchronisés ou non à une horloge commune (2.3.1), ou bien le traitement des données est différent d'un système à l'autre (2.3.2).

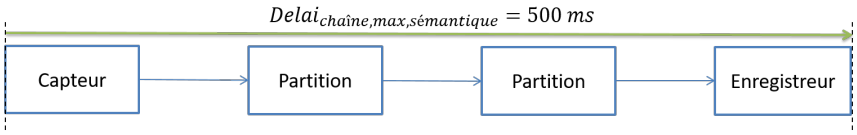


FIGURE 2.10 Exemple de contrainte de bout-en-bout d'une chaîne de données

2.3.1 Synchronisme vs. Asynchronisme des processeurs et des réseaux

Deux processeurs sont synchronisés si leurs cycles d'exécution démarrent au même instant : leur horloge est commune. Autrement, ils sont asynchrones. De la même manière, un réseau est synchrone si l'envoi des messages est effectué selon un espace-temps cyclique prédéfini. Dans le cadre des architectures IMA, les processeurs et les réseaux sont asynchrones. À l'inverse, les architectures DIMA spécifiées dans [15, 20, 49, 77–81] reposent sur une architecture où les processeurs et les réseaux sont synchrones. Par ailleurs, les communications sont synchronisées avec les processeurs [15, 49, 77].

Cette architecture distribuée proposée en premier dans le standard DO-297 [15] repose sur un système complètement périodique où tout est réservé (*Time-Triggered*) : l'exécution d'une partition dans un processeur et la transmission d'un message sur le réseau. L'avantage de cette architecture est de déterminer à l'avance les motifs d'exécution et de transmission des applications les plus critiques [78], mais une fois conçue, l'architecture sera difficilement modifiable [82]. Grâce à un réseau TTEthernet (cf. 2.2.3) par exemple, il est tout de même possible de transmettre des flux sporadiques ou apériodiques entre deux messages périodiques. Tout le système est synchronisé : tous les processeurs doivent se mettre d'accord sur une horloge commune.

Dans le cadre de cette thèse, nous travaillons sur une architecture DIMA asynchrone, que ce soit au niveau des processeurs ou des réseaux.

2.3.2 Sémantiques de bout-en-bout

En 2008, Feiertag et al. [83] soulignent que beaucoup de travaux traitent de l'évaluation des latences de transmission tandis que les latences de bout-en-bout dans des systèmes multi-périodiques ont reçu peu d'attention jusqu'alors. Ils proposent quatre sémantiques de bout-en-bout représentées Figure 2.11 pour étudier les réactions générales de tous types de systèmes automobiles, en considérant les mécanismes et effets spécifiques typiquement présents dans les plates-formes d'exécution automobile tels que le sur-échantillonnage, le sous-échantillonnage et la gigue.

Chaque système a ses propres contraintes. Prenons deux exemples parlants dans l'automobile où la réaction du système doit être déterministe. Le système de déclenchement des airbags doit être réactif pour sauver des vies. Pour valider ce système, l'instant le plus défavorable pour la « première réaction » du système doit être déterminé et ne pas excéder une limite, e.g. 150 millisecondes dans cas du déclenchement de l'airbag depuis le volant du conducteur pour éviter un traumatisme fatal. Un second exemple concerne les applications de contrôle du véhicule telles que le régulateur de vitesse automatique

ou bien le correcteur électronique de trajectoire. Ces applications peuvent être réparties sur un certain nombre de processeurs, impliquant le transit des données sur le réseau : l'âge des données augmente avec un impact évident sur la qualité des modèles de contrôle d'origine. Pour les concepteurs de système, il est de la plus haute importance de contrôler ces effets au plus tôt pour éviter une nouvelle conception du système tardive (et coûteuse).

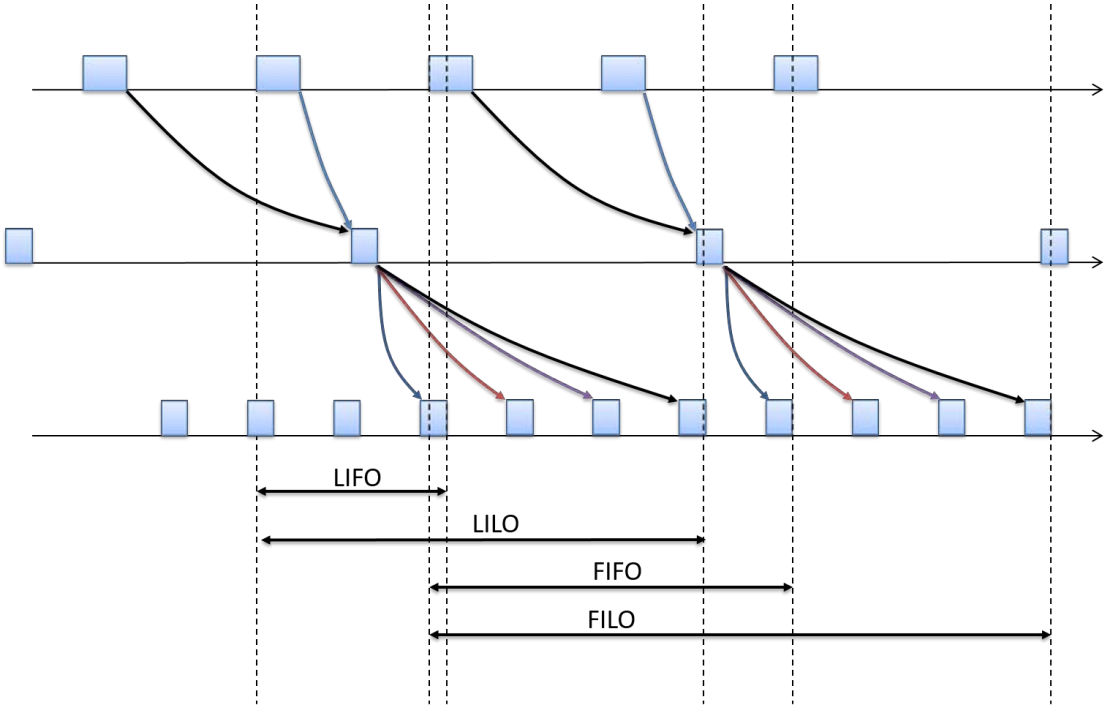


FIGURE 2.11 Sémantiques de bout-en-bout

Les quatre sémantiques sont les suivantes :

1. Le délai *Last In - First Out* (LIFO) : ce délai est égal au temps écoulé entre la dernière entrée non écrasée courante de la première partition de la chaîne de données et sa première utilisation par la dernière partition de la chaîne.
2. Le délai *Last In - Last Out* (LILO) ou « **âge maximal d'une donnée** » : ce délai est égal au temps écoulé entre la dernière entrée non écrasée courante de la première partition de la chaîne de données et sa dernière utilisation par la dernière partition de la chaîne. Ce retard trouve son importance dans de nombreux systèmes multi-périodiques tels que les systèmes de contrôle où l'intérêt réside dans la fraîcheur ou l'âge des données produites. Nous pouvons remarquer que la dernière entrée non écrasée qui se propage vers la dernière partition est autant prise en compte dans les délais LIFO et LILO.

3. Le délai FIFO ou « **délai de première réaction** » : ce délai est égal au temps écoulé entre la première entrée non écrasée courante de la première partition de la chaîne de données et sa première utilisation par la dernière partition de la chaîne. Ce retard trouve son importance dans des systèmes devant être réactif face à un événement tels que le déclenchement d'un airbag.
4. Le délai *First In - Last Out* (FILO) : ce délai est égal au temps écoulé entre la première entrée non écrasée courante de la première partition de la chaîne de données et sa dernière utilisation par la dernière partition de la chaîne. Il spécifie le temps le plus long d'existence de la donnée émise par la première partition.

Dans le contexte avionique, nous nous intéressons au délai de première réaction pour des systèmes critiques tels que l'affichage des données pour les pilotes ou l'enregistrement des données de vol.

2.4 Stratégies de recherche d'allocations valides

Nous nous intéressons dans cette thèse à l'allocation des différentes partitions dans une architecture distribuée. Cette architecture doit vérifier les exigences associées à l'IMA [15]. Pour être certifiée, toute l'architecture doit être prédictible, i.e. l'état d'un système doit être connu à tout instant, et respecter les contraintes temporelles de bout-en-bout des différents systèmes avioniques. En 2.4.1, nous mettons en avant la complexité d'allouer, d'ordonnancer et de vérifier les exigences de bout-en-bout des systèmes avioniques. En 2.4.2, nous détaillons les différentes stratégies d'allocation prenant en compte ces trois étapes dans différents domaines.

2.4.1 Allocation, ordonnancement, exigence

Dans le contexte avionique traité dans cette thèse, le problème est d'ordonnancer hors-ligne un ensemble de partitions périodiques avec des périodes harmoniques. Korst et al. [22] montrent que l'ordonnancement de fonctions non-préemptées périodique est NP-complet dans le sens fort dans le cas d'un monoprocesseur, mais que ce problème se résout en un temps polynomial si les périodes sont harmoniques. Eisenbrand et al. [54, 55] montrent qu'un ordonnancement de fonctions avec des périodes harmoniques sur un processeur existe si et seulement si un tel ordonnancement basé sur une séquence d'intervalles cycliques appelés « *bin* » existe (cf. 2.1.2.3). Ceci revient à ordonnancer les fonctions par période croissante dans les *bins*. La taille d'un *bin* correspond à la plus petite période des fonctions présentes dans le processeur. Leur nombre dans une

séquence est donné par le ratio entre la plus longue et la plus petite période. Étant donné que les périodes sont harmoniques, le plus petit commun multiple (PPCM) des périodes est la plus grande période de toutes les fonctions : cette valeur est appelée « hyper-période ». La séquence générée par l'algorithme des *bins* est répétée à chaque hyper-période. L'algorithme dans [54, 55] s'adapte bien à notre contexte puisque nous considérons des partitions avec des périodes harmoniques.

Lorsque des partitions communicantes sont distribuées sur un ensemble de processeurs, une analyse des délais de bout-en-bout est nécessaire afin de vérifier que les contraintes temporelles de bout-en-bout sont satisfaites.

Le premier aspect de cette analyse de bout-en-bout concerne le délai de transmission sur le média de communication. Les approches classiques dans le contexte avionique sont basées sur le calcul réseau et l'approche par trajectoire [84, 85].

Le second aspect concerne l'analyse temporelle d'une chaîne de communication complète, i.e. l'analyse du délai entre le début de l'exécution de la première partition de la chaîne jusqu'à la fin de l'exécution de la dernière partition de cette même chaîne. Cette analyse est basée sur des sémantiques de bout-en-bout définies par Feiertag [83]. Comme tous les systèmes doivent être prédictibles dans l'avionique, le type de chaîne considéré dans cette thèse est une chaîne dite de données (cf.2.2.2) [61]. Différentes analyses pour de telles chaînes ont été proposées, e.g. [3, 86–89]. Le point de départ de toutes ces approches est une connaissance de l'allocation des fonctions sur les différents processeurs. Ceci signifie qu'il faut avoir construit au préalable les allocations pour pouvoir les appliquer.

Toute la difficulté de ce problème est de pouvoir évaluer la satisfaction des exigences de bout-en-bout tout en construisant l'architecture avionique. Différentes stratégies peuvent être mises en place.

2.4.2 Stratégies d'allocation

Le processus d'allocation des ressources est effectué par l'intégrateur système. Il propose une allocation initiale des partitions hôtes sur les ressources matérielles offertes par une architecture donnée. Il vérifie ensuite que les ressources prévues ne vont pas sur-allouer le système [7]. Mais cette démarche doit être répétée tant qu'une allocation ne respecte pas toutes les exigences, ce qui est coûteux en temps et par conséquent en argent [14].

Différents travaux proposent de répartir un ensemble de partitions sur une architecture distribuée depuis quelques années. Les premières propositions sur les architectures DIMA sont de chercher des allocations et des ordonnancements sur des architectures complètement synchrones [15, 20, 49, 77–81].

Deux stratégies d'allocation peuvent être implémentées dans le cadre d'architectures synchrones ou asynchrones. La première stratégie dite « complète » assigne toutes les partitions aux différents processeurs, avant de rechercher des ordonnancements validant les contraintes de bout-en-bout. La deuxième dite « itérative » alloue les partitions au fur et à mesure que des allocations avec un ordonnancement validant ces contraintes ont été trouvées. Nous détaillons ces deux stratégies ci-dessous.

2.4.2.1 Allocation complète

Une allocation complète consiste à allouer les partitions sur les processeurs disponibles et à les ordonnancer. Cette ordonnancement dépend à la fois des communications et des exigences de bout-en-bout à valider. Connaître le placement des partitions sur les différents processeurs permet de revenir à un ordonnancement par partitionnement (c.f. 2.1.2.3), i.e. un ordonnancement monoprocesseur. Deux groupes de solutions s'appuient sur ce principe.

Le premier groupe de solutions porté par Al Sheikh [90] procède en deux étapes : dans un premier temps, il ordonnance les partitions sur les différents processeurs, puis il route les différents flux sur le réseau avionique. Sur le même principe, Badache [71], en fonction d'une allocation prédéfinie, cherche les ordonnancements permettant de satisfaire les contraintes de bout-en-bout. De telles approches ne peuvent pas être directement appliquées à notre problème. Ces méthodes reviennent à d'abord allouer les partitions sur les processeurs disponibles de telle sorte qu'au moins un ordonnancement valide existe pour chaque processeur, puis vérifier que les contraintes de délai de bout-en-bout sont satisfaites. Le premier problème soulevé est le nombre d'allocations candidates qui explose avec un nombre de partitions croissant. Le deuxième problème concerne l'ordonnancement dans chaque processeur qui reste un problème NP-complet dans le sens fort [22].

Le deuxième groupe de solutions, proposé dans les travaux de Tindell et al. [91] et Ekelin et Jonsson [92], recherche une allocation des partitions qui minimise les coûts de communication. Tindell et al. [91] proposent un algorithme d'ordonnancement hors-ligne multiprocesseur pour ordonnancer des partitions en prenant en compte le temps de traversé des communications avec un protocole spécifique, i.e. Token Ring : les sous-systèmes communicants sont regroupés en cluster pour minimiser le trafic sur le bus et améliorer le séquencement des partitions. Ekelin et Jonsson [92] regroupent eux-aussi les fonctions dépendantes sur les mêmes machines de façon à minimiser les communications sur le réseau. Ces approches se focalisent sur les coûts de communication qui sont un problème mineur dans notre cas : nous devons être capables de valider des contraintes de bout-en-bout.

Ces méthodes, utilisées lorsque l'architecture finale est connue, permettent de définir l'allocation géographique des partitions, les coûts de communication, et par conséquent évaluer les WCTTs. Des algorithmes d'optimisation tels que les méthodes exactes ou bien des méta-heuristiques peuvent être utilisés pour trouver des ordonnancements en fonction d'une allocation donnée. Malheureusement, si l'allocation ne fonctionne pas à cause du placement de deux partitions dans deux processeurs distincts par exemple, il faut vérifier tous les ordonnancements, ce qui amène à une suppression tardive des allocations impossibles.

2.4.2.2 Allocation itérative

Une allocation itérative est une approche intégrée qui consiste à allouer et ordonnancer chaque partition sur les processeurs disponibles et à vérifier que les exigences de bout-en-bout sont satisfaites avant de recommencer la même procédure jusqu'à ce que la dernière partition soit allouée. Ce type d'approche est le plus prometteur pour notre problème puisque l'allocation associée à un ordonnancement et les délais de bout-en-bout sont dépendants l'un de l'autre.

Zhu et al. [93] et Mehiaoui et al. [94] adressent le problème d'allocation et d'ordonnancement avec un respect de contraintes temporelles dans le contexte automobile. Basés sur une Optimisation Linéaire en Nombres Entiers (OLNE), ces deux approches permettent d'allouer les tâches et les messages en fonction de priorités qui leurs sont assignés. Cette OLNE répond à des systèmes de petite taille. Pour résoudre le problème d'allocation sur des systèmes de taille industrielle, Mehiaoui et al. [94] combinent cette OLNE avec un algorithme génétique. Dans [95–97], les auteurs s'intéressent aux modifications possibles des contraintes dans une allocation donnée. Ces modifications peuvent être de deux ordres. Le premier, nommé « *extensibility* », consiste à évaluer à quel point les temps d'exécution des fonctions peuvent être augmentés avant de violer les contraintes du système. Le deuxième, nommé « *flexibility* », consiste à évaluer si des allocations données nécessitent un nombre minimal de modifications en cas de mise à jour d'exigences. À la différence de l'IMA où un ordonnancement statique strictement périodique est construit hors-ligne, ces approches se placent dans un modèle d'exécution sur événements en-ligne.

Il existe peu de travaux qui traitent des problèmes d'allocation et d'ordonnancement multiprocesseur de fonctions non préemptées strictement périodiques avec des contraintes de dépendances et de bout-en-bout. Les travaux de Kermia [59] se rapprochent de notre problématique mais diffèrent sur certains points. Kermia recherche une allocation valide en trouvant un ensemble d'ordonnancement sur des processeurs synchrones. Une allocation est inacceptable si les exigences de bout-en-bout, définies implicitement avec la

sémantique délai de première réaction, ne sont pas satisfaites. Dans l'avionique civile, tous les processeurs ainsi que les moyens de communication sont actuellement asynchrones. Par ailleurs, Kermia [59] essaie de regrouper les tâches en fonction de leur dépendance et ainsi de réduire le nombre de processeurs. Afin d'obtenir une distribution accrue des partitions, nous recherchons de notre côté à trouver des allocations valides avec le plus grand nombre de processeurs possibles.

2.4.2.3 Réduction de l'espace de recherche

Comme nous l'avons évoqué en 2.4.1, une des complexités amenée par une distribution accrue est le nombre d'allocations qui peut exploser. Différents travaux proposent des méthodes pour allouer des partitions en fonction des ressources ou des problématiques de redondance.

Certaines fonctions requièrent des ressources spécifiques comme les E/Ss telles que des capteurs, des actionneurs et des écrans. Dans [56, 98–100], les auteurs prennent en compte ces contraintes de ressources pour restreindre les processeurs auxquels une fonction peut être attribuée : ceci réduit l'espace de recherche des allocations. Par ailleurs, des mécanismes de redondance sont appliqués dans le cadre de systèmes temps-réel critiques en répliquant les fonctions sur plusieurs processeurs [99, 101]. Islam et al. [102] allouent en premier les fonctions périodiques répliquées avant d'allouer les non-périodiques afin de réduire l'espace de recherche des allocations. À chaque fois qu'ils attribuent une fonction à un processeur, ils essaient de trouver un ordonnancement valide.

2.5 Positionnement

Dans cette section, nous positionnons les objectifs de la thèse parmi l'ensemble des solutions proposées dans la littérature.

2.5.1 La recherche d'architectures distribuées dans un contexte IMA

La recherche d'architectures distribuées est menée dans les domaines de l'aéronautique et de l'automobile.

Dans l'aéronautique, l'allocation des fonctions avioniques s'effectue sur une architecture matérielle donnée. Une recherche des ordonnancements satisfaisant les exigences de bout-en-bout revient à vérifier que les ordonnancements monoprocesseurs satisfassent les contraintes temporelles. L'idée de distribuer plus fortement les partitions a été amenée par l'annexe D de la DO-297 [15] sous le nom de DIMA : il s'agit de mettre en place une architecture complètement synchrone, que ce soit au niveau des processeurs ou des communications.

Dans l'automobile, les fonctions automotives sont allouées et ordonnancées sur un modèle d'exécution par événement. La distribution de ces fonctions s'effectue par une approche itérative : l'allocation, l'ordonnancement et la vérification des exigences systèmes sont traités en même temps.

Dans ces différents travaux, le souhait reste de minimiser le nombre de processeurs utilisés.

Dans notre contexte industriel illustré en Annexe A, nous souhaitons une distribution accrue des partitions : nous cherchons un ensemble de solutions avec différents nombres de processeurs afin d'ajouter de nouvelles fonctionnalités ainsi que des nouveaux équipements. Le besoin défini par ce contexte est de garder les propriétés de l'IMA, i.e. un ordonnancement hors-ligne, l'asynchronisme des processeurs voire des moyens de communication, et de respecter les exigences des systèmes avioniques. Nous répondons à ce problème dans les Chapitres 3 et 4 en proposant deux approches intégrant en même temps l'allocation, l'ordonnancement et la satisfaction des exigences de bout-en-bout.

2.5.2 Intégration des temps de transmission dans une architecture distribuée

Une répartition accrue des partitions implique un transit plus important des données sur le réseau, augmentant par conséquent les délais d'une chaîne de partitions communicantes [83]. Pour les concepteurs de systèmes, il est de la plus haute importance de contrôler ces effets au plus tôt pour éviter une nouvelle conception du système tardive (et coûteuse).

Le choix du réseau de communication trouve son importance ici car selon le protocole et les liens physiques choisis, un message ne mettra pas le même temps pour être transmis. Étant donné que l'architecture physique est construite au fur et à mesure, les temps de transmission des messages restent dépendants de l'allocation des partitions et des réseaux traversés. Au lieu de spécifier les réseaux de communication entre certains types de processeurs, nous proposons, une fois que toutes les partitions sont placées et ordonnancées, de spécifier les temps de transmission maximum autorisés. Le choix du-des réseau-x est ensuite laissé à l'intégrateur qui vérifiera que les temps de transmission sont dans les bornes admissibles. Cette spécification est présentée dans le Chapitre 5.

Chapitre 3

Recherche exhaustive d'allocations valides

Dans cette partie, nous nous intéressons à une distribution des partitions sur un nombre borné de processeurs. L'objectif est le suivant :

Un ensemble de n ***partitions*** $\mathcal{P} = \{P_1, ..., P_n\}$ ***doit être alloué sur un ensemble d'au plus*** $maxNbPE$ ***processeurs identiques*** PE ***noté*** $\mathcal{E} = \{PE_1, ..., PE_{maxNbPE}\}$ ***tout en respectant leurs contraintes temporelles.***

Si $maxNbPE$ n'est pas spécifié, il prend la valeur de n, i.e. chaque partition peut avoir son propre processeur.

Illustrons notre objectif avec l'exemple suivant. Dans l'exemple de la Table 3.1, six partitions strictement périodiques, avec des périodes T_i harmoniques et des pires temps d'exécution C_i, doivent être allouées sur au plus trois processeurs. Nous recherchons donc les allocations possibles sur un à trois processeurs.

Table 3.1 Spécification temps-réel des paramètres de l'exemple illustratif

Partitions	C_i (ms)	T_i (ms)
P_1	3	10
P_2	2	10
P_3	2	20
P_4	4	40
P_5	1	40
P_6	4	40

Ces partitions échangent des données entre elles. Des contraintes de délai de bout-en-bout entre le début de l'exécution de la partition source et la fin de l'exécution de

la partition destinatrice $D_{ch_k,max}$ peuvent exister. Chaque donnée définie une chaîne de partitions ch_k. Reprenons l'exemple de la Table 3.1 : dans cet exemple illustratif, trois chaînes de communication sont définies ainsi :

$$\mathcal{C}h = \{ch_1, ch_2, ch_3\}$$

$$ch_1 = \{P_1, P_2, P_3\}, D_{ch_1,max} = 30ms$$

$$ch_2 = \{P_2, P_5\}, D_{ch_2,max} = 40ms$$

$$ch_3 = \{P_4, P_5, P_6\}, D_{ch_3,max} = 60ms$$

Pour simplifier l'illustration, nous considérons que les latences de communication sont constantes. Dans cet exemple, lorsque deux partitions sont distantes, i.e. les deux partitions sont sur des processeurs distincts, cette latence vaut 1 ms. À l'inverse, cette latence est nulle lorsque celles-ci sont sur le même processeur.

Pour trouver les allocations avec des ordonnancements validant les contraintes de bout-en-bout, nous considérons le modèle décrit ci-après en 3.1. Nous proposons deux algorithmes selon une démarche incrémentale : une approche intégrée décrite en 3.2 et une heuristique gloutonne détaillée dans le Chapitre 4.

3.1 Modèle de distribution

Notre modèle se décompose en cinq parties : la définition et le paramétrage des partitions en 3.1.1, le choix d'allouer des partitions dans un ensemble particulier de processeurs en 3.1.2, les contraintes d'ordonnancement dans le contexte IMA en 3.1.3, l'asynchronisme des processeurs en 3.1.4 et son impact sur les contraintes de bout-en-bout en 3.1.5. Nous terminons cette partie par le problème général d'allocation en 3.1.6.

3.1.1 Partitions strictement périodiques de période harmonique

Dans le cadre de l'ARINC 653 [1], les partitions sont strictement périodiques de période harmonique. Chaque partition est caractérisée par le couple (période, pire temps d'exécution) noté (T_i, C_i) illustré Figure 3.1. Étant donné que tous les processeurs sont identiques, le WCET d'une partition est le même quel que soit le processeur. Par ailleurs, la stricte périodicité des partitions implique que la durée entre deux exécutions consécutives est exactement de T_i comme dans la Figure 3.1.

Chaque partition fonctionne avec une sémantique « Lecture-Traitement-Ecriture ». Les partitions s'échangent des données au travers des canaux virtuels, appelés canaux

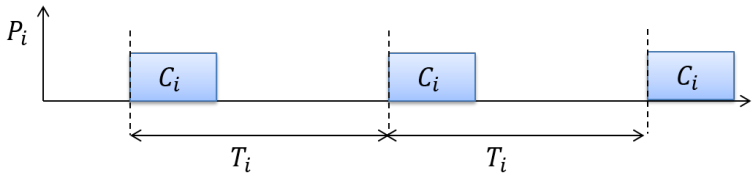


FIGURE 3.1 Partition strictement périodique

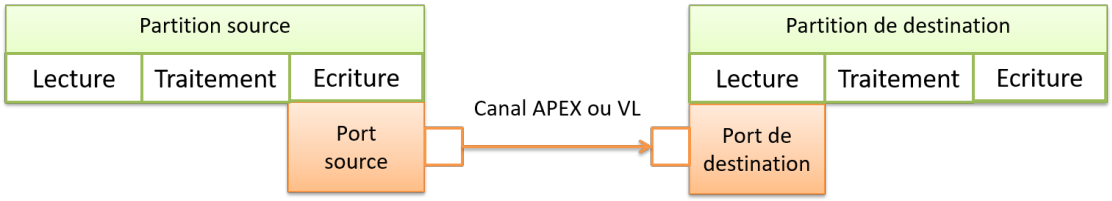


FIGURE 3.2 Exécution et traitement des messages par une partition

APEX dans l'ARINC 653 [1] illustrés Figure 3.2.

Nous considérons dans ce manuscrit la sémantique de délai de première réaction (cf. 2.3.2). Par conséquent, les ports des partitions sont en mode « *queuing* » : à chaque exécution, la partition destinatrice utilise toutes les données reçues dans la file d'attente afin de récupérer la première donnée émise par la partition source.

3.1.2 Allocation

Notre modèle doit associer une architecture matérielle et une architecture fonctionnelle. Le nombre de processeurs n'est pas fixé à l'avance. Nous faisons donc l'hypothèse que les partitions sont distribuées sur un nombre borné de processeurs identiques.

Chaque partition P_i doit être allouée sur un processeur PE_j. Une partition donnée ne peut pas être allouée sur n'importe quel processeur.

— Certaines partitions doivent être allouées sur des processeurs spécifiques. Par exemple, une partition qui échantillonne un capteur doit être allouée sur le processeur gérant ce capteur.

— La redondance est souvent implémentée pour des raisons de tolérance aux fautes. Ceci signifie qu'une fonction donnée est répliquée. Deux réplicas d'une même fonction doivent être impérativement alloués sur des processeurs différents.

Ceci signifie qu'un ensemble de processeurs candidats $\mathcal{E}_{P_i}$ est associé à chaque partition, avec $\mathcal{E}_{P_i} \subset \mathcal{E}$. Les processeurs sont retirés de $\mathcal{E}_{P_i}$ lorsque des réplicas de P_i sont alloués.

3.1.3 Ordonnancement

L'ordonnancement sur chaque processeur PE_j est défini par l'IMA. Il est basé sur une construction hors-ligne d'une MAF notée MAF_j : celle-ci définit statiquement le motif d'exécution de toutes les partitions allouées sur PE_j. Une telle MAF est illustrée dans la Figure 3.3. Cette figure reprend l'exemple de la Table 3.1 présenté au début de ce chapitre où les partitions P_1 et P_3 sont allouées sur le processeur PE_1. La taille Hyp_j de cette MAF dans PE_j est obtenue en prenant le plus petit commun multiple de toutes les périodes des partitions contenues dans $\mathcal{P}_{PE_j}$. Dans notre contexte, les périodes sont harmoniques : la taille de la MAF est alors la plus grande des périodes parmi toutes les partitions allouées sur $\mathcal{P}_{PE_j}$:

$$Hyp_j = LCM_{P_i \in \mathcal{P}_{PE_j}}(T_i) = \max_{P_i \in \mathcal{P}_{PE_j}}(T_i) \tag{3.1}$$

Dans la Figure 3.3, la taille de la MAF correspond à la période de P_3, i.e 20. En effet, nous avons $T_1 = 10$ et $T_3 = 20$.

Chaque MAF est décomposée en un sous-ensemble d'intervalles périodiques :

$$MAF_j = \{s_1^j, ..., s^j{}_{\frac{Hyp_j}{t^j_{intervalle}}}\} \tag{3.2}$$

où $t^j_{intervalle}$ est la plus petite des périodes des partitions contenues dans $\mathcal{P}_{PE_j}$:

$$t^j_{intervalle} = \min_{P_i \subset PE_j}(T_i) \tag{3.3}$$

Chaque partition P_i dans $\mathcal{P}_{PE_j}$ se voit allouer un slot tous les $\frac{T_i}{t^j_{intervalle}}$ intervalles. Dans la Figure 3.3, P_1 obtient un slot tous les intervalles, tandis que P_3 obtient un slot tous les deux intervalles.

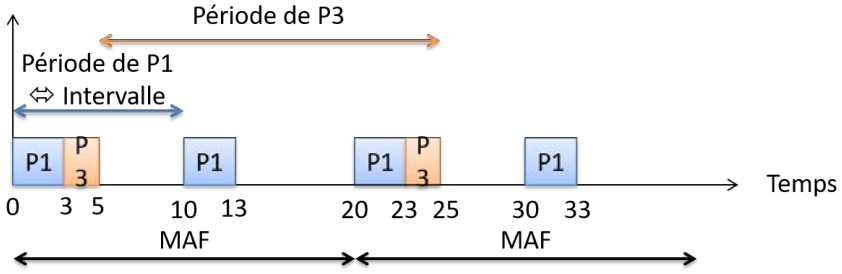


FIGURE 3.3 Motif d'exécution dans le processeur PE_1

r_i définit la durée entre le début de la MAF MAF_j et le début du premier slot alloué à la partition P_i. Dans l'exemple Figure 3.3, nous avons

$$r_1 = 0 \; et \; r_3 = 3.$$

Soit $\mathcal{P}_{PE_j}$ l'ensemble des partitions alloué sur le processeur PE_j. Une condition nécessaire pour que $\mathcal{P}_{PE_j}$ soit ordonnançable sur PE_j consiste à vérifier que la charge temporelle du processeur reste inférieure à 1 :

$$\sum_{P_i \in \mathcal{P}_{PE_j}} \frac{C_i}{T_i} \leq 1. \tag{3.4}$$

Cette condition n'est pas suffisante dans le contexte d'un ordonnancement hors-ligne de partitions non-préemptées [103]. Prenons deux partitions avec des périodes différentes : $P_1 = (10, 5)$ et $P_2 = (20, 8)$. Dans ce cas,

$$\sum_{P_i \in \mathcal{P}_{PE_j}} \frac{C_i}{T_i} = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{5}{10} + \frac{8}{20} = \frac{9}{10} \leq 1.$$

Comme nous l'illustrons Figure 3.4, P_1 utilise 5 unités de temps tous les 10 unités de temps, ne laissant que 5 unités de temps pour que d'autres partitions puissent s'exécuter, ce qui ne laisse pas assez de temps pour que P_2 puisse s'exécuter sans être préemptée.

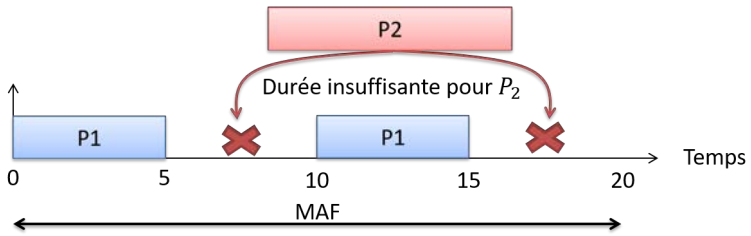


FIGURE 3.4 Espace temporel insuffisant pour allouer une partition dans une MAF

Une solution pour évaluer un ordonnancement des partitions dans $\mathcal{P}_{PE_j}$ sur le processeur PE_j est de trouver un ordonnancement faisable. Par ailleurs, un tel ordonnancement est nécessaire afin de vérifier que les contraintes de délai sont vérifiées.

La charge temporelle dans chaque intervalle de la MAF ne doit pas excéder $t^j_{intervalle}$:

$$\forall k \in \{1, \cdots, \frac{Hyp_j}{t^j_{intervalle}}\}, \sum_{P_i \subset s^j_k} C_i \leq t^j_{intervalle} \tag{3.5}$$

Nous proposons en 3.2.1 un algorithme de recherche d'un ordonnancement faisable.

3.1.4 Asynchronisme des processeurs

Dans un système IMA, il n'y a aucune hypothèse de synchronisation des processeurs. En effet, chaque processeur est indépendant des autres, ce qui permet d'éviter ou de réduire la propagation des fautes si elles existent [104]. Chaque processeur a sa propre horloge et exécute les partitions en fonction de la MAF prédéfinie. L'application doit fonctionner quel que soit le décalage des partitions sur les différents processeurs.

Les communications sont aussi asynchrones : chaque processeur envoie des données dès que les partitions en génèrent. Des délais introduits par ces communications asynchrones doivent être pris en compte. Ils dépendent des messages passant par chaque lien. Comme nous le verrons en 3.2 et 4, notre architecture matérielle est construite au fur et à mesure que nous allouons les partitions. Une borne de pire latence de traversée doit donc être évaluée à chaque fois pour chaque donnée transmise, en fonction de l'ensemble des données transmises et des caractéristiques du réseau de communication.

3.1.5 Contraintes de bout-en-bout

Comme nous l'avons mentionné précédemment, les partitions s'échangent des données. Une chaîne de partitions communicantes est une séquence ordonnée de partitions avec une contrainte applicative de bout-en-bout entre la première et la dernière partition de la chaîne. Typiquement, la première partition transmet un message à la deuxième, qui elle-même transmet un message à la troisième et ainsi de suite.

Plus formellement, un ensemble de chaînes de partitions communicantes $\mathcal{C}h = \{ch_1, ..., ch_m\}$ est considéré. Chaque chaîne ch_k est un n-uplet composé d'un sous-ensemble de $\mathcal{P}$:

$$ch_k = \{P_a, ..., P_e\} \tag{3.6}$$

La contrainte de délai de bout-en-bout d'une chaîne ch_k est notée $D_{ch_k,max}$. Celle-ci est définie selon la sémantique « délai de première réaction »[83] présentée en 2.3.2 : cette sémantique permet de valider de façon déterministe une réaction du système pour par exemple sauver des vies, e.g. l'ouverture d'un airbag lors d'un accident de voiture. Dans l'avionique, un système doit réagir à une commande du pilote en un temps maximal donné afin d'éviter tout dommage qui pourrait être fatal. Cette sémantique consiste à trouver le pire temps écoulé entre la première entrée reçue courante de la première partition de la chaîne de données et sa première utilisation par la dernière partition de la chaîne comme nous l'illustrons Figure 3.5.

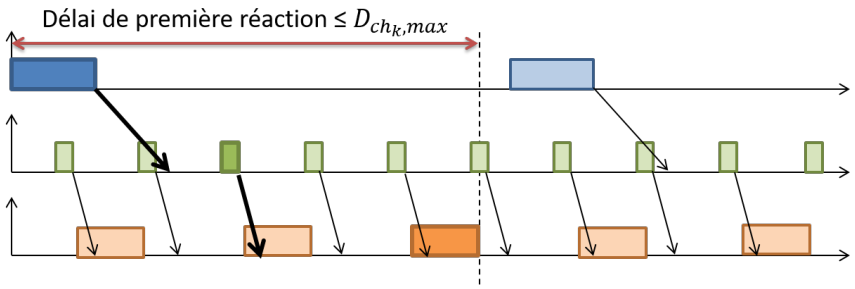


FIGURE 3.5 Sémantique « délai de première réaction »

D'autres sémantiques de délai de bout-en-bout explicitées en 2.3.2 ont été proposées par Feiertag [83], par exemple l'« âge maximal d'une donnée » qui considère la dernière utilisation de la donnée par la partition destinatrice. Dans le contexte des applications avioniques, c'est le plus souvent la première utilisation qui compte.

Le délai de bout-en-bout d'une chaîne ch_k inclut le temps d'exécution des partitions (WCET) et la durée entre leurs exécutions. Pour une allocation donnée de partitions dans $\mathcal{P}$ dans les processeurs $\mathcal{E}$, le pire délai de bout-en-bout D_{ch_k} d'une chaîne ch_k est obtenu par le calcul suivant :

$$D_{ch_k} = \sum_{(P_i,P_j)\in ch_k} distance(i,j) + \sum_{P_i \in ch_k} C_i \tag{3.7}$$

où $distance(i,j)$ est la plus grande distance possible entre la fin de l'exécution de P_i et le début de l'exécution de P_j.

La pire distance $distance(i,j)$ dépend de l'allocation.

Si les deux partitions communicantes P_i et P_j sont sur le même processeur, la communication est locale : le délai entre l'exécution des partitions source et destination dépend de la MAF comme nous l'avons décrit dans la Figure 3.6. Dans ce cas, la première exécution de la partition destinatrice juste après l'exécution de la partition source doit être considérée. Soit r_i (respectivement r_j) l'instant de réveil de la partition source (respectivement destination). Le délai est donné par :

$$0 \leq distance(i,j) = \max_{M\in\{0,..,\left\lceil \frac{T_j}{T_i} \right\rceil -1\}} \left(\min_{L\in\{0,..,\left\lceil \frac{T_i}{T_j} \right\rceil\}} (r_j + L\cdot T_j - (r_i + C_i + M\cdot T_i)) \right) < T_j \tag{3.8}$$

Dans l'exemple Figure 3.6, nous avons $r_j = 0$, $r_i = C_j$ et $T_i = 2 \cdot T_j$. Ainsi, $M = 0$ et

$L \in \{0, 1, 2\}$. Le calcul de la distance entre P_i et P_j donne

$$0 \leq distance(i,j) = \min_{L \in \{0,1,2\}} (L \cdot T_j - (C_j + C_i)) < T_j.$$

Dans ce cas, la seule solution valide est obtenue lorsque $L = 1$. La distance vaut

$$distance(i,j) = T_j - C_j - C_i.$$

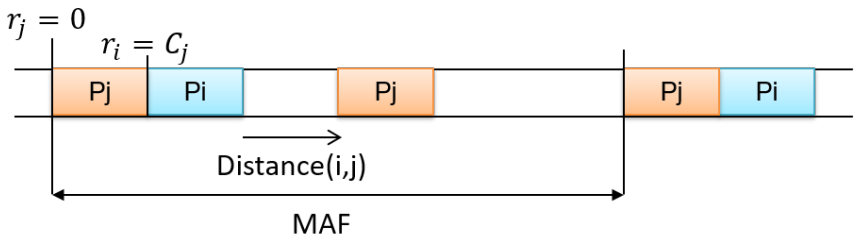


FIGURE 3.6 Distance entre deux partitions communicantes dans le même processeur

Si les partitions communicantes P_i et P_j sont sur des processeurs différents, la latence de transmission ainsi que le délai d'attente entre la réception du message et l'exécution de P_j doivent être pris en compte dans le calcul du délai. Cette latence de transmission peut être évaluée à la volée, à condition de connaître les éléments du réseau traversés et tous les flux les traversant [105]. Comme l'architecture est construite au fur et à mesure que les partitions sont allouées, nous considérons que chaque latence d'un VL est bornée par le pire temps de traversée ($WCTT(i,j)$) d'un message généré par la partition source P_i pour la partition destinatrice P_j. Le délai d'attente est borné par la période T_j de la partition P_j. Ainsi, comme nous l'illustrons Figure 3.7, nous avons :

$$distance(i,j) = WCTT(i,j) + T_j \tag{3.9}$$

Calculons maintenant le pire délai de bout-en-bout de la chaîne $ch_3 = (P_4, P_5, P_6)$. Si P_4, P_5 et P_6 sont allouées sur trois processeurs distincts, en faisant l'hypothèse que tous les WCTTs sont égaux à 1 ms, le calcul du pire délai illustré Figure 3.8 partie b est le

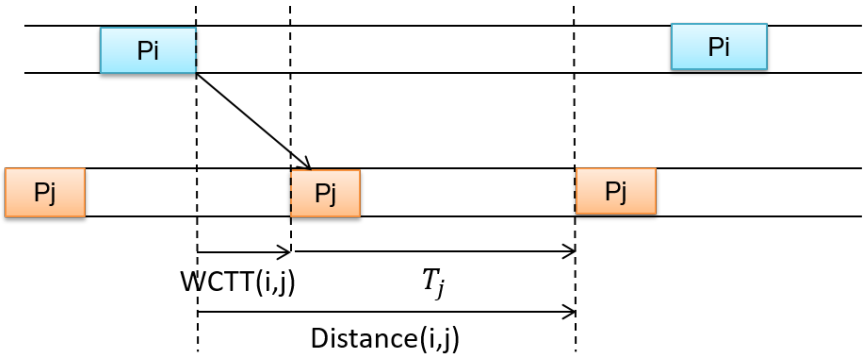


FIGURE 3.7 Distance entre deux partitions communicantes allouées sur deux processeurs distincts

suivant :

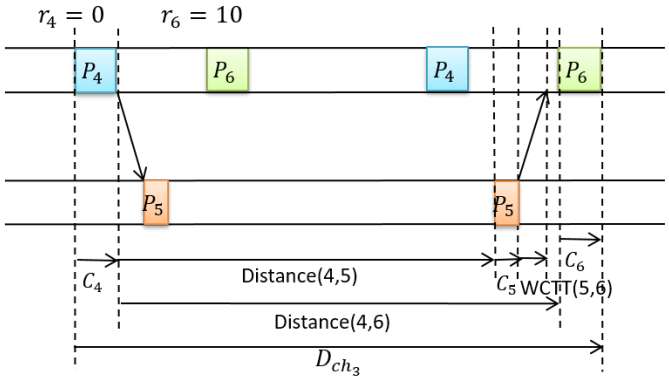
$$\begin{aligned} D_{ch_3} &= distance(4,5) + distance(5,6) + C_4 + C_5 + C_6 \\ &= WCTT(4,5) + T_5 + WCTT(5,6) + T_6 \\ &\quad + C_4 + C_5 + C_6 \\ &= 1 + 40 + 1 + 40 + 4 + 1 + 4 \\ &= 91 \end{aligned}$$

Étant donné que la contrainte de bout-en-bout est $D_{ch_3,max} = 60$, une telle allocation n'est pas valide.

