Vers une nouvelle génération de systèmes de test et de simulation avionique dynamiquement reconfigurables

George Afonso

Université Lille1 Sciences et Technologies

THÈSE

Présentée et soutenue publiquement le 02 juillet 2013

pour obtenir le titre de

Docteur en Informatique

par

George AFONSO

Vers une nouvelle génération de systèmes de test et de simulation avionique dynamiquement reconfigurables

préparée en collaboration avec EADS IW, Eurocopter et l'Université de Valenciennes

Jury :

Président :	Pr. Serge PETITON	-	Université de Lille1
Rapporteurs :	Pr. Frederic PETROT	-	Université de Grenoble
	Pr. Sébastien PILLEMENT	-	Université de Nantes
Directeur :	Pr. Jean-Luc DEKEYSER	-	Université de Lille1
Co-encadrants :	Dr. Rabie BEN ATITALLAH	-	Université de Valenciennes
	Dr. Nicolas BELANGER	-	Eurocopter
Examinateurs :	Dr. Werner DE-RAMMELAERE	-	Airbus
	Dr. Benoît MIRAMOND	-	Université de Cergy-Pontoise
Invité :	Mr. Wenceslas GODARD	-	EADS Innovation Works

Je remercie mes parents, Maria de Lurdes et Hilario Afonso qui ne m'ont jamais imposé de limites quand aux ambitions que j'ai pu avoir et que j'ai encore aujourd'hui. C'est grâce à leur soutient et à leurs encouragements que vous lisez ce document aujourd'hui.

Je tiens à remercier Isabelle Terrasse et Nicolas Bélanger sans qui cette thèse n'aurait pas eut lieu.

Je remercie mes encadrants, Jean-Luc Dekeyser, Rabie Ben Atitallah et Nicolas Bélanger pour le temps précieux qu'ils m'ont accordé, le savoir qu'il ont partagé avec moi et leur soutient sans faille. Ce fut un honneur et un véritable plaisir pour moi de bénéficier d'un tel encadrement. Je joins également à ces remerciements, David Duvivier qui m'a offert son aide et ses connaissances lors de ma présence à l'université de Valenciennes.

J'exprime toute ma gratitude envers ceux qui ont accepté de valider ce travail de thèse. Leurs questions et remarques pertinentes m'ont permis d'améliorer le manuscrit ainsi que de préparer ma soutenance dans les meilleures conditions. Je remercie Frédéric Pétrot et Sébastien Pillement pour avoir accepté de rapporter mon manuscrit dans un délai très court. Je remercie également Wenceslas Godard, Werner de Rammelaere, Benoît Miramond et Serge Petiton pour l'intérêt qu'ils ont témoigné par la lecture de mon manuscrit et par leur présence à la soutenance publique.

Je remercie tout particulièrement ma compagne Marion Dufour, pour son soutient, ses encouragements, son amour et surtout, sa patience...

Merci à tout ceux qui m'ont aidé de près ou de loin au bon déroulement de cette thèse, aux relecteurs qui ont permis (ou non) ma participation aux différents évènements scientifiques et industriels, aux étudiants de l'université de Valenciennes que j'ai pu côtoyer lors de mes visites dans le "grand-nord", à mes collègues et moutons d'Eurocopter.

Enfin, je remercie mes amis et ma famille, merci, obrigado, thank you !

Table des matières

TABLE DES MATIÈRES

TABLE DES MATIÈRES

Liste des figures

LISTE DES FIGURES

LISTE DES FIGURES

Glossaire

La liste des principaux acronymes utilisés dans ce document sont rassemblés dans la table 1

TABLE 1 – Glossaire

Acronyme	Définition
ADEOS	*Adaptive Domain Environment for Operating Systems*
AMD	*Advanced Micro Devices*
AMP	*Asymmetric MultiProcessing*
API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
CPU	*Central Processing Unit*
DDR	*Double Data Rate*
DMA	*Direct Memory Access*
DSP	*Digital Signal Processing*
EADS	*European Aeronautic Defence and Space Company*
FIFO	*First In First Out*
FPGA	*Field-Programmable Gate Array*
I/O	*Input/Output*
LPT	*Longest Processing Time*
MLPT	*Modified Longest Processing Time*
OS	*Operating System*
PC	*Personal Computer*
PID	*Process IDentifier*
RAM	*Random Access Memory*
RISE	*Real tIme Simulation Environment*
RT	*Real Time*
RTOS	*Real Time Operating System*
SIGUSR	*SIGnal reserved for the USeR*
SMP	*Symmetric MultiProcessing*
TCP	*Transmission Control Protocol*
TTM	*Time To Market*
T & S	*Test and Simulation*
UUT	*Unit-Under-Test*
VME	*Versa Module Eurocard*

LISTE DES FIGURES

Chapitre 1

Introduction

1.1 Contexte

Au cours des vingt dernières années, les phases de test et de simulation ont toujours été considérées comme des phases cruciales du cycle de développement d'un nouvel hélicoptère. En effet, en plus de la validation du bon fonctionnement des nouveaux équipements, ces phases permettent de faciliter les phases de certification cruciales dans le domaine aéronautique.

Comme décrit la figure 6.13, le cycle de développement d'un nouvel équipement suit un cycle en V où les phases de test et de simulation occupent des étapes primordiales. Dans un premier temps, lors de la phase de développement, la spécification permet de réaliser une première étude à plusieurs niveaux de ce que sera l'architecture de ce nouvel équipement. Ensuite, lors de la phase d'intégration, les spécifications définies précédemment sont transmises à l'équipementier qui effectuera le développement répondant à toutes ces dernières. Enfin, l'équipement sera testé sur banc de test pour une validation ultérieure en vol par les pilotes d'essais. En marge de la phase de test, la phase de simulation permet aux pilotes de se former sur des nouveaux appareils. Pour ce faire, de nombreux modèles de simulation sont développés pour offrir un environnement réaliste dans lequel le pilote et son appareil simulé pourront évoluer.