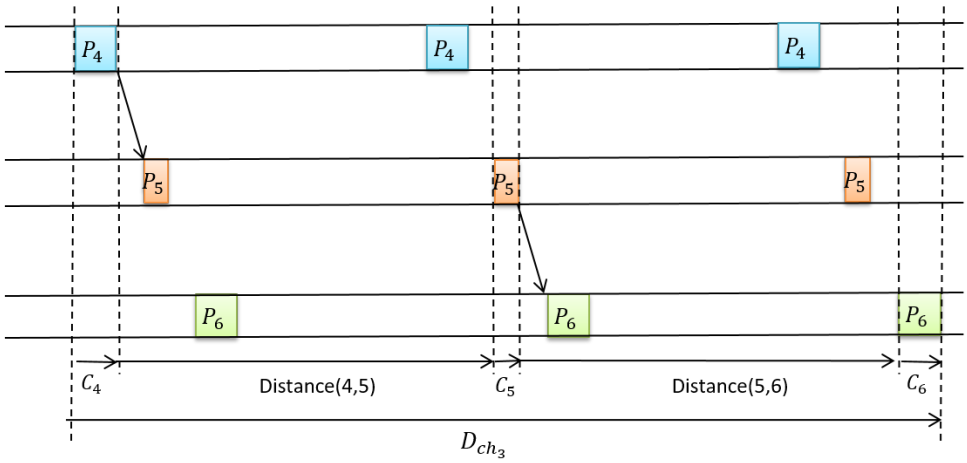
Prenons une autre allocation où P_4 et P_6 sont sur le même processeur et P_5 est allouée sur un autre processeur. Dans ce cas, la chaîne ch_3 reboucle sur le premier processeur. Nous avons le même calcul du délai de bout-en-bout puisque les messages doivent être transmis de P_4 à P_5, puis de P_5 à P_6. Cependant, ce calcul est pessimiste comme nous l'illustrons Figure 3.8 partie a.

Calculons le pire délai de bout-en-bout dans le cas de cette boucle illustrée Figure 3.8 partie a. Supposons que P_4 et P_6, ordonnancées dans le premier processeur, soient séparées d'un *offset* de 10 comme illustré dans la Figure 3.8 partie a. La distance maximale entre la fin de l'exécution de P_4 et l'instant d'arrivée du message pour P_6 est de 43 : $2 \times \text{WCTT}(2) + C_5(1) + T_5(40)$. Il suffit maintenant de déterminer la distance qui sépare la fin de l'exécution de P_4 et le début de l'exécution de P_6. Cette distance doit être la plus grande valeur entre 43 et $43 + T_6$. Ainsi, nous avons :

$$43 \leq distance(4,6) < 43 + T_6 \tag{3.10}$$



a) P_4 et P_6 sur le même processeur



b) P_4, P_5 et P_6 sur différents processeurs

FIGURE 3.8 Calcul du délai dans le cas d'une boucle

avec

$$distance(4,6) = \max_{M\in\{0,..,\lceil \frac{T_6}{T_4}\rceil -1\}} (\min_{L\in\{0,..,\lceil \frac{43+T_4}{T_6}\rceil\}} (r_6 + L\cdot T_6 - (r_4 + C_4 + M\cdot T_4))) \quad (3.11)$$

Afin de trouver la bonne valeur de L, nous devons considérer dans son calcul de borne la distance maximale entre l'émission du message et son retour sur le processeur (ici, 43). La distance entre P_4 et P_6 est donnée par :

$$\begin{aligned} distance(4,6) &= \max_{M\in\{0\}} (\min_{L\in\{0,1,2,3\}} (r_6 + L\cdot T_6 - (r_4 + C_4 + M\cdot T_4))) \\ &= \min_{L\in\{0,1,2,3\}} (r_6 + L\cdot T_6 - (r_4 + C_4)) \\ &= \min_{L\in\{0,1,2,3\}} (10 + L\cdot 40 - 4) \\ &= 46 \text{ avec } L = 1 \end{aligned}$$

Le délai de bout-en-bout de la chaîne ch_3 doit prendre en compte les WCET des partitions d'extrémité, i.e. P_4 et P_6 :

$$\begin{aligned} D_{ch_3} &= C_4 + distance(4,6) + C_6 \\ D_{ch_3} &= 4 + 46 + 4 = 54 \end{aligned}$$

La contrainte est donc respectée.

Dans le cas général, prenons $ch'_k = \{i_0, i_1, \cdots, i_x, i_y\}$ une chaîne de communication où i_0 est allouée sur le même processeur que i_y. Soit $distance_{min}(i_0, i_y)$ la distance minimum entre la fin de l'exécution de i_0 et l'instant d'arrivée du message pour i_y donnée par :

$$distance_{min}(i_0, i_y) = \sum_{0\le z<x} distance(i_z, i_{z+1}) + C_{i_{z+1}} + WCTT(i_x, i_y) \quad (3.12)$$

La distance entre la fin de l'exécution de P_{i_0} et le début de P_{i_y} illustrée Figure 3.9 est donnée par :

$$distance(i_0, i_y) = \max_{M\in\{0,..,\lceil \frac{T_j}{T_i}\rceil -1\}} (\min_{L\in\{0,..,\lceil \frac{distance_{min}(i_0,i_y)+T_i}{T_j}\rceil\}} (r_j + L\cdot T_j - (r_i + C_i + M\cdot T_i))) \quad (3.13)$$

Celle-ci est bornée par :

$$distance_{min}(i_0, i_y) \le distance(i_0, i_y) < distance_{min}(i_0, i_y) + T_{i_y} \quad (3.14)$$

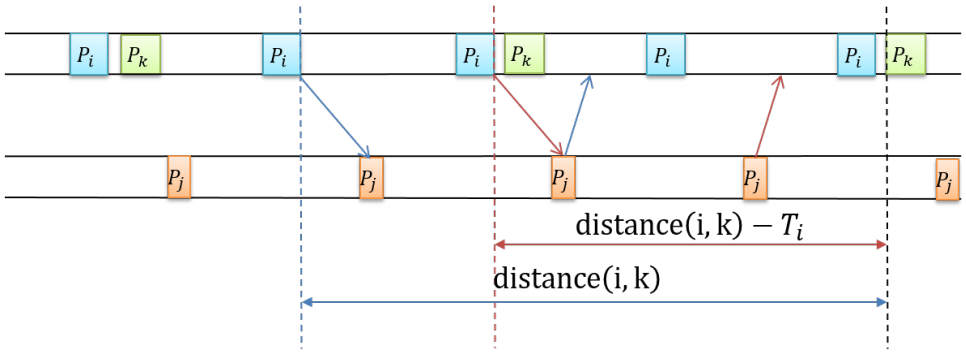


FIGURE 3.9 Calcul général du délai dans le cas d'une boucle

Le délai de bout-en-bout $D_{ch'_k}$ est donné par :

$$D_{ch'_k} = C_{i_0} + distance(i_0, i_y) + C_{i_y} \tag{3.15}$$

3.1.6 Problème général d'allocation

Le problème général consiste à trouver des allocations a de telle sorte que :

— les partitions allouées à un processeur PE_k soient ordonnançables sur PE_k,

— les contraintes de délai de bout-en-bout D_{ch_i} soient garanties pour toutes les chaînes de communication.

Une allocation a vérifiant ces propriétés est valide.

La recherche d'allocations distribuées peut s'effectuer en deux temps. Dans un premier temps, il s'agit de vérifier l'existence d'au moins une solution. Si elle existe, toutes les allocations peuvent être déterminées dans un second temps. Parmi toutes les solutions possibles, une allocation donnée pourra être sélectionnée en fonction de critères tels que le nombre de processeurs utilisés, leur charge, etc.

Dans la suite de ce manuscrit, nous proposons deux approches pour résoudre ce problème.

3.2 Recherche des allocations distribuées valides par une approche intégrée exhaustive

Dans cette partie, nous proposons une première solution pour trouver des allocations distribuées avec leur ordonnancement respectant toutes les contraintes de bout-en-bout. Nous cherchons toutes les allocations valides grâce à un parcours d'arbre en 3.2.1, en testant les allocations partielles générées à chaque nouveau nœud. En 3.2.2, nous évaluons

la complexité de ce premier algorithme et montrons les avantages et les inconvénients de celui-ci sur une étude de cas hélicoptère en 3.2.3 et sur une étude de passage à l'échelle en 3.2.4.

3.2.1 Algorithme général de notre approche intégrée

Comme défini en 3.1.6, il s'agit de construire les allocations valides à partir d'un ensemble de processeurs vides. Cet algorithme consiste à assigner chaque partition à un processeur de telle sorte que :

— les partitions allouées à un processeur PE_k soient ordonnançables sur PE_k,
— les contraintes de délai de bout-en-bout D_{ch_i} soient garanties pour toutes les chaînes de communication.

Une allocation a vérifiant ces propriétés est valide. À la fin d'un processus réussi, nous obtenons une allocation totale valide. À l'inverse, une allocation valide a est dite partielle si au moins une partition n'est pas encore allouée. Lorsque les contraintes d'une chaîne ch_k sont vérifiées, la pire distance $distance(i,j)$ est inconnue si P_i ou P_j n'est pas allouée. Dans ce cas, une borne inférieure de $distance(i,j)$, i.e. 0, est considérée. Par conséquent, la formule 3.7 devient :

$$D_{ch_k,a} = \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset a}} \big(distance(i,j)\big) + \sum_{P_i\in ch_k} C_i \tag{3.16}$$

Pour obtenir cette allocation a, il faut réussir à construire une MAF dans chaque processeur PE_k exécutant au moins une partition. Si une MAF existe, il devient possible de calculer les contraintes de bout-en-bout de toutes les chaînes avec l'équation 3.16 et de vérifier que leur contrainte n'est pas dépassée. Si une contrainte n'est pas satisfaite, alors un nouvel ordonnancement doit être trouvé.

Notre approche consiste en la construction d'un arbre dont tous les nœuds sont créés au fur et à mesure qu'une partition est allouée sur un processeur particulier. Pour cela, nous créons un algorithme récursif initialisé dans l'Algorithme 1. Nous détaillons les choix d'assigner les partitions selon leur périodicité pour ensuite développer notre Algorithme 2 d'allocation et d'ordonnancement qui est validé par l'Algorithme 3. Ce dernier teste si toutes les contraintes de bout-en-bout sont satisfaites.

3.2.1.1 Tri des partitions par période croissante

L'ajout d'une partition P_j à une MAF peut modifier ses caractéristiques. Nous détaillons en Annexe B.1.1 ces modifications dans le cas où la période T_j de P_j est

Algorithme 1 Initialisation de l'approche intégrée

Pré-conditions : les partitions triées et numérotées par période croissante, le nombre maximum de processeurs $MaxNbPE$

1: Soit a_0 une allocation de $MaxNbPE$ processeurs vides
2: **pour** $x = 1..MaxNbPE$ **faire**
3: Le processeur PE_x est représenté par un tableau de T_1 unités de temps
4: **fin pour**
5: Ordonnancement de la partition P_1 sur le processeur PE_1 de l'allocation a_0 (Algorithme 2)

différente des périodes des autres partitions déjà allouées que nous résumons ci-dessous :

— Si $T_j < t_{intervalle}$, alors il faut découper chaque intervalle de la MAF en $\frac{t_{intervalle}}{T_j}$ intervalles, réduire la valeur de $t_{intervalle}$ à T_j, sélectionner dans quel intervalle les slots des partitions déjà ordonnancées doivent être placés et ordonnancer P_j strictement périodiquement dans tous les intervalles comme le montre la Figure 3.10. Pour remédier à ce problème, une solution consiste à définir que la durée de tous les intervalles de chaque MAF soit toujours égale à la plus petite des périodes des partitions. Ceci implique qu'une MAF, dès qu'elle est créée, peut contenir plus d'un intervalle. Nous utilisons ce principe dans la suite de cette thèse.

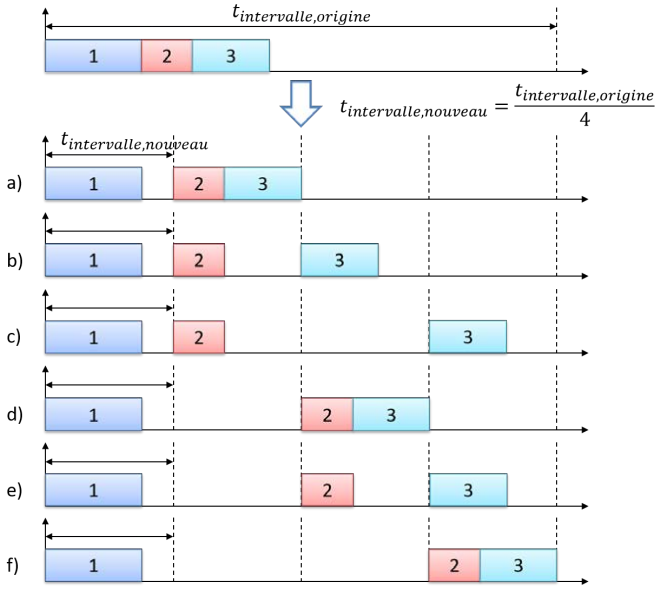


FIGURE 3.10 Création de nouvelles MAFs lorsque la valeur de l'intervalle est divisée par 4

Dans ce premier algorithme, nous choisissons d'effectuer le tri des partitions par

période croissante avant de les assigner à un processeur. Dans ce cas, la période d'une partition allouée dans un processeur PE_k sera supérieure ou égale à $t^k_{intervalle}$. Les partitions sont ainsi triées et numérotées par période croissante (Algorithme 1, Pré-conditions).

— Si $Hyp < T_j$, alors le motif d'ordonnancement doit être dupliqué $\frac{T_j}{Hyp}$ fois pour que la nouvelle hyper-période $Hyp_{nouvelle}$ devienne égal à T_j. Il suffit ensuite d'allouer la partition P_j dans les $\frac{Hyp}{t_{intervalle}}$ premiers intervalles de la MAF, les intervalles suivants étant la répétition de la MAF d'origine comme le montre la Figure 3.11.

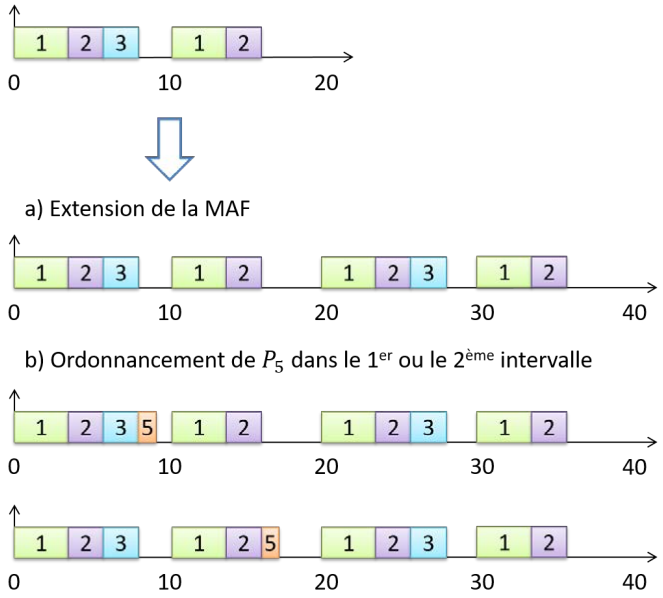


FIGURE 3.11 Extension de la MAF lorsque la valeur de l'hyper-période est doublée

Dans chaque nouveau processeur k, un premier tableau vide de T_1 unités de temps, i.e. la plus petite période de l'ensemble des partitions, est généré (Algorithme 1, lignes 2-4). Ce tableau représente la MAF_k du processeur k de durée Hyp_k.

La première partition peut être ordonnancée (Algorithme 1, ligne 5).

3.2.1.2 Sélection d'une allocation et d'un ordonnancement valide

La création d'un nœud dans notre arbre correspond à l'allocation de la partition P_i dans le processeur $PE_k, k \in [1, Nb_{PE} + 1]$ donnée par l'Algorithme 3. Nb_{PE} représente le nombre de processeurs contenant au moins une partition, et par conséquent une MAF. Si $k \leq Nb_{PE}$, la partition est alors allouée dans un processeur contenant une MAF (Algorithme 2 lignes 15-27) : la nouvelle partition doit alors être placée dans un

ordonnancement pré-établi dans un nœud amont. Si $k = Nb_{PE} + 1$, alors la partition est allouée dans un nouveau processeur (Algorithme 2 lignes 5-14). Nous pouvons choisir indifféremment un processeur puisqu'ils sont tous identiques. Comme précisé en 3.1.2, deux réplicas d'une même fonction doivent être impérativement alloués sur des processeurs différents : cette contrainte nous permet d'éliminer un certain nombre de nœuds lors de l'assignation des partitions sur les différents processeurs.

La première MAF d'un processeur est composée d'une unique partition P_i. Le motif créé contient un seul intervalle de temps T_i, qui représente aussi l'hyper-période de la MAF. La partition P_i débute son exécution à l'instant 0 (Algorithme 2 ligne 6).

Choisir l'emplacement d'un slot pour une partition peut s'avérer complexe. Différents choix détaillés en Annexe B.1.2 s'offrent à nous :

— Effectuer autant de permutations qu'il y a de partitions dans le processeur. Comme le nombre d'intervalles dans une MAF peut être supérieur à 2, il faut effectuer ces permutations dans tous les intervalles. La complexité augmente si plusieurs partitions se situent dans plusieurs intervalles : il faut vérifier que toutes les partitions s'exécutent de façon strictement périodique comme le montre la Figure 3.12. Si nous ajoutons la satisfaction des contraintes de bout-en-bout, la validité d'un ordonnancement dans un processeur peut dépendre des ordonnancements dans les autres processeurs. Ceci signifie que pour chaque partition ajoutée, la recherche d'un autre ordonnancement dans d'autres processeurs peut résoudre le problème de satisfaction des exigences de bout-en-bout. La difficulté revient à retenir et à ne pas revérifier des ordonnancements invalides, ce qui est coûteux en temps et en mémoire.

— Intercaler la nouvelle partition dans le motif déjà prédéfini, en prenant en compte la stricte périodicité des partitions. Ceci revient à décaler temporellement les slots alloués à d'autres partitions pour exécuter le slot de la partition ajoutée au processeur, comme nous l'illustrons Figure 3.13.

Nous choisissons d'intercaler la partition ajoutée dans le processeur PE_k dans les slots disponibles de sa MAF_k. Cette solution permet de figer la MAF à chaque niveau de l'arbre et d'éviter de répéter des ordonnancements infaisables.

Ainsi une nouvelle partition P_j de période T_j et de durée d'exécution C_j peut tenter d'être ordonnancée dans la MAF_k si $T_j \leq Hyp_k$. Si ce n'est pas le cas, la MAF_k est répliquée $\frac{T_j}{Hyp_k}$ fois (Algorithme 2, lignes 1-4). La nouvelle longueur de la MAF_k devient $Hyp_k = T_j$ (Algorithme 2, ligne 3). Si toutes les unités de temps sont libres, alors P_j est ordonnancée dans les C_j premières unités de temps de la MAF_k (Algorithme 2, lignes 5-6). Sinon, P_j est ordonnançable dans MAF_k s'il existe un intervalle libre de longueur

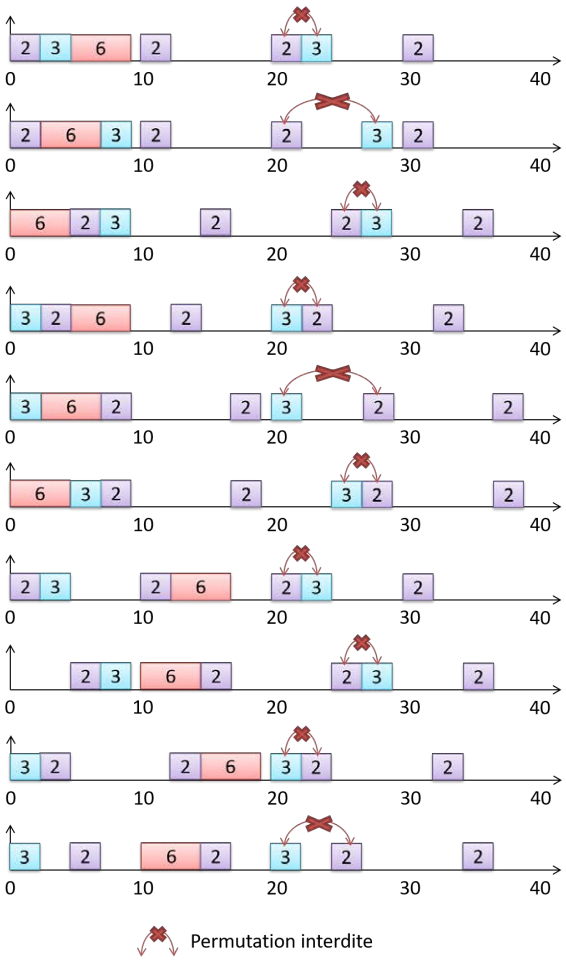


FIGURE 3.12 Tous les ordonnancements de P_2, P_3 et P_6 possibles obtenus par permutations et décalages dans les intervalles d'un processeur

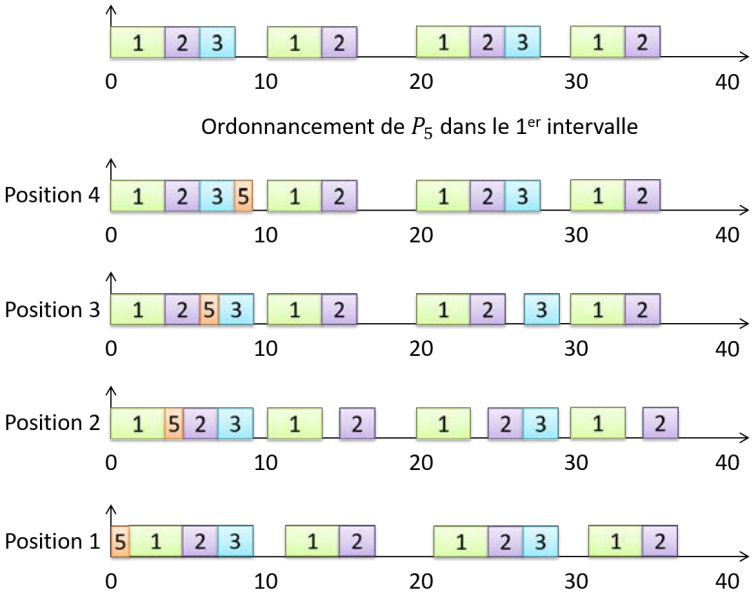


FIGURE 3.13 Décalage des partitions lors de l'ajout d'une nouvelle partition dans une MAF

C_j (Algorithme 2, lignes 16-17, Algorithme 3, lignes 1-3) :

$$MAF_k\,[x, x + C_j - 1] \quad \text{telle que} \quad \begin{cases} x \geq 0 \\ x + C_j - 1 \leq T_j \\ \text{l'intervalle } [x, x + C_j - 1] \text{ est libre.} \end{cases} \tag{3.17}$$

3.2.1.3 Validité d'une architecture

Pour vérifier qu'une allocation respecte bien les contraintes de délai de bout-en-bout, il faut qu'une MAF existe dans chaque processeur utilisé. En effet, l'évaluation des distances entre la fin de l'exécution de la partition source et le début de l'exécution de la partition destinatrice est nécessaire dans le calcul des délais pour chaque chaîne, comme nous l'avons montré Figures 3.6 et 3.7.

Si une nouvelle partition intercalée à une position dans une MAF donnée ne permet pas de satisfaire les contraintes de bout-en-bout comme la partition P_6 illustrée Figure 3.14 partie a, cette partition est déplacée et à nouveau intercalée à une autre position de la MAF comme dans la Figure 3.14 partie b. Si aucun ordonnancement dans le processeur donné ne fonctionne, alors les partitions les plus récemment intercalées dans

Algorithme 2 Ordonnancement d'une partition P_j sur un processeur PE_k d'une allocation a_l

1: **si** $T_j > Hyp_k$ **alors**
2: Répliquer la MAF_k $\frac{T_j}{Hyp_k}$ fois
3: $Hyp_k \leftarrow T_j$
4: **fin si**
5: **si** MAF_k vide **alors**
6: **si** a_m=Segment possible (0, PE_k, P_j, a_l) existe **alors**
7: **si** la partition P_j était la dernière partition à ordonnancer **alors**
8: a_m est une allocation totale valide
9: **sinon**
10: **pour** $h = 1..\min(\text{nombre de processeurs utilisés} + 1, MaxNbPE)$ **faire**
11: Ordonnancement de la partition P_{j+1} sur le processeur PE_h de l'allocation a_m (Algorithme 2)
12: **fin pour**
13: **fin si**
14: **fin si**
15: **sinon**
16: **pour** $i = 0..Hyp_k - (C_j + 1)$ **faire**
17: **si** a_m=Segment possible (i, PE_k, P_j, a_l) existe (Algorithme 3) **alors**
18: **si** la partition P_j était la dernière partition à ordonnancer **alors**
19: a_m est une allocation totale valide
20: **sinon**
21: **pour** $h = 1..\min(\text{nombre de processeurs utilisés} + 1, MaxNbPE)$ **faire**
22: Ordonnancement de la partition P_{j+1} sur le processeur PE_h de l'allocation a_m (Algorithme 2)
23: **fin pour**
24: **fin si**
25: **fin si**
26: **fin pour**
27: **fin si**

les nœuds amonts sont déplacées comme la partition P_5 dans la Figure 3.14 partie c et la partition, ici P_6, est à nouveau intercalée à une autre position dans la MAF de leur nœud. Si l'ensemble des ordonnancements d'un nœud respecte les contraintes de bout-en-bout, alors une allocation partielle valide a été trouvée. Dans ce cas, $Nb_{PE} + 1$ nœuds enfants, Nb_{PE} étant le nombre de processeurs utilisés dans le nœud parent, sont créés afin d'assigner les partitions non allouées soit dans l'un des Nb_{PE} processeurs déjà utilisés, soit dans un nouveau processeur. Si toutes les partitions sont assignées à un processeur, alors une allocation totale valide a été trouvée.

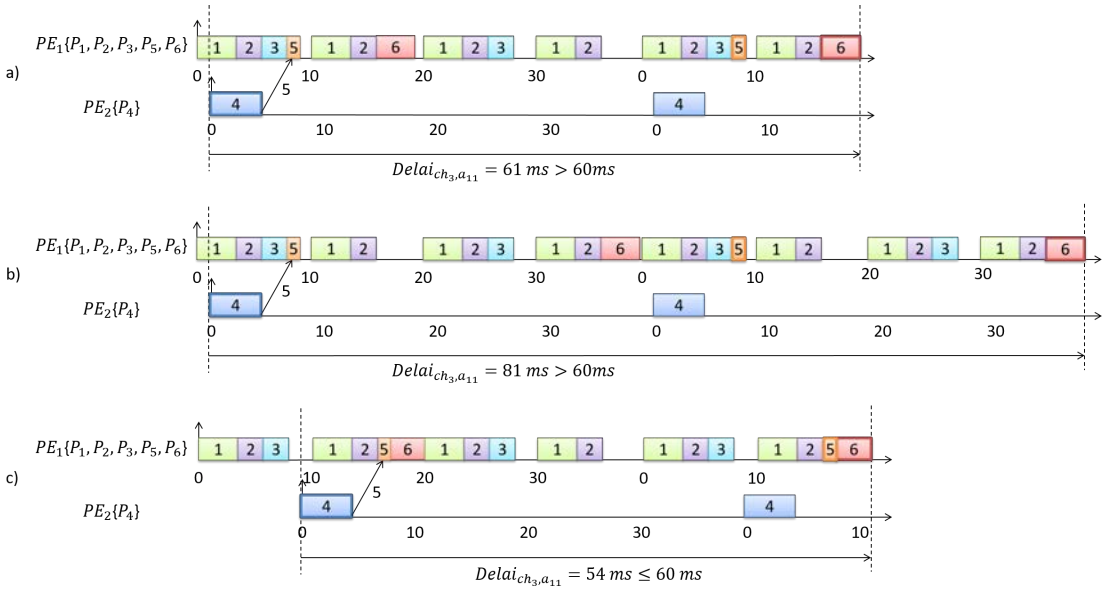


FIGURE 3.14 Modification d'ordonnancements pour valider les contraintes de bout-en-bout de la chaîne ch_3

Un intervalle est possible si toutes les contraintes de bout-en-bout sont satisfaites (Algorithme 3, lignes 4-6). Si c'est le cas, une nouvelle allocation a_m est générée et retournée à l'Algorithme 2 (Algorithme 3, lignes 7-9), sinon, l'intervalle est impossible (Algorithme 3, lignes 10-11).

Soit a_m une allocation partielle valide (Algorithme 2, lignes 6 et 17). Cette allocation est totale si P_j est la dernière partition à allouer (Algorithme 2, lignes 7-8 et 18-19). Sinon, il faut allouer la partition suivante. Celle-ci est assignée sur un des processeurs utilisés, ou sur un nouveau : pour chaque assignation, une nouvelle allocation est créée (Algorithme 2, lignes 10-12 et 21-23). Pour éviter de repasser par des nœuds dont tous les nœuds enfants sont des allocations valides, nous retenons ces allocations partielles dans une variable $\mathcal{A}_{valides}$: nous vérifions à chaque fois qu'une partition est allouée sur

un processeur si la future allocation n'existe pas dans $\mathcal{A}_{valides}$.

Algorithme 3 Déterminer si le segment démarrant à l'instant i dans le processeur PE_k de l'allocation a_l est possible pour la partition P_j

1: **si** le segment $[i, i + C_j - 1]$ est libre **alors**
2: a_m est l'allocation partielle intégrant la partition P_j dans le processeur PE_k de l'allocation a_l à l'instant i
3: **fin si**
4: **pour** $p = 1$..nombre de chaînes **faire**
5: Vérifier que $D_{ch_p,a_m} \leq D_{ch_p,max}$
6: **fin pour**
7: **si** toutes les chaînes $D_{ch_p,a_m} \leq D_{ch_p,max}$ **alors**
8: a_m est une allocation partielle valide
9: Retourner a_m
10: **sinon**
11: Segment impossible
12: **fin si**

3.2.1.4 Illustration de l'algorithme

Nous illustrons cet algorithme sur l'ensemble des partitions $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ qui nécessitent d'être allouées sur au plus trois processeurs, i.e. $\mathcal{E} = \{PE_1, PE_2, PE_3\}$. Les caractéristiques de ces partitions sont spécifiées dans la Table 3.1, qui donne pour chaque partition P_i son pire temps d'exécution (égal à C_i) et sa période (T_i).

Trois chaînes de communication sont traitées :

$$\mathcal{C}h = \{ch_1, ch_2, ch_3\}$$

$$ch_1 = \{P_1, P_2, P_3\}, D_{ch_1,max} = 30ms$$

$$ch_2 = \{P_2, P_5\}, D_{ch_2,max} = 40ms$$

$$ch_3 = \{P_4, P_5, P_6\}, D_{ch_3,max} = 60ms$$

Dans cet exemple, nous supposons que la latence de communication pire cas entre deux partitions distantes, i.e. sur deux processeurs différents, est de 5 millisecondes quelles que soient les partitions. Cela simplifie le calcul dans la mesure où il n'est pas nécessaire d'effectuer une analyse de la latence de communication pire cas pour chaque allocation. En revanche, cette latence pire cas globale doit englober toutes ces allocations possibles. Elle ajoute donc du pessimisme.

L'algorithme de recherche en profondeur est développé sur l'exemple ci-dessous et partiellement illustré Figure 3.15.

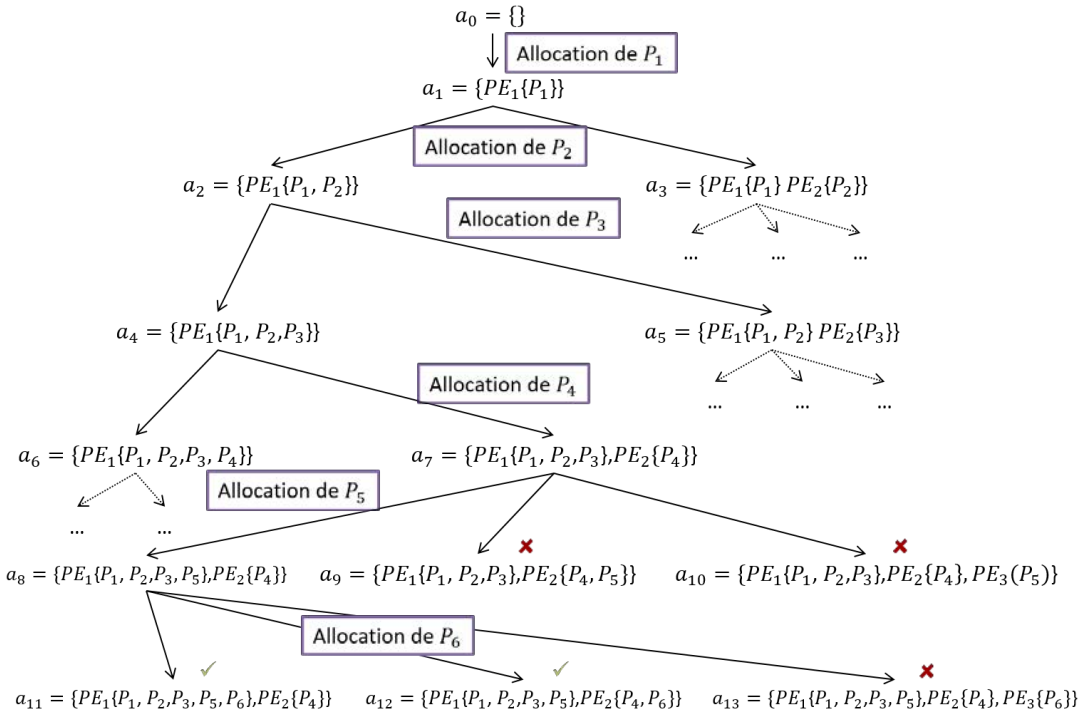


FIGURE 3.15 Une recherche en profondeur dans l'arbre d'allocations

La première étape consiste à assigner la première partition P_1 au premier processeur PE_1 dans l'allocation partielle a_1. La MAF est composé d'un intervalle qui dure 10 ms, i.e. la période de la partition P_1. La MAF est seulement composée d'un intervalle où la partition P_1 commence son exécution à $r_1 = 0$, i.e. au début de la MAF comme le montre la Figure 3.16 (Algorithme 2 lignes 5-14).

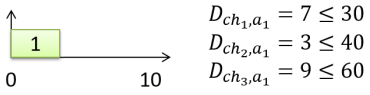


FIGURE 3.16 Allocation de P_1 sur le processeur PE_1 dans l'allocation a_1

Comme il n'y a qu'une seule partition allouée, les délais de première réaction à cet instant pour D_{ch_1,a_1}, D_{ch_2,a_1} et D_{ch_3,a_1} sont respectivement égaux à 7, 3, 9. Comme la MAF respecte les contraintes d'ordonnancement et temporelle, l'allocation partielle a_1 est valide (Algorithme 3 lignes 7-9).

La seconde étape consiste à allouer P_2 en considérant la précédente allocation partielle valide, a_1. Ici, P_2 peut être allouée soit sur PE_1, soit sur un nouveau processeur PE_2 : deux nouvelles allocations partielles enfants sont générées, a_2 et a_3.

Si nous prenons l'allocation partielle a_2, la partition P_2 a la même période que P_1. P_2 est allouée dans le même intervalle, juste après P_1 comme nous l'illustrons Figure 3.17 (Algorithme 2 lignes 16-26).

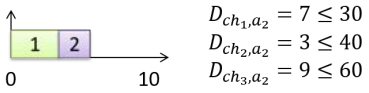


FIGURE 3.17 Allocation de P_2 sur le processeur PE_1 dans l'allocation a_2

En ajoutant cette partition, nous devons vérifier le délai de bout-en-bout de la chaîne partielle ch_1, D_{ch_1,a_2}, qui peut être calculé ainsi :

$$D_{ch_1,a_2} = distance(1,2) + C_1 + C_2 + C_3 \tag{3.18}$$

$distance(1,2)$ correspond à la distance maximale entre la fin de l'exécution de la partition P_1 et le début de l'exécution de P_2 comme nous l'illustrons Figure 3.6. Ce calcul donne :

$$0 \leq distance(1,2) = \max_{M\in\{0\}} (\min_{L\in\{0,1\}} (r_2 + L \cdot T_2 - (r_1 + C_1 + M \cdot T_1))) < T_2$$

$$0 \leq distance(1,2) = \min_{L\in\{0,1\}} (3 + L \cdot 10 - 3) < 10$$

$$0 \leq distance(1,2) = 0 \text{ avec } L = 0.$$

Ainsi, nous avons :

$$\begin{aligned} D_{ch_1,a_2} &= distance(1,2) + C_1 + C_2 + C_3 \\ &= 0 + 3 + 2 + 2 = 7 \leq D_{ch_1,max} \end{aligned} \tag{3.19}$$

Étant donné que cette allocation partielle est valide, nous essayons d'ajouter la partition P_3 soit sur ce processeur PE_1, soit sur un nouveau processeur PE_2. Ainsi, deux nouvelles allocations partielles potentielles se dessinent : a_4 et a_5. Sur le nœud a_4, P_3 est allouée sur le même processeur que P_1 et P_2 mais P_3 n'a pas la même période que ces dernières. L'hyper-période de la MAF doit être modifiée pour prendre la valeur de la période de la nouvelle partition. Les intervalles précédents Figure 3.17 sont répliqués afin d'étendre la MAF comme nous l'illustrons Figure 3.18 (Algorithme 2 lignes 1-4). Il suffit de vérifier que l'allocation de la nouvelle partition sur les premiers intervalles

est faisable, étant donné que les intervalles suivants sont une répétition des premiers intervalles répliqués.

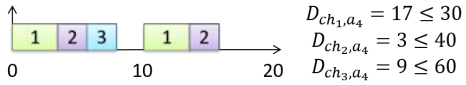


FIGURE 3.18 Allocation de P_3 sur le processeur PE_1 dans l'allocation a_4

Le délai de bout-en-bout de la chaîne ch_1 étendue doit être vérifié :

$$D_{ch_1,a_4} = distance(1,2) + distance(2,3) + C_1 + C_2 + C_3 \tag{3.20}$$

Dans ce cas :

$$0 \leq distance(2,3) = \max_{M\in\{0,1\}} (\min_{L\in\{0,1\}} (r_3 + L \cdot T_3 - (r_2 + C_2 + M \cdot T_2))) < T_3$$

$$0 \leq distance(2,3) = \max_{M\in\{0,1\}} (\min_{L\in\{0,1\}} (5 + L \cdot 20 - (3 + 2 + M \cdot 10))) < 20$$

$$distance(2,3) = \begin{cases} 0 & \text{si } M = 0 \quad L = 0 \\ 20 & \text{si } M = 0 \quad L = 1 \\ -10 & \text{si } M = 1 \quad L = 0 \\ 10 & \text{si } M = 1 \quad L = 1 \end{cases}$$

$$0 \leq distance(2,3) = 10 < 20 \text{ avec } (M,L) = (1,1).$$

$$D_{ch_1,a_4} = 0 + 10 + 3 + 2 + 2 = 17 \leq D_{ch_1,max} \tag{3.21}$$

Toutes les contraintes sont vérifiées (Algorithme 3 lignes 4-6), ainsi a_4 est une allocation partielle valide (Algorithme 3 lignes 7-9).

Dans la quatrième étape, nous essayons d'allouer P_4, soit sur le même processeur PE_1 dans l'allocation partielle a_6, soit sur un nouveau processeur PE_2 dans l'allocation partielle a_7. Si nous nous intéressons à a_7, la MAF_1 sur PE_1 n'est pas modifiée. La MAF_2 étant de taille $T_1 < T_4$, la MAF_2 est dupliquée 4 fois pour atteindre la taille T_4 (Algorithme 2 lignes 1-4). La MAF_2 étant vide, P_4 est ordonnancée à l'instant 0 Figure 3.19 (Algorithme 2 lignes 5-14). Comme D_{ch_3,a_7} est égal à 9, a_7 est un nœud valide (Algorithme 3).

La cinquième étape consiste en l'allocation de P_5, soit sur PE_1, soit sur PE_2, soit sur un nouveau processeur PE_3 qui correspondent respectivement aux allocations a_8, a_9 et

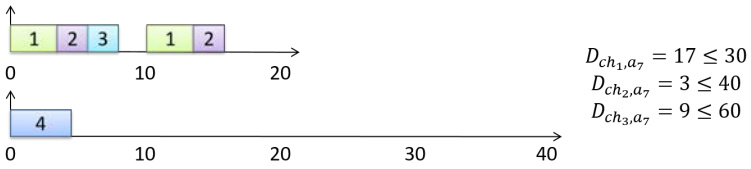


FIGURE 3.19 Allocation de P_4 sur le processeur PE_2 dans l'allocation a_7

a_{10}. Si nous nous concentrons sur le nœud a_{10}, la communication entre P_2 et P_5 mène à :

$$D_{ch_2,a_{10}} = distance(2,5) + C_2 + C_5 \tag{3.22}$$

$$distance(2,5) = WCTT(2,5) + T_5 = 5 + 40 = 45 \tag{3.23}$$

$$D_{ch_2,a_{10}} = 45 + 2 + 1 = 48 > D_{ch_2,max} \tag{3.24}$$

Nous constatons que si les partitions P_2 et P_5 ne sont pas sur le même processeur, la contrainte de bout-en-bout D_{ch_2} n'est pas satisfaite (Algorithme 3 lignes 10-11). Les nœuds a_9 et a_{10} ne peuvent pas avoir de nœud enfant.

Sur le nœud de l'allocation partielle a_8, nous ajoutons P_5 dans le processeur PE_1 Figure 3.20. Comme la partition P_5 a une plus grande période que toutes les partitions présentes dans MAF_1, la MAF_1 est dupliquée 2 fois afin que $Hyp_1 = T_5$ (Algorithme 2 lignes 1-4). Comme il reste assez de temps dans le premier intervalle pour que P_5 puisse s'exécuter, son slot est alloué dans cet intervalle comme le montre la Figure 3.20. Allouer P_5 dans le premier processeur permet de respecter le délai de bout-en-bout $D_{ch_2,max}$ (Algorithme 3).

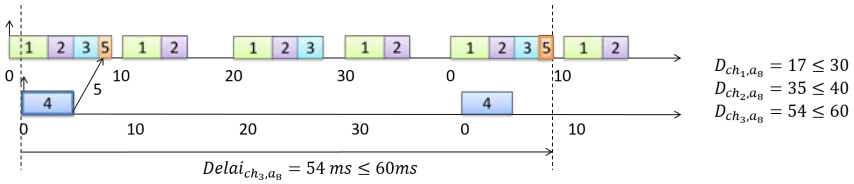


FIGURE 3.20 Allocation de P_5 sur le processeur PE_1 dans l'allocation a_8

Dans une sixième et dernière étape, nous essayons d'allouer P_6. Trois nœuds enfants sont générés : a_{11}, a_{12} et a_{13}. a_{13} n'est pas valide à cause des distances trop importantes entre les trois partitions communicantes P_4, P_5 et P_6. a_{11} et a_{12} sont valides après plusieurs modifications de l'ordonnancement. Dans le cas de l'allocation a_{12}, la partition P_6 est retardée pour débuter à 15 ms afin de valider D_{ch_3} avec une boucle comme nous l'illustrons Figure 3.21 (Algorithme 2 ligne 17). Dans le cas de l'allocation a_{11}, la partition

P_6 peut seulement être ordonnancée à la fin du second ou du quatrième intervalle comme nous le détaillons Figure 3.22, autrement l'ordonnancement ne respecte pas l'équation 3.5. Nous décalons la dernière partition ordonnancée : P_5 est ici décalée d'une unité de temps dans le premier intervalle (Algorithme 2 ligne 17). P_6 ordonnancée dans le second intervalle permet de valider D_{ch_3} (Algorithme 3).