Les systèmes de tests actuellement utilisés chez Eurocopter ont montré leur limite pour satisfaire la demande croissante de puissance de calculs nécessaire pour les phases de validation des nouveaux équipements. En effet, ces nouveaux équipements, leurs nouvelles fonctionnalités associées ainsi que leur capacité croissante de calcul complexifient leur validation. Dès lors, il est nécessaire d'augmenter la capacité de calcul des systèmes de test et

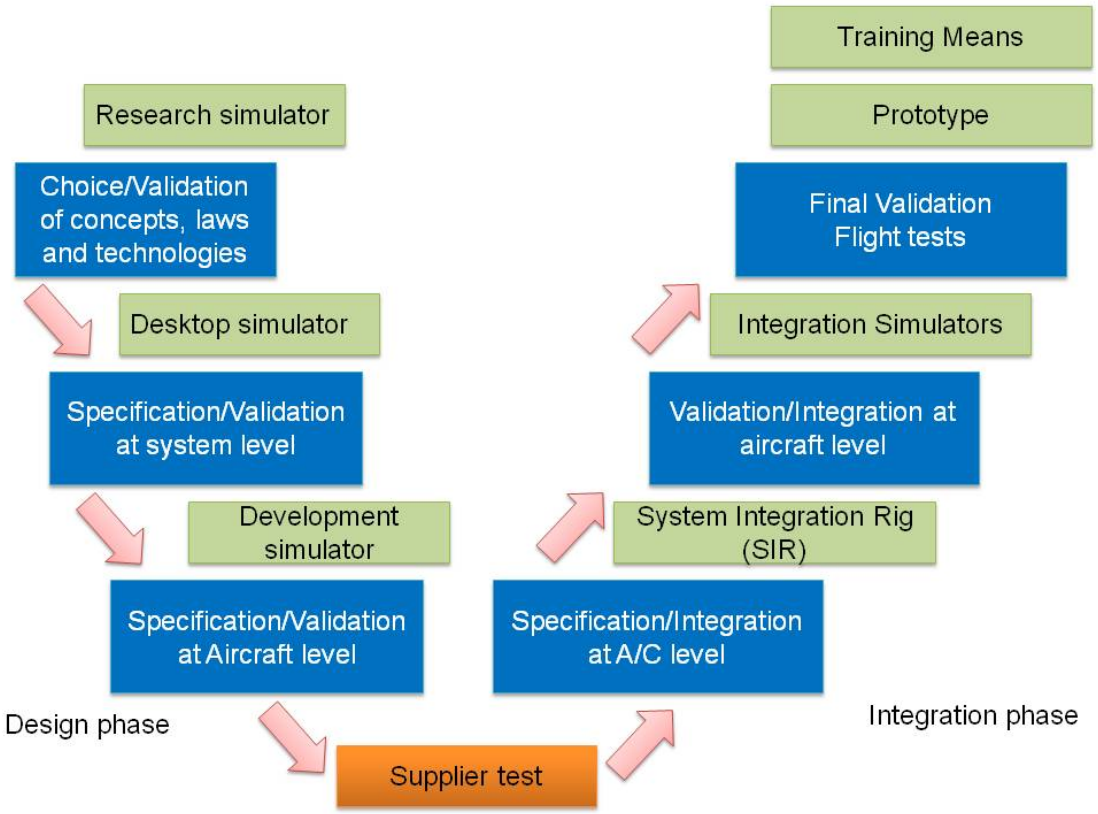

FIGURE 1.1 – Cycle de développement d'un nouvel équipement chez Eurocopter

de simulation. En 2008, le bureau d'études d'Eurocopter a entreprit une profonde réflexion sur la vocation pro-active des systèmes de test. Le choix de repenser ces systèmes dans leur globalité était nécessaire. De plus, cette réflexion est commune à plusieurs entités du groupe EADS qui est fortement impliqué dans le domaine aéronautique.

1.2 Problématique

Jusqu'alors, les systèmes de test étaient basés sur des solutions très spécifiques en termes logiciel et matériel. En effet, d'un point de vue architecture, les systèmes de tests actuels sont basés sur des fonds de panier VME [39] comme montré dans la figure 1.2.

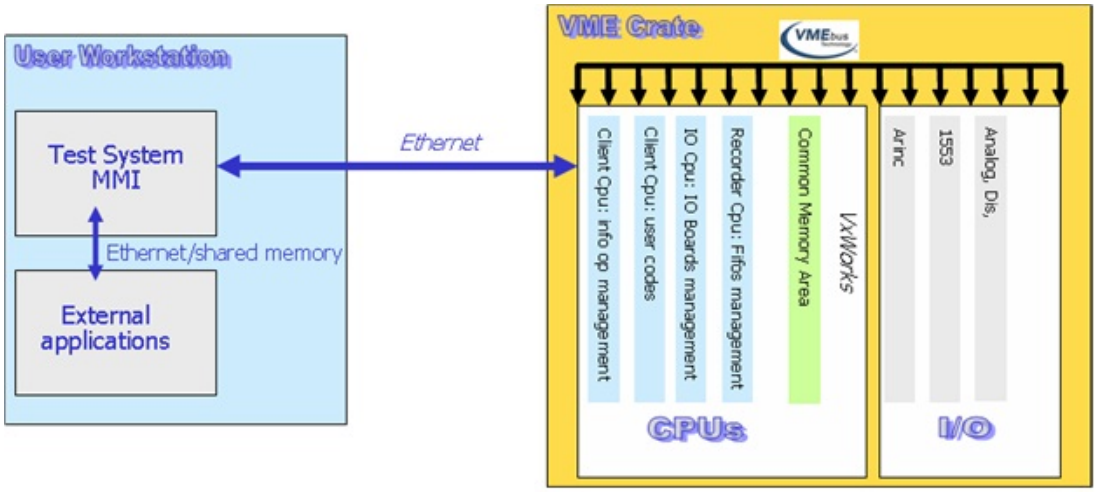

FIGURE 1.2 – Architecture VME des bancs de test avionique

Sur cette architecture, la capacité de calcul est concentrée sur des cartes CPU spécialisées. En effet, chacune d'entre elle possède une fonction particulière, comme la fonction d'enregistrement, la fonction de gestion des entrées-sorties, la gestion des informations opérationnelles et la gestion des codes utilisateurs. A ces dernières sont associées des cartes d'entrées-sorties interférant avec les équipements sous test en utilisant leur bus avionique associé (Arinc 429, 1553, CAN, etc). D'un point de vue logiciel, le domaine d'application requiert un environnement capable de répondre à des contraintes temps-réel. A l'heure actuelle les systèmes de test sont basés sur l'environnement temps-réel dur VxWorks proposé par WindRiver.

Une première limitation de cette solution concerne les cartes CPU. En effet, les processeurs embarqués ne sont plus capables de délivrer une fréquence de calcul suffisante pour

envisager l'ajout de fonctionnalités supplémentaires à la phase de test tel que le test simultané de plusieurs équipements. De plus, l'utilisation du système d'exploitation temps-réel VxWorks de WindRiver augmente la complexité et le coût de ce produit.

Une seconde limitation concerne la gestion de l'obsolescence des cartes CPU, Eurocopter s'est vu contraint d'investir dans trois coûteux retrofits durant les dix dernières années. Cette limitation concerne également les cartes d'entrées-sorties, de plus en plus rares et donc, coûteuses à l'achat. Ces technologies à la fois non évolutives et coûteuses deviennent un frein pour le développement rapide de nouveaux équipements.

Une troisième limitation concerne l'importante gestion de ces systèmes de tests, en effet, chaque équipement de chaque hélicoptère possède une configuration particulière (i.e. nombre de cartes CPU défini, nombre et type de cartes d'entrées-sorties défini) qui doit être maintenue dès lors qu'un appareil en vol possède cet équipement. Cette gestion est également contraignante lors de la maintenance de ces systèmes, pouvant rendre inactives les équipes liées à la vérification des équipements.

Dans un tel contexte, il est devenu nécessaire de repenser ces environnements de test dans le but de répondre à la problématique imposée par le système actuel ainsi qu'aux nouveaux besoins. Le travail proposé se place plus exactement à l'intersection de trois axes de recherche.
Le premier proposera un modèle d'exécution dynamique adapté aux contraintes imposées par le domaine d'application ainsi que les capacités du système proposé. Le second concerne le support matériel pour la prochaine génération des bancs de test et de simulation qui permettra de répondre à l'ensemble des contraintes liées au domaine d'application. Le dernier concerne le support logiciel qui permettra le respect des contraintes temps-réel et en particulier le placement dynamique des modèles composant les applications de test et de simulation à travers les unités de traitement composant le support matériel des systèmes de test et de simulation.

L'augmentation des besoins en termes de capacités de calcul, de généricité de l'architecture ainsi que de la réduction des coûts liés à l'architecture actuelle nous mène à une réflexion sur les architectures des systèmes de test et de simulation ainsi que les environnements d'exécution répondant aux besoins de notre domaine d'application. Cette réflexion devra également porter sur les outils liés au développement des systèmes de test et de simulation afin de faciliter leur intégration chez Eurocopter. Ce nouvel environnement logiciel et matériel dédié au test et à la simulation devra pallier aux limites actuelles des systèmes existants et devra répondre à plusieurs nouvelles exigences. Dans ce cadre, nous

avons identifié trois facteurs définissant la problématique de notre thèse.

Quel modèle d'exécution pour les futurs bancs de test et de simulation ?

Le défi est ici de proposer un modèle d'exécution adapté aux besoins du domaine d'application accélérant le cycle de validation des nouveaux équipements chez Eurocopter. En effet, ce modèle d'exécution devra permettre aux systèmes de test de s'adapter à l'équipement à tester et aux contraintes dynamiques pouvant faire échouer la session de test par le non respect des contraintes temps-réel imposées. De plus, ce modèle devra être ouvert permettant ainsi des évolutions, par exemple des nouvelles fonctionnalités sur les systèmes de test.

Quel support d'exécution matériel pour la prochaine génération de systèmes de test et de simulation ?

Le défi est ici de proposer un support d'exécution matériel capable de répondre tant au niveau des performances requises par le domaine d'application qu'offrant des capacités évolutives. En effet, chaque banc de test est aujourd'hui dédié à un équipement d'un appareil spécifique. En effet, dans le but de pouvoir reproduire une session de test, le système qui a servi à la validation doit être identique à chaque session. Cela contraint Eurocopter à conserver les architectures en leur état pendant des durées pouvant atteindre plusieurs décennies. Le défi ici est de concevoir une architecture générique pouvant s'adapter à l'équipement à tester. Dans un premier temps, cela permettrait d'améliorer la productivité, chaque banc pouvant tester plusieurs équipements. Ensuite, la généricité de l'architecture permettrait une meilleure gestion des stocks des différentes unités de calcul et d'entrées-sorties.

Quel support logiciel pour la prochaine génération de systèmes de test et de simulation ?

Une fois le modèle d'exécution ainsi que le support d'exécution matériel définis, il est nécessaire d'implémenter un environnement logiciel qui supportera le modèle d'exécution ainsi que l'architecture choisis. De plus, cet environnement devra respecter les contraintes temps-réel imposées et devra collecter efficacement les données nécessaires à la supervision de l'architecture tout en répondant aux contraintes industrielles. Pour ce faire, les tâches liées à l'application de test et de simulation seront allouées de façon optimales sur les unités de traitement disponibles du support matériel et pourront être ré-allouées dynamiquement en fonction des contraintes imposées lors de la session en cours d'exécution.

1.3 Contributions

Un modèle d'exécution pour les futurs bancs de test et de simulation

1. Nous avons définit les fonctionnalités que nous mettrons en place dans le cadre d'une nouvelle génération de systèmes de test et de simulation avioniques permettant d'accélérer le cycle de développement des équipements embarqués dans les futurs hélicoptères.
2. Nous avons proposé un modèle d'exécution dynamique qui s'appuie sur une architecture hétérogène dédié aux systèmes de test avionique. Ce modèle d'exécution apportera des capacités de flexibilité, de généricité et d'adaptabilité au système proposé. Pour ce faire, nous avons spécifié l'ensemble des fonctions nécessaires à la mise en place de ce modèle d'exécution sur notre système.

Un nouveau support d'exécution matériel pour la prochaine génération de systèmes de test et de simulation

1. Un support matériel répondant aux contraintes liées au domaine d'application a été proposée. Cet dernier est basé sur deux nœuds de calcul hétérogènes CPU-FPGA offrant différents compromis entre capacité de calcul haute performance, généricité et adaptabilité aux événements pouvant perturber le bon déroulement du test.
2. Nous avons exploré et implémenté des moyens de communication entre les différents nœuds de calcul proposés. De plus, nous avons étudié leurs capacités et optimisations possibles le but de réduire le coût de cette communication au sein de notre support matériel hétérogène. Nous avons proposé une solution "clef en main" dans la perspective de réduire la complexité du développement d'applications sur ce type d'architecture.
3. Nous avons proposé une méthodologie de développement permettant un passage aisé du profilage des modèles de test et de simulation vers une nouvelle implémentation tirant parti des capacités offertes par le support matériel et logiciel proposés. Cette méthodologie a permit de favoriser l'implémentation des modèles "legacy" au sein de notre système. Cette méthodologie a été appliquée sur plusieurs modèles de test et de simulation avioniques.

Un support logiciel pour la prochaine génération de systèmes de test et de simulation

1. Différentes solutions temps-réel ont été explorées dans le but de trouver le meilleur compromis entre performance, temps de développement et coût de la maintenance. Une solution a été développée afin de répondre à l'ensemble des critères requis par l'industriel.
2. Un environnement de supervision a été développé afin de détecter les moments critiques nécessitant une reconfiguration et ainsi que de contrôler dynamiquement le placement des tâches composant l'application de test et de simulation, qu'elles soient logicielles ou matérielles. Cet environnement embarque une heuristique de placement des tâches conçue dans le but de trouver un placement optimal.
3. Nous avons mis à contribution l'ensemble des capacités de notre architecture ainsi que de notre support logiciel dans le but de proposer un environnement unifié pour le test et la simulation avionique chez Eurocopter.

1.4 Plan

En respectant l'ordre de nos contributions décrites ci-dessus, notre manuscrit est organisé selon le plan suivant :

Chapitre 2 : État de l'art. Dans ce chapitre, nous présenterons le contexte de nos travaux qui abordent les systèmes de tests avioniques, les architectures hétérogènes, les environnements temps-réel ainsi que les algorithmes de placement de tâches sur architectures hétérogènes. A partir de cette étude, nous positionnerons nos travaux et nous donnerons les grandes lignes de nos contributions.