Pour finir, les allocations totales valides sont a_{11} et a_{12} (Algorithme 2 ligne 19).

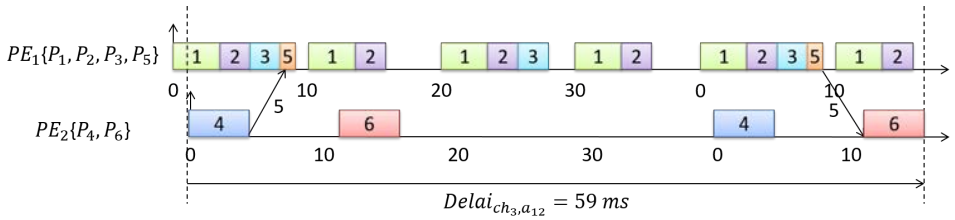


FIGURE 3.21 Ordonnancement valide dans l'allocation a_{12}

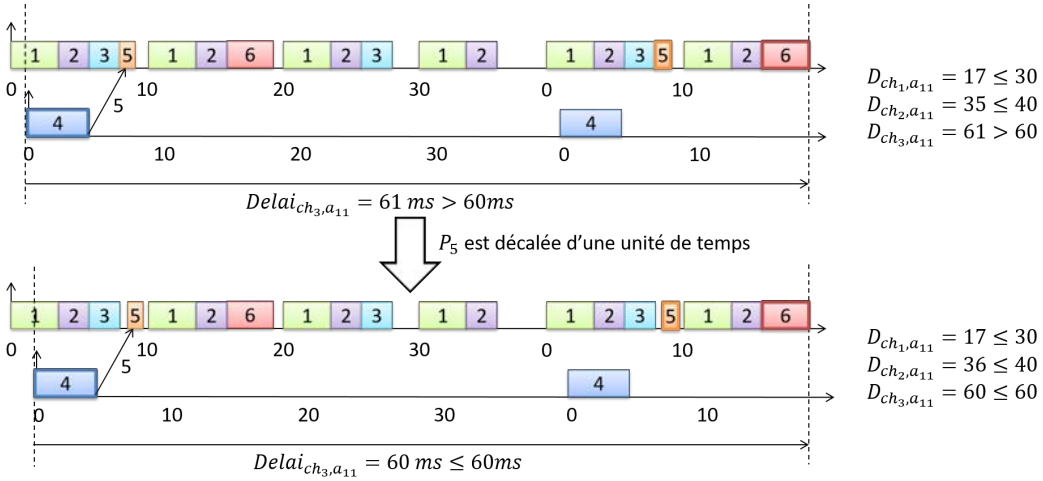


FIGURE 3.22 Modification de l'ordonnancement dans l'allocation a_{11}

3.2.2 Étude de complexité

La complexité d'un tel problème provient de la recherche des allocations des partitions et de leur ordonnancement devant respecter des exigences de bout-en-bout. En effet, avec certaines allocations, aucun ordonnancement ne pourra satisfaire les contraintes de bout-en-bout à cause des communications distantes trop importantes par rapport aux contraintes. Dans certains cas, il faut tester tous les ordonnancements même si aucun ne va fonctionner, ce qui peut revenir à évaluer un nombre important d'ordonnancements.

La première complexité provient du nombre d'allocations possibles : ceci revient à compter le nombre de façons de partitionner un ensemble de n éléments, i.e. nos partitions, en k sous-ensembles non vides, i.e. nos processeurs, défini par le nombre de Stirling de deuxième espèce :

$$\forall k > 0, S(n,k) = S(n-1,k-1) + kS(n-1,k) \tag{3.25}$$

$$\text{avec les conditions initiales} \begin{cases} S(0,0) = 1 \\ S(n,0) = S(0,n) = 0 \quad \forall n > 0, \end{cases} \tag{3.26}$$

Le nombre de Stirling de deuxième espèce est donné par la formule explicite suivante :

$$S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n \tag{3.27}$$

Dans cette thèse, nous recherchons toutes les allocations pour un nombre maximum donné de processeurs. Au pire des cas, le nombre maximal de processeurs est n, i.e. le nombre de partitions à distribuer. Ainsi, le nombre maximum d'allocations potentielles est :

$$\sum_{k=1}^{n} S(n,k) = B_n \tag{3.28}$$

Le résultat donné est le n-ième nombre de Bell.

La complexité pour trouver le nombre d'allocations est de $\mathcal{O}(\frac{n^n}{n!})$, n étant le nombre de partitions à distribuer.

La deuxième complexité provient de la recherche d'un ordonnancement des différentes partitions dans un processeur. Korst et al. [22] ont montré que le problème d'ordonnancement de tâches périodiques non-préemptées est NP-complet dans le sens fort dans le cas d'un unique processeur, mais que ce problème se résout en un temps polynomial si les périodes et les temps d'exécution sont divisibles.

Dans le cadre de notre algorithme, nous cherchons le nombre d'ordonnancements qu'il est possible de générer dans un processeur.

Prenons dans un premier temps un processeur où toutes les partitions qui lui sont assignées ont la même période. Soit p le nombre de partitions. Le nombre d'ordonnancements possible est $p!$ selon la définition suivante :

Définition 1. *Dénombrement des permutations*

Si X est un ensemble fini de cardinal n, alors l'ensemble des permutations de X est fini, de cardinal $n\,!$

Soit un processeur qui contient p partitions avec des fréquences d'exécution différentes. Soit n_{f_i} le nombre de partitions de fréquence f_i. Soit $N = n_{f_1} + n_{f_2} + \cdots + n_{f_k}$ le nombre total de partitions allouées sur un processeur avec $f_1 > f_2 > \cdots > f_k$. Nos périodes sont harmoniques de 2^i, ce qui nous donne $f_i = \frac{1}{2^{i-1}} f_1$ avec $i \in \mathbb{N}^*$.

Ordonnançons dans un premier temps les partitions de fréquence f_1. Il y a alors $n_{f_1}!$ façons possibles de les ordonnancer.

Ordonnançons dans un second temps les partitions moins fréquentes de f_2 à f_i.

Définition 2. *Soit $N_i = n_{f_1} + \cdots + n_{f_i}$ le nombre total de partitions de fréquences f_1 à f_i. Le nombre d'ordonnancements possibles O_i est :*

$$\begin{cases} O_i = \sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j})!}{(N_i+\alpha_{i,j}-1)} & \text{pour } i \geq 2 \\ \text{avec } 3 - i \leq \alpha_{i,j} \leq 1 \end{cases}$$

sachant que le nombre possible d'ordonnancements $O_1 = N_1!$.

Démonstration. — Si $i = 2$,

La première partition de fréquence f_2 est ordonnancée seulement dans le premier intervalle, ce qui revient à tester $(n_{f_1} + 1)!$ ordonnancements possibles. En effet, si l'ordre des partitions de fréquence f_1 ne convient pas, il faut à nouveau modifier leur ordonnancement. Les partitions suivantes de fréquence f_2 peuvent être ordonnancées soit dans le premier intervalle, soit dans le second. Soit $N_2 = n_{f_1} + n_{f_2}$. Le nombre d'ordonnancements possibles pour la deuxième partition de fréquence f_2 sera de :

$$(n_{f_1} + 2)! + (n_{f_1} + 1)!$$

Pour la troisième :

$$(n_{f_1} + 3)! + (n_{f_1} + 2)!$$

$$\vdots$$

Pour la n_{f_2}-ième partition de fréquence f_2 :

$$(n_{f_1} + n_{f_2})! + (n_{f_1} + n_{f_2} - 1)!$$

$$= (N_2 - 1)!(N_2 + 1) = \frac{(N_2 + 1)!}{N_2} \text{ avec } \alpha_{2,1} = 1$$

— Soit $i \geq 2$.

Supposons que $\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j})!}{(N_i+\alpha_{i,j}-1)}$ et montrons que $\sum_{j=1}^{2^{i-1}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)}$.

La première partition de fréquence f_{i+1} est ordonnancée seulement dans les i premiers intervalles, ce qui revient à tester $\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+1)!}{N_i+(\alpha_{i,j})}$ ordonnancements possibles. En effet, si l'ordre des partitions de fréquences inférieures à f_{i+1} ne convient pas, nous devons à nouveau modifier leur ordonnancement. Les partitions suivantes de fréquence f_{i+1} peuvent être ordonnancées dans les 2^i intervalles. Le nombre d'ordonnancements possibles pour la deuxième partition de fréquence f_{i+1} sera de :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+1)!}{(N_i+\alpha_{i,j})} + \sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+2)!}{(N_i+\alpha_{i,j}+1)}$$

Pour la troisième :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+2)!}{(N_i+\alpha_{i,j}+1)} + \sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+3)!}{(N_i+\alpha_{i,j}+2)}$$

$$\vdots$$

Pour la $n_{f_{i+1}}$-ième partition de fréquence f_{i+1} :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+n_{f_{i+1}}-1)!}{(N_i+\alpha_{i,j}+n_{f_{i+1}}-2)} + \sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j}+n_{f_{i+1}})!}{(N_i+\alpha_{i,j}+n_{f_{i+1}}-1)}$$

$$= \sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j}-1)!}{(N_{i+1}+\alpha_{i,j}-2)} + \sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j})!}{(N_{i+1}+\alpha_{i,j}-1)}$$

Le premier terme de cette somme donne :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j}-1)!}{(N_{i+1}+\alpha_{i,j}-2)}$$

$$= \sum_{k=2^{i-2}+1}^{2^{i-1}} \frac{(N_{i+1}+\alpha_{i,k-2^{i-2}}-1)!}{(N_{i+1}+\alpha_{i,k-2^{i-2}}-2)} \text{ avec } k = j+2^{i-2}$$

$$= \sum_{j=2^{i-2}+1}^{2^{i-1}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)} \text{ avec } \alpha_{i+1,j} = \alpha_{i,k-2^{i-2}}-1$$

Le second terme de cette somme donne :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j})!}{(N_{i+1}+\alpha_{i,j}-1)} = \sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)} \text{ avec } \alpha_{i+1,j} = \alpha_{i,j}$$

Ainsi :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j}-1)!}{(N_{i+1}+\alpha_{i,j}-2)} + \sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i,j})!}{(N_{i+1}+\alpha_{i,j}-1)}$$

$$= \sum_{j=2^{i-2}+1}^{2^{i-1}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)} + \sum_{j=1}^{2^{i-2}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)}$$

$$= \sum_{j=1}^{2^{i-1}} \frac{(N_{i+1}+\alpha_{i+1,j})!}{(N_{i+1}+\alpha_{i+1,j}-1)} \text{ avec } \begin{cases} \alpha_{i+1,j} = \alpha_{i,j} & \text{si } 1 \leq j \leq 2^{i-2} \\ \alpha_{i+1,j} = \alpha_{i,j-2^{i-2}} - 1 & \text{si } 2^{i-2}+1 \leq j \leq 2^{i-1} \end{cases}$$

Pour $i \geq 2$, $3-i \leq \alpha_{i,j} \leq 1$. En effet, le plus petit $\alpha_{i,j}$ est donné par $j = 2^{i-1}$:

$$\alpha_{i,2^{i-1}} = \alpha_{i-1,2^{i-2}} - 1 = \cdots = \alpha_{2,2} - i + 2 = 1 + 2 - i = 3 - i$$

.

On a montré par récurrence que, pour tout entier naturel $i \geq 2$, $O_i = \sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j})!}{N_i+(\alpha_{i,j}-1)}$

□

Ce calcul peut aussi s'écrire de la forme :

$$\sum_{j=1}^{2^{i-2}} \frac{(N_i+\alpha_{i,j})!}{(N_i+\alpha_{i,j}-1)} = (N_i+1-i)! \times (N_i^{i-1} + \sum_{j=1}^{i-2} \beta_j \times N_i^{i-2-j})$$

La complexité pour trouver un ordonnancement dans un processeur est alors de $\mathcal{O}(N! \times N^{k-1})$, N étant le nombre de partitions allouées sur le processeur et k le nombre de fréquences différentes.

Nous sur-approximons le nombre de partitions dans un processeur par le nombre total de partitions à distribuer :

$$N_k = n = N.$$

La complexité totale de cette algorithme est alors de :

$$\mathcal{O}(\frac{N^N}{N!} \times N! \times N^{k-1}) = \mathcal{O}(N^{N+k-1}).$$

3.2.3 Etude de cas

Le système à distribuer s'appelle le « VMSA », ou « Système de surveillance du véhicule ». Le but de ce système est d'une part d'alerter le pilote dès qu'un paramètre commence à dépasser certaines limites avant d'atteindre un seuil d'alarme, et d'autre part de fournir les valeurs de différents paramètres dès que le pilote les demande. Nous le décrivons dans le premier point et le distribuons à l'aide de notre approche complètement intégrée sur celui-ci dans un second point.

3.2.3.1 Présentation de l'étude de cas

Le système VMSA peut être décomposé en deux parties : une partie physique correspondant aux matériels utilisés et une partie logique avec ses fonctions avioniques. Nous décrivons dans les prochains paragraphes les différents équipements utilisés ainsi que le rôle des partitions présentes dans chacun de ses équipements. Nous détaillons ensuite les contraintes de bout-en-bout que ce système doit respecter. Une description plus détaillée de l'application « VMSA » est donnée en Annexe A.

La partie physique est décrite Figure 3.23. Ce système avionique est composé de deux bi-processeurs appelés « *Aircraft Management Computer* (AMC) », quatre écrans appelés « *Multi-Function Display* (MFD) » et des E/S locales [106, 107].

Chaque processeur d'un AMC contient sept partitions, notées P_1 à P_7. Les quatre premières partitions, de P_1 à P_4, s'occupent de l'enregistrement de l'état du véhicule et du pilotage. Des comparaisons d'états/de commandes sont effectuées fréquemment entre les deux processeurs d'un même AMC. Les trois dernières partitions, de P_5 à P_7, gèrent et surveillent les équipements où elles sont allouées : ces partitions sont identiques dans tous les processeurs des AMCs. Ces processeurs n'ont plus assez de place, que ce soit au niveau temporel pour contenir de nouvelles partitions, i.e. 86% d'utilisation comme le montre la Figure 3.23 au niveau de l'AMC 1 processeur A, ou bien au niveau physique pour connecter de nouvelles E/S. Une solution est de distribuer ces vingt-huit partitions, i.e. les 7 partitions implémentées sur quatre processeurs, sur un plus grand nombre de processeurs qui peuvent être moins puissants.

Concentrons-nous sur un seul processeur d'un AMC. Les résultats obtenus pour ce dernier peuvent être reproduits sur les autres processeurs. Nous devons prendre en considération que les partitions P_5 à P_7 doivent être recopiées sur tous les nouveaux processeurs, tandis que les partitions P_1 à P_4 devront être assignées dans un seul processeur, et leurs réplicas, au nombre de trois, devront être alloués sur d'autres processeurs pour des raisons de sûreté de fonctionnement. Ces quatre partitions sont

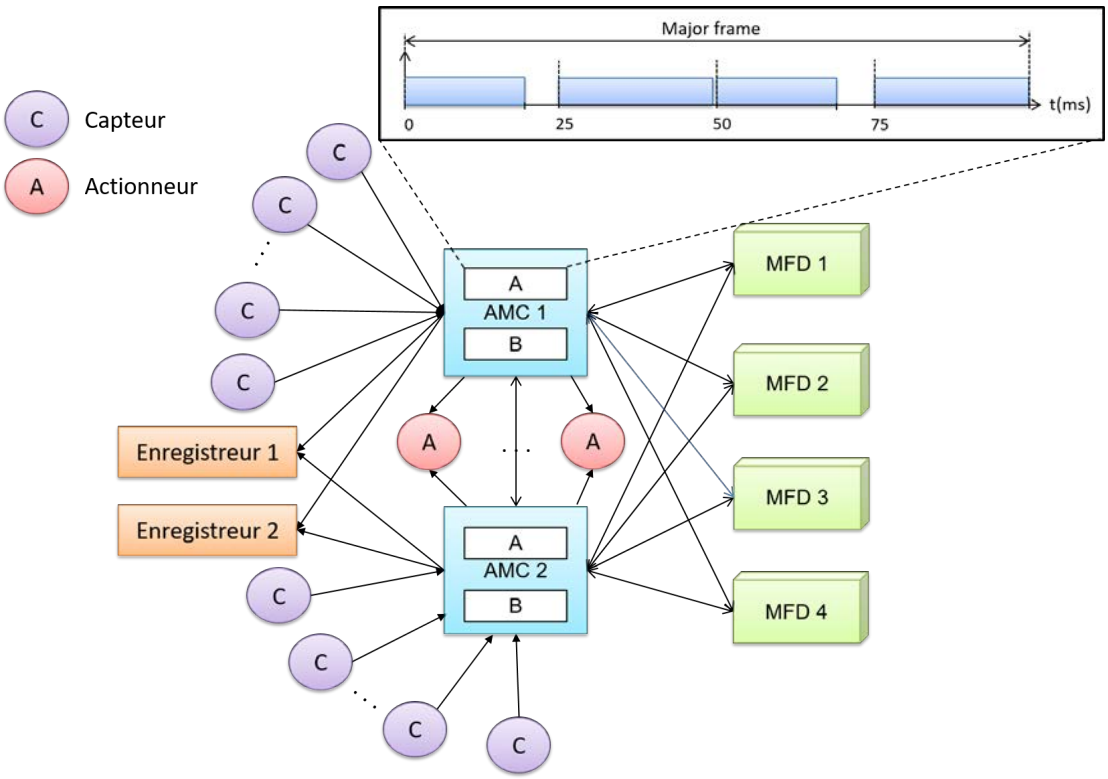


FIGURE 3.23 Architecture physique et utilisation des AMCs dans le système VMSA

notées $P_{i,XY}$, avec X le numéro de l'AMC (1 ou 2) et Y le processeur sur lequel la partition est allouée (A ou B). Nous considérons quatre types de processeurs plus ou moins puissants dans la Table 3.2. Les caractéristiques de ces différentes partitions (C_i et T_i) sont données Table 3.2 en fonction du processeur où elles sont allouées, le processeur $Proc_1$ étant le processeur utilisé actuellement.

TABLE 3.2 Spécifications temps-réel des partitions du système VMSA selon le processeur utilisé

Partitions	WCET				Période
	$Proc_1$	$Proc_2$	$Proc_3$	$Proc_4$	
P_1	10	12	13	14.5	25
P_2	10	10	11	11.5	50
P_3	6	6.5	7	8	100
P_4	6	6.5	7	8	50
P_5	5	5.5	6	6.5	100
P_6	2	2	2.5	2.5	100
P_7	1	1	1	1.5	100

Différentes informations sont remontées au pilote grâce aux MFD. Chaque MFD contient cinq partitions. Parmi ces partitions, une seule sert à traiter et comparer les entrées pour ensuite les afficher, les autres sont seulement présentes pour vérifier le bon fonctionnement de l'équipement. Cet équipement n'a pas besoin d'être modifié : nous considérons dans la suite de cette thèse seulement la partition d'affichage notée $P_{ecran} = (39, 50)$.

Le système VMSA récupère et envoie des données à une trentaine d'E/S. La communication est locale entre un E/S et un AMC, i.e. les E/S sont directement connectées aux différents AMC. Nous considérons que le temps d'exécution d'une E/S est nul. En effet, un capteur acquiert une donnée selon une période prédéfinie, tandis qu'un actionneur ou un enregistreur respectivement agit ou écrit en fonction des données reçues quasi instantanément. Les partitions associées aux deux enregistreurs de vol, notées $P_{enregistreur1}$ et $P_{enregistreur2}$, s'exécutent tous les 62,5 ms, selon les recommandations fournis par le *National Transportation Safety Board* [108]. Une donnée est acquise par un capteur selon une période stricte comprise entre 7 ms et 1,6 s.

Comme une symétrie existe entre tous les processeurs des AMC, nous nous concentrons seulement sur la distribution d'un seul d'entre eux, les résultats pouvant être reproduits sur les autres. Toutes les possibilités de distribution d'un processeur d'un AMC sont représentées Figure 3.25 : de l'allocation actuelle illustrée allocation 1 jusqu'à des ensembles de 2, 3 ou 4 processeurs. Les AMC, les MFD et les E/S sont connectés

directement.

Six contraintes temporelles doivent être prises en compte. Celles-ci sont illustrées Figure 3.24. La première, représentée Figure 3.24 partie a, consiste à récupérer toutes les informations acquises par la partition P_1 afin que P_2 puisse les traiter en moins de 50 ms. La deuxième, illustrée Figure 3.24 partie b, vérifie que le délai entre l'acquisition de la donnée d'un capteur et l'enregistrement de l'état de santé générale du véhicule donné par la partition P_3 soit inférieur à 500 ms [23]. La troisième, représentée Figure 3.24 partie c, consiste à afficher les données des capteurs en moins de 400 ms [3]. Les trois dernières chaînes illustrées Figure 3.24 parties d, e et f, enregistrent l'état des différents équipements en moins de 500 ms.

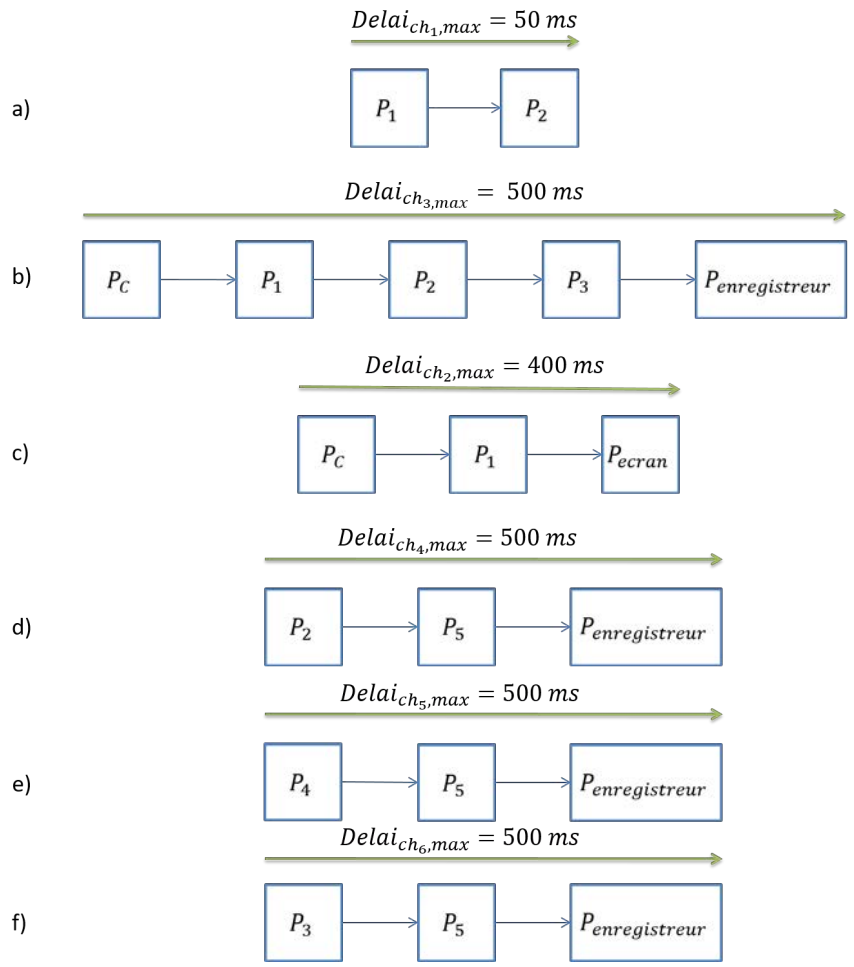


FIGURE 3.24 Les chaînes de communication dans le système VMSA

Toutes les distributions proposées Figure 3.25 doivent vérifier deux critères : toutes

les partitions sont ordonnançables dans les processeurs où elles sont assignées et toutes les contraintes de bout-en-bout sont satisfaites.

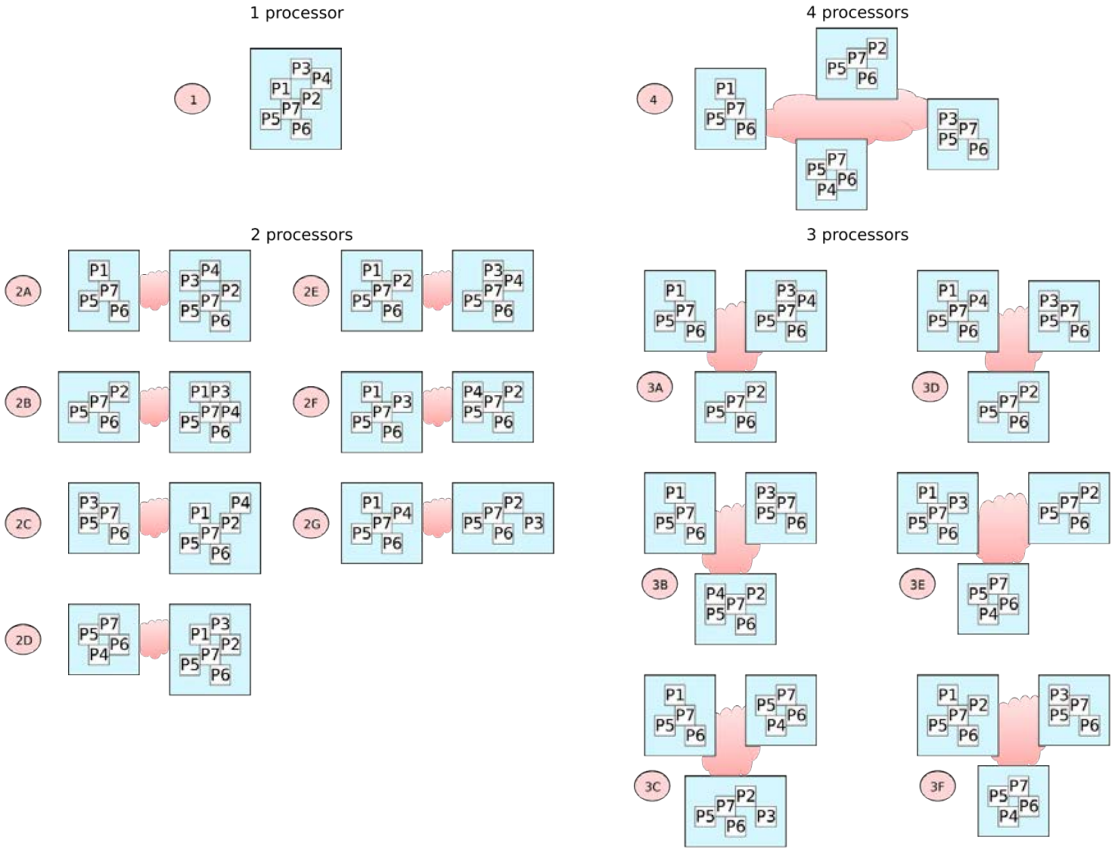


FIGURE 3.25 Allocations possibles des partitions d'un processeur d'un AMC sur au plus 4 processeurs

Dans la partie qui suit, nous recherchons l'ensemble des allocations valides sur un ensemble de processeurs identiques spécifiés Table 3.2, le processeur $Proc_1$ étant le processeur actuellement utilisé. Les communications distantes sont prises nulles, i.e. le WCTT est égal à 0 ms, afin de déterminer toutes les allocations valides sans prédéfinir un réseau de communication entre les processeurs. Une étude sur les pires latences de bout-en-bout peut être effectuée à ce niveau à condition de définir les équipements et le trajet des communications sur ces derniers. Les pires temps d'exécution des partitions sur différents types de processeurs sont détaillés Table 3.2.

TABLE 3.3 Nombre d'allocations valides selon le type de processeur avec l'approche complètement intégrée

Nombre de processeurs	$Proc_1$	$Proc_2$	$Proc_3$	$Proc_4$
1	1	1	0	0
2	3	3	2	0
3	1	1	1	0
4	0	0	0	0
Temps d'analyse (ms)	10	6	4	<1

3.2.3.2 Temps d'analyse et recherche d'ordonnancement

Dans cette partie, nous évaluons les performances de notre approche exhaustive sur le système VMSA. Les résultats, détaillés dans la Table 3.3, donnent le nombre d'allocations valides en fonction du nombre de processeurs utilisés. Le temps d'analyse pour chaque processeur est spécifié sur la dernière ligne de la Table 3.3.

Lorsque les WCET augmentent, nous constatons dans le Table 3.3 qu'aucune allocation n'est valide sur un unique processeur avec les processeurs $Proc_3$ et $Proc_4$. Si nous essayons d'ordonnancer les quatre premières partitions Figure 3.26, P_1 est allouée et ordonnancée en première position sur la MAF, puis la MAF est étendue pour ordonnancer P_2 dans le premier intervalle. Avec le processeur $Proc_4$, il est impossible d'ordonnancer P_2 durant 11,5 ms puisqu'il ne reste que 10,5 ms dans chaque intervalle de la MAF. Continuons la recherche d'ordonnancement avec le processeur $Proc_3$. P_4 est ensuite allouée : comme le temps restant dans le premier intervalle est inférieur à 1, P_4 est alors ordonnancée dans le second intervalle où il reste 12 ms à une partition pour s'exécuter. Il faut à nouveau étendre la MAF pour ordonnancer P_3 dans les deux premiers intervalles. Étant donné que le temps restant dans chaque intervalle vaut 1 ou 5, P_3 s'exécutant en 7 ms ne peut pas être ajoutée. Le temps d'exécution des partitions devenant plus important a un impact significatif dès lors que les partitions doivent être allouées sur un petit nombre de processeurs.

Nous remarquons dans la Table 3.3 qu'aucune allocation valide n'existe lorsque les quatre processeurs doivent être utilisés, tandis qu'il existe une unique allocation valide lorsque trois processeurs sont utilisés. Si nous reprenons les contraintes de délai de bout-en-bout, ch_1, contenant les partitions P_1 et P_2, est la chaîne la plus contrainte. Si nous essayons d'assigner ces deux partitions sur deux processeurs distincts comme nous l'illustrons Figure 3.27, aucune allocation est possible. En effet, le délai subit la période de P_2 qui vaut $D_{ch_1,max}$. Si nous ajoutons les temps d'exécution des partitions, nous dépassons la contrainte de bout-en-bout spécifiée. Assigner les partitions appartenant

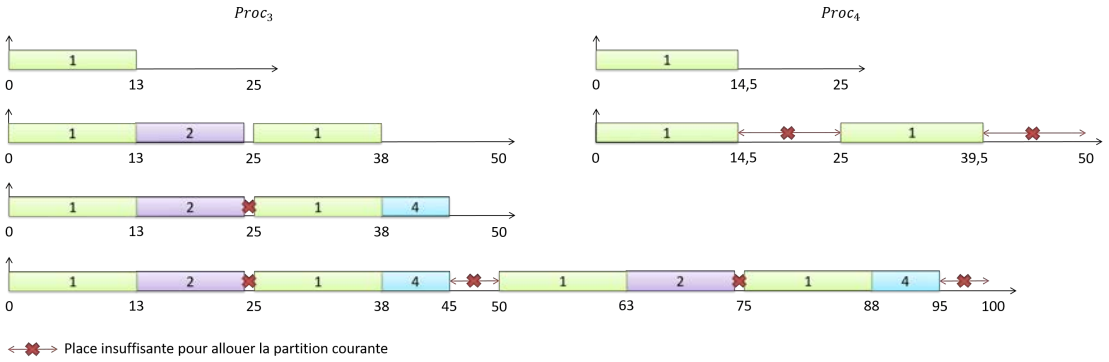


FIGURE 3.26 Temps manquant pour allouer sur un seul processeur $Proc_3$ ou $Proc_4$ toutes les partitions du VMSA

aux chaînes les plus contraintes dans un même processeur ou les considérer au plus tôt dans l'algorithme d'allocation nous permettrait de guider notre recherche en élaguant rapidement les allocations partielles non valides.

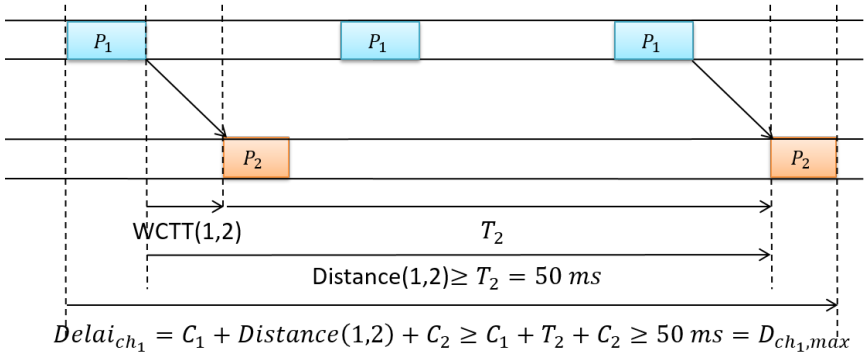


FIGURE 3.27 P_1 et P_2 sur des processeurs différents

Lorsque trois processeurs sont utilisés, nous remarquons qu'une seule allocation fonctionne dans tous les cas sauf avec le processeur $Proc_4$. Comme nous l'avons précisé dans le paragraphe ci-dessus, P_1 et P_2 doivent être sur le même processeur. C'est possible avec les processeurs $Proc_1$, $Proc_2$ et $Proc_3$. Or, avec le processeur $Proc_4$, il est impossible d'allouer ces deux partitions dans le même processeur car nous ne pouvons pas respecter les contraintes d'ordonnancement données par le standard ARINC 653 [1]. C'est aussi pour cette raison qu'il n'y a pas d'allocation valide avec deux processeurs $Proc_4$.

Lorsque deux processeurs sont utilisés, seul le cas avec le processeur $Proc_3$ admet deux allocations valides au lieu de trois. L'allocation valide manquante est l'allocation où les partitions P_1, P_2, P_4, P_5, P_6 et P_7 sont sur le premier processeur, tandis que P_3,

P_5, P_6 et P_7 sont sur le deuxième. Reprenons l'ordonnancement de ces six partitions sur le premier processeur : P_1 est allouée et ordonnancée en première position sur la MAF, puis la MAF est étendue pour ordonnancer P_2 dans le premier intervalle. P_4 est ensuite allouée : comme le temps restant dans le premier intervalle est inférieur à 1, P_4 est alors ordonnancée dans le deuxième intervalle où il reste 12 ms à une partition pour s'exécuter. Il faut à nouveau étendre la MAF pour ordonnancer P_5 dans les deux premiers intervalles. Étant donné que le temps restant dans chaque intervalle vaut 1 ou 5, P_5 s'exécutant en 6 ms ne peut être ajoutée ni dans le premier ni dans le deuxième intervalle.

Dans cette étude de cas, en considérant une latence de communication nulle, seule la chaîne la plus contrainte ch_1 impose que deux partitions, P_1 et P_2, soient allouées sur le même processeur afin de satisfaire les contraintes de bout-en-bout. Réduire la puissance des processeurs, et par conséquent augmenter le pire temps d'exécution de chaque partition, réduit le nombre d'allocations valides sur un petit nombre de processeurs puisqu'il devient impossible d'ordonnancer un certain nombre de partitions sur un même processeur, e.g. 6 partitions spécifiques sur le processeur $Proc_3$.

3.2.4 Passage à l'échelle

Une application avionique est généralement composée d'une dizaine de partitions, sans considérer ses réplicas. Par exemple, l'application ROSACE [109], pour « *Research Open-Source Avionics and Control Engineering* », est composée de 10 partitions de fréquences harmoniques entre 10 et 100 Hertz. Le FADEC, pour « *Full Authority Digital Engine Control* », gère le système moteur dans les avions. Il est composé de 9 partitions [110]. L'application de maintenance prédictive, ou « *Health Monitoring* », permet d'anticiper la maintenance des moteurs d'avion. Il est composé de 14 partitions [110].

Dans une architecture IMA, différentes applications, critiques ou non, peuvent être allouées sur un même processeur. L'algorithme de distribution doit être capable d'allouer et d'ordonnancer les partitions de ces différentes applications. Nous étudions dans cette partie les performances et les limites de notre algorithme intégré en faisant varier différents paramètres correspondant à la montée en charge des systèmes :

— le nombre de partitions en commençant par 10 partitions,
— toutes les partitions de période $T = 25ms$ voient leur WCET C varié entre 5 ms à 25 ms.

L'ensemble $\mathcal{C}h$ est composé de N chaînes ch de deux partitions communicantes : $ch_i = \{P_{2i-1}, P_{2i}\}$. Le délai de bout-en-bout $D_{ch_i,max}$ est le même pour toutes les chaînes : il prend pour valeur soit 20 ms (un délai très strict), soit 40 ms.

Ces partitions sont allouées sur un ensemble $\mathcal{E}$ contenant au minimum 10 processeurs

PE.

Un premier test consiste à allouer 10 partitions sur au plus 10 processeurs, en prenant en compte différentes valeurs de WCET. Ici, $D_{ch_i,max}$ est égal à 20 ms. Nous remarquons dans les Tables 3.4 et 3.5 que lorsque le WCET augmente, le temps d'analyse et le nombre de MAF générées décroissent. Un élagage rapide des branches est effectué à cause de partitions non ordonnançables ou de la non satisfaction des contraintes de délai de bout-en-bout dans les allocations partielles lorsque le WCET est grand.

Un second test consiste à relâcher la contrainte de délai de bout-en-bout $D_{ch_i,max}$, toujours avec 10 partitions. Si nous comparons les Tables 3.4 et 3.5 correspondant à l'analyse avec $D_{ch_i,max}$=20 ms et les Tables 3.6 et 3.7 correspondant à l'analyse avec $D_{ch_i,max}$=40 ms, nous constatons une explosion du nombre de MAF générées et du temps d'analyse lorsque le WCET est bas. Plus de branches sont parcourues car plus d'ordonnancements existent et peuvent être valides.

Un troisième test consiste à modifier le nombre de partitions à distribuer. Le WCET de toutes les partitions est égal à 5 ms et la contrainte de délai de bout-en-bout pour toutes les chaînes vaut 20 ms. D'une part, nous pouvons établir grâce à la Table 3.8 que le temps d'analyse atteint un seuil maximal à partir d'un nombre de processeurs, e.g. 6, 7 et 8, ... processeurs pour respectivement 10, 12, 14, ... partitions. Si nous corrélons ce résultat avec la Table 3.9, nous constatons que le nombre de MAF générées atteint une limite. D'autre part, nous remarquons que le temps d'analyse est multiplié par 10 lorsque 2 nouvelles partitions sont ajoutées : comme le nombre de possibilités d'ordonnancements augmente, un plus grand nombre de branches dans notre arbre de recherche d'allocations est parcouru. Pour finir, nous constatons dans la Table 3.8 que l'algorithme de recherche ne termine pas à partir de 20 partitions sur 6 processeurs. L'algorithme a généré en un mois plus d'un milliard d'ordonnancements (cf. Table 3.9), mais ce nombre reste ici insuffisant pour trouver dans ces cas, i.e. à partir de 6 processeurs, toutes les allocations valides.

Cet algorithme donne des résultats satisfaisants sur de petites architectures telles que le système VMSA ainsi que sur des architectures comportant un plus grand nombre de partitions. Néanmoins, dès que l'architecture grossit, avec des partitions où le WCET est faible par rapport à la période ou lorsque les délais sont peu contraints, une explosion du temps d'analyse apparaît et devient limitant, i.e. l'analyse ne finit pas au bout d'un mois. Ces limites proviennent dans ce cas du nombre d'ordonnancements qui augmente et accroît par conséquent la taille de l'arbre de recherche. Une heuristique est nécessaire pour réduire l'espace de recherche des ordonnancements.

TABLE 3.4 Temps d'analyse (s) pour distribuer 10 partitions lorsque $D_{ch_i,max} = 20ms$

Nombre de PEs	WCET (ms)				
	5	10	15	20	25
2	0.04	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
3	0.06	$1 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
4	0.08	$1 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
5 et plus	0.09	$2 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$

TABLE 3.5 Nombre de MAFs générées pour distribuer 10 partitions lorsque $D_{ch_i,max} = 20ms$

Nombre de PEs	WCET (ms)				
	5	10	15	20	25
2	8752	7	2	2	1
3	11103	11	2	2	1
4	16445	15	2	2	1
5	16946	18	2	2	1
6 et plus	16947	19	2	2	1

TABLE 3.6 Temps d'analyse (s) pour distribuer 10 partitions lorsque $D_{ch_i,max} = 40ms$

Nombre de PEs	WCET (ms)				
	5	10	15	20	25
2	0.06	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
3	235	$2 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
4	4956	$6 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
5	12866	$8 * 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
6	15463	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
7	15560	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
8	22520	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
9	21625	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
10	21054	$1 * 10^{-3}$	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$

TABLE 3.7 Nombre de MAFs générées pour distribuer 10 partitions lorsque $D_{ch_i,max} = 40ms$

Nombre de PEs	WCET (ms)				
	5	10	15	20	25
2	1415	10	2	2	2
3	450923	24	2	2	2
4	2415999	52	2	2	2
5	2908636	62	2	2	2
6	2967705	93	2	2	2
7	1456829	93	2	2	2
8	1457616	93	2	2	2
9	1457662	93	2	2	2
10	1457663	93	2	2	2

TABLE 3.8 Temps d'analyse (s) lorsque $C_i = 5ms$ et $D_{ch_i,max} = 20ms$

Nombre de PEs	Nombre de partitions					
	10	12	14	16	18	20
2	0.04	0.04	0.04	0.05	0.05	0.06
3	0.06	0.08	5.8	6.3	6.8	7.0
4	0.08	0.7	7.6	9.9	841.3	869.4
5	0.09	0.8	12.6	155.8	2061.6	4631.9
6	0.09	0.8	12.9	177.4	4877.0	?
7	0.09	0.8	12.9	176.2	4189.8	?
8	0.09	0.8	12.9	175.9	4038,4	?
9	0.09	0.8	12.9	172.8	3967,7	?
10	0.09	0.8	12.9	171.0	4068,0	?

TABLE 3.9 Nombre de MAFs générées lorsque $C_i = 5ms$ et$D_{ch_i,max} = 20ms$

Nombre de PEs	Nombre de partitions					
	10	12	14	16	18	20
2	8752	8752	8752	8752	8752	8752
3	11103	14424	822684	822684	822684	822684
4	16445	108876	913731	963771	77703186	77703186
5	16946	129861	1456356	10214991	82803486	83589411
6	16947	130864	1518394	12491602	140640952	?
7	16947	130865	1520158	12643182	148237480	?
8	16947	130865	1520159	12646020	148561189	?
9	16947	130865	1520159	12646021	148565468	?
10	16947	130865	1520159	12646021	148565469	?

3.3 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la distribution de partitions avioniques sur un plus grand nombre de processeurs. Un algorithme appelé « approche intégrée » a été créé, basé sur le parcours d'un arbre d'allocations valides. Chaque nouveau nœud correspond à l'allocation d'une partition dans un processeur : ce nœud est dit valide s'il existe un ordonnancement respectant le standard ARINC 653 [1], si les contraintes de communication de bout-en-bout sont respectées et s'il existe au moins un nœud enfant, i.e. une allocation totale valide représentée par une feuille de l'arbre.