Chapitre 3 : Un modèle d'exécution dynamique pour les architectures hétérogènes dédié au test et à la simulation. Dans ce chapitre, nous présenterons le besoin et les contraintes liés au domaine d'application et à l'industriel. Ensuite, nous exposerons notre réflexion concernant les capacités nécessaires à implanter au sein des futurs systèmes de test et de simulation. Enfin, nous proposerons le modèle d'exécution répondant aux contraintes et aux fonctionnalités précédemment décrites.

Chapitre 4 : Un support matériel pour la prochaine génération de systèmes de test et de simulation. Dans ce chapitre, nous présenterons l'architecture hétérogène

choisie pour les futurs systèmes de test et de simulation. Une méthodologie de développement permettant des optimisations logicielles et matérielles sera détaillé afin que le passage à ce nouveau support matériel soit facilité chez l'industriel. Nous mettrons également en avant les nouvelles possibilités envisageables pour cette architecture en exploitant les capacités intrinsèques des nœuds de calculs afin d'améliorer significativement les temps d'exécution.

Chapitre 5 : Un support logiciel pour la prochaine génération de systèmes de test et de simulation. Dans ce chapitre, nous décrirons l'ensemble des fonctions développées composant l'environnement logiciel proposé. De plus, nous mettrons en avant l'heuristique développée dans le cadre de la fonction de placement de tâches sur notre architecture hétérogène. Cette heuristique est embarquée dans l'environnement de supervision qui permettra de vérifier l'état de notre architecture avec une grande précision temporelle afin de respecter les contraintes temps-réel requises par l'application.

Chapitre 6 : Étude de cas et Validation expérimentale. Dans ce chapitre, nous décrirons les étapes techniques nécessaires pour la mise en place de l'environnement logiciel proposé. Ensuite, nous présenterons l'étude d'une session de test et de simulation soumise à des contraintes sévères afin de mettre en avant la validation fonctionnelle ainsi que la stabilité de l'environnement. Enfin, nous projetterons les concepts présentés dans le cadre de cette thèse vers d'autres domaines d'applications.

Chapitre 7 : Conclusion et Perspectives. Nous concluons cette thèse par le bilan des travaux effectués et nous détaillons les contributions apportées avant d'aborder plusieurs perspectives à nos travaux.

Chapitre 2

Etat de l'art

2.1 Introduction

La motivation principale de notre travail est la mise au point d'une nouvelle génération de bancs de test et de simulation qui permettra d'obtenir les performances exigées par la phase de test et simulation chez Eurocopter. L'apport d'une nouvelle architecture devra s'accompagner d'une méthodologie de développement. En effet, l'une des difficultés majeures des concepteurs se situe dans le développement d'applications destinées à être exécutées sur une nouvelle architecture. Cette méthodologie permettra de mieux maitriser l'architecture et de réduire le temps de développement. Une des exigences de cette nouvelle architecture est sa généricité qui passera par la nécessité d'une capacité de reconfiguration accompagné d'un environnement qui permettra de la contrôler.
Les travaux entrepris dans le cadre de cette thèse visent plusieurs points :

1. Architectures de test et de simulation avionique
2. Architectures hétérogènes dynamiquement reconfigurables
3. Environnements temps-réel

Cet ensemble d'objectifs nous amène à diviser ce chapitre sur l'état de l'art en 3 sections. Une section est dédiée à chaque objectif. Ainsi la section 2 introduit les architectures de test et de simulation actuelles. Un tour d'horizon des technologies et solutions proposées dans le cadre des architectures hétérogènes dynamiquement reconfigurables dans la section 3. Enfin, nous aborderons les différentes solutions temps-réel ainsi que les heuristiques de placement de tâches.

2.2 Systèmes de test et de simulation avionique

2.2.1 Notions élémentaires sur le domaine du test et de la simulation avionique

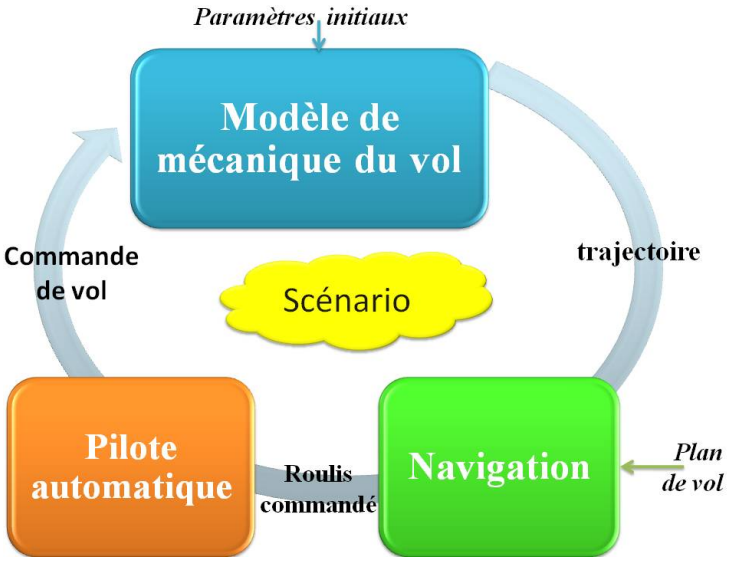

FIGURE 2.1 – Boucle de test simplifiée

Avant d'aborder l'état de l'art concernant les systèmes de test avioniques, il convient d'introduire quelques notions liées à ce domaine d'application.
Lors d'une session de test et de simulation, il est nécessaire de simuler le comportement global de l'appareil concerné ainsi que tous les paramètres environnementaux comme les équipements non présents, les conditions climatiques, l'environnement géographique dans lequel va évoluer l'hélicoptère, etc. La figure 2.1 présente une version simplifiée d'une boucle de test nécessaire incluant trois modèles. La *mécanique de vol*, la *navigation* ainsi que le *pilote automatique*. Lors de la phase d'initialisation, le modèle de mécanique de vol va recevoir plusieurs paramètres tels que sa position initiale dans l'espace, sa vitesse initiale et l'hélicoptère concerné par le scénario.

Dans le cadre de ce projet de thèse, nous nous sommes plus particulièrement intéressés à l'implémentation de la mécanique du vol décrite dans la figure 2.2. Ce modèle prend en compte plusieurs paramètres comme le cap, l'assiette, le roulis, etc. Chacun de ces paramètres est calculé grâce à une ou plusieurs équations. Il possède quatre entrées qui représentent la position absolue du manche de l'appareil. Les sorties viennent alimenter

FIGURE 2.2 – Modèle de mécanique du vol

des capteurs de type accéléromètre ou encore anémomètre pouvant être représentés par d'autres modèles ou bien les équipements réels. Elles seront ainsi utilisées par le modèle suivant dans la figure 2.1 : Le **pilote automatique**.

De cette phase d'initialisation le modèle de mécanique de vol va retourner une position dite d'"équilibre" de l'hélicoptère dans son environnement simulé. Lors de la session de test, la mécanique de vol fournira au modèle de navigation la trajectoire actuelle de l'appareil ainsi que ses données de vol. Ce dernier va déterminer grâce au plan de vol la nouvelle trajectoire à prendre ou à conserver au modèle de pilote automatique qui agira sur les commandes de l'appareil via le modèle mécanique du vol en conséquence. Cette boucle de test est très simplifiée par rapport au cas réel. En effet, le nombre de modèles cohabitant dans la même session peut attendre la centaine.

C'est le scénario décrit par l'utilisateur qui va déterminer l'utilisation des modèles ainsi que la présence d'équipements réels. De plus, c'est ce scénario qui va déterminer les périodes de temps attribuées à l'exécution de ces modèles de simulation. Le respect de ces périodes est la condition nécessaire pour la validité de la session de test. Ainsi, dans le cadre d'une simulation complète, c'est à dire sans aucun équipement réel (pilote automatique, altimètre, etc) dans la boucle, il n'y a pas nécessité d'utilisation de cartes d'entrées/sorties vers ces équipements comme décrit précédemment. Dans ce cas, ce sont les versions simulées de ces équipements qui seront dans la boucle de test. Dans le cadre d'une session de test, un équipement simulé comme le pilote automatique pourra être présent dans la boucle précédemment décrite permettant alors sa validation.

2.2.2 Les systèmes de test chez Eurocopter

En 1984, lors du lancement de la conception du Tigre, un outil performant permettant de tester les équipements a été créé. Cet outil de test, appelé MONA, est utilisé pour la mise au point des hélicoptères en permettant la simulation des interfaces en avance de phase, le test d'un équipement avec son environnement complet simulé (STB : Software Test Bench) ainsi que le test d'équipements entre eux (SIR : System Integration Rig).

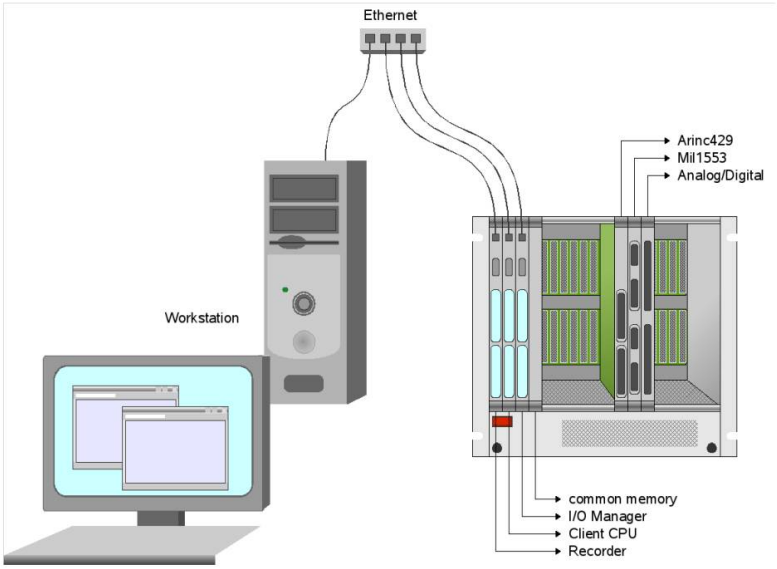

FIGURE 2.3 – Système de test avionique

Les étapes STB et SIR font l'objet de qualifications officielles imposant de fortes contraintes. En ce qui concerne la partie matérielle, les bancs de tests sont composés d'un châssis VME (fond de panier) comme celui présenté sur la figure 2.3, de cartes d'entrées-sorties effectuant les interfaces avec les équipements ainsi que de cartes CPU accueillant la partie software temps-réel et les codes utilisateurs de test. Le système d'exploitation temps-réel est pSOS et les stations de travail permettant de faire toute la configuration des bancs se basent sur des machines spécifiques (Workstation sur la figure 2.3).

Dix ans plus tard, lors de la conception du NH90, l'outil MONA est mis à jour pour permettre l'ajout de fonctionnalités. Ce nouvel outil est baptisé ANAIS. Le système d'exploitation temps-réel VxWorks de WindRiver est désormais utilisé afin de permettre la chaîne de compilation ADA.

A la fin des années quatre-vingt dix, la fusion avec le groupe Allemand a conduit à

l'homogénéisation des outils de tests, ANAIS et l'outil Allemand ATTILA sont unifiés et en résultera ARTIST qui est encore aujourd'hui déployé chez Eurocopter.

Après des années d'utilisation, ARTIST a besoin d'évoluer pour répondre à tous les besoins actuels. D'une part, l'outil a été limité par les performances du matériel. A l'époque, la nécessité de réécrire des parties du système d'exploitation ou des drivers était indispensable à l'obtention des performances temps-réel requises, ce qui a été très coûteux. À l'heure actuelle, les performances des architectures disponibles ne nous limitent plus vis-à-vis de notre application à contraintes temps-réel molles et ainsi notre outil de test pourra évoluer vers une standardisation avec l'interface matérielle.

Ensuite, une part importante des contraintes est de garder la compatibilité de l'outil avec l'ancienne configuration. En effet, la durée de vie de l'outil de test est la même que celle de l'hélicoptère. Le Tigre est toujours en production et l'outil de test actuel doit permettre d'effectuer toutes les opérations désirées. De plus le profil des utilisateurs d'ARTIST n'est plus un profil informatique mais un profil spécialiste systèmes avioniques, ce qui impose un support de bancs plus important de la part des concepteurs et des développeurs. Il peut donc être intéressant de proposer un outil intuitif, masquant la complexité système inhérente aux bancs.

L'utilisation de matériels et logiciels standards permettrait de réduire les coûts de déploiement de bancs de tests aujourd'hui importants. Le matériel utilisé est spécifique (Châssis VME), considéré comme obsolète chez Eurocopter depuis 2009 et donc très coûteux à maintenir et le coût des licences est élevé. L'utilisation de matériels et logiciels standardisés permettrait de réduire significativement ces coûts.

Depuis quelques années, RISE [4], un nouvel outil de simulation est proposé. Ce dernier permet aux pilotes souhaitant se former aux nouveaux appareils un environnement simulé proposant une vue à 180 ° diffusée dans une sphère de simulation comme celle présentée dans la figure 2.4. Le système utilisé est composé d'une architecture multi-cœur embarquée dans un PC standard et d'un environnement Linux. RISE et ARTIST partagent leurs modèles permettant ainsi une mutualisation de ces derniers. Toutefois, cet outil ne permet ni de respecter les contraintes temps-réel ni de communiquer avec des équipements sous-test.

2.2.3 Les solutions pour le test avionique disponibles

Les architectures des interfaces d'entrées-sorties des bancs de tests avioniques ont été jusque lors dominées par l'utilisation de châssis VME. D'ailleurs, les systèmes de test déployés dans le domaine aéronautique sont similaires à ceux encore aujourd'hui déployés

FIGURE 2.4 – Sphère de simulation avionique

chez Eurocopter. En effet, parmis ceux développés par les entreprises reconnues dans le domaine, nous pouvons citer AIDASS [30] utilisé notamment pour l'EuroFighter d'EADS Military Air Systems, U-test [14] développé par EADS Test & Services et ADS2 [59] par Techsat GmbH.