Cet algorithme recherche en profondeur tous les ordonnancements valides d'une allocation partielle tant qu'un des nœuds enfants n'est pas valide. Si aucun ordonnancement respectant les contraintes de communication de bout-en-bout dans le nœud courant ou dans les nœuds enfants est trouvé, la branche est alors élaguée. Cette approche a été testée sur une étude de cas hélicoptère : les résultats obtenus permettent de sélectionner un nombre de processeurs ainsi qu'une distribution des partitions associées au système VMSA avec un ordonnancement satisfaisant les contraintes de communication de bout-en-bout. Cette approche permet de supprimer les allocations impossibles, par exemple deux partitions doivent obligatoirement être assignées sur le même processeur. Nous avons ensuite passé à l'échelle cette approche. Elle montre ses limites dès lors qu'une vingtaine de partitions doivent être allouées sur plus de six processeurs : générer la MAF sur chaque processeur devient coûteux en temps. Par ailleurs, dès que les contraintes de communication de bout-en-bout deviennent très relâchées, le nombre d'ordonnancements possibles augmentent, créant un arbre d'allocations valides très large et par conséquent ne permettant pas de trouver toutes les allocations valides en un temps raisonnable. Pour parer à ces limitations, nous proposons dans le Chapitre 4 une heuristique gloutonne.

Chapitre 4

Recherche heuristique d'allocations valides

Dans l'approche intégrée présentée Chapitre 3, une limite a été mise en avant : une recherche des allocations valides qui ne finit pas dès lors que 20 partitions doivent être allouées sur plus de 6 processeurs. Nous mettons en avant les limites de ce premier algorithme en 4.1 afin de proposer une heuristique basée sur des choix d'allocations partielles selon les contraintes de bout-en-bout. Ces choix sont basés sur la méthode exacte de « séparation et évaluation » d'un parcours d'arbre. Cette méthode est détaillée en 4.2. Nous développons une heuristique qui n'a pas besoin de sauvegarder les nœuds déjà parcourus valides grâce à la sélection de l'ordonnancement offrant le plus de variabilité pour les nœuds enfants en 4.3. Dans le cadre de cette heuristique, nous montrons en 4.4 qu'il est plus intéressant d'allouer les partitions en fonction des contraintes des chaînes plutôt que de les traiter par période croissante. Notre étude de complexité en 4.6 montre un gain important d'un facteur de $\frac{1}{N!}$ par rapport à notre approche intégrée précédente, N étant le nombre de partitions à distribuer. Nous comparons en 4.7 et en 4.8 les deux algorithmes définis dans ce chapitre et le chapitre précédent sur l'exemple d'étude de cas présentée en 3.2.3 et sur leur passage à l'échelle.

4.1 Limites de l'approche exhaustive

Dans l'approche intégrée ci-dessus, nous vérifions qu'une allocation partielle satisfait les contraintes d'ordonnancement et les délais de bout-en-bout. Reprenons rapidement cette approche avec l'exemple de la Table 3.1. Un arbre contenant toutes les allocations possibles est parcouru en profondeur. Nous représentons une partie de cet arbre Figure

4.1. La racine de notre arbre est une allocation vide. Sur le premier niveau de l'arbre, P_1 est allouée dans un premier processeur (cela peut être n'importe quel processeur puisqu'ils sont tous identiques). Sur le niveau suivant, la partition suivante P_2 peut-être allouée sur le même processeur que P_1 (branche de gauche) ou sur un autre (branche de droite). Dans chaque nœud, un ordonnancement faisable vérifiant les contraintes des délais de chaque chaîne doit être trouvé. Lorsqu'une partition est ajoutée dans un ordonnancement existant, toutes les positions que son slot associé peut prendre sont potentiellement testées.

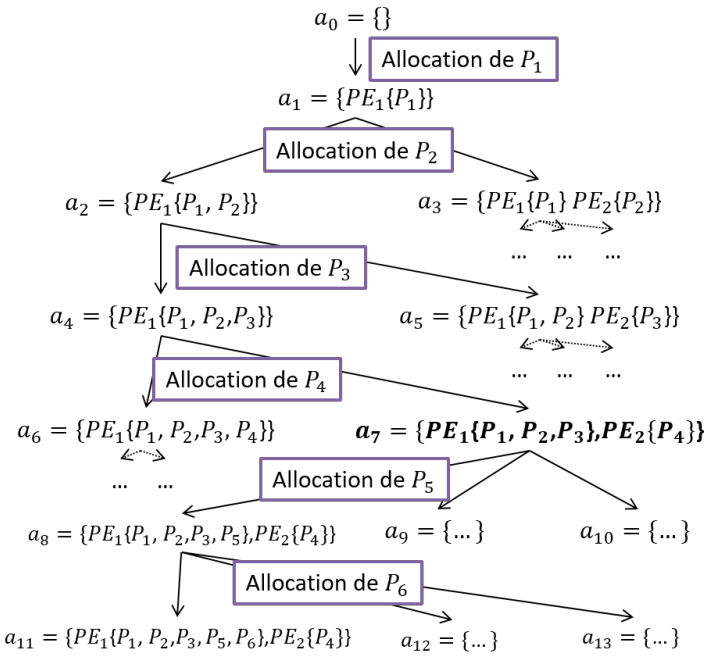


FIGURE 4.1 Parcours partiels de l'arbre avec l'exemple Table 3.1

Considérons la branche où toutes les partitions sont allouées dans le premier processeur et imaginons qu'aucune allocation valide existe. Si nous sommes obligés de descendre jusqu'à l'allocation de P_6, toutes les positions possibles de chaque partition doivent être testées. La Figure 4.2 illustre l'ordonnancement d'une partition $P_{j,j=1..6}$ laissant le plus de place pour l'ordonnancement des autres partitions. P_1 est ordonnancée à l'instant 0. P_2 peut commencer son exécution à 3, 4, 5, 6, 7 ou 8 dans la MAF_1. Elle peut donc être ordonnancée dans $Hyp_1 - (C_1 + C_2) + 1 = 10 - (3 + 2) + 1 = 6$ positions dans la MAF_1. P_3 est ordonnancée dans le premier intervalle pour éviter des doublons. Elle peut prendre $10-(3+2+2)+1 = 4$ positions. P_4 est ordonnancée dans le deuxième intervalle puisqu'il n'y a pas assez de temps libre dans le premier intervalle. Elle peut prendre $10-(3+2+4)+1 = 2$

positions dans la MAF_1. P_5 peut être ordonnancée dans tous les intervalles : elle peut prendre $40-(3+2+2+1)+1-(3+2+4+1)+1-(3+2+2+1)+1-(3+2+1)+1 = 12$ positions. P_6 ici ne peut être ordonnancée que dans le quatrième intervalle et prendre 2 positions. Si aucune position de P_6 fonctionne, il faut ordonnancer autrement les cinq partitions précédentes. Ceci revient à faire au maximum $6 \times 4 \times 2 \times 12 = 576$ retours en arrière juste pour cette branche.

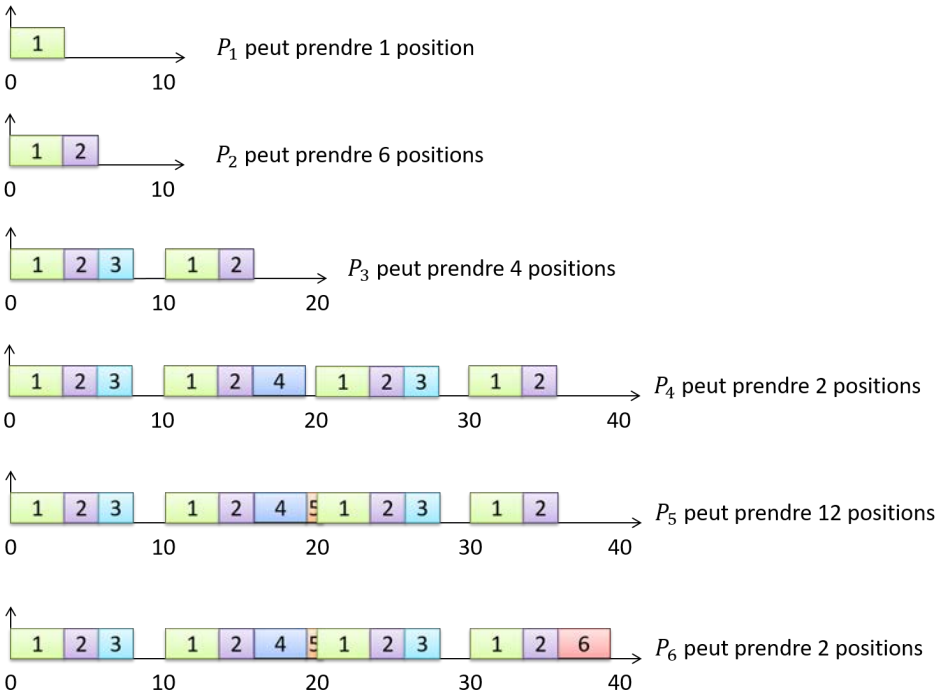


FIGURE 4.2 Ordonnancement des partitions laissant le plus de marge pour l'allocation des autres partitions

Un retour arrière est effectué jusqu'à ce qu'une allocation valide soit trouvée ou qu'aucune allocation valide existe. Ceci revient à tester tous les ordonnancements dans le pire des cas. Nous avons montré en 3.2.2 que lorsque le nombre de partitions allouées sur un processeur augmente, le nombre d'ordonnancements possibles explose. C'est le problème principal du précédent algorithme.

4.2 Méthode de « séparation et évaluation »

La recherche d'allocations et d'ordonnancements de façon itérative fait partie des problèmes d'optimisation combinatoire dits durs. Deux ensembles de techniques peuvent

être utilisés pour résoudre ce problème : les méthodes dites exactes et les heuristiques. L'avantage des méthodes exactes est d'effectuer une recherche exhaustive afin de trouver une solution optimale si elle existe, mais leur inconvénient majeur provient de leur temps d'exécution qui augmente avec la taille du système. Pour contrer ce temps d'exécution, des heuristiques ou des méta-heuristiques telles que le recuit simulé, la recherche Tabou, les algorithmes génétiques, peuvent être implémentées, mais elles ne conduisent pas en général à une solution optimale en un temps fini. [111]

Dans cette partie, nous souhaitons trouver toutes les allocations qui satisfont les contraintes d'ordonnancement et les contraintes de communication de bout-en-bout. Le but est de trouver pour chaque branche l'ordonnancement partiel le plus intéressant, tout en écartant les mauvaises solutions au plus tôt. Pour cela, nous allons nous appuyer sur la méthode dite de « séparation et évaluation ».

Cette méthode peut être illustrée par le problème du sac à dos qui est un problème NP-complet [112]. Ce problème consiste à remplir un sac à dos de capacité b avec des produits x_1, x_2, $\cdots$, x_n qui ont un poids b_1, b_2, $\cdots$, b_n et rapportent un coût c_1, c_2, $\cdots$, c_n par unité, de façon à maximiser le profit. Le système d'équation associé à ce problème est donné équation 4.1.

$$\begin{aligned} \max Z = \quad & c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n \\ \text{tel que} \quad & b_1 \cdot x_1 + b_2 \cdot x_2 + \cdots + b_n \cdot x_n \quad \leq b \\ & x_i \in \mathbb{R} \end{aligned} \tag{4.1}$$

Prenons le problème du sac à dos suivant :

$$\begin{aligned} \max Z = \quad & x_1 + 2 \cdot x_2 + 3 \cdot x_3 + 4 \cdot x_4 \\ \text{tel que} \quad & 4 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 + 2 \cdot x_4 \quad \leq 12 \\ & x_1 \in \{0, 1, 2, 3\} \\ & x_2 \in \{0, 1, 2\} \\ & x_3 \in \{0, 1\} \\ & x_4 \in \{0, 1, 2, 3, 4\} \end{aligned} \tag{4.2}$$

Toutes les combinaisons de remplissage du sac à dos respectant la contrainte de poids $b \leq 12$ sont données Figure 4.3. La valeur obtenue de Z est donnée en dessous de chaque feuille solution.

La méthode de « séparation et évaluation », ou « *branch and bound* », est un outil qui a été proposé par Land et Doig [113] pour résoudre les problèmes d'optimisation

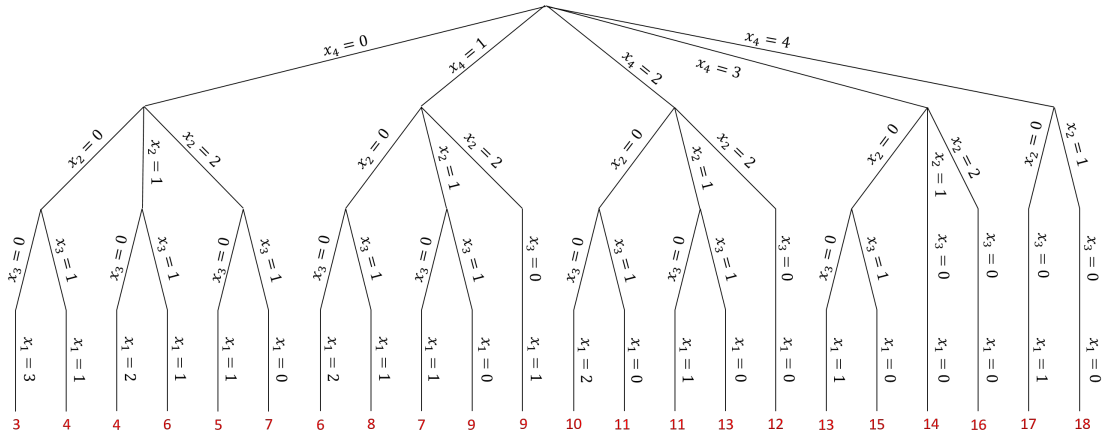


FIGURE 4.3 Solutions possibles du problème du sac à dos

NP-durs. Il s'agit de parcourir un arbre afin d'en arriver aux extrémités, les feuilles, représentant une solution complète satisfaisant les contraintes du problème ; les solutions temporaires sont représentées par un nœud.

À chaque nouveau nœud, la décision d'explorer la nouvelle branche est prise en fonction des deux axes suivants : la séparation et l'évaluation. La séparation permet d'obtenir une méthode générique pour énumérer toutes les solutions tandis que l'évaluation évite l'énumération systématique de toutes les solutions. Plus précisément, l'évaluation permet de réduire l'espace de recherche en éliminant les sous-ensembles qui ne contiennent pas la solution optimale. Le nœud candidat généré est comparé au nœud qui a le coût le plus intéressant déjà exploré dans l'arbre : si le coût du nœud le plus intéressant reste meilleur que celui du nœud candidat, alors la nouvelle branche est élaguée car les solutions sous-jacentes resteront par supposition moins bonnes, sinon, la branche courante est explorée. La séparation consiste à réduire le problème en un ensemble de sous-problèmes. Grâce à l'évaluation, les sous-problèmes pouvant potentiellement contenir la solution optimale sont retenus.

Reprenons l'exemple de l'équation 4.2. Chaque produit est ajouté dans le sac à dos dans l'ordre x_4, x_2, x_3, x_1, i.e. par ordre décroissant du rapport coût/poids. Dans ce cas, cette méthode est dite exacte car nous sommes sûrs de trouver le maximum global. Si les produits sont ajoutés dans un autre ordre, le risque est de se retrouver bloqué dans un maximum local. Construisons l'arbre de recherche illustré Figure 4.4.

1. Une première étape consiste à évaluer tous les nœuds x_4 générés.
   — Le nœud $x_4 = 4$ correspond à mettre quatre produits x_4 dans le sac à dos. L'évaluation de ce nœud prend en compte à la fois le coût réel (16) et le coût

estimé avec le produit suivant. Ce dernier est donné par le coût du produit suivant ($c_2 = 2$) et leur nombre potentiel proportionnel au poids restant dans le sac à dos sur le poids du produit suivant ($\frac{b-4\times b_4}{b_2}$). L'évaluation du nœud est alors de $16 + 2 \times \frac{12-8}{3} = 18,67$. Le coût de chaque feuille sous-jacente à ce nœud courant ne dépassera pas cette valeur.

— Le nœud $x_4 = 3$ a pour évaluation $12 + 2 \times \frac{12-6}{3} = 16$.

— Le nœud $x_4 = 2$ a pour évaluation $8 + 2 \times \frac{12-4}{3} = 13,33$.

— Le nœud $x_4 = 1$ a pour évaluation $4 + 2 \times \frac{12-2}{3} = 10,67$.

— Le nœud $x_4 = 0$ a pour évaluation $0 + 2 \times \frac{12-0}{3} = 8$.

Le nœud offrant la meilleure évaluation est le nœud $x_4 = 4$. Ce dernier est développé.

2. Une deuxième étape consiste à diviser le nœud ayant obtenu la meilleure évaluation. Ici, il s'agit du nœud $x_4 = 4$. x_2 peut prendre deux valeurs ici : 0 ou 1. L'évaluation de ces deux nœuds nous donne respectivement 18,4 et 18,6. Comme il n'y a pas d'évaluation supérieure dans notre arbre de recherche, le nœud $(x_4, x_2) = (4, 1)$ est développé.

3. La troisième étape développe le nœud $(x_4, x_2) = (4, 1)$ qui a obtenu la meilleure évaluation. x_3 de poids 5 ne peut pas être ajouté au sac à dos : le sac à dos est chargé à 11 ne laissant que 1 pour ajouter un nouveau produit. L'évaluation du nœud obtenu est alors de $18 + \frac{1}{4} = 18,25$, qui est inférieur à l'évaluation du nœud $(x_4, x_2) = (4, 0)$. Ce dernier est alors développé.

4. Pour les mêmes raisons que le point précédent, x_3 ne peut pas être ajouté au sac à dos. L'évaluation du nœud obtenu est alors de $16 + \frac{12-8}{4} = 17$. Cette valeur est inférieure à l'évaluation du nœud $(x_4, x_2, x_3) = (4, 1, 0)$.

5. Le nœud $(x_4, x_2, x_3) = (4, 1, 0)$ est développé : x_1 ne peut prendre que la valeur 0. L'évaluation exacte est alors de 18. Comme aucune autre branche a une évaluation supérieure, nous avons obtenu la valeur de la solution optimale qui consiste à prendre 4 unités du produit x_4 et 1 unité du produit x_2.

Dans cette thèse, nous recherchons toutes les allocations valides avec des ordonnancements respectant les contraintes de bout-en-bout. Le premier problème consiste à choisir le processeur où est allouée une partition donnée. Le deuxième problème doit spécifier l'intervalle et la position de la partition dans un processeur donné. Le tout doit satisfaire les exigences de bout-en-bout. Ce dernier problème revient à un problème de remplissage d'un sac à dos : déterminer le meilleur ordonnancement d'un ensemble de n partitions. La méthode de « séparation et évaluation » nous permet de réduire l'espace de recherche

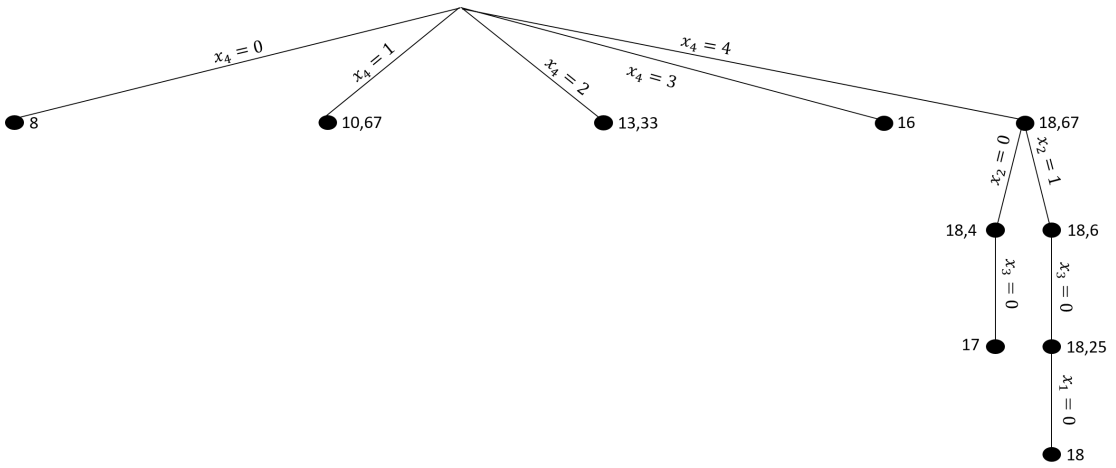


FIGURE 4.4 Recherche d'une solution optimale au problème du sac à dos avec la méthode de séparation et évaluation

afin d'énumérer les solutions valides. Cette méthode est adaptée et implémentée dans notre heuristique pour la sélection des ordonnancements.

4.3 Sélection des ordonnancements

Dans cette partie, nous souhaitons réduire le temps de recherche de notre analyse. Nous proposons ici une heuristique gloutonne visant à supprimer le retour en arrière lors des recherches d'ordonnancements, le problème majeur du précédent algorithme. Lorsqu'une nouvelle partition est allouée sur un processeur, tous les intervalles où la partition peut être ordonnancée sont testés. Chaque ordonnancement représente un nouveau nœud. Grâce à la méthode de séparation et évaluation, nous évaluons le potentiel de chaque nœud. Seul l'ordonnancement avec le potentiel le plus élevé est retenu (les autres sont définitivement abandonnés). Le potentiel d'un ordonnancement est évalué, basé sur la somme f_c des marges des chaînes de communication ma_{ch_i,a_m} :

$$f_c = \sum_{ch_i \in Ch \subset a_m} ma_{ch_i,a_m} \tag{4.3}$$

où

$$ma_{ch_i,a_m} = D_{ch_i,max} - D_{ch_i,a_m} \tag{4.4}$$

La marge d'une chaîne de communication est la différence entre la contrainte de bout-en-bout de celle-ci et son délai de bout-en-bout courant calculé grâce à l'équation 3.16. Le but de cette métrique est de choisir l'ordonnancement qui maximise le nombre de

possibilités pour allouer les partitions restantes appartenant aux différentes chaînes de communication dans les futurs ordonnancements des nœuds enfants en leur laissant le plus de marge sur les communications, qu'elles soient locales ou distantes. Cette métrique permet de maximiser le potentiel d'un ordonnancement et ainsi de rapprocher toutes les partitions communicantes dans un même processeur tout en minimisant l'impact des communications sur l'architecture créée.

Prenons l'exemple Figure 4.5. Lorsque la partition P_5 est allouée sur le premier processeur, plusieurs ordonnancements candidats existent. Nous en illustrons que deux Figure 4.5 parties b et d pour des raisons de lisibilité. Le potentiel du premier ordonnancement est :

$$f_c = (30 - 17) + (40 - 35) + (60 - 54) = 24$$

et celui du second ordonnancement est :

$$f_c = (30 - 17) + (40 - 33) + (60 - 54) = 26.$$

Le second ordonnancement offre une marge supplémentaire de 2 millisecondes par rapport au premier ordonnancement. Si nous comparons les parties b et d de la Figure 4.5, nous constatons que cette marge de 2 millisecondes est obtenue grâce à l'exécution consécutive des partitions P_2 et P_5 dans le second intervalle Figure 4.5 partie d alors que P_3 sépare ces deux partitions dans le premier intervalle Figure 4.5 partie b. Ainsi, le second ordonnancement est sélectionné. Nous trouvons ensuite l'ordonnancement contenant P_6 illustré Figure 4.5 partie e qui amène à une allocation valide de toutes les partitions.

4.4 Sélection des partitions

Dans les travaux [59], Kermia propose de regrouper les chaînes les plus longues en nombre de partitions dans les mêmes processeurs pour réduire le nombre d'allocations possibles. Par conséquent, cette méthode ne permet pas d'offrir une distribution des partitions sur un plus grand nombre de processeurs. Par ailleurs, il ne prend pas en considération qu'une chaîne courte peut être plus contrainte qu'une chaîne plus longue telle que dans notre exemple la chaîne ch_2 (2 partitions) qui est plus contrainte que la chaîne ch_3 (3 partitions).

Dans notre approche précédente, les partitions étaient traitées par période croissante. Ceci a permis de limiter l'espace de recherche dans les niveaux les plus proches de la base de l'arbre. En effet, allouer les partitions avec de longues périodes augmente la taille de la MAF, entraînant par conséquent un plus grand nombre d'ordonnancements

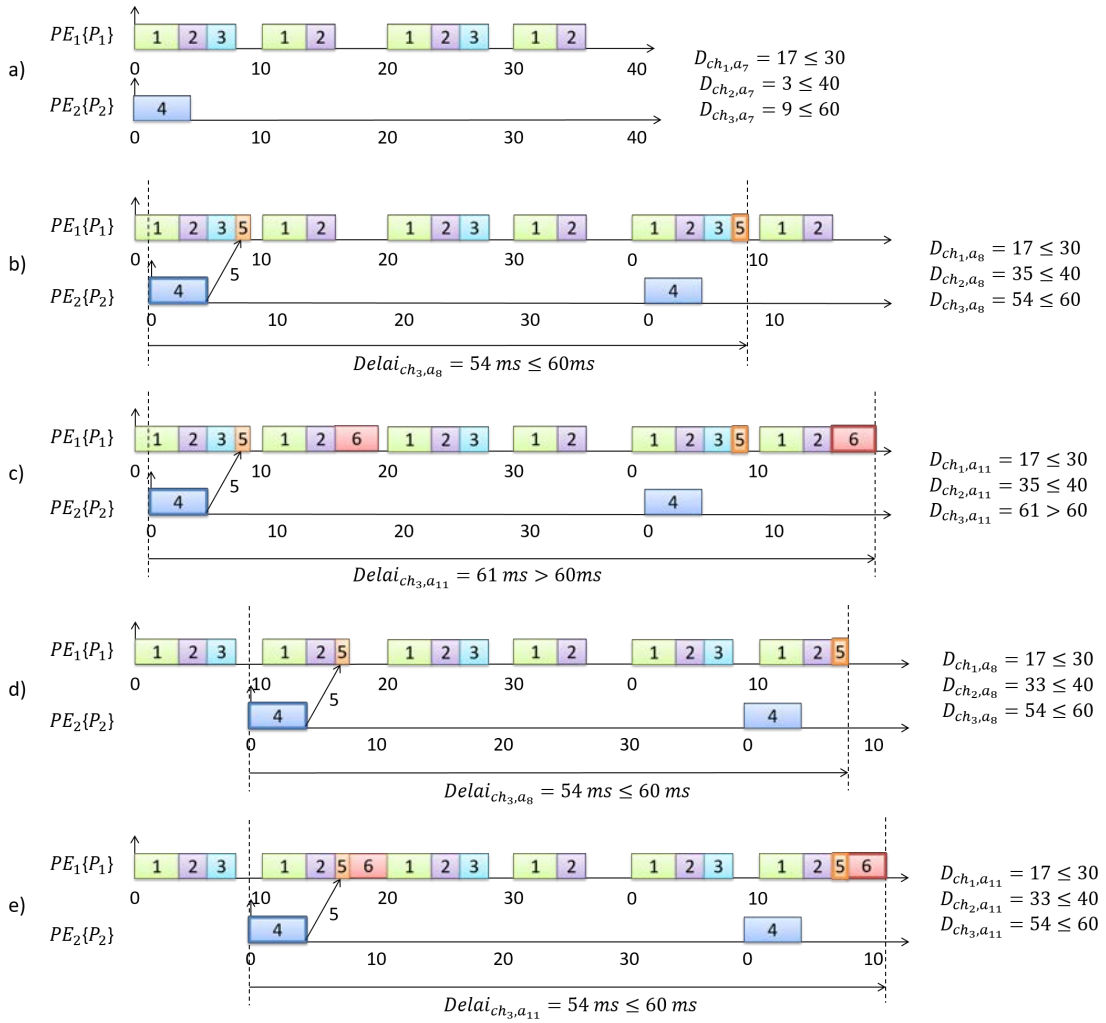


FIGURE 4.5 Recherche d'ordonnancements valides

candidats. Retarder le plus possible l'allocation de ces partitions limite ainsi le nombre d'ordonnancements candidats.

Dans notre nouvelle approche présentée ici, le problème est un peu différent, étant donné qu'un seul ordonnancement est sélectionné, sans retour en arrière. Il en découle que le nombre d'ordonnancements considérés par le processus de sélection dans chaque nœud reste très petit, même pour des MAFs longues.

Pour obtenir la marge la plus forte pour chaque chaîne, il faut que les partitions communicantes dans un même processeur soient le plus proche possible. Il faut aussi que toutes les chaînes respectent leur contrainte de bout-en-bout.

Nous choisissons de ce fait de traiter les partitions en fonction de la marge croissante des chaînes afin d'allouer en priorité les partitions les plus contraintes, i.e. celles appartenant aux chaînes avec les plus petites marges. Cet ordre est obtenu en triant en premier les chaînes par marge croissante, basé sur l'équation 3.16 où aucune partition n'est allouée sur l'ensemble des processeurs. Ensuite, les partitions sont triées en fonction de leur ordre d'apparition dans les chaînes. Si une partition apparaît dans plusieurs chaînes, celle-ci est seulement considérée dans la chaîne avec la plus petite marge.

Comme nous l'avons précisé ci-dessus, les partitions communicantes appartenant à une même chaîne et allouées dans un même processeur doivent être les plus proches possibles. Il faut aussi que toutes les chaînes respectent leur contrainte de bout-en-bout. Pour respecter en même temps ces deux contraintes, nous considérons qu'un ensemble de partitions alloué dans le même intervalle d'une MAF représente un bloc indissociable pour ne plus remettre en cause les chaînes les plus contraintes. Une nouvelle partition allouée dans cet intervalle peut seulement prendre deux positions : la première ou la dernière. Elle est accolée par ailleurs au bloc indissociable.

4.5 Algorithme général de notre heuristique

Notre heuristique consiste en la construction d'un arbre dont tous les nœuds sont créés au fur et à mesure qu'une partition est allouée sur un processeur particulier. Pour cela, nous créons un algorithme récursif initialisé dans l'Algorithme 4. Les partitions sont triées et numérotées en fonction de la marge croissante des chaînes (Algorithme 4, Pré-conditions). Dans chaque nouveau processeur k, un premier tableau vide de T_{min} unités de temps, i.e. la plus petite des périodes de l'ensemble des partitions $\mathcal{P}$, est généré (Algorithme 4, lignes 3-7). Ce tableau représente la MAF_k du processeur k de durée Hyp_k dont chaque intervalle a une durée T_{min}. La première partition peut être ordonnancée (Algorithme 4, ligne 8).

Nous créons une nouvelle variable $a_{potentielle}$ qui retient dans le nœud courant l'allocation ayant le plus grand potentiel selon la métrique Équation 4.3. Dans un premier temps, cette allocation est vide. Nous considérons que la somme de ces marges de communication $f_{c,a_{potentielle}}$ est égal à -1 (Algorithme 5, lignes 1-2). Une nouvelle partition P_j de période T_j et de durée d'exécution C_j peut tenter d'être ordonnancée dans la MAF_k si $T_j \leq Hyp_k$. Si ce n'est pas le cas, la MAF_k est répliquée $\frac{T_j}{Hyp_k}$ fois (Algorithme 5, lignes 3-6). La nouvelle longueur de la MAF_k devient $Hyp_k = T_j$ (Algorithme 5, ligne 5).

Si toutes les unités de temps sont libres, alors P_j est ordonnancée dans les C_j premières unités de temps de la MAF_k (Algorithme 5, lignes 5-6). Sinon, nous considérons que les partitions allouées dans chaque intervalle représente un ensemble indissociable. P_j est ordonnancée en première et en dernière position de chaque intervalle de la MAF_k (Algorithme 5, lignes 10-20).

L'algorithme 6 va ordonnancer de deux façons différentes les partitions, selon qu'elles doivent être placées en première ou dernière position d'un intervalle. Si la partition prend la première place d'un intervalle (Algorithme 6, lignes 1-7), il faut dans un premier temps décaler les partitions présentes pour laisser C_j unités de temps, et décaler si nécessaire les autres partitions pour l'exécution périodiques des partitions (Algorithme 6, lignes 2-3). Si la partition prend la dernière place d'un intervalle (Algorithme 6, lignes 8-13), il faut déterminer l'instant h où se termine la dernière partition ordonnancée dans l'intervalle (Algorithme 6, ligne 9). Comme les partitions ne sont pas allouées par période croissante, il faut vérifier que chaque partition s'exécute de façon strictement périodique (Algorithme 6, lignes 4 et 10) :

$$MAF_k\left[x, x + C_j - 1\right] \quad \text{telle que} \quad \begin{cases} x \geq 0 \\ x + C_j - 1 \leq T_j \\ \text{les segments } \left[x + n \times T_j, x + n \times T_j + C_j - 1\right], \\ \qquad n \in \{0, .., \left\lceil \frac{Hyp_k}{T_j} \right\rceil - 1\} \text{ sont libres.} \end{cases} \tag{4.5}$$

Si l'ordonnancement respecte les strictes périodicités des partitions, une allocation a_m est générée (Algorithme 6, lignes 5 et 11). Dans ce cas, les contraintes de bout-en-bout sont vérifiées (Algorithme 6, lignes 15-23). Si a_m est valide, i.e. la partition est ordonnancée de façon à satisfaire les contraintes de bout-en-bout, une allocation est retournée (Algorithme 5, ligne 20).

Si a_m existe (Algorithme 5, lignes 12-13), sa métrique est calculée (Algorithme 5, ligne 14). L'allocation a_m remplace l'allocation potentielle courante si et seulement si sa métrique est supérieure à celle de $a_{potentielle}$ (Algorithme 5, lignes 15-17).

Si $a_{potentielle}$ existe (Algorithme 5, ligne 22), une partition suivante est allouée (Algorithme 5, lignes 26-28), ou s'il n'en reste plus, nous obtenons une allocation totale valide (Algorithme 5, lignes 23-24).

Algorithme 4 Initialisation de l'heuristique

Pré-conditions : les partitions triées et numérotées en fonction de la marge croissante des chaînes, le nombre maximum de processeurs $MaxNbPE$

1: Soit a_0 une allocation de $MaxNbPE$ processeurs vides
2: Soit $T_{min} = \min_{P_i \in \mathcal{P}}(T_i)$ la plus petite des périodes de $\mathcal{P}$
3: **pour** $x = 1..MaxNbPE$ **faire**
4: Le processeur PE_x est représenté par un tableau de T_{min} unités de temps
5: $Hyp_x \leftarrow T_{min}$
6: $t^x_{intervalle} \leftarrow T_{min}$
7: **fin pour**
8: Ordonnancement de la partition P_1 sur le processeur PE_1 de l'allocation a_0 (Algorithme 5)

4.6 Etude de complexité

Comme en 3.2.2, nous recherchons toutes les allocations valides.

L'algorithme d'ordonnancement est différent de la partie 3.2.2. Soit k le nombre de fréquences de l'ensemble des partitions $\mathcal{P}$. Un ordonnancement est choisi parmi 2^{2k} choix possibles dans chaque nœud parcouru, i.e. dans chaque intervalle, une nouvelle partition pourra prendre au maximum deux positions, la première ou la dernière. Soit N le nombre de partitions à distribuer, le nombre total de nœuds pouvant être parcourus est donné par :

$$\sum_{i=1}^{N} B_i = \mathcal{O}(\frac{N^N}{N!})$$

La complexité de cette heuristique est de l'ordre de $\mathcal{O}(\frac{2^{2k} \times N^N}{N!})$.

4.7 Comparaisons sur l'étude de cas

Reprenons l'étude de cas présentée en 3.2.3 et regardons les différences apportées par l'heuristique, que ce soit au niveau des solutions trouvées ou du temps d'exécution et l'utilisation de la mémoire.

Comparons dans un premier temps le nombre d'allocations valides pour chaque processeur avec les Tables 3.3 et 4.1. L'heuristique manque une allocation valide sur les 13 trouvées avec l'approche intégrée, soit un taux de réussite de 92%. L'allocation

Algorithme 5 Ordonnancement d'une partition P_j sur un processeur PE_k d'une allocation a_l

1: Soit $a_{potentielle}$ une allocation vide
2: $f_{c,a_{potentielle}} \leftarrow -1$ la somme des marges des chaînes de communication de $a_{potentielle}$
3: **si** $P_j > Hyp_k$ **alors**
4: Répliquer la MAF_k $\frac{T_j}{Hyp_k}$ fois
5: $Hyp_k \leftarrow T_j$
6: **fin si**
7: **si** MAF_k vide **alors**
8: $a_{potentielle} \leftarrow$ Ordonnancement (P_j, 1, 1, PE_k, a_l) (Algorithme 6)
9: **sinon**
10: **pour** $i = 1..\left\lceil \frac{Hyp_k}{t^k_{intervalle}} \right\rceil$ **faire**
11: **pour** $p = 1$, position de la dernière partition ordonnancée dans l'intervalle $i+1$ **faire**
12: $a_m \leftarrow$ Ordonnancement (P_j, i, p, PE_k, a_l) (Algorithme 6)
13: **si** $a_m \neq vide$ **alors**
14: Calculer f_{c,a_m}
15: **si** $f_{c,a_m} > f_{c,a_{potentielle}}$ **alors**
16: $a_{potentielle} \leftarrow a_m$
17: **fin si**
18: **fin si**
19: **fin pour**
20: **fin pour**
21: **fin si**
22: **si** $a_{potentielle} \neq$ vide **alors**
23: **si** la partition P_j était la dernière partition à ordonnancer **alors**
24: $a_{potentielle}$ est une allocation totale valide
25: **sinon**
26: **pour** $h = 1..\min$(nombre de processeurs utilisés $+ 1, MaxNbPE$) **faire**
27: Ordonnancement de la partition P_{j+1} sur le processeur PE_h de l'allocation $a_{potentielle}$ (Algorithme 5)
28: **fin pour**
29: **fin si**
30: **fin si**

Algorithme 6 Déterminer si l'ordonnancement de la partition P_j dans l'intervalle i $p^{ième}$ position dans le processeur PE_k de l'allocation a_l est possible

si $p = 1$ **alors**
  Décaler les groupes de partitions appartenant aux intervalles $i + n \times \frac{T_j}{t^k_{intervalle}}, n \in \{0, .., \frac{Hyp_k}{T_j} - 1\}$ pour laisser C_j unités de temps au début de ces intervalles
  Décaler les groupes de partitions appartenant aux autres intervalles afin de respecter la stricte périodicité des partitions.
  si les segments $\left[(n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j, (n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j + C_j - 1\right]$ sont libres **alors**
    a_m est l'allocation partielle intégrant la partition P_j dans le processeur PE_k de l'allocation a_l aux instants $(n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j, n \in \{0, .., \frac{Hyp_k}{T_j} - 1\}$
  fin si
fin si
si $p =$dernière position de l'intervalle i **alors**
  Soit h l'instant où se termine la dernière partition ordonnancée dans l'intervalle i.
  si les segments $\left[h + (n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j, h + (n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j + C_j - 1\right]$ sont libres **alors**
    a_m est l'allocation partielle intégrant la partition P_j dans le processeur PE_k de l'allocation a_l aux instants $h + (n + i \pmod{\frac{T_j}{t^k_{intervalle}}}) \cdot T_j, n \in \{0, .., \frac{Hyp_k}{T_j} - 1\}$
  fin si
fin si
si a_m existe **alors**
  pour $q = 1$..nombre de chaînes **faire**
    Vérifier que $D_{ch_q,a_m} \leq D_{ch_q,max}$
  fin pour
  si toutes les chaînes $D_{ch_q,a_m} \leq D_{ch_q,max}$ **alors**
    a_m est une allocation partielle valide
    Retourner a_m
  sinon
    Retourner une allocation vide
  fin si
sinon
  Retourner une allocation vide
fin si

manquante se situe sur le processeur $Proc_2$. Regardons les raisons pour lesquelles cette allocation n'est pas trouvée. Au lieu d'être triées par période croissante, les partitions sont triées en fonction de l'importance des chaînes. Ceci nous donne P_1, P_2, P_3, P_4, P_5, P_6 puis P_7, correspondant au tri des chaînes ch_1, ch_2, ch_3, ch_4, ch_5, ch_6, qui ont pour potentiel respectif 28, 349, 472, 485, 489 et 489.

TABLE 4.1 Nombre d'allocations valides selon le type de processeur avec l'heuristique

Nombre de processeurs	$Proc_1$	$Proc_2$	$Proc_3$	$Proc_4$
1	1	0	0	0
2	3	3	2	0
3	1	1	1	0
4	0	0	0	0
Temps d'analyse (ms)	10	6	4	<1

Ordonnançons ces sept partitions sur le processeur $Proc_2$, illustré Figure 4.6. P_1 est allouée et ordonnancée en première position sur la MAF, puis la MAF est étendue pour ordonnancer P_2 dans le premier intervalle. P_3 est ensuite allouée : la MAF est à nouveau étendue pour pouvoir ordonnancer P_3 dans l'un des deux premiers intervalles. Comme il reste 3 ms et 13 ms respectivement dans le premier et le second intervalle, P_3 est allouée dans le second intervalle en deuxième position selon notre heuristique. P_4 est ensuite allouée dans le second et le quatrième intervalle de la MAF, respectivement à la troisième et à la deuxième position de ces intervalles. Comme le temps restant dans le premier et le troisième intervalle est inférieur à 1, et égal à 0 dans le deuxième et le quatrième intervalle, P_5 ne peut pas être ordonnancée. En effet, l'heuristique considère les partitions déjà ordonnancées comme un seul et même bloc indissociable, empêchant l'ajout de nouvelle partition entre deux partitions même si l'espace est suffisant. L'heuristique ne trouve pas toutes les solutions valides mais reste très proche des résultats de l'approche intégrée.

Intéressons-nous maintenant au temps d'analyse et à l'utilisation de la mémoire lorsque nous recherchons toutes les allocations valides avec au plus 12 processeurs $Proc_1$, correspondant à la distribution maximale du système VMSA avec ses réplicas, i.e. nous prenons en compte tous les processeurs des AMCs. Comparons l'approche exhaustive avec et sans tri des partitions et l'heuristique. Tous ces algorithmes trouvent 625 allocations valides. Dans ce cas, l'heuristique ne rate aucune solution. Pour cette application, avec les paramètres temps-réel fournis, le choix d'ordonnancement dans chaque nœud est judicieux.

La Figure 4.7 montre le temps d'exécution et le besoin en mémoire requis pour chaque

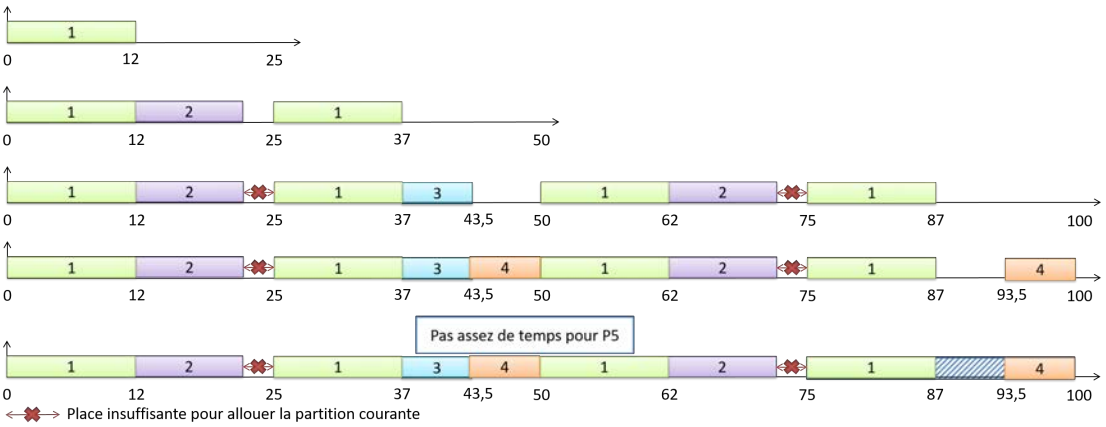


FIGURE 4.6 Allocation manquée avec l'heuristique

approche pour trouver la première solution valide. Toutes les approches finissent en moins de 100 ms, avec une utilisation de la mémoire quasi-identique.