Depuis plusieurs années, les fournisseurs de solutions proposent le même type de produit "fond de panier" où seul le médium de communication varie. En effet, l'utilisation d'autres bus tels que le PCI, compact PCI, PXI devient courante aujourd'hui.

La société SILKAN (anciennement ARION) [54] propose des solutions "clefs en main" pour les systèmes de test et les systèmes de simulation. Dans le cadre des systèmes de test, cette société propose des cartes d'entrées-sorties telles que celles présentées dans [55] intégrables dans des fonds de paniers propriétaires ARION-100 [56]. Avec cette gamme de produits, Arion propose une solution basée sur un réseau Ethernet pour exécuter facilement des applications en temps-réel. De plus, ces fonds de panier pourront s'interfacer avec des environnements logiciels variés tels que Microsoft Windows, Linux et le système d'exploitation temps-réel VxWorks de WIND-RIVER.

La société SHROFF [52] propose également des solutions basées sur des fonds de panier VME, PCI, etc. dans lesquels pourront s'interfacer des cartes d'entrées-sorties du commerce.

Au point de vue cœur de calcul, plusieurs sociétés telles que ACROMAG [53] et IOxOS

[35] proposent des cartes basées sur des processeurs standards tels que l'i5 ou l'i7 développés par INTEL permettant ainsi d'actualiser ce type de support matériel en fond de panier. Néanmoins la bande passante limitée du bus VME alliée à ce type de processeur n'est pas judicieuse tant le fossé de performances est important.

Toutefois, c'est dans cette optique de standardisation qu'Eurocopter [14] avait déjà envisagé ce type de solution en embarquant le calcul sur une architecture PC standard avec un environnement temps-réel. De plus, une solution basée sur une architecture PC embarquant une carte d'extension PCIe [34] permettant la communication avec un fond de panier associé avait également été envisagée [13].

La non-généricité de ces architectures décrites ci-dessus basées sur des fonds de paniers et des cartes associées reporterait finalement le problème d'obsolescence et de coût de maintenance actuel dans le temps.

De nouvelles solutions commencent à émerger depuis quelques années, celles-ci sont basées sur l'utilisation de cœurs IP (Intellectual Property) implantés sur architectures FPGA. D'ailleurs XILINX, ALTERA, et d'autres constructeurs de composants FPGA proposent et supportent aujourd'hui des IPs d'entrées-sorties avioniques tels que le MIL-1553 [60], l'Arinc 429 [62], le CAN [64], etc.

Cette nouvelle offre laisse présager la possibilité de concevoir des architectures basées sur l'utilisation de cartes FPGA génériques permettant de résoudre la problématique actuelle de la spécialisation des cartes d'entrées-sorties encore utilisées. De plus, l'augmentation en performances des cœurs de calcul devra nécessairement être accompagnée d'une augmentation de la bande passante vers les interfaces d'entrées-sorties.

De plus, XILINX se lance également dans les solutions certifiables DO-254[1] basées sur leurs composants commercialisés [65] permettant ainsi une intégration de ce type de technologies dans les futurs systèmes embarqués dans le domaine aéronautique.

Les attraits de cette nouvelle tendance technologique nous amène à considérer le composant FPGA comme une solution clef pour proposer des solutions innovantes dans le domaine du test et de la simulation avionique. En effet, notre contribution majeure dans le cadre de cette thèse s'appuiera sur l'utilisation de la technologie reconfigurable.

1. http ://www.do254.com

2.3 Architectures hétérogènes dynamiquement reconfigurables

Sur les architectures SMP (Symetric Multi Processing), comme sur les architectures multi-cœurs, un superviseur est systématiquement utilisé afin de pouvoir gérer les affinités caractéristiques de ces systèmes (affinité : attachement d'une unité de calcul à un cœur particulier). Il permet d'équilibrer la charge sur les différents CPU en s'efforçant de respecter les contraintes temps-réel.

Dans le cadre d'architectures AMP (Asymetric Multi Processing) du type station de travail - Ethernet - CPU VME/VXI ou Compact PCI/PXI communément rencontrées dans le monde du test aéronautique le problème a dans ce cas là aussi été résolu mais sa résolution, même automatisée, est lourde à mettre en œuvre et sans aucune contrainte temps-réel.

Les solutions existantes par nature dynamique ne permettent pas de prendre en compte la puissance de calcul adaptative offerte par les composants FPGA actuels. C'est la nouveauté même du type d'architecture considérée et décrite précédemment avec des fonctionnalités de reconfiguration dynamique étendues aux FPGAs qui impose naturellement le concept de superviseur hétérogène dont un des rôles principaux consiste à gérer de façon optimale l'utilisation du FPGA en tant que nouvelle entité de calcul à prendre en compte dans un système temps-réel distribué.
La reconnaissance de ce composant en tant que candidat éligible pour l'hébergement de modèles réside dans la possibilité d'implanter des algorithmes présentant de forts degrés de parallélisation. En effet, si ces modèles présentent peu de dépendances de données intrinsèques dans leur exécution, ils deviendront éligibles à une implantation matérielle pouvant atteindre un haut niveau de performances. A titre d'exemple, dans [12], les auteurs comparent l'exécution d'un même algorithme sur plusieurs supports matériels. C'est sur ce même critère que les composants FPGAs ont démontré des temps d'exécution inférieurs à ceux obtenus sur CPU et GPU.

Dans le cadre de calculs hautes performances, les MPSoC (Multiprocessor Systems on-Chip) ont permis d'apporter un nouveau type de solutions embarqués sur puce. Ce type de solution introduit néanmoins des difficultés telles que la communication et la synchronisation entre les différents processeurs, parfois hétérogènes implantés sur puce.

Dans [15], les auteurs se sont intéressés aux architectures MPSoC et plus précisément à la communication entre les différents processeurs hétérogènes pouvant composer ce type

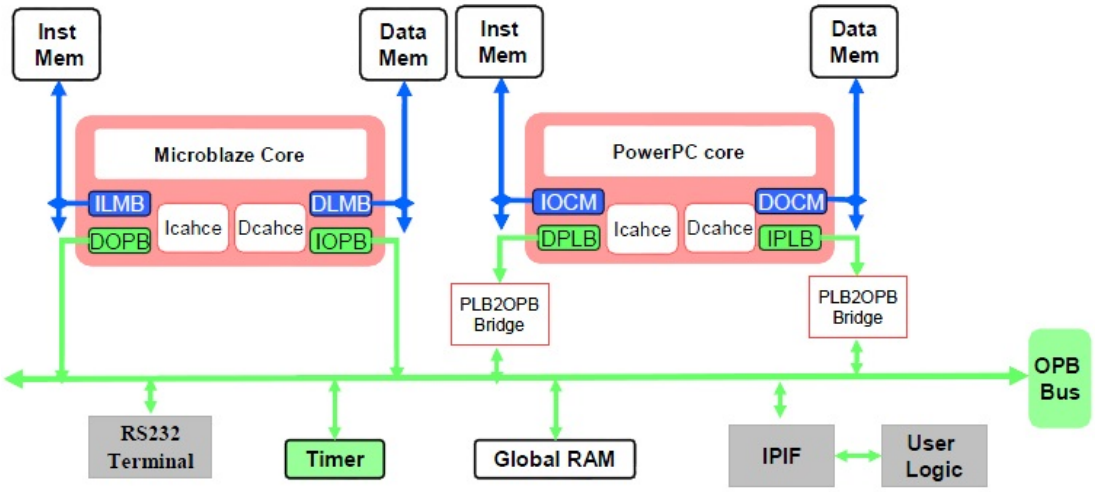

FIGURE 2.5 – Exemple d'architecture MPSOC [15]

d'architectures. Dans le cadre de leur recherche, les auteurs proposent un middleware de contrôle des communications inter-processeurs sur le type d'implémentations décrit dans la figure 2.5. Leur solution est basée sur une implémentation "matérielle" de sémaphores qui leur permet de communiquer et synchroniser des données au sein de leur architecture matérielle comme cela est montré dans la figure 2.6. Ainsi, l'architecture hétérogène globale reste cohérente par le partage et la synchronisation efficaces des données de l'application globale transitant ici.

Dans [31], les auteurs proposent un mécanisme de migration de tâches au sein d'un MPSoC hétérogène dont l'architecture est présentée sur la figure 2.7. Leur proposition permet d'attribuer à chacune des tâches composant leur application globale un processeur tout en prenant compte que chacune d'entre elles n'est bien-sûr pas exécutable sur tous les processeurs et qu'elles n'ont pas toutes la même priorité d'exécution. Pour ce faire, les auteurs ont proposé un algorithme prenant en compte la priorité de chacune des tâches dans l'application globale ainsi que le meilleur processeur éligible pour son exécution. Ainsi, chacune des tâches composant l'application peut être migrée dynamiquement d'un processeur vers un autre dans le but d'obtenir les meilleures performances possibles par rapport à une exécution standard sur un processeur unique.

Les travaux présentés ci-dessus ont été proposés dans le cadre d'applications embarquées. Néanmoins les concepts étudiés peuvent être adaptés voire judicieux dans le cadre de notre étude portant sur les architectures CPU-FPGA hautes performances dynamiquement

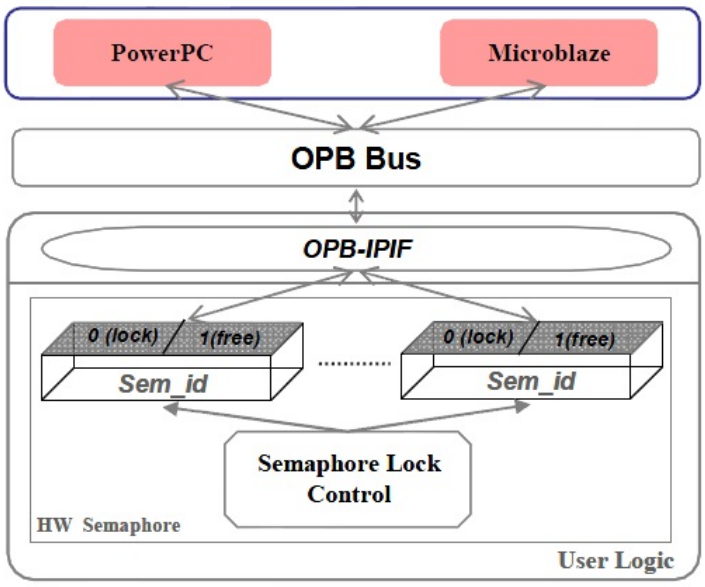

FIGURE 2.6 – Synchronisation matérielle sur une architecture hétérogènes [15]

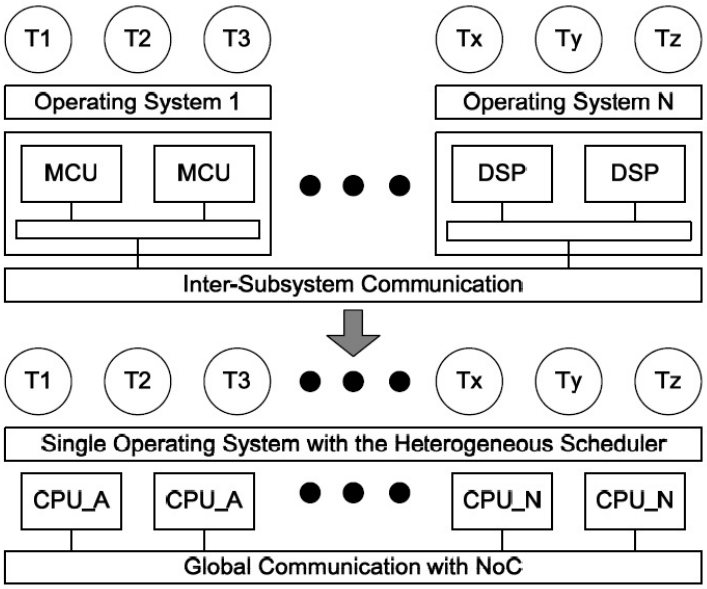

FIGURE 2.7 – Architecture MPSoC hétérogène [31]

reconfigurables.

Il existe aussi des systèmes fortement distribués de type CPU-FPGA ou plusieurs configurations sont déjà prédéfinies et figées, donc forcément prêtes à l'emploi en fonction de l'état d'un paramètre du système, respectant de facto les contraintes hautes performances et temps-réel imposées. Dans [71], les auteurs comparent l'exécution du Quasi-Monte Carlo (simulation financière) sur différentes architectures (FPGA, GPU, GPP). Les composants FPGA apportent dans ce cas un temps d'exécution inférieure d'un facteur 3 par rapport à celui sur un GPU. De plus, depuis quelques années apparaissent de nouvelles architectures composées de CPU et FPGA. La société Kontron [61] propose aujourd'hui une architecture composée d'un processeur x86 et d'un composant FPGA Altera [22] pour les équipements embarqués. La société Xilinx propose également un nouveau type d'architecture, les EPP (Extensible Processing Platform) [63] réunissant les mondes du développement logiciel et matériel permettant ainsi de tirer parti des avantages de ces derniers.

Dès lors, il semble que l'association de composants CPU pour les calculs séquentiels et FPGA pour des algorithmes présentant un haut degré de parallélisation ainsi que pour la gestion des entrées-sorties vers les équipements sous-test se montre judicieuse dans le cadre de notre projet.

Il sera donc intéressant de s'appuyer sur des architectures CPU-FPGA afin de proposer un modèle d'exécution dynamique et hétérogène logiciel-matériel dans la perspective de concevoir une nouvelle génération de bancs de test avioniques génériques et versatiles.