Lorsque toutes les allocations valides sont recherchées, l'heuristique surpasse les approches exhaustives comme le montre la Figure 4.8, que ce soit au niveau du temps d'exécution (11 secondes versus 162 lorsque les partitions sont triées par période croissante et 381 lorsqu'elles ne le sont pas) ou de l'utilisation de la mémoire. Ceci est dû à la limitation drastique des ordonnancements considérés.

4.8 Passage à l'échelle

Nous considérons une première configuration incluant 30 partitions. Chaque partition a une période $T = 25ms$ et un WCET $C = 5ms$. Comme en 3.2.4, deux partitions communicantes composent chacune des chaînes ch_i qui sont au nombre de 15. La contrainte de délai de bout-en-bout de chaque chaîne vaut 20 ms. Ces partitions sont allouées sur un ensemble $\mathcal{E}$ contenant au minimum 2 processeurs.

Nous comparons dans un premier temps les deux approches, i.e. celle intégrée et l'heuristique, pour trouver une allocation valide. Ces deux approches trouvent la même première solution, ce qui signifie que le choix d'ordonnancement n'a pas d'impact sur le résultat. La Figure 4.9 montre que l'heuristique réduit considérablement le temps d'exécution de l'analyse : lorsque le nombre de processeurs est égal à 5, notre analyse met 177 026 secondes pour se terminer avec l'approche intégrée tandis qu'elle met 3 secondes avec l'heuristique. Nous observons par ailleurs que dans les deux cas, le temps d'exécution augmente jusqu'à une limite de sept processeurs, ensuite il diminue. Comme

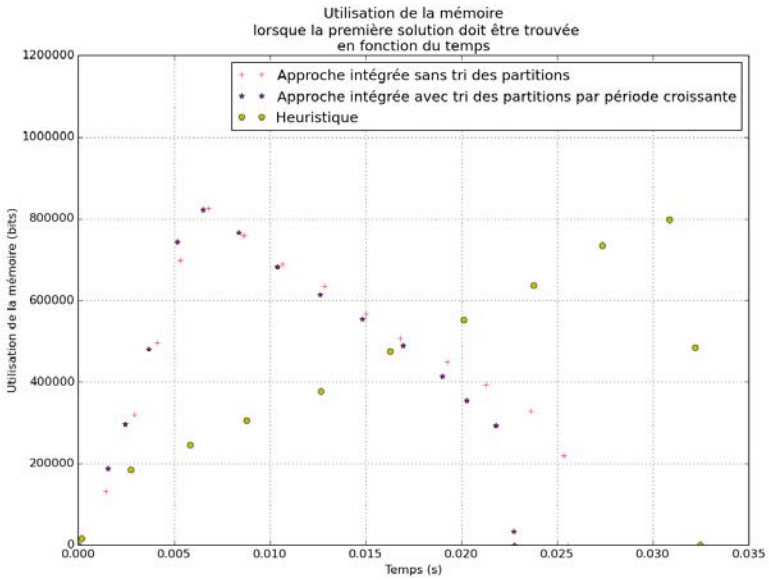


FIGURE 4.7 Temps et mémoire pour trouver la première allocation valide du système VMSA avec le processeur $Proc_1$

nous parcourons l'arbre d'allocations en commençant par les branches les plus à droite, nous cherchons alors à utiliser le plus de processeurs possibles. Sur cette configuration particulière, une allocation valide est facilement trouvée lorsqu'au minimum 8 processeurs composent l'architecture. Ce n'est pas le cas avec moins de processeurs où aucune allocation ne fonctionne.

Des résultats similaires sont obtenus avec différents nombres de partitions comme le montre les courbes dans les Figures 4.10 (heuristique gloutonne) et 4.11 (approche intégrée).

Lorsque toutes les allocations valides sont recherchées, l'approche intégrée ne finit pas en un temps raisonnable (certaines analyses ont dû être arrêtées au bout de 3 mois), tandis que l'heuristique gloutonne les trouve en moins de 1000 secondes pour toutes les configurations testées, exceptée celle devant allouer 30 partitions sur 8 processeurs comme le montre la Figure 4.12.

La Figure 4.13 montre que le temps d'exécution nécessaire pour trouver toutes les allocations valides pour 10 partitions sur au plus 10 processeurs augmente considérablement lorsque la contrainte des chaînes est relâchée à 40 ms. Dans ce cas, le nombre d'allocations valides croît radicalement. Très peu de branches sont élaguées et l'algorithme doit par conséquent visiter un plus grand nombre de nœuds de l'arbre d'allocations. Cependant,

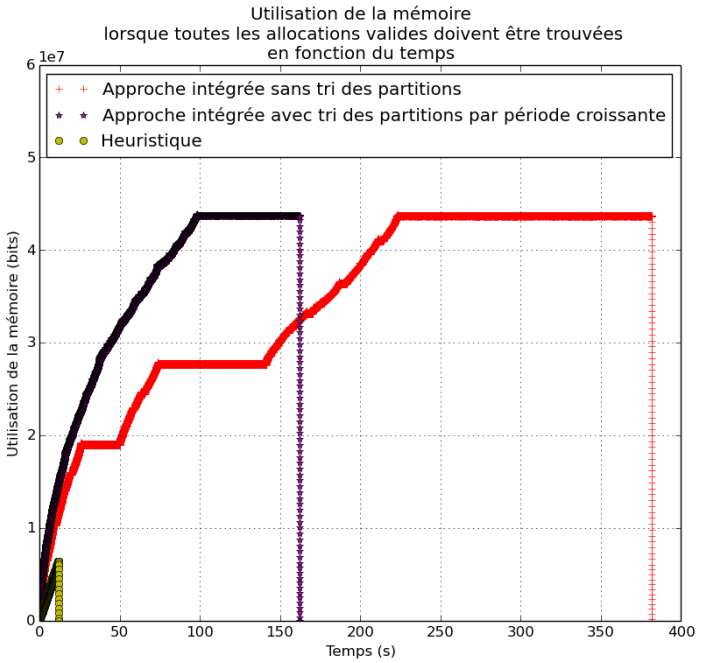


FIGURE 4.8 Temps et mémoire pour trouver toutes les allocations valides du système VMSA avec le processeur $Proc_1$

l'heuristique termine toujours en moins de 4000 secondes.

Tous ces résultats montrent que notre heuristique gloutonne finit dès lors qu'une solution, la première, est trouvée jusqu'à une trentaine de partitions quel que soit le nombre de processeurs, à la différence de l'approche intégrée présentée en 3.2. Concernant la recherche de toutes les solutions, l'heuristique atteint des limites dès que le nombre de branches à évaluer devient supérieur à 25 millions, i.e. le cas de 30 partitions étant allouées sur au moins 8 processeurs, alors que l'approche intégrée peine à allouer une dizaine de partitions sur 7 processeurs. Cette heuristique est une solution très prometteuse pour distribuer un ensemble d'applications avioniques telles que le VMSA, ROSACE, le FADEC et le *Health Monitoring* étant donné que la taille de ces applications ne dépasse pas généralement la taille des configurations que nous avons considéré dans cette partie.

4.9 Conclusion

Dans ce chapitre, nous avons cherché à améliorer grâce à une démarche incrémentale l'≪ approche intégrée ≫ présentée dans le Chapitre 3. Le résultat obtenu est une heuristique gloutonne réduisant le coût associé à la génération de la MAF sur chaque processeur.

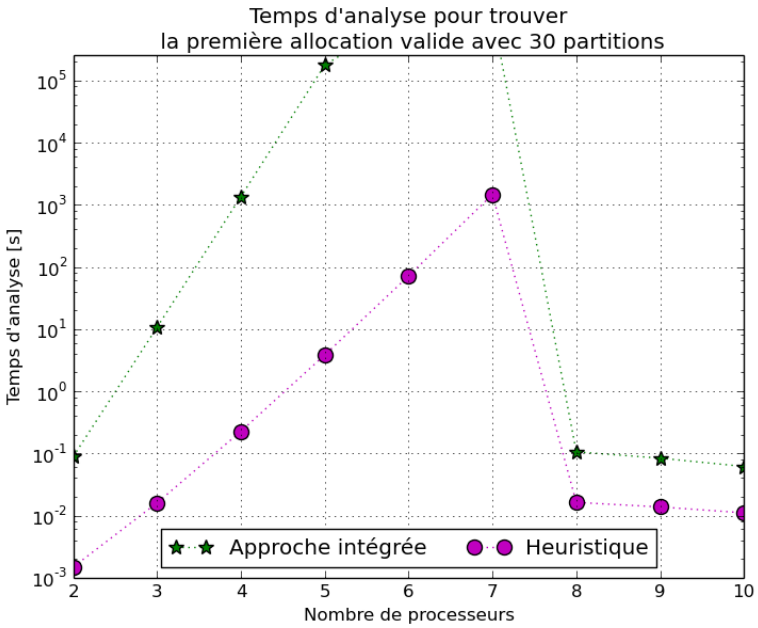


FIGURE 4.9 Temps d'analyse pour distribuer 30 partitions et trouver la première solution avec l'approche intégrée et l'heuristique, $D_{ch_i,max} = 20ms$

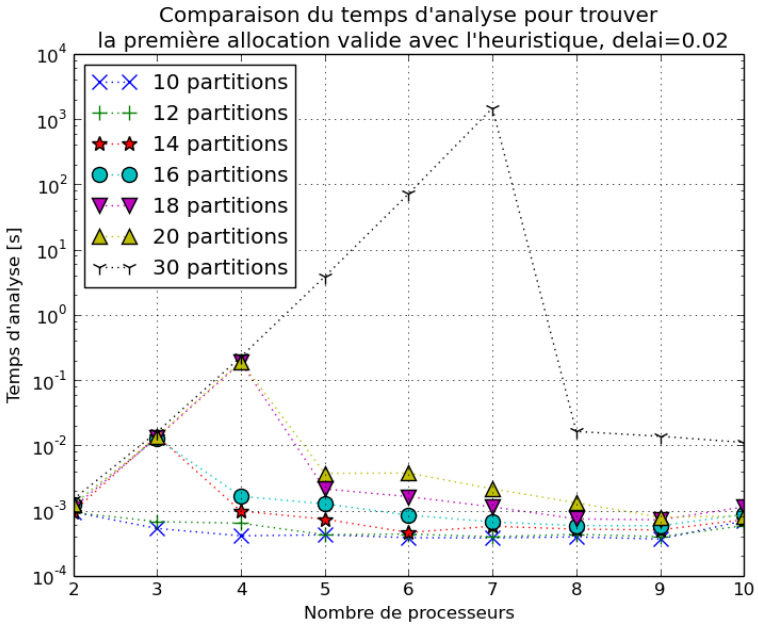


FIGURE 4.10 Temps d'analyse pour trouver une solution avec l'heuristique, $D_{ch_i,max} = 20ms$

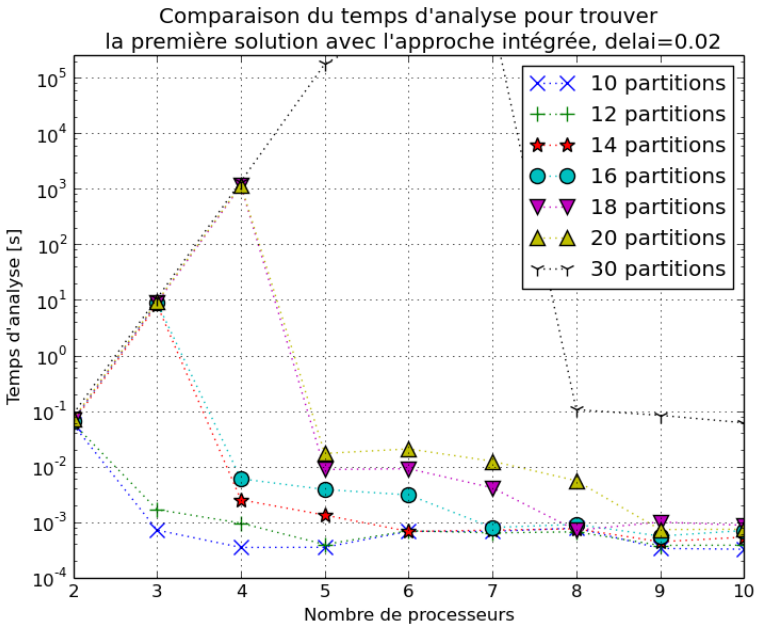


FIGURE 4.11 Temps d'analyse pour trouver une solution avec l'approche intégrée, $D_{ch_i,max} = 20ms$

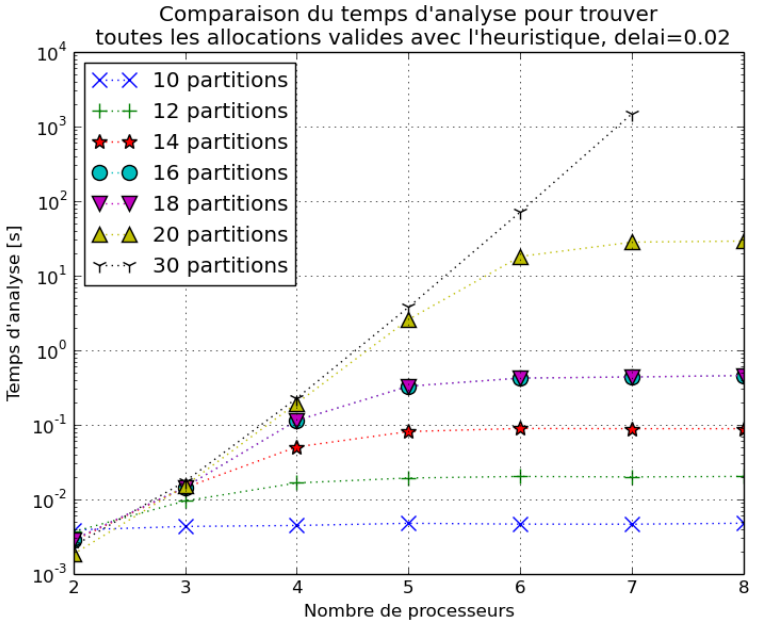


FIGURE 4.12 Temps d'analyse pour trouver toutes les solutions avec l'heuristique, $D_{ch_i,max} = 20ms$

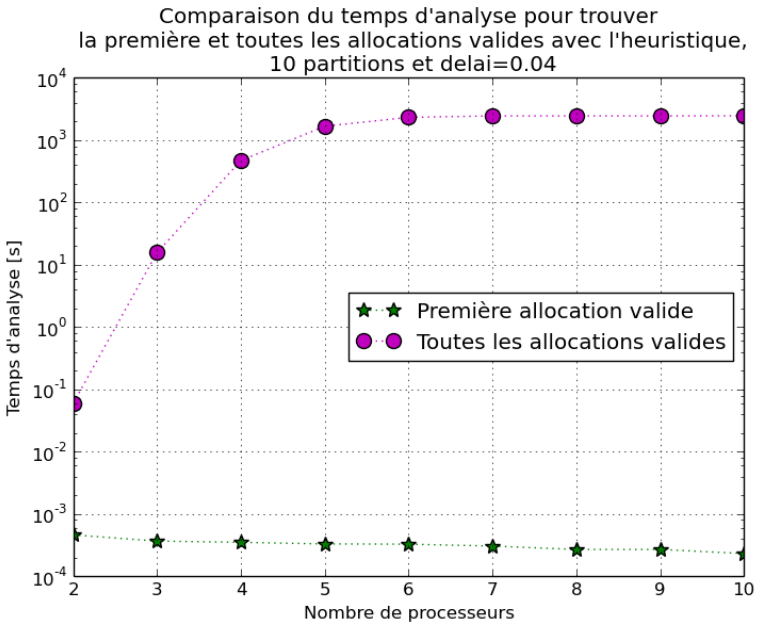


FIGURE 4.13 Comparaison du temps d'analyse pour trouver la première et toutes les solutions avec l'heuristique gloutonne, 10 partitions $P_i = (25, 5)$ et $D_{ch_i,max} = 40ms$

Au lieu de proposer un ordonnancement satisfaisant les contraintes de bout-en-bout aux nœuds enfants et de le modifier s'il n'y a pas une allocation sous-jacente valide, un ordonnancement pour une allocation donnée est figé pour réduire l'espace de recherche. À chaque nouveau nœud, l'ordonnancement le plus prometteur pour ordonnancer les partitions non-assignées est retenu, les autres étant éliminés. Sur l'étude de cas hélicoptère simplifiée, cette heuristique manque seulement une allocation valide parmi les treize trouvées avec l'approche intégrée, soit un taux de réussite de 92%. Sur l'étude de cas hélicoptère complète, cette heuristique permet d'obtenir toutes les allocations valides en 3 secondes au lieu de 158 secondes avec l'approche intégrée. Nous retrouvons sur une analyse de passage à l'échelle des résultats similaires : l'heuristique permet de réduire considérablement l'utilisation de la mémoire et le temps d'analyse lorsque nous recherchons une solution valide, permettant ainsi de terminer la recherche dans la majorité des cas. Par ailleurs, toutes les solutions valides sont trouvées en moins de trois heures si nous essayons d'allouer une trentaine de partitions sur au maximum sept processeurs. L'algorithme implanté dans cette heuristique est prometteur pour l'allocation et l'ordonnancement de systèmes avioniques sur des architectures plus distribuées.

Dans les Chapitres 3 et 4, nous avons considéré que tous les processeurs étaient connectés par un seul et même réseau avec une latence de communication fixe. Or, dans

une même architecture, différents réseaux interconnectant chacun deux types distincts de processeurs peuvent être implémentés. Nous proposons d'évaluer dans le chapitre suivant les latences de communication admissibles entre les différents processeurs pour permettre à un intégrateur système de sélectionner la/les meilleure/s technologie/s de communication pour une architecture donnée et de déterminer la position des E/S par rapport aux partitions distribuées.

Chapitre 5

Choix des latences de communication pour une distribution donnée

Dans le chapitre précédent, nous nous sommes seulement intéressés à la distribution des partitions avioniques sur un plus grand nombre de processeurs, en considérant que les latences maximales de communication (WCTT) sont fixées. Ces latences maximales dépendent de la technologie de communication utilisée et des flux qui y sont transmis. Le choix de la technologie est donc crucial pour le respect des contraintes de bout-en-bout et, par conséquent, pour l'admissibilité d'une allocation donnée.

Dans ce chapitre, nous proposons de rechercher toutes les allocations validant les contraintes de bout-en-bout, ainsi que les latences de communication maximales admissibles pour chacune d'entre elles. Le résultat de cette recherche offre à un architecte système plusieurs solutions de répartitions et d'ordonnancements des partitions avioniques sur un plus grand nombre de processeurs ainsi que des spécifications temporelles pour le choix des moyens de communication qu'il peut utiliser.

5.1 Recherche des latences admissibles

5.1.1 Formulation du problème

Comme nous l'avons vu dans les Chapitres 3 et 4, les allocations valides dépendent des ordonnancements. Si nous reprenons l'allocation valide a_{11} Figure 3.22, cette allocation est valide avec un WCTT de 5 ms. Si nous prenons maintenant une latence maximale de

communication de 12 ms, le calcul du délai est alors égal à :

$$\begin{aligned}
D_{ch_3,a_{11}} &= C_4 + distance(4,5) + C_5 + distance(5,6) + C_6 \\
&= C_4 + WCTT(4,5) + T_5 + C_5 \\
&\quad + \max_{M\in\{0,..,\lceil \frac{T_6}{T_5} \rceil - 1\}} \left(\min_{L\in\{0,..,\lceil \frac{T_5}{T_6} \rceil\}} (r_6 + L\cdot T_6 - (r_5 + C_5 + M\cdot T_5)) \right) \\
&\quad + C_6 \\
&= 4 + 12 + 40 + 1 \\
&\quad + \max_{M\in\{0\}} \left(\min_{L\in\{0,1\}} (16 + L\cdot 40 - (15 + 1 + M\cdot 40)) \right) + 4 \\
&= 4 + 12 + 40 + 1 + \min_{L\in\{0,1\}} (16 + L\times 40 - (15+1)) + 4 \\
&= 4 + 12 + 40 + 1 + 0 + 4 \text{ avec } L = 0 \\
&= 61 \text{ ms} \geq D_{ch_3,max}
\end{aligned} \tag{5.1}$$

Cette allocation, en prenant en compte un WCTT plus élevé, devient impossible. Les ordonnancements ainsi que les latences maximales de communication doivent être prises en compte en même temps dans la recherche d'allocations distribuées. Or, ces latences dépendent des différents flux traversant le réseau.

Différentes méthodes d'analyse pire cas du réseau existent. Nous pouvons citer le calcul réseau, l'approche par trajectoires, la vérification de modèles évoqués en 2.2.4. Elles nécessitent de connaître tous les flux transitant sur les câbles de communication afin d'évaluer les WCTTs entre deux partitions communicantes. Étant donné que l'architecture physique nous est inconnue, nous ne pouvons pas établir si un-des moyen-s de communication, pour une allocation donnée, répond-ent aux contraintes de délai de bout-en-bout. L'objectif est le suivant :

Comment évaluer les latences maximales de communication admissibles afin de définir les réseaux à implémenter dans une architecture donnée ?

Dans ce chapitre, nous proposons d'évaluer les latences maximales admissibles pour une allocation donnée au lieu de les figer. Cette technique permet de déterminer les réseaux qui peuvent être implémentés. Une fois que l'architecture de communication est choisie par l'intégrateur système, il lui suffit de vérifier que les latences des moyens de communication implémentés ne dépassent pas les contraintes de WCTT données par notre outil pour une allocation avec un ordonnancement valide donnée.

En 5.1.2, nous montrons que notre problème se modélise par un système de m inéquations à n inconnues. Ce type de système se résout grâce à la méthode d'élimination

de Fourier-Motzkin que nous illustrons en 5.1.3. Un polyèdre donne l'ensemble des solutions d'un système de m inéquations à n inconnues. Nous le représentons par une méthode graphique en 5.1.4. Nous détaillons en 5.1.5 la méthode d'élimination de Fourier-Motzkin.

5.1.2 Mise en équation du problème

Comme nous l'avons vu ci-dessus, le calcul du délai de bout-en-bout dépend en même temps de l'ordonnancement des partitions dans les différents processeurs et des latences de communication entre ces derniers. Si nous reprenons l'équation 3.16 donnant le délai de la chaîne ch_k pour une allocation a :

$$D_{ch_k,a} = \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset a}} \big(distance(i,j)\big) + \sum_{P_i \in ch_k} C_i \tag{5.2}$$

Ce calcul varie selon les distances entre les différentes partitions communicantes. Si les partitions communicantes P_i et P_j sont allouées sur le même processeur, la distance les séparant dépend de l'ordonnancement dans la MAF. À l'inverse, si elles sont assignées sur des processeurs différents, alors la distance les séparant dépend de la latence de communication, i.e. le WCTT dans le cadre d'une analyse pire-cas, et de la période de la partition destinatrice. Seuls les pires temps d'exécution, i.e. les WCET notés C_i, sont connus et fixes.

Ces délais doivent respecter les différentes contraintes de bout-en-bout spécifiées par un système donné. Dans les chapitres 3 et 4, nous supposions que les latences maximales de bout-en-bout étaient fixées. Ici, nous cherchons leurs limites. Étant donné que ces latences maximales dépendent des allocations avec leur ordonnancement ainsi que des contraintes de bout-en-bout, nous devons rechercher dans un premier temps les allocations valides satisfaisant les contraintes de bout-en-bout avec des latences nulles. Dans un second temps, nous recherchons les latences maximales autorisées pour une allocation donnée.

Pour une allocation donnée, chaque chaîne peut contenir de 0 à $NB_{\mathcal{P}\subset ch_k} - 1$ communications distantes, $NB_{\mathcal{P}\subset ch_k}$ étant le nombre de partitions dans la chaîne ch_k. Une chaîne doit satisfaire ses contraintes de bout-en-bout, ce qui signifie que :

$$D_{ch_k,a} \leq D_{ch_k,max}, \tag{5.3}$$

ce qui revient à vérifier dans notre cas que :

$$\sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset a}} \big(distance(i,j)\big) \leq D_{ch_k,max} - \sum_{P_i \in ch_k} C_i. \tag{5.4}$$

La somme des distances peut-être décomposée en deux parties comme nous le montrons équation 5.5 : la première partie correspond à la somme des distances entre deux partitions communicantes allouées dans un même processeur, tandis que la deuxième partie correspond à la somme des distances entre deux partitions communicantes allouées sur différents processeurs.

$$\sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset a}} distance(i,j) = \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset PE_u\subset a}} distance(i,j) + \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} distance(i,j) \tag{5.5}$$

Dans le premier cas, ces distances dépendent de l'ordonnancement des partitions dans le processeur PE_u : ces distances sont connues grâce à l'équation 3.8. Dans le second cas, nous ne connaissons qu'une valeur minimale de la somme puisque chaque distance dépend de la pire latence de communication et de la période de la partition destinatrice (cf. équation 3.9). Cette dernière somme peut-être décomposée en une partie variable et une partie fixe comme nous le montrons dans l'équation ci-dessous :

$$\sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} distance(i,j) = \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} WCTT(i,j) + \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} T_j \tag{5.6}$$

Pour chaque chaîne, pour une allocation a donnée, il suffit alors de vérifier que :

$$\begin{aligned} \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} WCTT(i,j) \quad \leq \quad & D_{ch_k,max} - \sum_{P_i\in ch_k} C_i - \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} T_j \\ & - \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset PE_u\subset a}} distance(i,j) \end{aligned} \tag{5.7}$$

Un système tel que le VMSA est composé de plusieurs chaînes. Une même latence peut se retrouver dans plusieurs chaînes, et par conséquent une même latence peut être plus ou moins contrainte selon les chaînes où elle se situe. Cette contrainte doit être reportée dans le calcul des autres latences afin d'être le plus précis possible dans les

valeurs de WCTT valides. Par ailleurs, il faut prendre en compte que les latences de communication sont positives. Notre problème revient alors à résoudre un système de m inéquations à n inconnues, avec m le nombre de chaînes contraintes et n le nombre de latences maximales à trouver.

Nous illustrons la résolution d'un tel système sur un exemple très simple, ne correspondant pas forcément à notre problème de latence maximale.

5.1.3 Exemple illustratif

Prenons le système de 7 inégalités à 2 inconnues suivant :

$$\begin{cases} -4x_1 - x_2 & \leq -9 \\ -x_1 - 2x_2 & \leq -4 \\ -2x_1 + x_2 & \leq 0 \\ -x_1 - 6x_2 & \leq -6 \\ x_1 + 2x_2 & \leq 11 \\ 6x_1 + 2x_2 & \leq 17 \\ x_2 & \leq 4 \end{cases}$$

La méthode d'élimination de Fourier-Motzkin permet de résoudre ce système. Appliquons la phase de descente de cet algorithme explicité en 5.1.5 sur celui-ci. L'objectif est de trouver un système d'inéquations solutions à une inconnue.

La première étape consiste à éliminer la variable x_1. Pour cela, séparons la variable x_1 du reste des inéquations :

$$\begin{cases} -4x_1 & \leq -9 + x_2 \\ -x_1 & \leq -4 + 2x_2 \\ -2x_1 & \leq -x_2 \\ -x_1 & \leq -6 + 6x_2 \\ x_1 & \leq 11 - 2x_2 \\ 6x_1 & \leq 17 - 2x_2 \\ x_2 & \leq 4 \end{cases} \iff \begin{cases} x_1 & \geq \frac{9-x_2}{4} \\ x_1 & \geq 4 - 2x_2 \\ x_1 & \geq \frac{x_2}{2} \\ x_1 & \geq 6 - 6x_2 \\ x_1 & \leq 11 - 2x_2 \\ x_1 & \leq \frac{17-2x_2}{6} \\ x_2 & \leq 4 \end{cases}$$

Nous en déduisons les conditions minimales et maximales sur x_1 :

$$\max(\frac{9 - x_2}{4}, 4 - 2x_2, \frac{x_2}{2}, 6 - 6x_2) \leq x_1 \leq \min(11 - 2x_2, \frac{17 - 2x_2}{6}) \tag{5.8}$$

La deuxième étape consiste à remplacer x_1 par l'encadrement obtenu précédemment. Le nouveau système S' après l'élimination de la variable x_1 est :

$$\begin{cases} \frac{9-x_2}{4} & \leq 11-2x_2 \\ 4-2x_2 & \leq 11-2x_2 \\ \frac{x_2}{2} & \leq 11-2x_2 \\ 6-6x_2 & \leq 11-2x_2 \\ \frac{9-x_2}{4} & \leq \frac{17-2x_2}{6} \\ 4-2x_2 & \leq \frac{17-2x_2}{6} \\ \frac{x_2}{2} & \leq \frac{17-2x_2}{6} \\ 6-6x_2 & \leq \frac{17-2x_2}{6} \\ x_2 & \leq 4 \end{cases} \iff \begin{cases} 9-x_2 & \leq 44-8x_2 \\ 4-2x_2 & \leq 11-2x_2 \\ x_2 & \leq 22-4x_2 \\ 6-6x_2 & \leq 11-2x_2 \\ 34-6x_2 & \leq 68-8x_2 \\ 24-12x_2 & \leq 17-2x_2 \\ 6x_2 & \leq 34-4x_2 \\ 36-36x_2 & \leq 17-2x_2 \\ x_2 & \leq 4 \end{cases} \iff \begin{cases} x_2 & \leq 5 \\ 4 & \leq 11 \\ x_2 & \leq \frac{22}{5} \\ -x_2 & \leq \frac{5}{4} \\ x_2 & \leq 17 \\ -x_2 & \leq \frac{-7}{10} \\ x_2 & \leq \frac{34}{10} \\ -x_2 & \leq \frac{-19}{34} \\ x_2 & \leq 4 \end{cases}$$

Nous en déduisons les conditions minimales et maximales sur x_2 :

$$\max(\frac{-5}{4}, \frac{7}{10}, \frac{19}{34}) \leq x_2 \leq \min(5, \frac{22}{5}, 17, \frac{34}{10}, 4) \Leftrightarrow \frac{7}{10} \leq x_2 \leq \frac{34}{10} \tag{5.9}$$

Le système S admet donc une solution réelle dans $\mathbb{R}^2$.

Appliquons maintenant la phase de remontée. Choisissons une valeur de x_2 dans l'intervalle $[\frac{7}{10}, \frac{34}{10}]$, par exemple $x_2 = 2$. Les conditions sur x_1 nous donne :

$$\max(\frac{7}{4}, 0, 1, -6) \leq x_1 \leq \min(7, \frac{13}{6}) \Leftrightarrow \frac{7}{4} \leq x_1 \leq \frac{13}{6} \tag{5.10}$$

Sur la droite de coordonnée $x_2 = 2$, x_1 peut prendre n'importe quelle valeur entre $\frac{7}{4}$ et $\frac{13}{6}$ pour satisfaire le système S. Une solution quelconque du système S est $x = [2\ 2]^T$.

5.1.4 Polyèdre des solutions

L'ensemble des solutions d'un système d'inégalités linéaires est défini par le polyèdre :

$$P = \{x \in \mathbb{R}^n | Gx \leq h\} \tag{5.11}$$

où G est une matrice de m lignes et n colonnes et h un vecteur réel de m composantes.

Pour obtenir le polyèdre solution par une méthode graphique, il faut transformer chaque inéquation en une équation afin de définir les limites de l'espace des solutions. La Figure 5.1 montre les étapes pour déterminer l'espace des solutions du couple (x_1, x_2)

associé au système d'inéquation de notre exemple illustratif en 5.1.3 rappelé ci-dessous :

$$\begin{cases} -4x_1 - x_2 & \leq -9 \\ -x_1 - 2x_2 & \leq -4 \\ -2x_1 + x_2 & \leq 0 \\ -x_1 - 6x_2 & \leq -6 \\ x_1 + 2x_2 & \leq 11 \\ 6x_1 + 2x_2 & \leq 17 \\ x_2 & \leq 4 \end{cases}$$

La première étape consiste à définir l'ensemble des solutions possibles pour le couple (x_1, x_2) avec la première inéquation du système représenté Figure 5.1a. La deuxième étape associe au premier ensemble solution les couples solutions de la seconde inéquation Figure 5.1b. Et ainsi de suite, nous ajoutons à chaque fois une nouvelle inéquation du système Figures 5.1c, 5.1d, 5.1e, 5.1f afin de réduire l'espace des solutions.

La Figure 5.2 trace toutes les droites associées à chaque inégalité du système S. L'ensemble des solutions valides consiste à ne garder que les valeurs de x_2 supérieures à $9-4x_1$, $\frac{4-x_1}{2}$ et $\frac{6-x_1}{6}$ et inférieures à $2x_1$, $\frac{11-x_1}{2}$, $\frac{17-6x_1}{2}$ et 4. Nous représentons l'ensemble des coordonnées (x_1, x_2) respectant le système S dans la partie hachurée Figure 5.2.

5.1.5 Résolution par la méthode d'élimination de Fourier-Motzkin

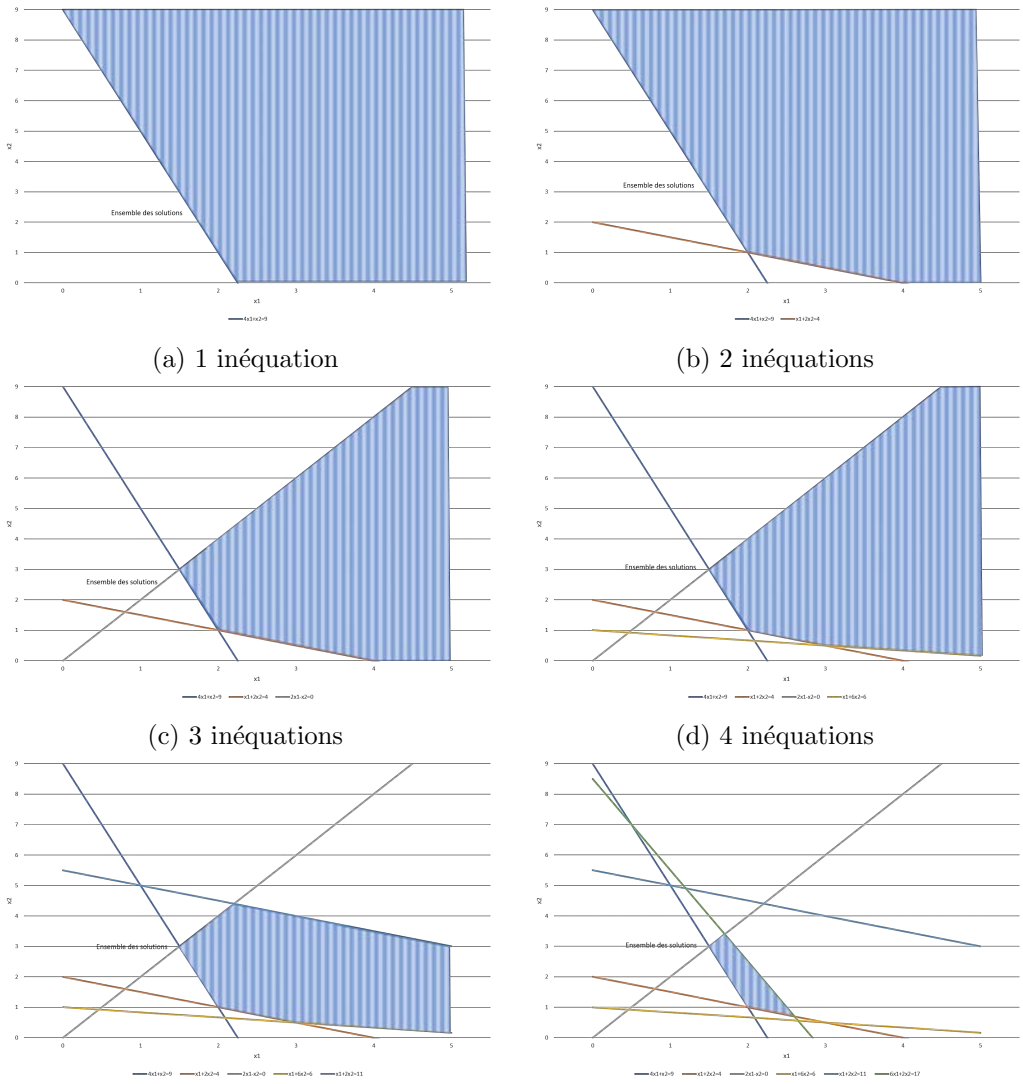
La résolution d'un système de m inéquations linéaires à n inconnues a été introduite par Fourier en 1827 [114], puis redécouverte par Dines en 1918-9 [115] et Motzkin en 1936 [116]. (Le lecteur peut se référer au livre de Schrijver [117] pour plus de détails).

Soit G une matrice réelle ayant m lignes et n colonnes, h un vecteur de $\mathbb{R}^m$ et x un vecteur contenant les variables du système de taille $m \times 1$. Un tel système s'écrit sous la forme :

$$Gx \leq h. \tag{5.12}$$

Chaque coefficient de G noté $g_{i,j}$ correspond au coefficient associé à la variable x_j. Dans notre cas, x_j est une des latences à trouver tandis que h_j représente la valeur fixe à ne pas dépasser donnée par le second membre de l'équation 5.7.

Nous expliquons ci-dessous l'objectif de la méthode d'élimination de Fourier-Motzkin, puis l'algorithme associé dans le cas général afin de l'implémenter dans notre étude. Nous détaillons ensuite la représentation des solutions ainsi que la complexité de cet



(a) 1 inéquation

(b) 2 inéquations

(c) 3 inéquations

(d) 4 inéquations

(e) 5 inéquations

(f) 6 inéquations

FIGURE 5.1 Obtention du polyèdre solution du système d'inéquations par méthode graphique

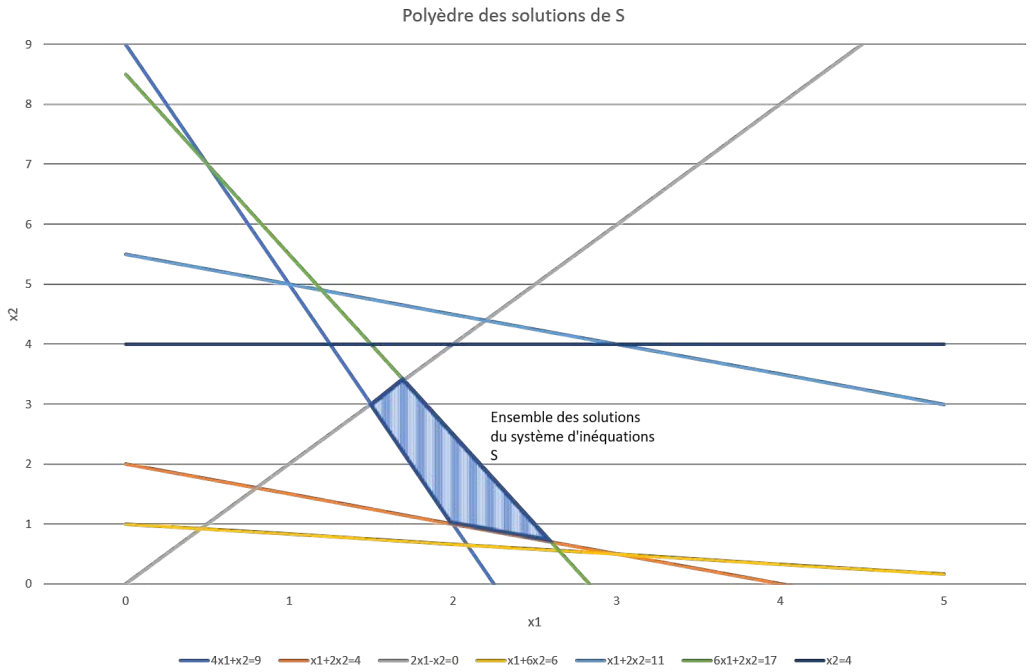


FIGURE 5.2 Ensemble des solutions du système S

algorithme.

5.1.5.1 Objectif de la méthode

L'objectif de la méthode de Fourier-Motzkin est de savoir s'il existe une région de l'espace dans $\mathbb{R}^m$ solution du système d'inéquations linéaires 5.12. Cette méthode consiste à éliminer les variables du système, générant de nouvelles inégalités ôtées des variables éliminées, jusqu'à obtenir un système à une inconnue que nous pouvons résoudre. Elle se base sur deux phases :

1. une phase de descente : elle permet de vérifier l'existence d'au moins une solution du système d'inéquations linéaires 5.12,
2. une phase de remontée : elle permet de trouver une solution quelconque si elle existe.

La méthode d'élimination de Fourier-Motzkin s'apparente à la méthode de pivot de Gauss-Jordan, sauf qu'elle s'applique à des inégalités plutôt qu'à des égalités.

5.1.5.2 Algorithme de Fourier-Motzkin

Nous décrivons les deux phases de l'algorithme.

— Phase de descente :
Soit G une matrice $\mathcal{M}_{m,n}(\mathbb{R})$ et h un vecteur de $\mathbb{R}^m$. Considérons le système d'inégalités linaires suivant : $S = \{Gx \leq h\}$ avec $x \in \mathbb{R}^m$. L'élimination des variables x_1, ..., x_l consiste à calculer un nouveau système S' ayant les mêmes solutions que S sur les variables restantes x_{l+1}, ..., x_m. Considérons l'élimination de la première variable x_1 du système S. Nous multiplions chaque ligne du système S par un scalaire tel que le coefficient de la première colonne appartienne à $\{-1, 0, 1\}$. Le système S peut se réécrire de la manière suivante :

$$S \begin{cases} +x_1 + g_{i,2}x_2 + \cdots + g_{i,n}x_n \leq h_i & , \ i \in \{1, \cdots, m_1\}, \\ -x_1 + g_{j,2}x_2 + \cdots + g_{j,n}x_n \leq h_j & , \ j \in \{m_1+1, \cdots, m_2\}, \\ 0x_1 + g_{k,2}x_2 + \cdots + g_{k,n}x_n \leq h_k & , \ k \in \{m_2+1, \cdots, m\}. \end{cases} \tag{5.13}$$

En considérant les deux premières lignes du système S en 5.13, nous déduisons une borne inférieure et une borne supérieure à la variable x_1 :

$$\max_{j \in \{m_1+1, \cdots, m_2\}} (\sum_{l=2}^{n} g_{j,l}x_l - h_j) \leq x_1 \leq \min_{i \in \{1, \cdots, m_1\}} (h_i - \sum_{l=2}^{n} g_{i,l}x_l) \tag{5.14}$$

La variable x_1 peut être éliminée du système S. Nous obtenons un système réduit S' de la forme suivante :

$$S' \begin{cases} \sum_{l=2}^{n} (g_{i,l} + g_{j,l})x_l \leq h_i + h_j & , \ i \in \{1, \cdots, m_1\}, j \in \{m_1+1, \cdots, m_2\}, \\ \sum_{l=2}^{n} g_{k,l}x_l \leq h_k & , \ k \in \{m_2+1, \cdots, m\}. \end{cases} \tag{5.15}$$

qui ne contient plus que $n-1$ variables. Nous réitérons cet algorithme jusqu'à obtenir un système à 1 inconnue.

— Phase de remontée :
À la fin de la phase de descente, nous savons s'il existe au moins une solution au système S. Si elle existe, nous pouvons remonter grâce aux conditions sur x_n aux conditions sur x_{n-1} et ainsi de suite jusqu'à la variable x_1.

5.2 Application de l'algorithme de Fourier-Motzkin à la recherche des latences maximales admissibles

Comme nous l'avons montré en 5.1.2, la recherche des latences maximales admissibles pour une allocation donnée peut se modéliser par un système d'inéquations données par l'inéquation 5.7. En 5.2.1, nous proposons un modèle prenant en compte la connexion de

différents équipements ainsi que le choix d'une allocation distribuée. Nous adaptons en 5.2.2 l'algorithme d'élimination de Fourier-Motzkin à notre modèle. Nous évaluons son efficacité sur l'étude de cas « VMSA » en 5.2.3 et déterminons sa complexité en 5.2.4.

5.2.1 Modèle de la recherche des latences maximales

Un système est composé d'un ensemble de chaînes à respecter temporellement. Les partitions les composant peuvent être de plusieurs natures et ainsi être associées à des classes d'équipements, i.e. les E/S telles que les capteurs et les actionneurs, les écrans, les enregistreurs, les passerelles et les calculateurs. Nous modélisons dans un premier temps la connexion entre tous les éléments constituant une architecture. Dans un second temps, nous nous intéressons au choix d'un ordonnancement dans une allocation distribuée afin de résoudre le système d'inéquations obtenu.

5.2.1.1 Connexion des éléments d'une architecture

Une architecture de type avionique est composée de divers équipements tels que des capteurs, des actionneurs, des écrans, des calculateurs, etc. Selon la nature des partitions, celles-ci sont allouées à ces équipements particuliers. Deux partitions de différentes natures, allouées par conséquent sur deux équipements différents, peuvent communiquer ensemble : une partition échantillonnant les données d'un capteur, e.g. un capteur de pression, est allouée au niveau de celui-ci, tandis qu'une partition de traitement, e.g. retourner la vitesse de l'aéronef, ne peut donner une information qu'en fonction de la donnée produite en amont. Ces partitions sont donc dépendantes et reliées par un canal APEX [1].

Dans un aéronef, il existe différents systèmes tels que le VMSA, le FADEC pour le contrôle moteur, le FWS pour alerter le pilote lorsqu'un problème apparaît par exemple. Ces différents systèmes peuvent partager les mêmes équipements grâce à la ségrégation des partitions de la norme ARINC 653 [1]. Ils peuvent aussi partager les mêmes moyens de communication. Pour évaluer les latences de communication dans le cadre d'un seul système, il faut avoir par conséquent une connaissance de tous les flux, i.e. les canaux APEX, pouvant emprunter les mêmes chemins que les flux qui nous intéressent. Nous faisons une sur-approximation de la latence maximale subie par chaque canal APEX afin de simplifier les calculs.

L'ensemble des canaux APEX est mappé sur différents moyens de communication connectant les différents équipements. Nous considérons que des équipements de même nature sont connectés ensemble par des moyens de communication identiques. Par exemple,

un ensemble de calculateurs est connecté grâce un réseau commuté de type AFDX, ou bien par un bus de type MIL-STD-1553 (ceci n'est pas une énumération exhaustive). Des équipements de différentes natures, par exemple un capteur et un calculateur, peuvent être connectés quant à eux soit par le même moyen de communication tel qu'un lien mono-directionnel comme un ARINC 429 ou un bus de terrain comme un CAN, soit par l'interconnexion de différents réseaux. Une communication entre deux partitions est alors spécifiée par la nature des équipements où elles sont allouées, en prenant en compte l'ordre (nature source - nature destination), e.g. $com_{\text{E/S}-calculateur}$, $com_{calculateur-\text{E/S}}$ ou $com_{calculateur-ecran}$ par exemple. Nous faisons l'hypothèse que les communications de mêmes natures ont la même latence maximale.

Différents équipements tels que certains capteurs ou actionneurs peuvent être connectés à des passerelles afin de réduire le nombre de connexions directes de ces derniers aux calculateurs. Ces passerelles peuvent être des IOMs, des RDCs, etc. Ces dernières sont modélisées par une unique partition, noté P_{GW} s'exécutant à une période donnée. Son rôle est d'envoyer ponctuellement les données reçues par les divers équipements vers les équipements destinations. La latence subie par un flux traversant une passerelle est la somme des communications entre l'élément source et la passerelle, cette dernière et l'équipement destination, et l'attente maximale pour transférer le message dans la passerelle.

Regrouper les différents flux dans des classes de communication permet ainsi de réduire la taille de notre système d'inéquations.

5.2.1.2 Choix d'un ordonnancement pour une allocation valide

La contrainte principale de l'algorithme d'élimination de Fourier-Motzkin est de déterminer s'il existe au moins une solution à un système d'inéquations de la forme $Gx \leq h$. Cette solution n'est identifiée qu'à la fin de la phase de descente, à condition que le résultat donné pour la dernière variable soit cohérent, i.e. $c \leq x_u \leq d$ avec $c \leq d$. Pour s'affranchir d'une recherche infructueuse, seules les allocations valides cherchées avec des latences de communication nulles sont modélisées par un système d'inéquations décrit en 5.1.2.

5.2.2 Résolution par la méthode d'élimination de Fourier-Motzkin

Chaque système avionique est composé d'un nombre de partitions de natures différentes définies par classe. Chaque classe de partitions est allouée dans une classe d'équipements (ces dernières sont détaillées en 2.1.1). Les types de communication se déduisent des

natures des partitions : $com_{nature1,nature2}$ correspond à une communication entre une partition émettrice de nature $nature1$ et une partition destinatrice de nature $nature2$. En considérant toutes les classes de partitions définies dans cette thèse, nous pouvons énumérer huit types de communications illustrés Figure 5.3 :

1. $com_{E/S-Calculateur}$: le message envoyé par un capteur ne traverse qu'un seul réseau pour aller au calculateur,

2. $com_{E/S-GW}$: la donnée envoyée par le capteur passe nécessairement par une passerelle,

3. $com_{GW-Calculateur}$: la passerelle envoie les données qu'elle a reçu à un calculateur,

4. $com_{Calculateur-Calculateur}$: une partition d'un calculateur envoie une donnée à une autre partition d'un calculateur,

5. $com_{Calculateur-Ecran}$: la donnée envoyée par une partition d'un calculateur ne traverse qu'un seul réseau pour accéder à l'écran,

6. $com_{Calculateur-E/S}$: les données sont transmises sur un seul réseau du calculateur vers les E/S,

7. $com_{Calculateur-GW}$: les données émises par une partition d'un calculateur doivent traverser une passerelle avant d'accéder à des équipements tels que des actionneurs ou des enregistreurs,

8. $com_{GW-E/S}$: la passerelle transmet les données reçues aux équipements E/S.

Leur nombre peut être étoffé suivant les besoins de l'utilisateur.

Seules les partitions communicantes placées sur deux processeurs distincts subissent la latence du-des moyen-s de communication traversé-s (le lecteur peut se référer à la partie 3.1.5 de ce manuscrit pour plus de détails). Dans ce cas, le WCTT dépend du type de communication et est noté $WCTT_{nature1,nature2}$. Si nous reprenons la création du système d'inéquations définie en 5.1.2, ce système est composé d'inéquations basées

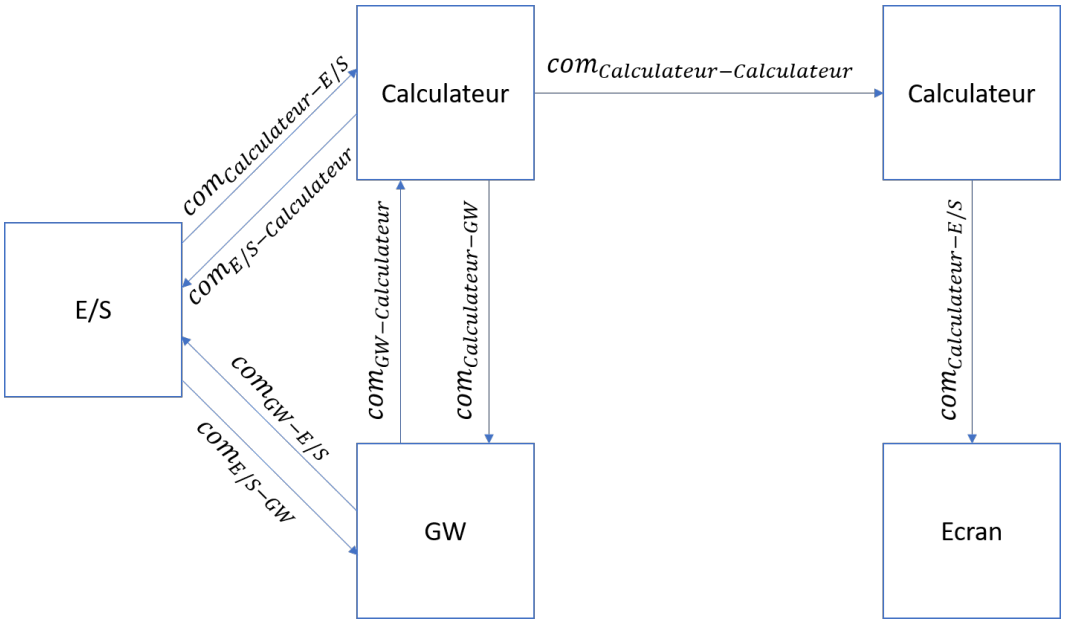


FIGURE 5.3 Types de communications

sur l'équation 5.7 de la forme suivante :

$$\begin{aligned}
\sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} WCTT(i,j) \;\leq\; & D_{ch_k,max} - \sum_{P_i\in ch_k} C_i - \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} T_j \\
& - \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset PE_u\subset a}} distance(i,j) \\
\Leftrightarrow \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} WCTT_{nature(i),nature(j)} \;\leq\; & D_{ch_k,max} - \sum_{P_i\in ch_k} C_i - \sum_{\substack{(P_i,P_j)\in ch_k \\ P_i\subset PE_u\subset a \\ P_j\subset PE_{v\neq u}\subset a}} T_j \\
& - \sum_{\substack{(P_i,P_j)\in ch_k \\ (P_i,P_j)\subset PE_u\subset a}} distance(i,j) \qquad (5.16)
\end{aligned}$$

où $WCTT(i,j)$ est remplacé par $WCTT_{nature(i),nature(j)}$.

Le système d'inéquations obtenu pour chaque allocation valide avec des latences nulles est résolu grâce à la librairie FM (pour Fourier-Motzkin) créée en langage C par Pouchet [118].

Nous illustrons cet algorithme sur l'exemple illustratif en introduction du chapitre 3 où un écran, deux E/S et une passerelle (GW) sont ajoutés. L'ensemble des partitions

peut être séparé en quatre classes :

$$\mathcal{P} = \{\mathcal{P}_{Distribuees}, \mathcal{P}_{Ecran}, \mathcal{P}_{E/S}, \mathcal{P}_{GW}\}.$$

$\mathcal{P}_{Distribuees} = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ représente l'ensemble des partitions qui nécessite d'être alloué sur au plus trois processeurs, i.e. $\mathcal{E}_{Distribues} = \{PE_1, PE_2, PE_3\}$. $\mathcal{P}_{Ecran} = \{P_{Ecran}\}$ est un ensemble d'une partition alloué sur un équipement de type écran PE_{Ecran} qui va gérer l'affichage des informations reçues. $\mathcal{P}_{E/S} = \{P_{E/S,1}, P_{E/S,2}\}$ est un ensemble de capteurs et actionneurs alloué dans l'ensemble des équipements d'entrées/sorties $\mathcal{E}_{E/S}$. $\mathcal{P}_{GW} = \{P_{GW}\}$ est composé d'une seule partition qui transfère les messages reçus d'un équipement sur un réseau particulier vers un équipement se situant sur un autre réseau. Les caractéristiques de ces partitions sont données dans la Table 5.1, qui donne pour chaque partition P_i son pire temps d'exécution (égal à C_i) et sa période (T_i).

TABLE 5.1 Spécification temps-réel des paramètres de l'exemple illustratif de l'algorithme d'élimination de Fourier-Motzkin

Partitions	C_i (ms)	T_i (ms)
P_1	3	10
P_2	2	10
P_3	2	20
P_4	4	40
P_5	1	40
P_6	4	40
P_{Ecran}	10	50
$P_{E/S,1}$	0	50
$P_{E/S,2}$	0	25
P_{GW}	0	25

Cinq chaînes de communication sont traitées :

$$\mathcal{C}h = \{ch_1, ch_2, ch_3, ch_4, ch_5\}$$

$$ch_1 = \{P_1, P_2, P_3\}, D_{ch_1,max} = 30ms$$

$$ch_2 = \{P_2, P_5\}, D_{ch_2,max} = 40ms$$

$$ch_3 = \{P_4, P_5, P_6\}, D_{ch_3,max} = 60ms$$

$$ch_4 = \{P_{E/S,1}, P_4, P_5, P_{GW}, P_{E/S,2}\}, D_{ch_4,max} = 100ms$$

$$ch_5 = \{P_{E/S,1}, P_4, P_5, P_{Ecran}\}, D_{ch_5,max} = 120ms$$

Dans cet exemple, parmi les huit types de communication définis précédemment, seulement $com_{E/S-Calculateur}$, $com_{Calculateur-Calculateur}$, $com_{Calculateur-Ecran}$, $com_{Calculateur-GW}$ et $com_{GW-E/S}$ ont un impact sur les contraintes de bout-en-bout. Les WCTT associés à ces communications sont pris nuls lors de la recherche d'architectures distribuées valides.

Si nous prenons en compte l'écran, les E/S ou la passerelle GW, et par conséquent les chaînes ch_4 et ch_5, nous passons de 25 allocations valides à 9. Ces neuf allocations valides sont possibles sur seulement deux et trois processeurs.

Afin d'appliquer la méthode d'élimination de Fourier-Motzkin, il est nécessaire de générer un ordonnancement valide pour obtenir le système d'inéquations à résoudre. Notre heuristique définie dans le Chapitre 4 peut répondre à ce besoin puisqu'il maximise localement les durées de communication. Prenons par exemple une allocation possible sur trois processeurs illustré Figure 5.4. Cette allocation valide obtenue grâce à l'heuristique consiste à allouer la partition P_1 sur le premier processeur, les partitions P_2 à P_5 sur le deuxième processeur et la partition P_6 sur le troisième processeur disponible. Les ordonnancements dans chacun de ces processeurs sont donnés Figure 5.4.

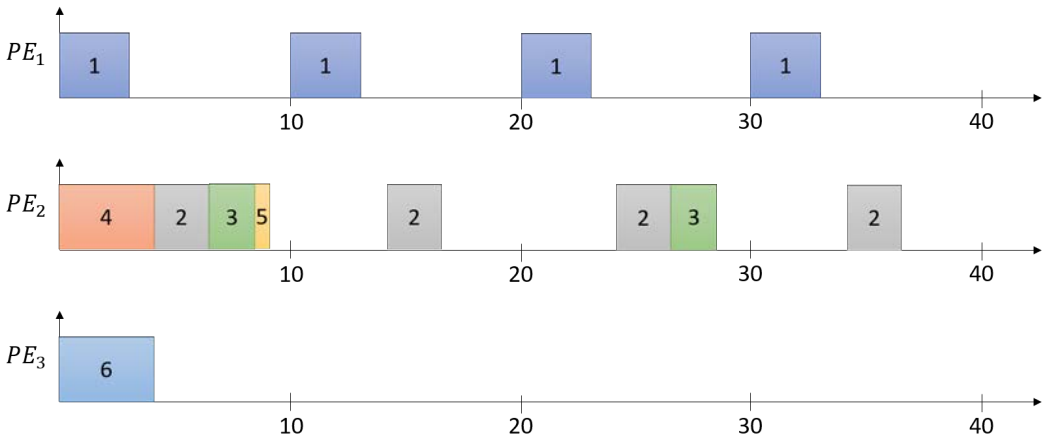


FIGURE 5.4 Une allocation valide sur 3 processeurs de l'exemple Tableau 5.1

Le système d'inéquations généré est le suivant :

$$S_{exemple,1} = \begin{cases} C_1 + distance(1,2) + C_2 + distance(2,3) + C_3 & \leq D_{ch_1,max} \\ C_2 + distance(2,5) + C_5 & \leq D_{ch_2,max} \\ C_4 + distance(4,5) + C_5 + distance(5,6) + C_6 & \leq D_{ch_3,max} \\ C_{E/S_1} + distance(E/S_1,4) + C_4 + distance(4,5) + C_5 & \\ +distance(5,GW) + C_{GW} + distance(GW,E/S_2) + C_{E/S_2} & \leq D_{ch_4,max} \\ C_{E/S_1} + distance(E/S_1,4) + C_4 + distance(4,5) + C_5 & \\ +distance(5,Ecran) + C_{Ecran} & \leq D_{ch_5,max} \end{cases}$$

Seules les partitions P_2, P_3, P_4 et P_5 sont sur le même processeur. Les communications sont donc locales entre P_2 et P_3, P_2 et P_5, P_4 et P_5. Ainsi, les distances entre ces partitions sont respectivement de 10, 32 et 4 millisecondes selon l'équation 3.8. Dans tous les autres cas, la distance entre les différentes partitions est donnée par l'équation 3.9 :

$$distance(i,j) = WCTT(i,j) + T_j.$$

Le système $S_{exemple,1}$ est équivalent à :

$$S_{exemple,1} \Leftrightarrow \begin{cases} distance(1,2) \leq D_{ch_1,max} - C_1 - C_2 - distance(2,3) - C_3 \\ 0 \leq D_{ch_2,max} - C_2 - distance(2,5) - C_5 \\ distance(5,6) \leq D_{ch_3,max} - C_4 - distance(4,5) - C_5 - C_6 \\ distance(E/S_1,4) + distance(5,GW) + distance(GW,E/S_2) \\ \qquad \leq D_{ch_4,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{GW} - C_{E/S_2} \\ distance(E/S_1,4) + distance(5,Ecran) \\ \qquad \leq D_{ch_5,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{Ecran} \end{cases}$$

$$\Leftrightarrow \begin{cases} WCTT_{Calculateur,Calculateur} + T_2 \leq D_{ch_1,max} - C_1 - C_2 - distance(2,3) - C_3 \\ 0 \leq D_{ch_2,max} - C_2 - distance(2,5) - C_5 \\ WCTT_{Calculateur,Calculateur} + T_6 \leq D_{ch_3,max} - C_4 - distance(4,5) - C_5 - C_6 \\ WCTT_{E/S,Calculateur} + T_4 + WCTT_{Calculateur,GW} + T_{GW} + WCTT_{GW,E/S} \\ \qquad +T_{E/S_2} \leq D_{ch_4,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{GW} - C_{E/S_2} \\ WCTT_{E/S,Calculateur} + T_4 + WCTT_{Calculateur,Ecran} + T_{Ecran} \\ \qquad \leq D_{ch_5,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{Ecran} \end{cases}$$

$$\Leftrightarrow \begin{cases} WCTT_{Calculateur,Calculateur} \leq D_{ch_1,max} - C_1 - C_2 - distance(2,3) - C_3 - T_2 \\ 0 \leq D_{ch_2,max} - C_2 - distance(2,5) - C_5 \\ WCTT_{Calculateur,Calculateur} \leq D_{ch_3,max} - C_4 - distance(4,5) - C_5 - C_6 - T_6 \\ WCTT_{E/S,Calculateur} + WCTT_{Calculateur,GW} + WCTT_{GW,E/S} \\ \quad \leq D_{ch_4,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{GW} - C_{E/S_2} \\ \quad -T_4 - T_{GW} - T_{E/S_2} \\ WCTT_{E/S,Calculateur} + WCTT_{Calculateur,Ecran} \\ \quad \leq D_{ch_5,max} - C_{E/S_1} - C_4 - distance(4,5) - C_5 - C_{Ecran} - T_4 - T_{Ecran} \end{cases}$$

Le système final $S_{exemple,1}$ à résoudre se résume à :

$$S_{exemple,1} \Leftrightarrow \begin{cases} WCTT_{Calculateur,Calculateur} \leq 30 - 3 - 2 - 10 - 2 - 10 = 3 \\ WCTT_{Calculateur,Calculateur} \leq 60 - 4 - 4 - 1 - 4 - 40 = 7 \\ WCTT_{E/S,Calculateur} + WCTT_{Calculateur,GW} + WCTT_{GW,E/S} \\ \quad \leq 100 - 0 - 4 - 4 - 1 - 0 - 0 - 40 - 25 - 25 = 1 \\ WCTT_{E/S,Calculateur} + WCTT_{Calculateur,Ecran} \\ \quad \leq 120 - 0 - 4 - 4 - 1 - 10 - 40 - 50 = 11 \end{cases}$$

Grâce à la méthode de résolution de Fourier-Motzkin, nous obtenons les bornes suivantes sur les différents WCTTs :

$$\begin{cases} 0 \leq & WCTT_{E/S,GW} \\ 0 \leq & WCTT_{E/S,Calculateur} \leq 1 \\ 0 \leq & WCTT_{GW,Calculateur} \\ 0 \leq & WCTT_{Calculateur,Calculateur} \leq 3 \\ 0 \leq & WCTT_{Calculateur,Ecran} \leq 11 - WCTT_{E/S,Calculateur} \\ 0 \leq & WCTT_{Calculateur,E/S} \\ 0 \leq & WCTT_{Calculateur,GW} \leq 1 - WCTT_{E/S,Calculateur} \\ 0 \leq & WCTT_{GW,E/S} \leq 1 - WCTT_{E/S,Calculateur} - WCTT_{Calculateur,GW} \end{cases}$$

Nous constatons que les latences associées aux communications d'une entrée de type capteur au calculateur et d'un calculateur vers une sortie telle qu'un actionneur sont très contraintes pour valider les exigences de bout-en-bout. Il est difficile dans ce cas de

choisir des réseaux qui peuvent avoir une latence significative par rapport aux systèmes à distribuer. L'intégrateur système va choisir de préférence des liens directs tels qu'un ARINC 429 par exemple.

5.2.3 Étude de cas

Dans cette partie, nous considérons dans un premier temps tous les éléments qui composent le système « VMSA », i.e. les calculateurs, les capteurs, les actionneurs, les enregistreurs et les écrans. Un équipement supplémentaire est ajouté dans un second temps à cet ensemble : une passerelle. Nous comparons dans cette partie les différences entre les latences de communication valides obtenues par l'heuristique et par la méthode d'élimination de Fourier-Motzkin, ainsi que les modifications de ces latences lors de l'ajout de passerelles.

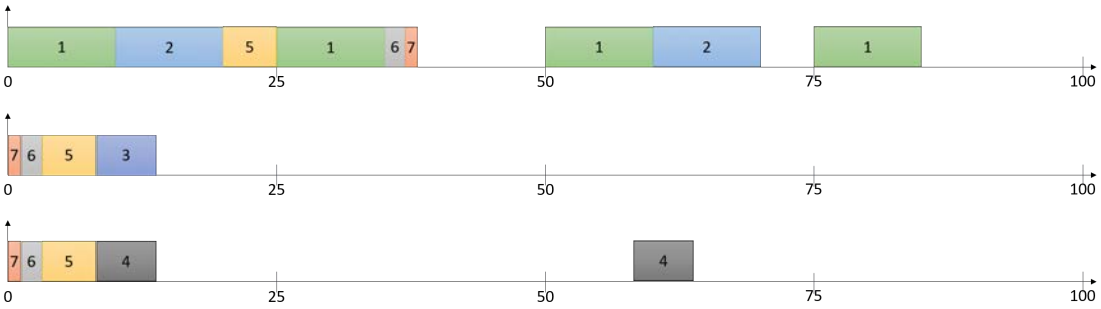
5.2.3.1 Architecture sans passerelle

Nous considérons dans cette première partie le système « VMSA » sans passerelle. Pour effectuer des allocations distribuées, nous spécialisons un certain nombre d'équipements afin d'assigner les partitions pré-allouées sur ces derniers, i.e. les capteurs, les actionneurs, les enregistreurs et les écrans. Les partitions restantes, i.e. les partitions à distribuer, sont quant à elles allouées sur un plus grand nombre de processeurs de type « calculateur ». Nous traitons ici l'allocation des 7 partitions d'une voie d'un AMC pour la lisibilité des résultats.

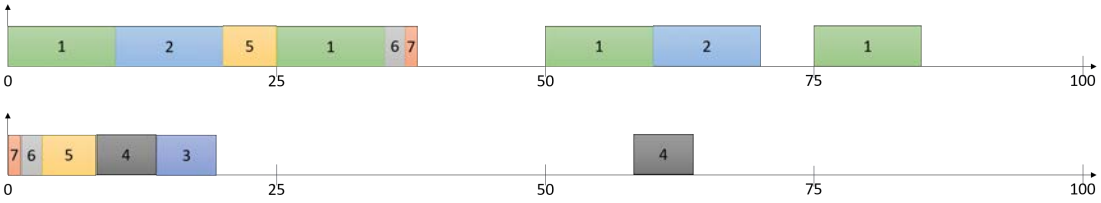
Une distribution plus importante des partitions de type « calculateur » est obtenue grâce à l'heuristique présentée partie 4 en considérant que les latences de communication sont nulles. Comme en 4.7, nous retrouvons les cinq allocations valides de notre étude de cas illustrées Figure 5.5 avec leurs ordonnancements.

À chaque allocation valide correspond un système d'inéquations solution que nous notons $S_{i,i\in 1..5}$, chaque i correspondant à une allocation illustrée Figure 5.5. Parmi ces cinq systèmes d'inéquations solutions définis ci-dessous, deux sont identiques : S_2 et S_3.

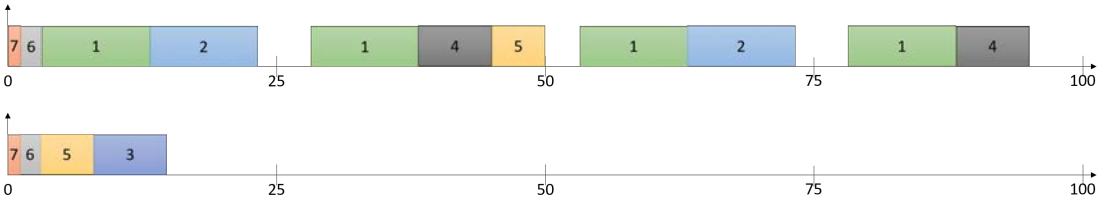
$$S_1 = \begin{cases} WCTT_{E/S,Calculateur} \leq 0,2865 \\ WCTT_{Calculateur,Calculateur} \leq 0,2865 - WCTT_{E/S,Calculateur} \\ WCTT_{Calculateur,Ecran} \leq 0,275 - WCTT_{E/S,Calculateur} \\ WCTT_{Calculateur,E/S} \leq \min(0,275 - WCTT_{Calculateur,Calculateur}; \\ \qquad\qquad 0,2865 - WCTT_{E/S,Calculateur} - WCTT_{Calculateur,Calculateur}) \end{cases}$$



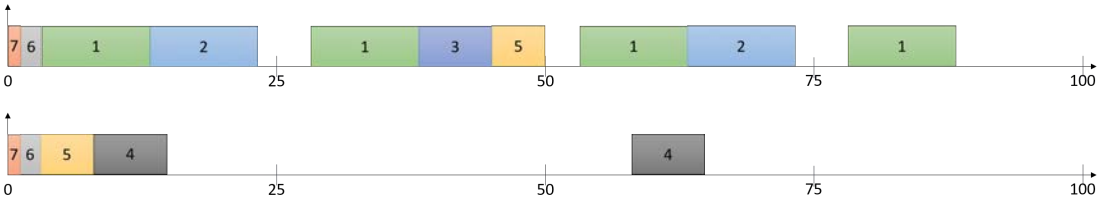
(a) Allocation 1 sur 3 processeurs



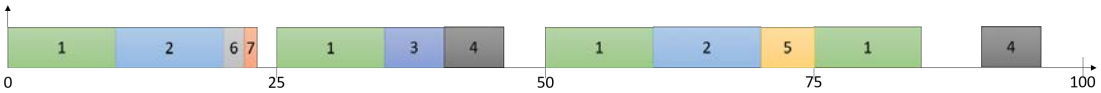
(b) Allocation 2 sur 2 processeurs



(c) Allocation 3 sur 2 processeurs



(d) Allocation 4 sur 2 processeurs



(e) Allocation 5 sur 1 processeur

FIGURE 5.5 Allocations et ordonnancements des 5 allocations valides du système VMSA - Hypothèse : distribution d'une seule voie de l'AMC

$$
\begin{aligned}
S_2 = S_3 = & \begin{cases}
WCTT_{E/S,Calculateur} \leq 0,2865 \\
WCTT_{Calculateur,Calculateur} \leq 0,2865 - WCTT_{E/S,Calculateur} \\
WCTT_{Calculateur,Ecran} \leq 0,275 - WCTT_{E/S,Calculateur} \\
WCTT_{Calculateur,E/S} \\
\qquad \leq 0,2865 - WCTT_{E/S,Calculateur} - WCTT_{Calculateur,Calculateur}
\end{cases} \\
S_4 = & \begin{cases}
WCTT_{E/S,Calculateur} \leq 0,275 \\
WCTT_{Calculateur,Calculateur} \leq 0,3265 \\
WCTT_{Calculateur,Ecran} \leq 0,275 - WCTT_{E/S,Calculateur} \\
WCTT_{Calculateur,E/S} \leq \min(0,3265 - WCTT_{Calculateur,Calculateur}; \\
\qquad 0,3715 - WCTT_{E/S,Calculateur})
\end{cases} \\
S_5 = & \begin{cases}
WCTT_{E/S,Calculateur} \leq 0,275 \\
WCTT_{Calculateur,Ecran} \leq 0,275 - WCTT_{E/S,Calculateur} \\
WCTT_{Calculateur,E/S} \leq 0,3715 - WCTT_{E/S,Calculateur}
\end{cases}
\end{aligned}
$$

Nous constatons que l'allocation et l'ordonnancement des partitions sur le même nombre de processeurs, ici deux, ne permettent pas d'atteindre les mêmes bornes de latences comme nous le montre les systèmes S_2 à S_4. Dans le cadre de l'allocation numéro 4 Figure 5.5d, les partitions P_1, P_2 et P_3 sont sur le même processeur. Dans cette configuration, il n'existe plus de communications inter-calculateurs, puisque la seule chaîne de communication ch_3 où apparaît cette latence a toutes ses partitions de type « calculateur » sur le même processeur. Dans les autres cas, cette latence dépend des autres étant donné que plusieurs communications différentes composent la chaîne de bout-en-bout ch_3, i.e. $WCTT_{E/S,Calculateur}$, $WCTT_{Calculateur,Calculateur}$ et $WCTT_{Calculateur,E/S}$.

5.2.3.2 Architecture avec passerelles

Une des contraintes fortes amenée par le système « VMSA » décrit dans l'Annexe A est la connexion des entrées/sorties sur l'ensemble des processeurs disponibles. Ce système ne peut plus accepter actuellement de nouvelles entrées/sorties car toutes les connectiques sont utilisées. Une première solution est une distribution plus forte des fonctions avioniques sur un plus grand nombre de processeurs afin d'augmenter le nombre de connecteurs disponibles. Une deuxième solution, qui peut être adjacente à la première, est d'intercaler des passerelles entre les E/S et les calculateurs.

Une passerelle est un système matériel et logiciel permettant de faire une liaison entre

deux réseaux qui peuvent être différents. Dans le cadre du système VMSA, la partition P_1 a besoin de la majorité des données des entrées/sorties. Cette partition doit obtenir toutes les dernières mises à jour des données à chaque exécution. Pour cette raison, nous prenons les caractéristiques suivantes pour les passerelles : $P_{GW} = (0,025; 0)$, i.e. une passerelle envoie les données qu'elle a reçu toutes les 25 millisecondes, en considérant que la génération des messages est négligeable. Nous considérons également que le réseau utilisé pour interconnecter la passerelle et les calculateurs est identique à celui utilisé pour interconnecter les calculateurs entre eux comme le montre la Figure 5.6. La latence de communication maximale entre une passerelle et un calculateur est égale à la latence de communication entre deux calculateurs :

$$WCTT_{GW,Calculateur} = WCTT_{Calculateur,Calculateur}.$$

Nous retrouvons les mêmes allocations et les mêmes ordonnancements que précédemment en ajoutant une passerelle entre les capteurs et les calculateurs lorsque les latences sont nulles. Néanmoins, les systèmes d'inéquations solutions sont différents. La latence $WCTT_{E/S,Calculateur}$ se décompose en la somme de deux latences : $WCTT_{E/S,GW}$ et $WCTT_{Calculateur,Calculateur}$, venant impacter le calcul du délai de bout-en-bout. Parmi les cinq systèmes d'inéquations solutions décrits ci-dessous, les trois premiers sont identiques.

$$S'_1 = S'_2 = S'_3 = \begin{cases} WCTT_{E/S,GW} \leq 0,2615 \\ WCTT_{Calculateur,Calculateur} \leq 0,13075 - \frac{WCTT_{E/S,GW}}{2} \\ WCTT_{Calculateur,Ecran} \leq 0,250 - WCTT_{E/S,GW} \\ \qquad -WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \leq 0,2615 - WCTT_{E/S,GW} \\ \qquad -2 \times WCTT_{Calculateur,Calculateur} \end{cases}$$

$$S'_4 = \begin{cases} WCTT_{E/S,GW} \leq 0,250 \\ WCTT_{Calculateur,Calculateur} \leq 0,250 - WCTT_{E/S,GW} \\ WCTT_{Calculateur,Ecran} \\ \qquad \leq 0,250 - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \leq \min(0,3265 - WCTT_{Calculateur,Calculateur}; \\ \qquad 0,3465 - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur}) \end{cases}$$

$$S'_5 = \begin{cases} WCTT_{E/S,GW} \leq 0,250 \\ WCTT_{Calculateur,Calculateur} \leq 0,250 - WCTT_{E/S,GW} \\ WCTT_{Calculateur,Ecran} \\ \quad \leq 0,250 - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \\ \quad \leq 0,3465 - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \end{cases}$$

L'ajout d'une passerelle dans notre étude de cas modifie fortement les contraintes sur les latences de communication et surtout sur la latence $WCTT_{Calculateur,Calculateur}$. Dans les trois premières allocations, cette latence maximale est divisée par deux, pouvant atteindre 130 millisecondes dans le meilleur des cas avec une passerelle au lieu de 261 millisecondes sans celle-ci. Dans la quatrième allocation, cette même latence est réduite et devient dépendante de la latence $WCTT_{E/S,GW}$. La latence $WCTT_{Calculateur,Calculateur}$ apparaît comme nouvelle inéquation dans le système solution S'_5 alors qu'elle n'existait pas dans S_5 auparavant. Selon les contraintes des systèmes, des choix peuvent être fait pour garder certaines E/S directement connectées aux calculateurs et d'autres regroupées derrière des passerelles configurées selon les besoins systèmes.

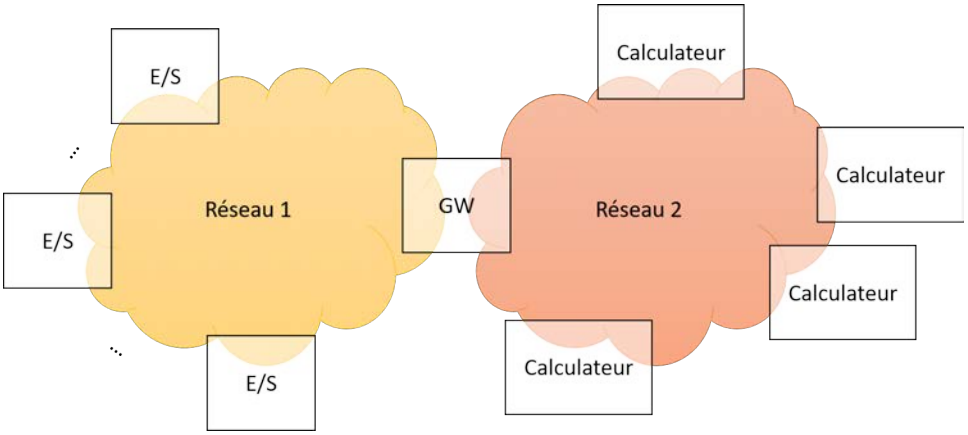


FIGURE 5.6 Réseau utilisé entre une passerelle et les calculateurs

5.2.3.3 Résultats

Les résultats ci-dessous ont été obtenus de deux façons. Dans les cas notés « Heuristique sans GW » Figure 5.7 et « Heuristique avec GW » Figure 5.9, nous avons lancé l'heuristique autant de fois qu'il y a d'ensemble de valeurs de WCTTs possibles. Dans les cas notés « FM sans GW » Figure 5.8 et « FM avec GW » Figure 5.10, l'heuristique

a été lancée une seule fois avec toutes les latences nulles, le nombre d'allocations valides étant obtenu grâce aux différents systèmes d'inéquations.

Que ce soit avec l'heuristique ou l'algorithme d'élimination de Fourier-Motzkin, à aucun moment un ordonnancement d'une allocation trouvé Figure 5.5 avec des latences nulles n'est modifié lorsque les différentes latences de communication varient. L'heuristique fige chaque partition allouée dans un processeur de façon à maximiser les latences de communication. Le seul cas où l'heuristique peut modifier l'ordonnancement d'une partition se situe seulement dans le cas où une chaîne reboucle sur un processeur : la dernière partition destinatrice sur ce processeur est alors ordonnancée de façon à réduire les temps d'attente entre l'arrivée d'une donnée sur le processeur et son utilisation par la partition destinatrice. Ce cas n'est pas présent dans notre étude.

Si nous comparons les Figures 5.7 et 5.8 qui correspondent au cas sans passerelle, nous constatons dans le cas de l'algorithme d'élimination de Fourier-Motzkin qu'il existe quatre allocations possibles au lieu de cinq dans le cas de l'heuristique. Cette différence provient de la première allocation : une des inéquations solutions obtenue grâce à l'algorithme d'élimination de Fourier-Motzkin lorsque $WCTT_{E/S,Calculateur} = 0$ spécifie que la valeur maximale de $WCTT_{Calculateur,E/S}$ ne doit pas dépasser :

$$WCTT_{Calculateur,E/S} \leq 0,275 - WCTT_{Calculateur,Calculateur},$$

or la limite donnée par l'heuristique est :

$$WCTT_{Calculateur,E/S} \leq 0,276 - WCTT_{Calculateur,Calculateur}.$$

L'algorithme d'élimination de Fourier-Motzkin perd 3102 allocations sur 743456 dû à un problème d'approximation dans l'implémentation de la librairie FM [118]. Les résultats sont ainsi identiques dans 99,6% des cas.

Si nous étudions plus particulièrement le nombre d'allocations possibles en fonction des paramètres $WCTT_{E/S,Calculateur}$, $WCTT_{Calculateur,Calculateur}$, $WCTT_{Calculateur,Ecran}$ et $WCTT_{Calculateur,E/S}$, nous constatons que les paramètres associés aux communications de et vers les E/S influent sur la possibilité d'obtenir des allocations valides. En effet, aucune allocation n'existe lorsque le $WCTT_{Calculateur,E/S}$ prend des valeurs supérieures respectivement à 0,372, 0,272 et 0,172 secondes lorsque le $WCTT_{E/S,Calculateur}$ est respectivement égal à 0, 0,100 et 0,200 secondes.

Nous remarquons également sur les Figures 5.7 et 5.8 que trois allocations deviennent de plus en plus contraintes dans le choix de la valeur de $WCTT_{Calculateur,Calculateur}$ dès lors que la valeur de $WCTT_{E/S,Calculateur}$ augmente. Ce dernier paramètre influe

fortement sur les trois premières allocations comme le montre les systèmes d'inéquations solutions S_1, S_2 et S_3 dans le paragraphe « Architecture sans passerelle ».

Les entrées/sorties et leurs liens vers les calculateurs ont une forte influence sur l'obtention et le choix d'une allocation plus distribuée puisqu'elles font parties de la majorité des chaînes étudiées (de ch_3 à ch_6).

Si nous comparons les Figures 5.9 et 5.10 qui correspondent au nombre d'allocations valides avec la passerelle définie précédemment, les résultats obtenus dans les deux cas sont identiques. L'ensemble des systèmes d'inéquations solutions obtenu pour chaque allocation grâce à la méthode d'élimination de Fourier-Motzkin donne les limites des polyèdres solutions de notre étude de cas identiques par rapport à l'heuristique.

Lorsque nous ajoutons les passerelles, nous constatons que toutes les latences dépendent des unes des autres. Cette dépendance provient de l'hypothèse $WCTT_{GW,Calculateur} = WCTT_{Calculateur,Calculateur}$ qui rend toutes les chaînes de communication dépendantes les unes des autres comme nous le montre les systèmes d'inéquations S'_1 à S'_5 dans le paragraphe « Architecture avec passerelle ». Une différence notable par rapport au cas sans passerelle est l'influence du paramètre $WCTT_{Calculateur,Ecran}$ sur le nombre d'allocations valides. Il n'existe pas d'allocation valide si $WCTT_{Calculateur,Calculateur}$ prend des valeurs supérieurs respectivement à 0,250 à 0,150 puis à 0,050 secondes pour les valeurs de $WCTT_{Calculateur,Ecran}$ respectivement égales à 0, 0,100 et 0,200 secondes, le tout est abaissé de 0,100 à chaque fois que $WCTT_{E/S,Calculateur}$ augmente de 0,100.

Pour laisser plus de marge pour la latence de communication entre calculateurs, une solution est de réduire la période de transmission des messages par la passerelle. Si nous réécrivons ci-dessous les systèmes d'inéquations solutions en laissant apparaître la période de la passerelle notée T_{GW}, nous remarquons que la période d'envoi des messages impacte toutes les latences de communication.

$$S''_1 = S''_2 = S''_3 = \begin{cases} WCTT_{E/S,GW} \leq 0,2865 - T_{GW} \\ WCTT_{Calculateur,Calculateur} \leq 0,14325 - \frac{T_{GW} + WCTT_{E/S,GW}}{2} \\ WCTT_{Calculateur,Ecran} \leq 0,275 - t_{GW} - WCTT_{E/S,GW} \\ \qquad\qquad -WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \leq 0,2865 - T_{GW} - WCTT_{E/S,GW} \\ \qquad\qquad -2 \times WCTT_{Calculateur,Calculateur} \end{cases}$$

$$S_4'' = \begin{cases} WCTT_{E/S,GW} \leq 0,275 - T_{GW} \\ WCTT_{Calculateur,Calculateur} \leq 0,275 - T_{GW} - WCTT_{E/S,GW} \\ WCTT_{Calculateur,Ecran} \\ \quad \leq 0,275 - T_{GW} - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \leq \min(0,3265 - WCTT_{Calculateur,Calculateur}; \\ \quad 0,3715 - T_{GW} - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur}) \end{cases}$$

$$S_5'' = \begin{cases} WCTT_{E/S,GW} \leq 0,275 - T_{GW} \\ WCTT_{Calculateur,Calculateur} \leq 0,275 - T_{GW} - WCTT_{E/S,GW} \\ WCTT_{Calculateur,Ecran} \\ \quad \leq 0,275 - T_{GW} - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \\ WCTT_{Calculateur,E/S} \\ \quad \leq 0,3715 - T_{GW} - WCTT_{E/S,GW} - WCTT_{Calculateur,Calculateur} \end{cases}$$

De ce fait, plusieurs paramètres d'architecture doivent être pris en compte en même temps : le choix des réseaux utilisés pour interconnecter les différents équipements ainsi que les paramètres de configuration des éléments de type passerelle.

5.2.4 Complexité de l'algorithme de Fourier-Motzkin

Pour déterminer la complexité de cet algorithme, il faut définir le nombre de nouvelles inéquations générées à chaque étape. Nous avons vu précédemment qu'une variable x_u peut être encadrée par $m_2 - m_1$ valeurs inférieures et m_1 valeurs supérieures dépendantes des variables restantes du vecteur x :

$$\max_{j \in \{m_1+1,\cdots,m_2\}} (\sum_{l=u+1}^{n} a_{jl}x_l - b_j) \leq x_u \leq \min_{i \in \{1,\cdots,m_1\}} (b_i - \sum_{l=u+1}^{n} a_{il}x_l) \tag{5.17}$$

Si nous retirons x_u de l'équation 5.17, nous créons un système de $(m - m_1 - m_2) + m_1 \times (m_2 - m_1)$ nouvelles inéquations indépendantes de x_u.

Dans le pire des cas où $m_1 = m_2 - m_1 = \frac{m}{2}$, la taille du système devient égale à $(\frac{m}{2})^2$. Après les n éliminations successives, ce système peut comprendre $(\frac{m}{2})^{2n}$ inéquations. L'algorithme d'élimination de Fourier-Motzkin est ainsi un algorithme de complexité exponentielle. Il doit être réservé à des problèmes de petite taille.

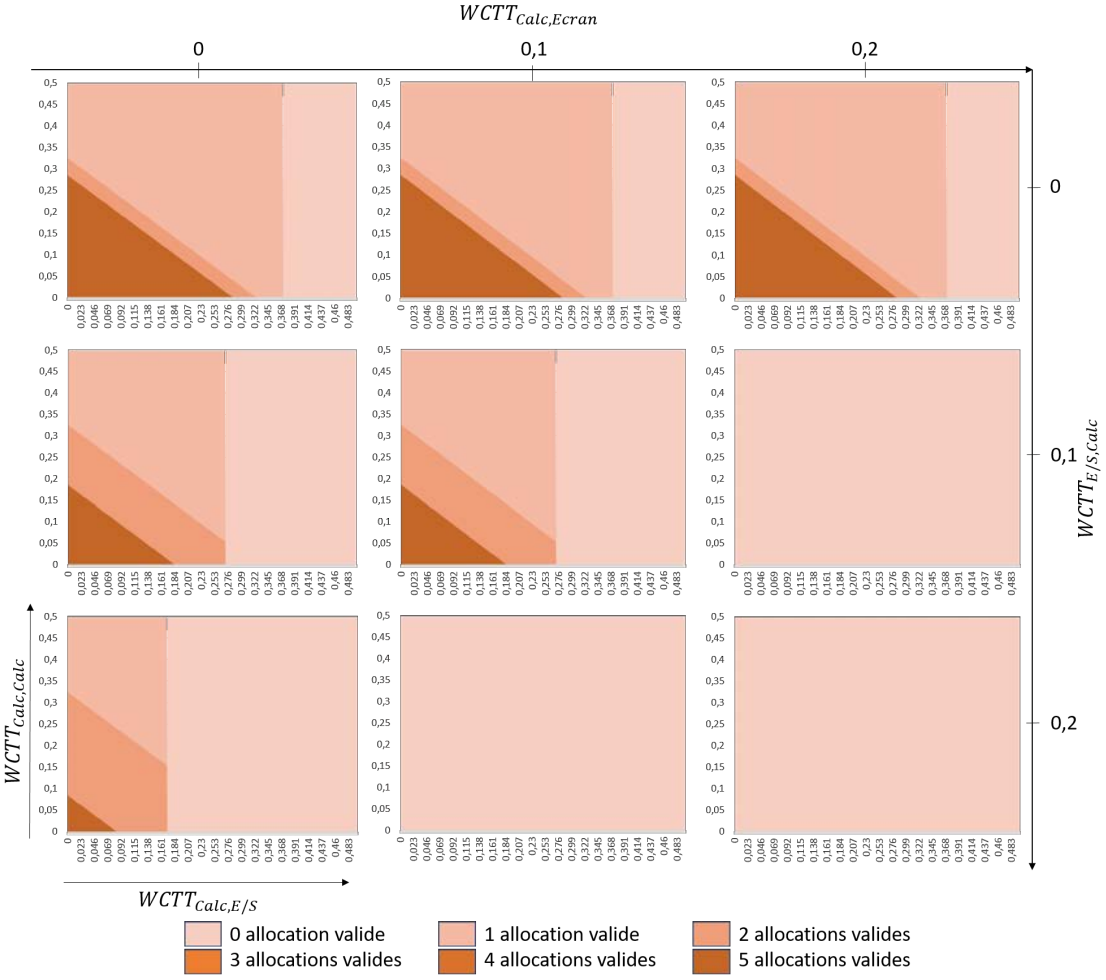


FIGURE 5.7 Nombre d'allocations valides sans passerelle obtenu avec l'heuristique

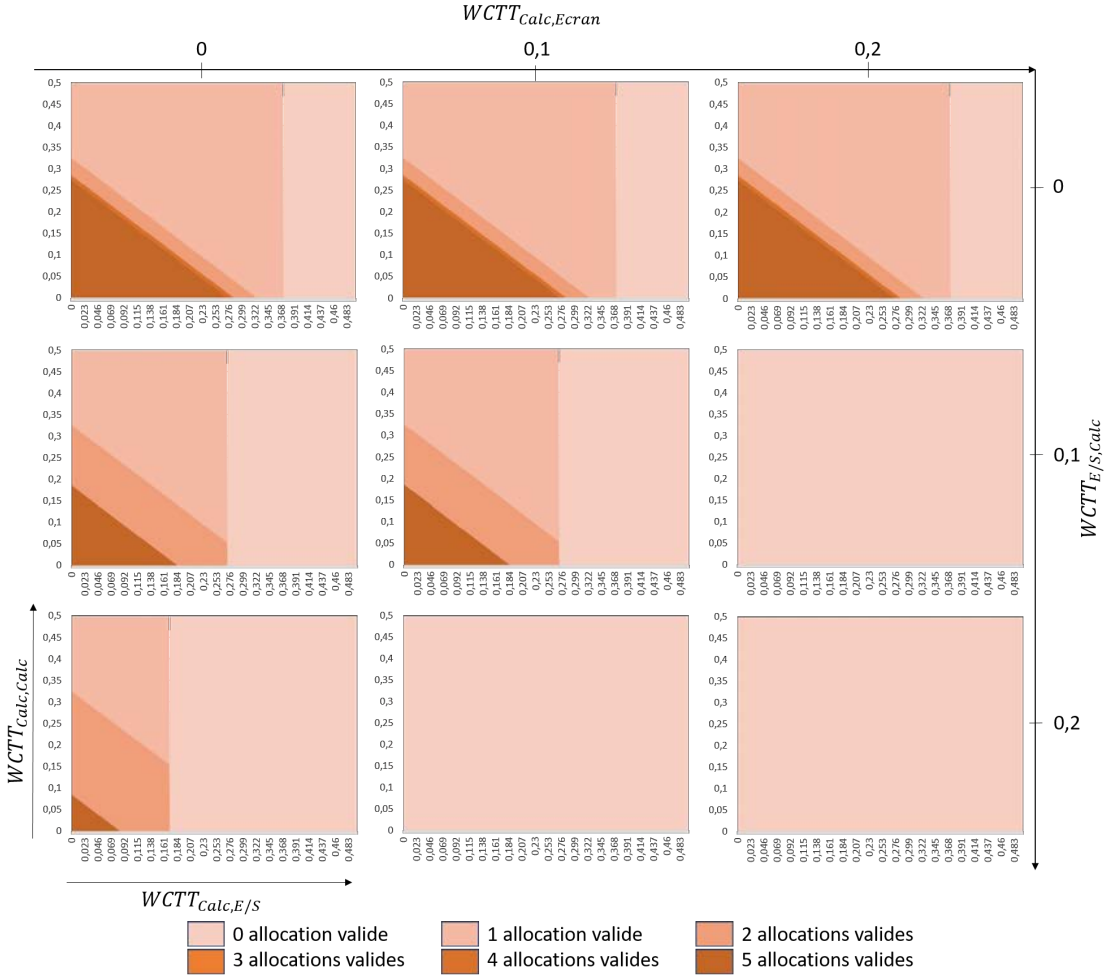


FIGURE 5.8 Nombre d'allocations valides sans passerelle obtenu avec l'algorithme d'élimination de Fourier-Motzkin

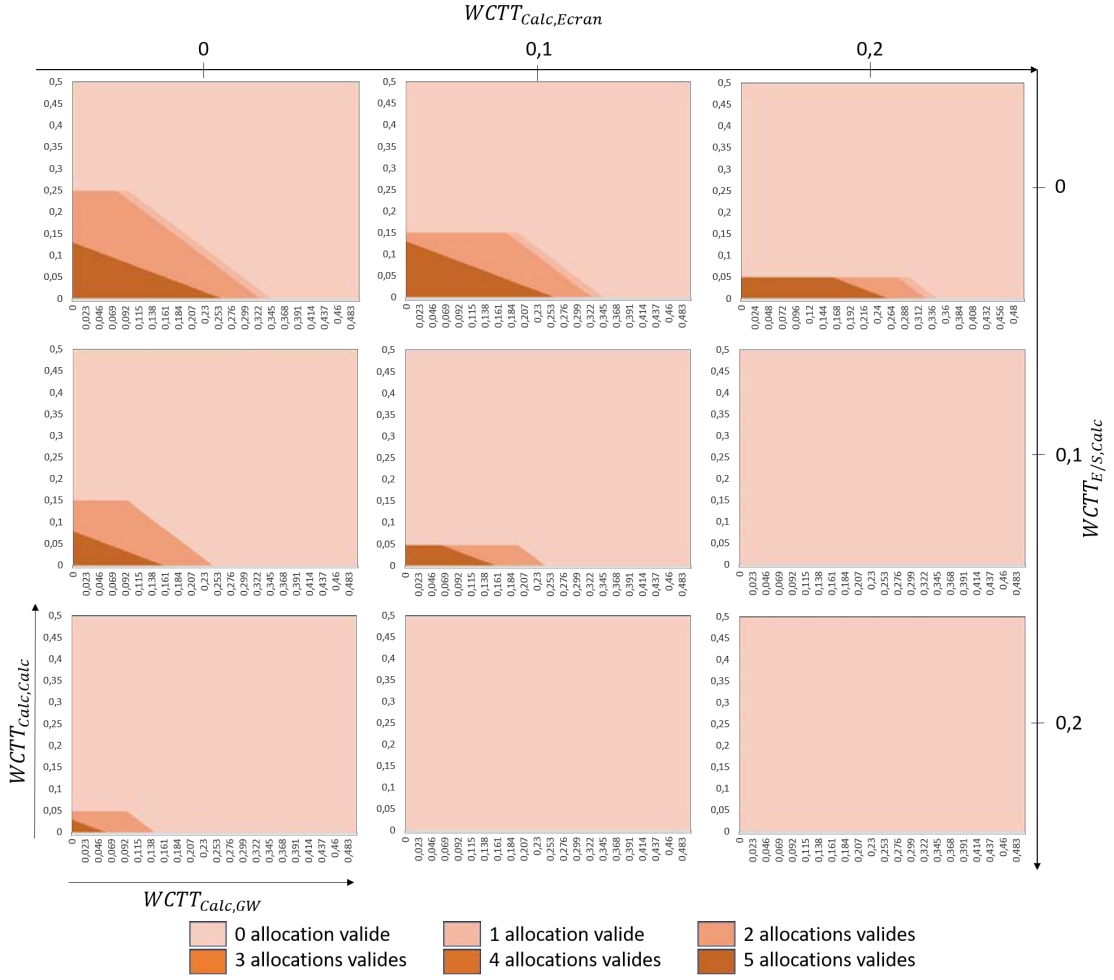


FIGURE 5.9 Nombre d'allocations valides avec passerelle obtenu avec l'heuristique - Hypothèses : 1/ $WCTT_{E/S,Calculateur}$ remplacé par $WCTT_{E/S,GW}$; 2/ $WCTT_{GW,Calculateur}$ = $WCTT_{Calculateur,Calculateur}$

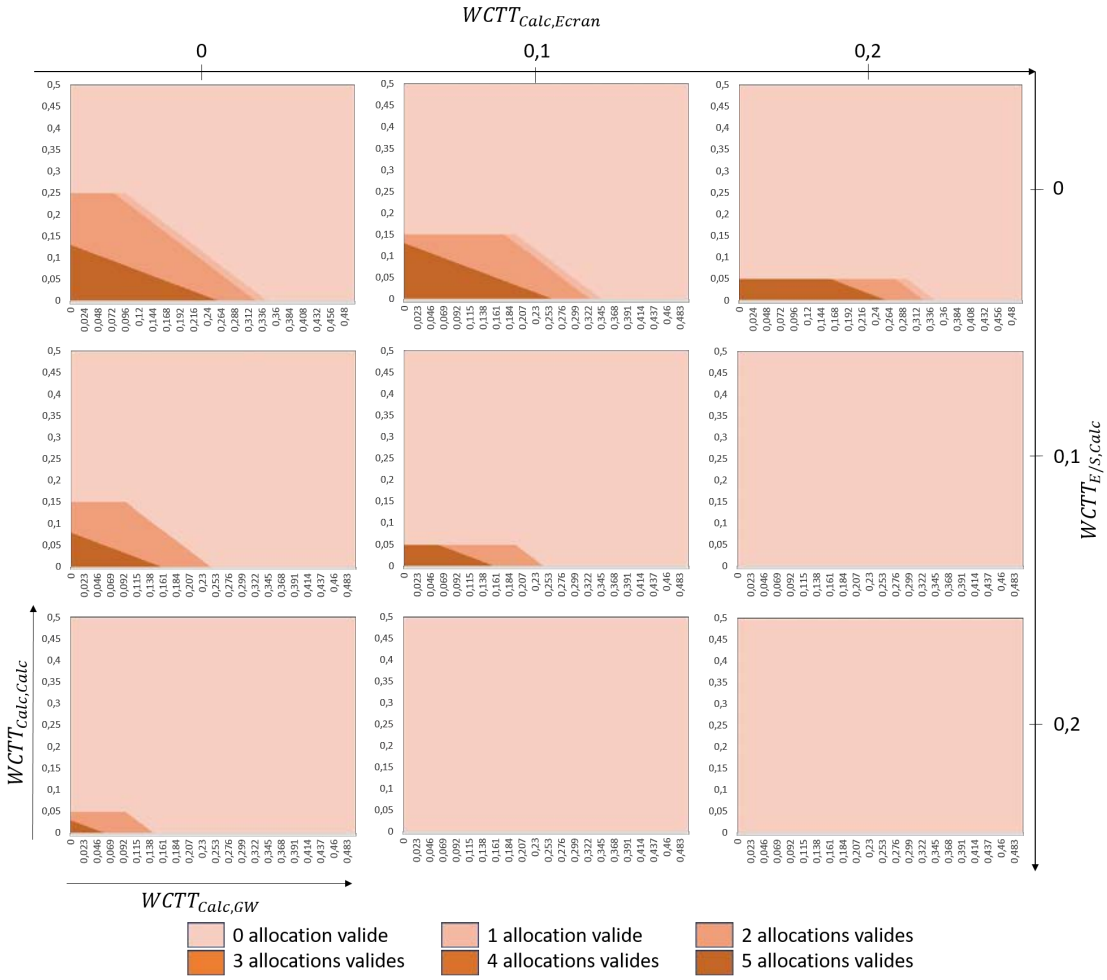


FIGURE 5.10 Nombre d'allocations valides avec passerelle obtenu avec l'algorithme d'élimination de Fourier-Motzkin - Hypothèses : 1/ $WCTT_{E/S,Calculateur}$ remplacé par $WCTT_{E/S,GW}$; 2/ $WCTT_{GW,Calculateur} = WCTT_{Calculateur,Calculateur}$

5.3 Conclusion

Dans ce chapitre, nous nous sommes penchés sur les moyens de déterminer les latences de communication maximales admissibles pour chaque architecture devant satisfaire des contraintes de bout-en-bout. Notre méthode se résume en deux étapes : obtenir les architectures valides et trouver les latences admissibles. La première étape consiste à rechercher les allocations, avec leurs ordonnancements et des latences nulles, répondant aux exigences de bout-en-bout. La deuxième étape recherche l'ensemble des latences admissibles entre chaque paire d'équipements. Nous avons montré que chaque architecture peut se modéliser par un système de m inéquations à n inconnues prenant en considération toutes les latences, les ordonnancements et les contraintes de bout-en-bout. Ce système se résout grâce à l'algorithme d'élimination de Fourier-Motzkin : le résultat obtenu est un système d'inéquations solutions donnant les contraintes entre les différences latences.

L'utilisation de la méthode d'élimination de Fourier-Motzkin donne des résultats identiques à l'heuristique à plus de 99%, le dernier pourcentage restant provenant d'une erreur d'approximation de la librairie FM [118] utilisée pour obtenir les valeurs limites du polyèdre solution du système d'inéquations. Cette méthode donne une image rapide de la flexibilité de chaque architecture par rapport aux réseaux implémentés : elle permet de déterminer à quel point les moyens de communication doivent être rapides.

Le choix des latences de communication à ne pas dépasser peut devenir périlleux dès lors que plusieurs chaînes de communication ont plusieurs latences communes. Il faut trouver un compromis entre les réseaux que nous souhaitons utiliser et les latences maximales à ne pas dépasser. Par ailleurs, l'ajout d'équipements dans une architecture tels que des passerelles augmente les valeurs des délais de bout-en-bout. La méthode d'élimination de Fourier-Motzkin offre à un intégrateur système la possibilité de définir et de configurer les périodes de transmission des messages transitant par des passerelles grâce aux systèmes d'inéquations solutions obtenus pour chaque allocation valide.

Conclusions et perspectives

Conclusions

Les architectures avioniques ont connu différentes évolutions en intégrant de nouvelles technologies, i.e. l'électronique puis l'informatique, à bord des différents aéronefs. Ces nouvelles technologies ont permis de réduire l'encombrement et le poids en remplaçant les équipements mécaniques. La ségrégation des différents systèmes est au cœur des préoccupations des intégrateurs systèmes : aucun système ne doit mettre en défaut un autre si une défaillance matérielle ou logicielle se produit. Une évolution majeure des architectures avioniques a été de pouvoir regrouper dans des modules centralisés physiquement dans des baies avioniques différents systèmes grâce au concept de l'IMA (Integrated Modular Architecture). Mais cette centralisation physique sur des processeurs puissants et complexes n'est pas possible pour de petits porteurs tels que les hélicoptères : une distribution des fonctions avioniques sur un plus grand nombre de processeurs, qui peuvent être moins puissants, est alors nécessaire.

Cette distribution doit répondre à deux critères : une allocation donnée de partitions sur un processeur donné doit être ordonnançable en respectant la norme ARINC 653 et doit satisfaire les exigences de communication de bout-en-bout entre partitions. Il s'agit de trouver l'allocation de partitions strictement périodiques ainsi que leur ordonnancement statique sur chacun des processeurs et de vérifier les contraintes des chaînes de transmission de données entre partitions. Les réseaux utilisés pour interconnecter les différents équipements influent sur les allocations valides : la vitesse du réseau peut simplifier la prise en compte des contraintes de communication de bout-en-bout. Le choix des moyens de communication est un facteur à ne pas négliger dans une architecture distribuée.

Dans cette thèse, nous souhaitions obtenir un outil qui permette d'identifier des allocations avec des ordonnancements valides satisfaisant des contraintes de bout-en-bout. Nous souhaitions aussi déterminer les limites des moyens de communication pour chaque

architecture trouvée.

Nous nous sommes donc focalisés sur le problème de distribution des partitions sur un plus grand nombre de processeurs tout en respectant les contraintes de communication de bout-en-bout. Après avoir formalisé un modèle de partitions communicantes, nous avons proposé des algorithmes de recherche d'allocations valides en tenant en compte des contraintes d'allocation et d'ordonnancement des partitions d'une part et des contraintes de communication de bout-en-bout entre partitions d'autre part. Nous avons présenté deux algorithmes qui permettent d'allouer et d'ordonnancer les partitions sur un plus grand nombre de processeurs, tout en respectant les exigences de bout-en-bout. Tous les deux sont basés sur la construction d'un arbre de recherche où chaque nœud représente une allocation partielle, i.e. certaines partitions ne sont pas encore allouées, avec un ordonnancement validant les contraintes de bout-en-bout. Ils ont été testés et validés sur une étude de cas hélicoptère. Dans le premier algorithme, l'ordonnancement d'une partition sur un nœud donné peut être modifié jusqu'à ce que tous les nœuds enfants soient ordonnançables ou qu'il n'y ait pas de solution. La génération coûteuse des ordonnancements rend cet algorithme viable que sur des systèmes de taille petite, i.e. une vingtaine de partitions allouées sur au plus cinq processeurs. Pour parer à cette limitation, nous avons proposé dans une démarche incrémentale une heuristique gloutonne basée sur la méthode de « séparation et évaluation ». À chaque nouveau nœud, l'ordonnancement le plus prometteur pour ordonnancer les partitions non-assignées est retenu, les autres étant éliminés. Cette heuristique permet de réduire considérablement l'utilisation de la mémoire et le temps d'analyse lorsque nous recherchons à vérifier la validité d'une solution. Par ailleurs, la recherche de toutes les solutions valides se termine en moins de trois heures pour une trentaine de partitions allouées sur au plus sept processeurs. L'algorithme implanté dans cette heuristique est prometteur pour l'allocation et l'ordonnancement de systèmes avioniques sur des architectures distribuées.

Ces deux algorithmes proposent une allocation et un ordonnancement en fonction de latences de communication dont les bornes supérieures peuvent être fixées. Or, une distribution donnée peut fonctionner avec un certain nombre de réseaux plus ou moins rapide. Nous avons proposé une méthode permettant de spécifier les exigences temporelles sur le réseau. Chaque allocation avec un ordonnancement donné valide se modélise par un système de m inéquations à n inconnues où chaque inconnue représente une latence de communication. Basé sur l'algorithme d'élimination de Fourier-Motzkin, nous pouvons alors déterminer les contraintes pour chaque latence. Les résultats obtenus avec cette méthode sont identiques, à plus de 99%, à une recherche à la volée avec l'heuristique. Cette méthode donne une image rapide de la flexibilité de chaque architecture par

rapport aux réseaux implémentés : elle permet de déterminer à quel point les moyens de communication doivent être rapides.

Perspectives

Suite à ces travaux, nous pouvons envisager plusieurs perspectives.

Pour éviter la propagation des fautes dans une architecture redondée, il est nécessaire d'avoir des codes implémentés sur des processeurs différents pour obtenir une redondance se basant sur la dissimilarité des processeurs. Dans cette thèse, nous avons fait l'hypothèse que le pire temps d'exécution des partitions répliquées étaient identiques et que les processeurs étaient homogènes. Nous faisons donc l'hypothèse que les codes implémentés pour une fonction répliquée dans plusieurs partitions ont leur pire temps d'exécution identique pour un processeur donné. Si nous prenons un ensemble de processeurs hétérogènes, une partition va voir son pire temps d'exécution être modifié en fonction de son allocation dans les processeurs donnés. Une solution serait de nous appuyer sur les travaux de Bate, Emberson, Zhu et al. [95–97] pour évaluer la variabilité, i.e. les modifications possibles, des architectures valides trouvées afin de définir la meilleure architecture valide parmi l'ensemble des architectures valides trouvées. Cette évaluation peut être de deux ordres. Le premier consiste à augmenter les temps d'exécution des partitions et à définir les limites avant la violation d'une des contraintes des systèmes (« *extensibility* »). Le deuxième propose de s'intéresser au nombre minimal de modifications si des mises à jour d'exigences ont lieu (« *flexibility* »).

Nous nous sommes essentiellement intéressés dans cette thèse à la distribution des partitions et à la satisfaction des contraintes de communication de bout-en-bout. D'autres contraintes telles que la cohérence des données [3] doivent être prises en compte. Cette cohérence de données passe dans notre exemple d'étude de cas par la vérification de données par une partition spécifique. Cette partition peut recevoir à la fois des données externes comme des données provenant de son homologue : elle doit par la suite les comparer. Actuellement, cette partition est synchronisée avec sa partition homologue dans un autre processeur. Or, l'IMA considère que tous les processeurs sont asynchrones. Une solution possible pour avoir cet asynchronisme est le sur-échantillonnage de certaines partitions afin que les données traitées divergent d'un laps de temps minimal. Nous pouvons nous appuyer sur les travaux de Badache [71] qui proposent une recherche de la périodicité des partitions de façon à garantir la fraîcheur des données. Ce sur-échantillonnage peut aussi amener à détacher du même processeur deux partitions avec des contraintes temporelles fortes. Il peut aussi permettre à une partition devenue plus

fréquente de s'exécuter en un temps plus court. Le problème qui s'ensuit peut être l'apparition d'un *overhead* devenant de plus en plus important, amenant à trouver un compromis entre la proportion de l'overhead pendant l'exécution de la partition et le sur-échantillonnage.

La localité des entrées/sorties, i.e. les entrées/sorties directement connectées aux processeurs, est une hypothèse que nous avons partiellement prise en compte dans nos travaux. Nous avons pu remarquer dans le Chapitre 5 qu'il était possible de placer certaines entrées/sorties derrière des passerelles. Cette dernière facilite le partage des ressources d'entrées/sorties. Or, cet éloignement réduit considérablement les latences maximales des réseaux traversés autorisées par les systèmes et rend certaines architectures impossibles, dû à de fortes contraintes sur certaines chaînes de partitions communicantes. Par exemple, certains asservissements nécessitent une (contre-)réaction rapide pour arriver à une valeur donnée. Une solution est de distinguer les différents flux associés aux entrées/sorties afin d'évaluer les contraintes temporelles sur ces derniers : elles permettront de spécifier si certaines entrées/sorties doivent être directement connectées ou peut être éloignées d'un processeur. Dans le cas où les entrées/sorties sont placées derrière des passerelles ou des bus de terrain tels que le bus CAN, il faut être capable de définir leurs priorités par rapport aux autres flux. Ces priorités dépendront des contraintes temporelles sur chaque flux pouvant être définies grâce à la méthode présentée dans le Chapitre 5.

Dans le cadre de cette thèse, nous avons fait l'hypothèse de la localité des communications entre deux partitions allouées sur un même processeur. Or, dans un contexte avionique, toutes les communications doivent pouvoir être analysées via le réseau de communication pour des raisons de maintenance. Deux cas sont possibles. Le premier consiste à répliquer la donnée envoyée localement afin de la transmettre par une communication distante vers un équipement de surveillance/d'analyse. Dans ce cas, un nombre de flux supplémentaires doit être pris en compte dans le calcul des délais pires cas des réseaux. Notre démarche n'est pas remise en cause ici. Le deuxième considère que toutes les communications sont distantes. Dans ce cas, notre heuristique va trouver un nombre plus faible d'architectures favorables. En effet, les ordonnancements générés cherchent à rapprocher le plus possible les partitions communicantes dans un même processeur. Or, les délais de bout-en-bout dans le cas où toutes les communications sont distantes doivent alors être calculés en considérant une latence de communication pire cas exacte et l'instant de prochaine activation de la partition destinatrice. Si deux partitions communicantes sont localisées sur le même processeur, cette hypothèse ne nous permet pas de générer le meilleur ordonnancement local sans une connaissance exacte des latences de communication pire cas sur le réseau. Créer des blocs de partitions indissociables dans

nos ordonnancements est une contrainte forte dans notre heuristique. Une solution serait de considérer que chaque partition allouée et ordonnancée sur un processeur ne fige plus temporellement un bloc indissociable dans la MAF, mais juste un slot temporel dans la MAF totale.

L'utilisation de nouvelles technologies tels que les pluri-cœurs amène à une réflexion sur l'allocation et l'ordonnancement des partitions sur ces derniers. Actuellement, les processeurs mono-cœurs sont remplacés par des processeurs multi-/pluri-cœurs. Ces processeurs connectent un ensemble de cœurs avec un premier niveau de cache. Les autres caches sont partagées dans une mémoire commune grâce à un réseau sur puce, ou *Network on Chip* (NoC). Pour des raisons de sûreté de fonctionnement/certification dans l'avionique, un seul cœur est souvent utilisé dans ces architectures. L'idée est de pouvoir utiliser toute la puissance de calcul offerte. Nos travaux peuvent s'adapter à ce contexte en considérant que les réseaux de type NoC sont reliés par une passerelle à un réseau fédérateur tel que l'AFDX par exemple pour définir l'allocation et l'ordonnancement des différents systèmes avioniques dans les architectures pluri-cœurs.

Publications

Conférences internationales :

— E. Deroche, J. L. Scharbarg, and C. Fraboul, "Performance evaluation of a distributed ima architecture," in *2015 IEEE 27th Euromicro Conference on Real-Time Systems (ECRTS) - Work-in-Progress Proceedings*, pp. 17–20, July 2015

— E. Deroche, J. L. Scharbarg, and C. Fraboul, "Mapping real-time communicating tasks on a distributed ima architecture," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, Sept. 2016

— E. Deroche, J.-L. Scharbarg, and C. Fraboul, "A greedy heuristic for distributing hard real-time applications on an ima architecture," in *12th IEEE International Symposium on Industrial Embedded Systems (SIES 2017)*, pp. 1–8, June 2017

Workshops internationaux :

— E. Berard Deroche, C. Fraboul, and F. Le Sergent, "Towards a more distributed avionics architecture (regular paper)," in *International Workshop on Real-Time Networks (RTN), Madrid, 08/07/2014-08/07/2014*, (http://www.ieee.org/), pp. 9–10, IEEE, juillet 2014

— E. Deroche, J. L. Scharbarg, and C. Fraboul, "Communication-aware scheduling on an ima architecture," in *2015 8th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pp. 1–2, Dec. 2015

Bibliographie

[1] "Arinc specification 653p1-4 : Avionics application software standard interface part 1 required services." SAE-ITC, Aug. 2015.

[2] "Arinc 651 : Design guidance for integrated modular avionics." Aeronautical Radio, INC., Mar. 1999.

[3] M. Lauer, *Une méthode globale pour la vérification d'exigences temps réel - Application à l'Avionique Modulaire Intégrée*. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, juin 2012. (Soutenance le 15/06/2012).

[4] J. Botti, *Carnet de vol : la fascinante aventure de l'aérospatiale*. Fondation d'entreprise EADS, June 2010.

[5] D. d'histoire de l'armement du Centre des hautes études de l'armement, ed., *Un demi-siècle d'aéronautique en France - Les équipements*, vol. I-II. Groupe COMAÉRO-Équipements, 2004.

[6] I. Moir and A. Seabridge, "Avionics architectures," in *Civil avionics systems*, pp. 159–203, Wiley-Blackwell, aug 2013.

[7] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pp. 2.A.1–1–2.A.1–10, Oct 2007.

[8] R. Wolfig and M. Jakovljevic, "Distributed ima and do-297 : Architectural, communication and certification attributes," in *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pp. 1.E.4–1–1.E.4–10, Oct 2008.

[9] V. Lopez-Jaquero, F. Montero, E. Navarro, A. Esparcia, and J. A. Catal'n, "Supporting ARINC 653-based dynamic reconfiguration," in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, Institute of Electrical and Electronics Engineers (IEEE), aug 2012.

[10] S. I. A. Jim Moore, *Digital Avionics Handbook, Second Edition - 2 Volume Set*, ch. Chapter 33. Advanced Distributed Architectures, pp. –. -, 2001.

[11] R. Alena, J. Ossenfort, K. Laws, A. Goforth, and F. Figueroa, "Communications for integrated modular avionics," in *Aerospace Conference, 2007 IEEE*, pp. 1 –18, march 2007.

[12] Y. Sorel, "Systèmes distribués temps réel sûrs et optimisés. méthodologie aaa : adéquation algorithme-architecture." http ://www.syndex.org/cours/web14.pdf. Accédé le 02/12/2016.

[13] I. Lee, J. Y.-T. Leung, and S. H. Son, *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC, 1st ed., 2007.

[14] "ARP 4754 A : Guidelines for Development of Civil Aircraft and Systems." SAE Aerospace, Dec. 2010.

[15] "Do-297 integrated modular avionics (ima) development guidance and certification considerations," Aug. 2005.

[16] S. M'Sirdi, W. Godard, and M. Pantel, "A multi-core interference-aware schedulability test for ima systems, as a guide for sw/hw integration," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[17] P. R. Joël Goossens, "Ordonnancement temps réel multiprocesseur." École d'été Temps Réel 2013, Aug. 2013.

[18] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, jan 1973.

[19] B. Annighöfer and F. Thielecke, "Multi-objective mapping optimization for distributed integrated modular avionics," in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pp. 6B2–1–6B2–13, Oct 2012.

[20] M. Ekman, "Avionic architecture : trends and challenges." SAAB. Accédé le 4 décembre 2012.

[21] R. E. Korf, "A new algorithm for optimal bin packing," in *Eighteenth National Conference on Artificial Intelligence*, (Menlo Park, CA, USA), pp. 731–736, American Association for Artificial Intelligence, 2002.

[22] J. Korst, E. Aarts, and J. K. Lenstra, "Scheduling periodic tasks," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 428–435, 1996.

[23] "Ed-112a : Minimum operational performance specification for crash protected airborne recorder systems." EUROCAE, September 2013.

[24] ICAO, *Annex 6 to the Convention on International Civil Aviation, Operation of Aircraft, Part I International Commercial Air Transport - Aeroplanes*. No. AN 6-1, 999 University Street, Montréal, Quebec, Canada H3C 5H7 : International Civil Aviation Organization, July 2010.

[25] "Arinc report 655 : Remote data concentrator(rdc) generic description," 1999.

[26] C. Pagetti, "Module de sûreté de fonctionnement." ENSEEIHT - 3ème année TR - option SE - 10 décembre 2012, Dec. 2012.

[27] M. Park, "Non-preemptive fixed priority scheduling of hard real-time periodic tasks," in *Computational Science – ICCS 2007*, pp. 881–888, Springer Nature, 2007.

[28] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, dec 1982.

[29] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," Research Report RR-2966, INRIA, 1996. Projet REFLECS.

[30] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *[1991] Proceedings Twelfth Real-Time Systems Symposium*, pp. 129–139, Dec 1991.

[31] M. Nasri and G. Fohler, "Non-work-conserving non-preemptive scheduling : Motivations, challenges, and potential solutions," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, Institute of Electrical and Electronics Engineers (IEEE), jul 2016.

[32] A. K.-L. Mok, *Fundamental design problems of distributed systems for the hard-real-time environment.* phdthesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, Cambridge, MA, USA, 1983.

[33] W. Li, K. Kavi, and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems," *Computers & Electrical Engineering*, vol. 33, no. 1, pp. 12 – 29, 2007.

[34] M. Nasri and M. Kargahi, "Precautious-rm : a predictable non-preemptive scheduling algorithm for harmonic tasks," *Real-Time Systems*, vol. 50, no. 4, pp. 548–584, 2014.

[35] M. Nasri and G. Fohler, "Non-work-conserving scheduling of non-preemptive hard real-time tasks based on fixed priorities," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems - RTNS '15*, Association for Computing Machinery (ACM), 2015.

[36] M. R. Garey, R. L. Graham, and J. D. Ullman, "Worst-case analysis of memory allocation algorithms," in *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, (New York, NY, USA), pp. 143–150, ACM, 1972.

[37] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *Journal of the ACM*, vol. 32, pp. 562–572, jul 1985.

[38] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing - STOC '93*, Association for Computing Machinery (ACM), 1993.

[39] J. H. Anderson and A. Srinivasan, "Pfair scheduling : beyond periodic task systems," in *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 297–306, 2000.

[40] J. Anderson and A. Srinivasan, "A new look at pfair priorities," *Submitted to Real-time Systems Journal*, 1999.

[41] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem : how much fairness is necessary ?," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 142–151, Dec 2003.

[42] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair : A simple model for understanding optimal multiprocessor scheduling," in *2010 22nd Euromicro Conference on Real-Time Systems*, pp. 3–13, July 2010.

[43] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, (Washington, DC, USA), pp. 101–110, IEEE Computer Society, 2006.

[44] E. Massa, G. Lima, and P. Regnier, "Revealing the secrets of RUN and QPS : new trends for optimal real-time multiprocessor scheduling," in *2014 Brazilian Symposium on Computing Systems Engineering, SBESC 2014, Manaus, Amazonas, Brazil, November 3-7, 2014*, pp. 150–155, 2014.

[45] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Real-time scheduling algorithms for multiprocessor," *Journal Européen des Systèmes Automatisés (JESA)*, vol. 48, pp. 613–639, Oct. 2015.

[46] E. Grolleau, *Ordonnancement temps réel hors-ligne optimal à l'aide des réseaux de Pétri en environnement monoprocesseur et multiprocesseur*. PhD thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, Nov. 1999.

[47] K. R. Baker and Z.-S. Su, "Sequencing with due-dates and early start times to minimize maximum tardiness," *Naval Research Logistics Quarterly*, vol. 21, pp. 171–176, mar 1974.

[48] J. Xu and D. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Transactions on Software Engineering*, vol. 16, pp. 360–369, mar 1990.

[49] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[50] R. Barreto, P. Maciel, M. Neves, E. Tavares, and R. Lima, "A novel approach for off-line multiprocessor scheduling in embedded hard real-time systems," in *IFIP International Federation for Information Processing*, pp. 157–166, Springer Nature, 2004.

[51] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Optimal scheduling using priced timed automata," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, pp. 34–40, mar 2005.

[52] C. Bernard and G. Dominique, "Une approche markovienne pour l'étude de systèmes temps-réel à contraintes strictes," *Technique et Science Informatiques*, vol. 2007, no. 10, pp. 1269–1303, 2007.

[53] K. Danne and M. Platzner, "A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware," in *International Conference on Field Programmable Logic and Applications, 2005.*, pp. 568–573, Aug 2005.

[54] F. Eisenbrand, K. Kesavan, R. S. Mattikalli, M. Niemeier, A. W. Nordsieck, M. Skutella, J. Verschae, and A. Wiese, "Solving an avionics real-time scheduling problem by advanced ip-methods," in *Proceedings of the 18th Annual European Conference on Algorithms : Part I*, ESA'10, (Berlin, Heidelberg), pp. 11–22, Springer-Verlag, 2010.

[55] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *Automata, Languages and Programming*, pp. 299–311, Springer Nature, 2010.

[56] K. Ramamritham, J. Stankovic, and P.-F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 184–194, apr 1990.

[57] L. Cucu and Y. Sorel, "Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints," in *Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG*, vol. 4, 2004.

[58] T. Shepard and J. A. M. Gagne, "A pre-run-time scheduling algorithm for hard real-time systems," *IEEE Transactions on Software Engineering*, vol. 17, pp. 669–677, Jul 1991.

[59] O. Kermia, *Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence.* PhD thesis, Université Paris XI - UFR scientifique d'Orsay, 2009. Thèse de doctorat dirigée par Sorel, Yves Informatique Université de Paris-Sud. Faculté des Sciences d'Orsay (Essonne) 2009.

[60] P. Bratley, M. Florian, and P. Robillard, "Scheduling with earliest start and due date constraints on multiple machines," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 165–173, 1975.

[61] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite : Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, pp. 453–482, 2013.

[62] "Arinc specification 429p1-18 : Digital information transfer system (dits) part 1 - functional description, electrical interfaces, label assignments and word formats." Aeronautical Radio, INC., Nov. 2012.

[63] "Arinc specification 825-2 : General standardization of can (controller area network) bus protocol for airborne use." Aeronautical Radio, INC., July 2011.

[64] "Mil-std-1553 tutorial." Condor Engineering, Inc., June 2000. Version 3.41.

[65] "Arinc 629 p1 : Multi-transmitter data bus part 1 technical description." Aeronautical Radio, INC., Mar. 1999.

[66] A. Gabillon and L. Gallon, "Availability constraints for avionic data buses," in *First International Conference on Availability, Reliability and Security (ARES'06)*, pp. 7 pp.–, April 2006.

[67] Y. Isik, "Arinc 629 data bus standard on aircrafts," in *Proceedings of the 9th WSEAS international conference on Circuits, systems, electronics, control & signal processing*, pp. 191–195, World Scientific and Engineering Academy and Society (WSEAS), 2010.

[68] "Arinc specification 664p7-1 : Aircraft data network part 7 - avionics full-duplex switched ethernet network." Aeronautical Radio, INC., Sept. 2009.

[69] "TTEthernet Specification." TTTech Computertechnik AG, Nov. 2008.

[70] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (tte) design," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pp. 22–33, IEEE, 2005.

[71] N. Badache, *Allocation temporelle de systèmes avioniques modulaires embarqués*. PhD thesis, Institut National Polytechnique de Toulouse (INP Toulouse), 2016. Thèse de doctorat dirigée par Fraboul, Christian et Scharbarg, Jean-Luc Réseaux, Télécommunications, Systèmes et Architecture Toulouse, INPT 2016.

[72] J.-Y. Le Boudec and P. Thiran, *Network calculus : a theory of deterministic queuing systems for the internet*, vol. 2050. Springer Science & Business Media, 2001.

[73] J. Grieu, *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, Institut National Polytechnique de Toulouse, Sept. 2004.

[74] S. Martin, *Mastering the time dimension of the quality of service in networks*. Theses, Université Paris XII Val de Marne, July 2004.

[75] E. M. Clarke, "The birth of model checking," in *25 Years of Model Checking*, pp. 1–26, Springer, 2008.

[76] M. Adnan, *Analyse pire cas exact du réseau AFDX*. PhD thesis, Institut National Polytechnique de Toulouse (INP Toulouse), 2013. Thèse de doctorat dirigée par Fraboul, Christian et Scharbarg, Jean-Luc Réseaux, Télécommunications, Systèmes et Architecture Toulouse, INPT 2013.

[77] Y. Han and F. He, "Design of DIMA scheduling algorithm based on network partition integrating model," in *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, Institute of Electrical and Electronics Engineers (IEEE), oct 2014.

[78] M. Jakovljevic, "Synchronous/asynchronous ethernet networking for mixed criticality systems," in *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pp. 1.E.3–1–1.E.3–10, Oct 2009.

[79] M. Fletcher, "Progression of an open architecture : from orion to altair and lss," *White paper S65–5000-20-0, Honeywell International, Glendale*, 2009.

[80] "TTTech Delivers Distributed IMA Test Bed with TTEthernet to Sikorsky Aircraft." TTTech, June 2010. Accédé le 6/12/2012.

[81] G. Gaderer, "Distributed IMA with TTEthernet - ARINC 653 Integration of TTEthernet," Oct. 2012.

[82] H. Kopetz, *Event-triggered versus time-triggered real-time systems*, pp. 86–101. Berlin, Heidelberg : Springer Berlin Heidelberg, 1991.

[83] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, 2008.

[84] H. Bauer, J. Scharbarg, and C. Fraboul, "Worst-case end-to-end delay analysis of an avionics afdx network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 1220–1224, March 2010.

[85] X. Li, J. Scharbarg, and C. Fraboul, "Worst-case delay analysis on a real-time heterogeneous network," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pp. 11–20, June 2012.

[86] M. Ashjaei, S. Mubeen, M. Behnam, L. Almeida, and T. Nolte, "End-to-end resource reservations in distributed embedded systems," in *The 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2016.

[87] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *The 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2016.

[88] N. Badache, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, "End-to-end delay analysis in an integrated modular avionics architecture," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pp. 1–4, Sept 2013.

[89] M.-Y. Nam, R. Pellizzoni, L. Sha, and R. Bradford, "Asiist : Application specific i/o integration support tool for real-time bus architecture designs," in *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*, pp. 11–22, June 2009.

[90] A. Al Sheikh, *Resource allocation in hard real-time avionic systems. Scheduling and routing problem.* PhD thesis, EDSYS, September 2011.

[91] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks : An np-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[92] C. Ekelin and J. Jonsson, "A lower-bound algorithm for minimizing network communication in real-time systems," in *Parallel Processing, 2002. Proceedings. International Conference on*, pp. 343–351, 2002.

[93] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. L. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Trans. Embedded Comput. Syst.*, vol. 11, no. 4, pp. 85 :1–85 :30, 2012.

[94] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-P. Babau, L. Lemarchand, and S. Gerard, "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems," *SIGPLAN Not.*, vol. 48, pp. 121–132, June 2013.

[95] I. Bate and P. Emberson, "Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems," in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pp. 221–230, April 2006.

[96] P. Emberson and I. Bate, "Stressing search with scenarios for flexible solutions to real-time task allocation problems," *IEEE Transactions on Software Engineering*, vol. 36, pp. 704–718, Sept 2010.

[97] Q. Zhu, Y. Yang, M. Natale, E. Scholte, and A. Sangiovanni-Vincentelli, "Optimizing the software architecture for extensibility in hard real-time distributed systems," *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 621–636, Nov 2010.

[98] S. Ronngren and B. A. Shirazi, "Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs," in *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, vol. 2, pp. 143–152 vol.2, Jan 1995.

[99] S. Islam, N. Suri, A. Balogh, G. Csertán, and A. Pataricza, "An optimization based design for integrated dependable real-time embedded systems," *Design Automation for Embedded Systems*, vol. 13, no. 4, pp. 245–285, 2009.

[100] K. Kuchcinski, "Constraints-driven scheduling and resource assignment," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, pp. 355–383, July 2003.

[101] Y. H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th*, vol. 1, pp. 1C2/1–1C2/8 vol.1, 2000.

[102] S. Islam, R. Lindstrom, and N. Suri, "Dependability driven integration of mixed criticality sw components," in *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, pp. 11 pp.–, April 2006.

[103] D. Decotigny, "Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps-réel." Révision : 1.5, Nov. 2012.

[104] M. Sghairi Haouati, *Architectures innovantes de systèmes de commandes de vol.* PhD thesis, Institut National Polytechnique de Toulouse, Sept. 2010.

[105] E. GROLLEAU, M. RICHARD, P. RICHARD, and F. RIDOUARD, "Ordonnancement temps réel ordonnancement réparti," *Techniques de l'ingénieur Supervision des systèmes industriels*, vol. base documentaire : TIB396DUO., Nov. 2013.

[106] "Aircraft accident report 1/2011 - report on the accident to eurocopter ec225 lp super puma, g-redu near the eastern trough area project (etap) central production facility platform in the north sea on 18 february 2009," tech. rep., Department for Transport, September 2011.

[107] "Ec145 t2 - technical data," Tech. Rep. 145 T2 14.100.01 E, Airbus Helicopters, Aeroport International Marseille Provence - 13725 Marignane Cedex - France, 2014.

[108] E. G. Engleman, "Safety recommendation a-03-048," tech. rep., National Transportation Safety Board (NTSB), Washington, D.C. 20594, Nov. 2003.

[109] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The ROSACE case study : From simulink specification to multi/many-core execution," in *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pp. 309–318, 2014.

[110] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "Reducing the contention experienced by real-time core-to-i/o flows over a tilera-like network on chip," in *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*, pp. 86–96, 2016.

[111] J. Puchinger and G. R. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification," in *Artificial Intelligence and Knowledge Engineering Applications : A Bioinspired Approach*, pp. 41–53, Springer Nature, 2005.

[112] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, pp. 85–103, Springer US, 1972.

[113] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[114] J. Fourier, *Mémoires de l'académie royale des sciences de l'Institut de France*, vol. Tome VII, ch. Analyse des travaux de l'Académie royale des Sciences, pendant l'année 1824. Partie mathématiques, pp. j–lv. De l'imprimerie de Firmin Didot, 1827.

[115] L. L. Dines, "Systems of linear inequalities," *Annals of Mathematics*, vol. 20, no. 3, pp. 191–199, 1919.

[116] T. Motzkin, *Beiträge zur Theorie der linearen Ungleichungen.* Azriel, 1936.

[117] A. Schrijver, *Theory of Linear and Integer Programming.* JOHN WILEY & SONS INC, 1998.

[118] L.-N. Pouchet, "Fm the fourier-motzkin library," Nov. 2008.

[119] E. Deroche, J. L. Scharbarg, and C. Fraboul, "Performance evaluation of a distributed ima architecture," in *2015 IEEE 27th Euromicro Conference on Real-Time Systems (ECRTS) - Work-in-Progress Proceedings*, pp. 17–20, July 2015.

[120] E. Deroche, J. L. Scharbarg, and C. Fraboul, "Mapping real-time communicating tasks on a distributed ima architecture," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, Sept. 2016.

[121] E. Deroche, J.-L. Scharbarg, and C. Fraboul, "A greedy heuristic for distributing hard real-time applications on an ima architecture," in *12th IEEE International Symposium on Industrial Embedded Systems (SIES 2017)*, pp. 1–8, June 2017.

[122] E. Berard Deroche, C. Fraboul, and F. Le Sergent, "Towards a more distributed avionics architecture (regular paper)," in *International Workshop on Real-Time Networks (RTN), Madrid, 08/07/2014-08/07/2014*, (http ://www.ieee.org/), pp. 9–10, IEEE, juillet 2014.

[123] E. Deroche, J. L. Scharbarg, and C. Fraboul, "Communication-aware scheduling on an ima architecture," in *2015 8th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pp. 1–2, Dec. 2015.

Annexe A

Description de l'application VMSA

L'application « *Vehicle Monitoring System DAL A* » (VMSA) gère les données moteur et véhicule, en vérifiant leur validité et leur cohérence. Le pilote doit être alerté dès qu'un paramètre dépasse des seuils donnés. Cette application doit aussi renvoyer les données des paramètres demandées par le pilote. Nous détaillons ce système ci-dessous.

A.1 Architecture matérielle

A.1.1 Les équipements

Le système VMSA illustré Figure A.1 est composé de 26 capteurs, 3 équipements d'asservissement, i.e. des équipements contenant une boucle de contre-réaction, 2 enregistreurs, 4 actionneurs, 2 AMC et 4 écrans notés MFD. Les quatre premiers types d'équipement sont illustrés par l'acronyme « I/O » Figure A.1.

Un AMC est un équipement duplex composé de deux voies, A et B, contenant chacun un processeur. Ces deux processeurs sont reliés par un lien interne RS-422. Les deux AMCs traitent les données reçues des capteurs afin de calculer les commandes à envoyer.

Un MFD est un processeur directement lié à l'écran. C'est un équipement actif jouant trois rôles : une première partie assure sa logistique, une deuxième partie traite les données provenant des liens de type ARINC 429, Ethernet, des composants discrets et analogiques, tandis qu'une troisième partie s'occupe de l'interface homme-machine avec l'écran. Les MFDs reçoivent des données de toutes les voies des AMCs dans un mode « COMmande/MONiteur (COM/MON) » : la voie A d'un AMC envoie une commande tandis que la voie B du même AMC envoie un statut.

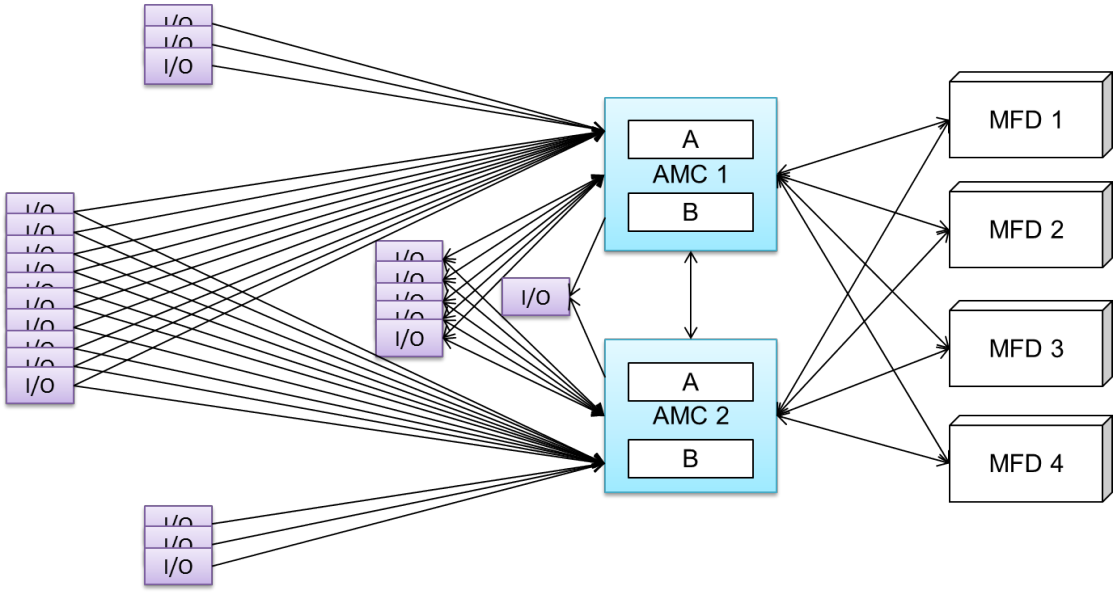


FIGURE A.1 Connection des équipements dans le système VMSA

A.1.2 Architecture réseau

Deux réseaux distincts, Ethernet et ARINC 429, sont employés pour interconnecter les équipements, chacun étant spécialisé pour un type de communication précis. Le réseau Ethernet gère les communications entre partitions associées à la surveillance des équipements. Le réseau ARINC 429 laisse passer les communications de données utiles au fonctionnement de l'hélicoptère : par exemple, les données des capteurs, l'envoi d'information vers les écrans, etc.

A.2 Architecture applicative

A.2.1 Allocation des partitions sur les équipements

A.2.1.1 Spécification des partitions

Une instance du VMSA est composée de sept partitions spécifiées Tableau A.1. Parmi ces sept partitions, quatre d'entre elles, i.e. de P_1 à P_4, s'occupent du pilotage (P_1) et de l'enregistrement de l'état du véhicule (de P_2 à P_4). Les trois autres partitions, i.e. de P_5 à P_7, gèrent et surveillent la voie où elles sont allouées. Ces sept partitions utilisent actuellement 86% du processeur sur chaque voie d'un AMC.

Chaque AMC reçoit les acquisitions d'une quinzaine de capteurs dédiés tels que

TABLE A.1 Spécifications temps-réel des partitions du système VMSA dans une voie d'un AMC (valeurs modifiées restant cohérentes par rapport au système VMSA)

Partitions	WCET	Période
P_1	10	25
P_2	10	50
P_3	6	100
P_4	6	50
P_5	5	100
P_6	2	100
P_7	1	100

TABLE A.2 Période d'acquisition des capteurs associés à l'application VMSA

Nombre de capteurs	1	2	3	2	3	5	1
Période d'acquisition	7	14	20	34	40	50	100

les accéléromètres pour évaluer l'accélération normale, longitudinale, latérale, l'attitude de tangage, de roulis, la puissance de chaque moteur, la vitesse du rotor, etc. Chaque capteur échantillonne à son rythme ses acquisitions. Le Tableau A.2 énumère la période d'acquisition des capteurs associés à l'application VMSA. Un capteur est relié à une unique voie ou aux deux voies d'un AMC. Les capteurs peuvent envoyer des signaux analogiques, discrets ou numériques (ARINC 429).

Le système VMSA pilote trois équipements asservis. Le système envoie une valeur de consigne à atteindre et mesure en permanence l'écart entre la valeur réelle et celle-ci : il calcule la commande appropriée à appliquer à l'actionneur associé de façon à réduire cet écart le plus rapidement possible. Ces trois équipements envoient leurs données à la partition P_1 toutes les 20 ms, P_1 leur renvoie une consigne toutes les 25 ms pour un équipement et toutes les 50 ms pour les deux autres.

Dans le système VMSA, deux types de données sont enregistrés : la santé générale du véhicule et l'état des équipements. Ces informations permettent d'évaluer les besoins de maintenance de l'hélicoptère. Un hélicoptère contient un enregistreur de vol dont les données sont utilisés en cas de crash. Cet équipement enregistre cycliquement les données reçues toutes les 62,5 ms selon les recommandations fournis par le *National Transportation Safety Board* [108]. Le système VMSA n'interagit pas avec l'enregistreur de vol, mais nous utilisons les valeurs préconisées par [23] et [108] pour les enregistrements de maintenance.

Le système VMSA est connecté à 4 écrans identiques. Ces écrans contiennent à la fois des partitions qui statuent sur le fonctionnement de l'appareil et des partitions qui gèrent

l'affichage des données reçues. Une partition d'une durée d'exécution de 39 ms toutes les 50 ms récupère les données envoyées par d'autres équipements, vérifient la validité des informations reçues grâce à des mécanismes de COM/MON avant de les afficher.

A.2.1.2 Allocation

L'application VMSA est instanciée quatre fois : chaque instance est allouée sur une voie d'un AMC. Toutes ces instances communiquent avec les quatre écrans. Le VMSA reçoit des données des capteurs ou des systèmes d'asservissement et envoie des données à ces derniers et aux actionneurs. La Figure A.2 représente à partir d'un AMC les communications entre la partition principale du VMSA noté P_1 et :

— les autres partitions de sa voie : P_2, P_3, P_4 et P_5, (P_6 et P_7 ne sont pas dessinées ici puisqu'elles n'interagissent pas directement avec P_1),
— les écrans,
— les entrées/sorties telles que les capteurs, les actionneurs, les asservissements et les enregistreurs.

Les E/S sont locales : elles sont directement connectées aux équipements qui ont besoin de leurs données. Ainsi une trentaine d'E/S sont rattachées à chaque AMC, que ce soit sur ses interfaces discrètes, analogiques ou numériques.

Avec un taux d'utilisation de 86% de chaque processeur des AMCs et un nombre d'interfaces disponibles n'étant plus suffisant, il est difficile d'ajouter à la fois de nouvelles fonctionnalités dans les AMCs ou d'y connecter de nouvelles entrées/sorties. Deux solutions, qui peuvent être indépendantes, s'offrent à nous : soit les partitions des voies des AMCs sont distribuées sur un plus grand nombre de processeurs qui peuvent être moins puissants, permettant d'obtenir un plus grand nombre d'interfaces disponibles, soit les entrées/sorties sont regroupées derrière des passerelles ou connectées sur des bus de terrain tels que le réseau CAN, permettant ainsi la diffusion des informations des E/S partagées entre tous les équipements.

A.2.2 Contraintes applicatives

A.2.2.1 Cohérence des données

Afin de vérifier la véracité des informations reçues dans une voie d'un AMC, plusieurs capteurs de même nature doivent être comparés. Cette étape se résume à faire une vérification croisée, ou « *cross-check* » en anglais. Chaque voie envoie les informations reçues des capteurs à l'autre voie grâce au lien interne. Cet échange est effectué par la partition P_1 : au début de son exécution, cette partition échange avec son homologue

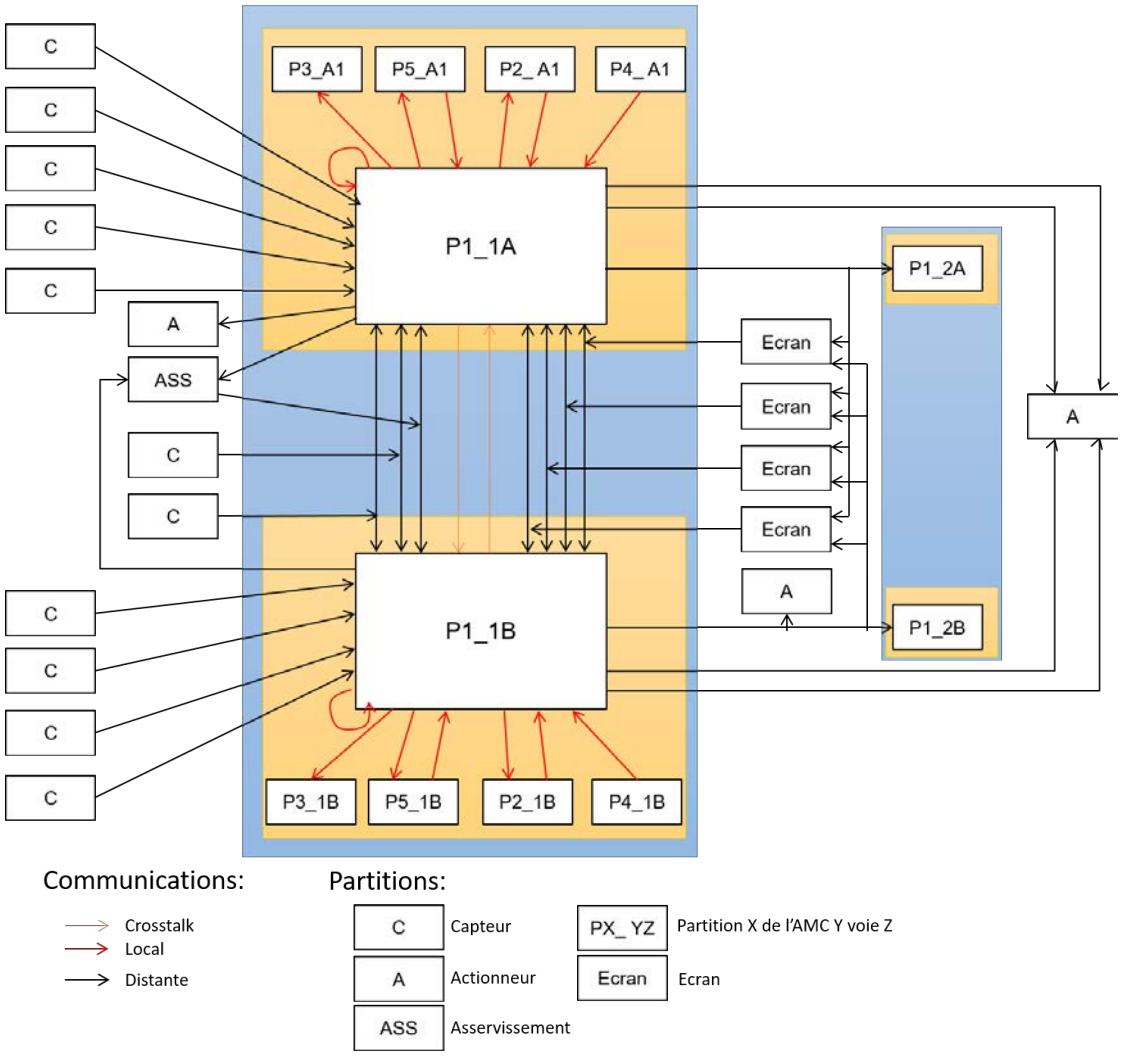


FIGURE A.2 Communication avec la partition principale d'un AMC

de l'autre voie les données que chacune a reçu. Ensuite, chaque partition P_1 croise les données reçues de capteurs de même nature. Pour terminer, P_1 génère les commandes pour les envoyer vers les asservissements, les actionneurs, les écrans et les enregistreurs.

Les écrans reçoivent d'un même AMC une commande de la voie A et un statut de la voie B. Afin de vérifier la véracité des informations reçues au niveau de l'écran, la commande et le statut doivent être envoyés au même instant. Les partitions P_1 dans un même AMC doivent échanger au même instant leurs données, ce qui implique que les deux processeurs contenant P_1 aient une horloge commune et communiquent sans latence ensemble.

A.2.2.2 Fraîcheur des données

Le système VMSA impose six contraintes temporelles associées à la fraîcheur des données illustrées Figure A.3. La première, représentée Figure A.3 partie a, consiste à récupérer toutes les informations acquises par la partition P_1 afin que P_2 puisse les traiter en moins de 50 ms. La seconde, illustrée Figure A.3 partie b, vérifie que le délai entre l'acquisition de la donnée d'un capteur et l'enregistrement de l'état de santé générale du véhicule donné par la partition P_3 soit inférieur à 500 ms [23]. La troisième, représentée Figure A.3 partie c, consiste à afficher les données des capteurs en moins de 400 ms [3]. Les trois dernières chaînes illustrées Figure A.3 parties d, e et f, enregistrent l'état des différents équipements en moins de 500 ms.

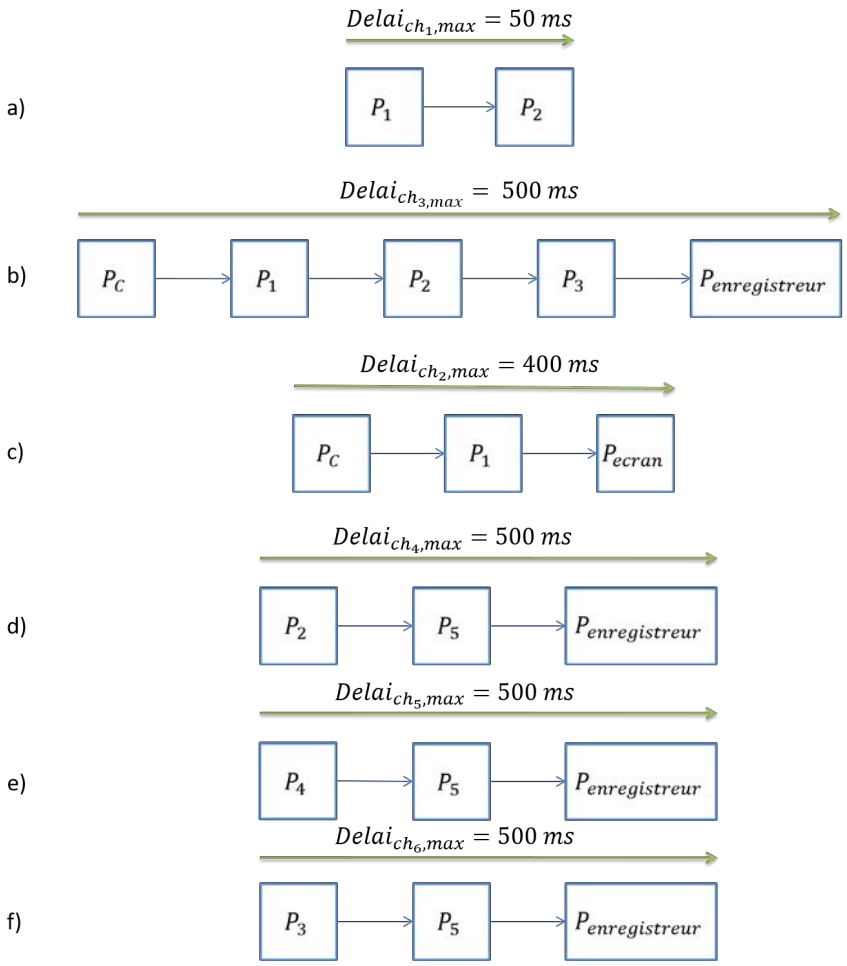


FIGURE A.3 Les chaînes de communication dans le système VMSA

Annexe B

Conception des ordonnancements

Nous présentons dans cette annexe nos choix de modélisation des MAF pour concevoir des ordonnancements à la volée. Nous détaillons les différentes complexités qui y sont associées afin d'amener nos choix de modélisation.

Nous montrons la complexité de définir un ordonnancement à la volée en nous appuyant sur l'exemple de la Table 3.1 avec ses trois chaînes de communication rappelées ci-dessous :

$$\mathcal{C}h = \{ch_1, ch_2, ch_3\}$$

$$ch_1 = \{P_1, P_2, P_3\}, D_{ch_1,max} = 30ms$$

$$ch_2 = \{P_2, P_5\}, D_{ch_2,max} = 40ms$$

$$ch_3 = \{P_4, P_5, P_6\}, D_{ch_3,max} = 60ms$$

Table B.1 Spécification temps-réel des paramètres de l'exemple illustratif

Partitions	C_i (ms)	T_i (ms)
P_1	3	10
P_2	2	10
P_3	2	20
P_4	4	40
P_5	1	40
P_6	4	40

Nous considérons une latence de communication de 1 ms si deux partitions sont distantes ; elle est nulle si celles-ci sont sur le même processeur.

B.1 Ajout d'une partition dans une MAF existante

L'ajout d'une partition P_j à une MAF peut modifier ses caractéristiques selon sa périodicité. Elle peut aussi ne pas être ordonnançable.

B.1.1 Périodicité des partitions différente

L'ordonnancement de partitions multi-périodiques est valide si toutes les partitions assignées à un slot peuvent s'exécuter dans un motif créé hors-ligne. Les caractéristiques de ce motif peuvent se retrouver modifiées selon la périodicité T_j de la partition ajoutée.

— Si $T_j < t_{intervalle}$, alors il faut découper chaque intervalle de la MAF en $\frac{t_{intervalle}}{T_j}$ intervalles, réduire la valeur de $t_{intervalle}$ à T_j, sélectionner dans quel intervalle les slots des partitions déjà ordonnancées doivent être placées et ordonnancer P_j strictement périodiquement dans tous les intervalles comme nous l'illustrons Figure B.1.
Sélectionner un intervalle pour un slot peut s'avérer complexe. En effet, un slot peut être ordonnancé dans le premier intervalle uniquement s'il s'agit de la première partition de l'intervalle d'origine, soit dans tous les intervalles, sachant que l'ordre des partitions n'est pas modifié. Si un intervalle k avant qu'il soit découpé contient N_k partitions, ceci revient à créer

$$\sum_{i=0}^{N_k-1} \binom{i}{2^{\frac{t_{intervalle}}{T_j}}} + i - 2$$

nouveaux motifs pour chaque intervalle, soit au maximum

$$\sum_{k=1}^{\frac{Hyp}{t_{intervalle}}} \sum_{i=0}^{N_k-1} \binom{i}{2^{\frac{t_{intervalle}}{T_j}}} + i - 2$$

nouveaux ordonnancements, avec N_k le nombre de partitions dans l'intervalle k de la MAF d'origine. Comme le montre la Figure B.1 lorsque nous essayons juste de réordonnancer 3 partitions dans 4 nouveaux intervalles, ceci amène une complexité supplémentaire non négligeable. En effet, dans cette illustration, nous constatons qu'il est impossible d'avoir les partitions notées 1, 2 et 3 dans le même intervalle. Par ailleurs, si nous devons ajouter une partition notée 4 dans tous les nouveaux intervalles, les cas a, d et f sont créés pour rien étant donné qu'il n'y a pas assez de temps pour que 4 s'exécute dans les intervalles où sont ordonnancés 2 et 3.

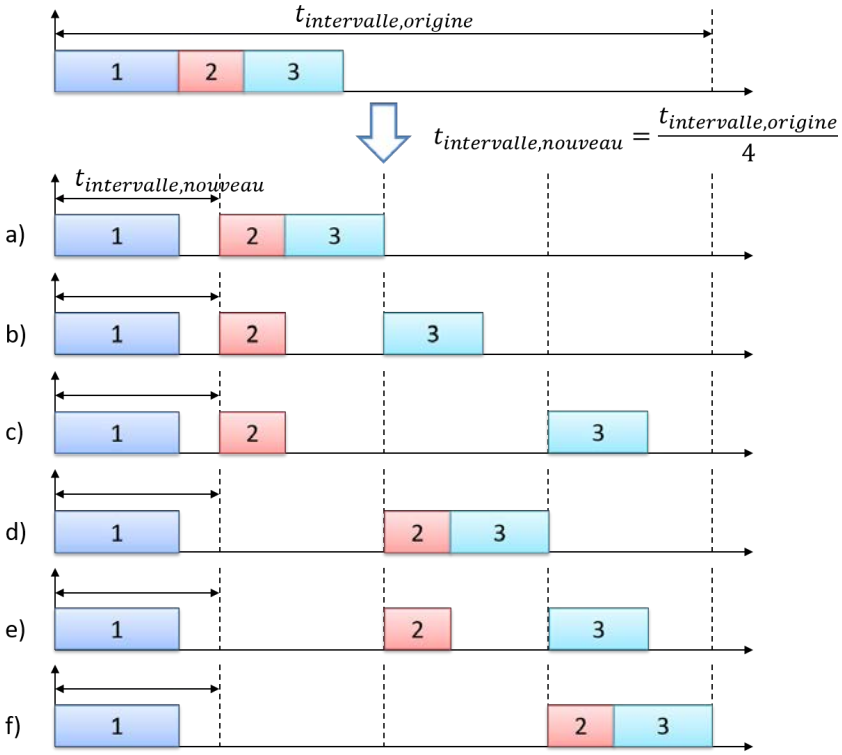


FIGURE B.1 Création de nouvelles MAFs lorsque la valeur de l'intervalle est divisée par 4

Pour remédier à ce problème, une solution consiste à définir que la durée de tous les intervalles de chaque MAF soit toujours égale à la plus petite des périodes des partitions. Ceci implique qu'une MAF, dès qu'elle est créée, peut contenir plus d'un intervalle. Nous utilisons ce principe dans cette thèse.

— Si $Hyp < T_j$, alors le motif d'ordonnancement doit être dupliqué $\frac{T_j}{Hyp}$ fois pour que la nouvelle hyper-période $Hyp_{nouvelle}$ devienne égal à T_j comme nous le montrons Figure B.2. Il suffit ensuite d'allouer la partition P_j dans les $\frac{Hyp}{t_{intervalle}}$ premiers intervalles de la MAF, les intervalles suivants étant la répétition de la MAF d'origine. Reprenons l'exemple Table 3.1. P_1, P_2 et P_3 sont déjà allouées et ordonnancées dans le premier processeur PE_1 comme le montre la Figure B.2. Dans ce cas, l'hyper-période de la MAF est égal à $T_3 = 20$ ms. Nous souhaitons ordonnancer P_5 qui a une période $T_5 = 40$ ms dans ce même processeur. La MAF précédente doit alors être dupliquée 2 fois (Figure B.2 partie a), puis P_5 peut être ordonnancée soit dans le premier intervalle, soit dans le deuxième (Figure B.2 partie b).

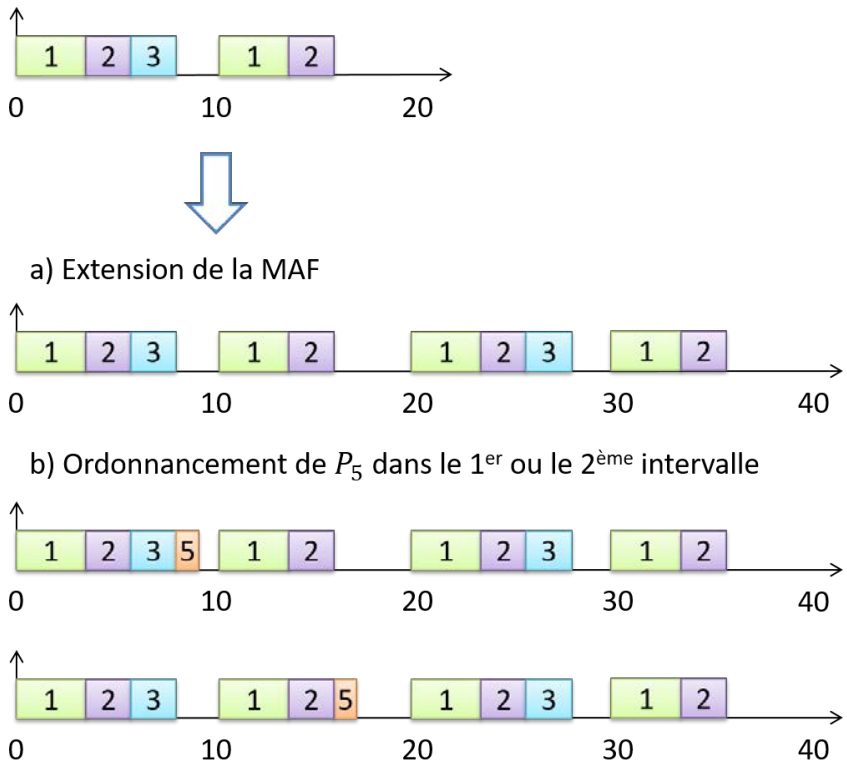


FIGURE B.2 Extension de la MAF lorsque la valeur de l'hyper-période est doublée

B.1.2 Choix d'emplacement d'une partition dans une MAF

Choisir l'emplacement d'un slot pour une partition peut s'avérer complexe. Différents choix s'offrent à nous :

— Effectuer autant de permutations qu'il y a de partitions dans le processeur. Comme le nombre d'intervalles dans une MAF peut être supérieur à 2, il faut effectuer ces permutations dans tous les intervalles. Dans l'exemple Figure B.3, il existe 18 façons d'ordonnancer les partitions P_1, P_4 et P_5 dans un même processeur.

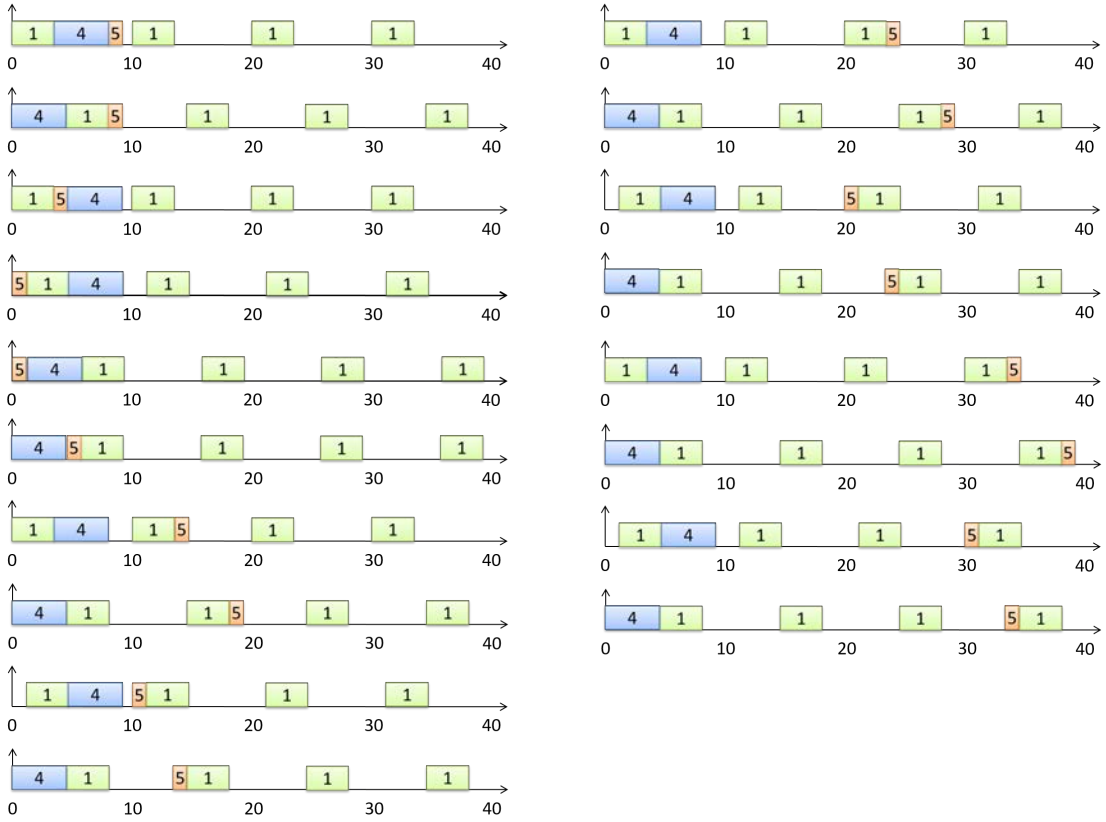


FIGURE B.3 Tous les ordonnancements de P_1, P_4 et P_5 possibles obtenus par permutations et décalages dans les intervalles d'un processeur

La complexité augmente si plusieurs partitions se situent dans plusieurs intervalles : il faut à la fois vérifier que toutes les partitions s'exécutent de façon strictement périodique et que les contraintes de bout-en-bout sont satisfaites. Si nous ordonnançons les autres partitions, i.e. P_2, P_3 et P_6, il existe 10 ordonnancements possibles présentés Figure B.4. Dans ces ordonnancements, P_2 et P_3 se situent dans deux intervalles : si elles sont permutées dans un intervalle, elles doivent

aussi être permutées dans tous les autres intervalles où elles existent afin que P_2 et P_3 soient exécutées dans le même ordre.

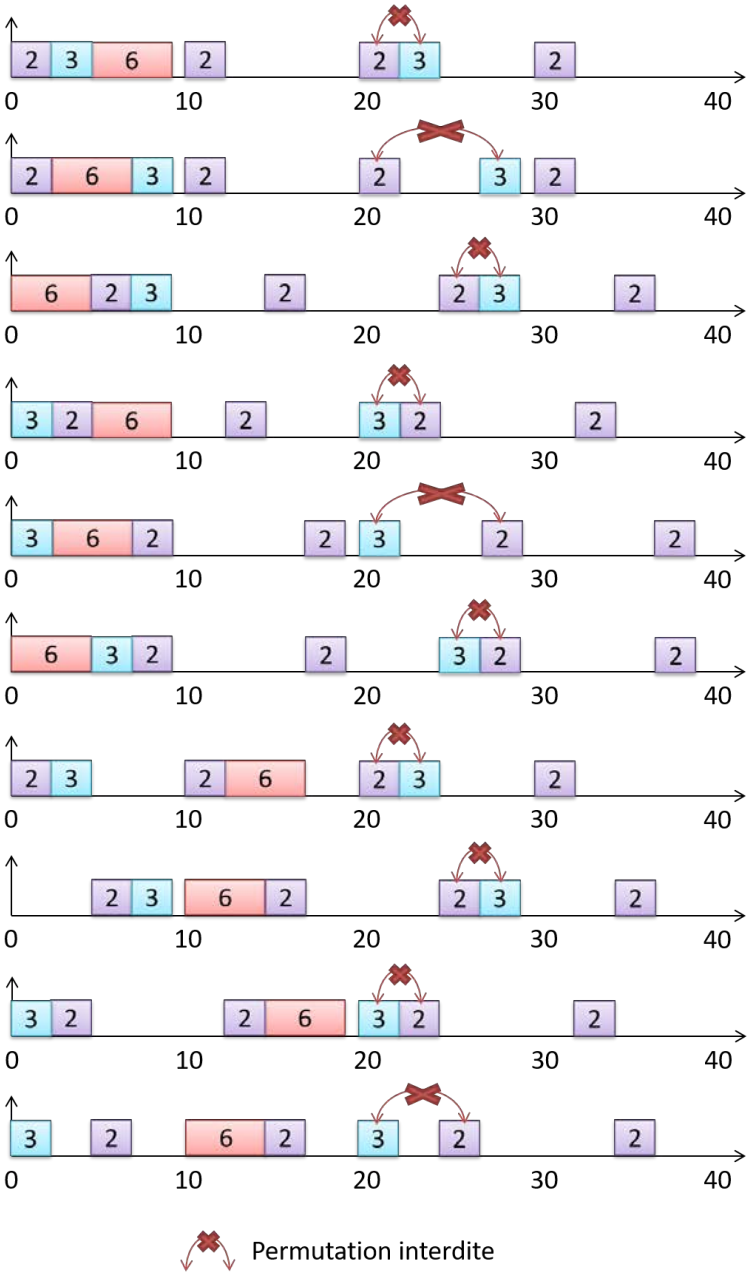


FIGURE B.4 Tous les ordonnancements de P_2, P_3 et P_6 possibles obtenus par permutations et décalages dans les intervalles d'un processeur

Par ailleurs, un ordonnancement dans un processeur peut être valide si les ordon-

nancements dans d'autres processeurs sont différents : pour un ordonnancement donné dans le processeur courant, il faut aussi modifier l'ordonnancement des autres processeurs si les contraintes de bout-en-bout ne sont pas respectées. Si nous prenons le premier ordonnancement Figure B.3 et nous souhaitons seulement valider les exigences des chaînes ch_1 et ch_3 avec une latence de communication de 1 ms, le premier, le troisième et le septième ordonnancements Figure B.4 valident ces exigences de bout-en-bout. Les autres ordonnancements ne satisfont pas les contraintes de bout-en-bout de ch_1 et ch_3, quel que soit l'ordonnancement de la Figure B.3. L'ajout d'une nouvelle partition dans un processeur modifie l'ordonnancement courant dans un processeur, insatisfaisant les exigences de bout-en-bout. Rechercher un nouvel ordonnancement dans un autre processeur peut résoudre ce problème. La difficulté revient à retenir et à ne pas revérifier des ordonnancements invalides, ce qui est coûteux en temps et en mémoire.

— Intercaler la nouvelle partition dans le motif déjà prédéfini, en prenant en compte la stricte périodicité des partitions. Ceci revient à décaler temporellement les slots alloués à d'autres partitions pour exécuter le slot de la partition ajoutée au processeur, comme nous l'illustrons Figure B.5.

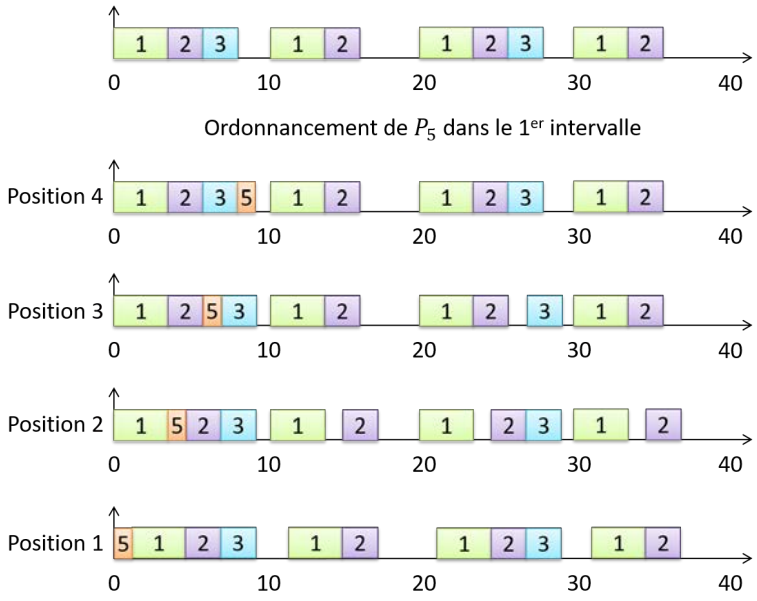


FIGURE B.5 Décalage des partitions lors de l'ajout d'une nouvelle partition dans une MAF

Résumé

Nous nous intéressons dans cette thèse à l'implémentation des architectures modulaires intégrées (IMA) dans les hélicoptères. À la différence des avions qui intègrent dans des baies avioniques des processeurs puissants, nous cherchons à répartir des systèmes hélicoptères sur un plus grand nombre de processeurs moins puissants. Notre étude de cas nous amène à trouver un compromis entre la distribution et la satisfaction des exigences de communication de bout-en-bout de ces systèmes. Nous répondons à cet enjeu en deux parties : 1/ la recherche d'allocations en considérant au même niveau l'ordonnancement et la satisfaction des exigences de bout-en-bout avec des latences de communication figées, 2/ les réseaux viables dans une architecture distribuée donnée, basés sur les latences de communication admissibles entre les différentes fonctions avioniques. Les méthodes que nous proposons répondent au besoin de l'étude de cas industriel, y compris pour des systèmes de plus grande taille.

Mots clés : Architecture avionique, Systèmes temps-réel embarqués, IMA distribuée, Ordonnancement multiprocesseur, Exigence de bout-en-bout, Asynchronisme.

Abstract

This thesis addresses the issue of implementing integrated modular architectures (IMA) in helicopters. Unlike aircraft which integrate powerful processors in avionics bays, we look for distributing helicopter systems on a larger number of less powerful processors. Our case study leads to find a trade-off between the distribution and the satisfaction of end-to-end communication requirements of helicopter systems. We answer at this topic in two steps : 1/ the search of allocations schemes considering altogether the scheduling and the satisfaction of end-to-end constraints with fixed communication latencies, 2/ the networks in a given distributed architecture based on eligible communication latencies between the different avionics functions. Our methods take into account an industrial case study need, including bigger size systems.

Keywords : Avionics architecture, Real-time embedded systems, Distributed IMA, Multiprocessor scheduling, End-to-end requirements, Asynchronism.