2.3.1 Commande de reconfiguration

La reconfiguration dynamique partielle des composants FPGA est soumises à plusieurs règles dépendant des contraintes imposées par l'application (diminution de la consommation, augmentation des performances, gestion des ressources composant disponibles). Dans cette partie, nous allons nous intéresser à la commande de ces reconfigurations.

2.3.1.1 Commande via des contrôleurs matériels internes

Une première solution consiste à commander ces reconfigurations en interne. En effet, une fois les régions dynamiquement reconfigurables ont été définies avec l'outil PlanAhead [44], l'utilisateur pourra implémenter une logique qui commandera les reconfigurations des différentes régions. Cette méthode a un grand intérêt, en effet, une commande purement matérielle présente plusieurs avantages.

Le premier est le faible coût de ce type d'implémentation en terme de ressources matérielles, en effet, dans le cadre d'applications simples où les reconfigurations vont dépendre d'équations logiques, la programmation de ces dernières se fera plus aisément via les outils disponibles.

Le second concerne la consommation de ce type d'implémentations qui sera faible en conséquence direct de la faible occupation des équations logiques sur le composant FPGA.

Enfin, le troisième avantage sera le déterminisme de cette implémentation, en effet les équations étant "câblées matériellement", le temps de passage sera identique à chaque itération permettant ainsi de respecter des contraintes temps-réel qui pourraient être imposées dans le cadre du développement d'une nouvelle application.

2.3.1.2 Commande via des contrôleurs externes

Dans le cadre de développement d'architectures s'appuyant sur des composants hétérogènes (CPU, GPU, DSP, FPGA, etc.), les commandes de reconfiguration dynamique pourront se faire depuis un composant extérieur au FPGA.

Pour ce faire, la décision se fera depuis un cœur de calcul distant du composant FPGA et sera communiqué par un médium de communication liant ces deux composants.

C'est dans ce cadre que les auteurs de [42] proposent une architecture sur laquelle le composant FPGA sera reconfiguré par une interface distante communiquant via le bus PCIe. En effet, l'acteur de commande distant envoie à travers le bus disponible la configuration souhaitée à un IP "Configuration Control" situé sur la puce qui agira directement sur la région reconfigurable sur le composant.

Le contrôle distant est également utilisé par les auteurs [11] dans le cadre de la TMR (Triple Modular Redundancy), les performances du bus de communication permettent une réactivité optimale du système global.

Ces solutions décrites ci-dessus permettent d'envisager la supervision globale d'une architecture composée de plusieurs puces FPGAs par un cœur de calcul distant et relié par un bus de communication.

2.3.1.3 Commande via des systèmes systèmes d'exploitations

Une autre solution possible est d'utiliser un système d'exploitation pour la commande des reconfigurations du composant FPGA.

Une des solutions existante est ReconOS [48] (Re-Configurable and Extensible Operative System) un environnement d'exécution proposant l'utilisation de threads logiciels et

matériels permettant au système d'utiliser au mieux les ressources logicielles et matérielles disponibles. Dans [25], les auteurs proposent plusieurs implémentations de cette solution sur différents systèmes d'exploitation (Linux, eCOS) sur différent processeurs (Power PC sur Virtex 2 et 4, MicroBlaze sur Virtex 4). Cette flexibilité laisse une grande liberté d'utilisation aux développeurs. Cette solution permettra la gestion d'exécution d'applications ainsi que la commande de reconfiguration partielle et dynamique des composants FPGAs.

Une autre solution existante est R3TOS [16] (Reliable Reconfigurable Real-Time Operating System), un ROS (Reconfigurable Operating System). Gordon Brebner [29] a été le premier a proposer ce type de système d'exploitation dans le but de coordonner l'exécution des différentes tâches matérielles afin d'utiliser au mieux les ressources disponibles. Ces ressources sont virtualisées [36] permettant à l'application utilisateur de les exploiter afin d'augmenter la puissance de calcul disponible. Ce type de service doit être fourni avec une API permettant l'adressage des ressources matérielles disponibles. C'est ce que propose l'API de R3TOS avec des appels systèmes similaires à ceux définis par la famille de standards POSIX. Dans [17], les auteurs proposent une implémentation de ce système dans un Picoblaze n'utilisant que 96 slices du composant FPGA. Une telle implémentation ne consommant que peu de ressources permet d'obtenir un système temps-réel minimaliste gérant la globalité du composant matériel.

Le projet ANR FOSFOR [26] (Flexible Operating System FOr Reconfigurable platform) a pour but de reconsidérer la structure purement logicielle du RTOS visant à le rendre flexible, distribué avec une interface homogène du point de vue de l'application. L'objectif sera de reconsidérer les trois services fondamentaux d'un système d'exploitation, l'ordonnancement, les entrées-sorties et la gestion de la mémoire disponible afin que les tâches composant l'application puissent s'exécuter sans connaissance de leur nature matérielle ou logicielle. Ce dernier point est très intéressant, en effet, un niveau d'abstraction permettant de traiter les tâches indépendamment de leur implémentation sur le support matériel global permet une gestion plus simple de l'application globale.

Ces solutions offrent un grand intérêt dans le cadre de développement d'applications embarquées sur des composants FPGAs. Dans le cadre de notre projet, nous envisageons d'utiliser le composant FPGA dans un contexte de calcul reconfigurable à haute performance afin de satisfaire les besoins requis par le domaine d'application. Notre contribution vise à commander la ré-allocation hétérogène des tâches composant l'application de test depuis une machine hôte reliée au composant matériel par un bus de communication rapide.

2.3.2 Reconfiguration Rapide

Dans le cadre du développement d'architectures dynamiquement reconfigurables, les performances de ces reconfigurations sont primordiales. En effet, dans le cadre de reconfiguration en ligne, c'est à dire que l'application continue de s'exécuter alors qu'une partie du composant FPGA se reconfigure, il est essentiel que le composant soit prêt à exécuter les nouvelles fonctionnalités contenues dans la région reconfigurée. Un coût de reconfiguration trop important pourrait mener l'application à ralentir voire échouer si cette dernière est soumise à ces échéances temporelles.

Les auteurs de [27], présentent FaRM (Fast Reconfiguration Manager) un contrôleur de reconfiguration dynamique offrant des débits de reconfiguration allant jusqu'à 800 Mega-Octets par seconde. Ce résultat a été obtenu par l'utilisation de DMA (Direct Access Memory), des modules de compression du bitstream et l'overclocking de l'ICAP (Internal Configuration Access Port) [72].

Les auteurs de [18] proposent UParC (Ultra-fast power-aware reconfiguration controller), un contrôleur de reconfiguration dynamique sur puce permettant d'atteindre une vitesse de reconfiguration allant jusqu'à 1433 Mega-Octets par seconde, ce qui est quasiment 2 fois supérieur aux vitesses jusque-lors atteintes par FARM[27]. Comme cela est décrit dans la figure 2.8, le principe réside dans une architecture basée sur un DMA et un contrôleur pipeline qui transmet les bitstream compressés à implanter dans les régions reconfigurables.

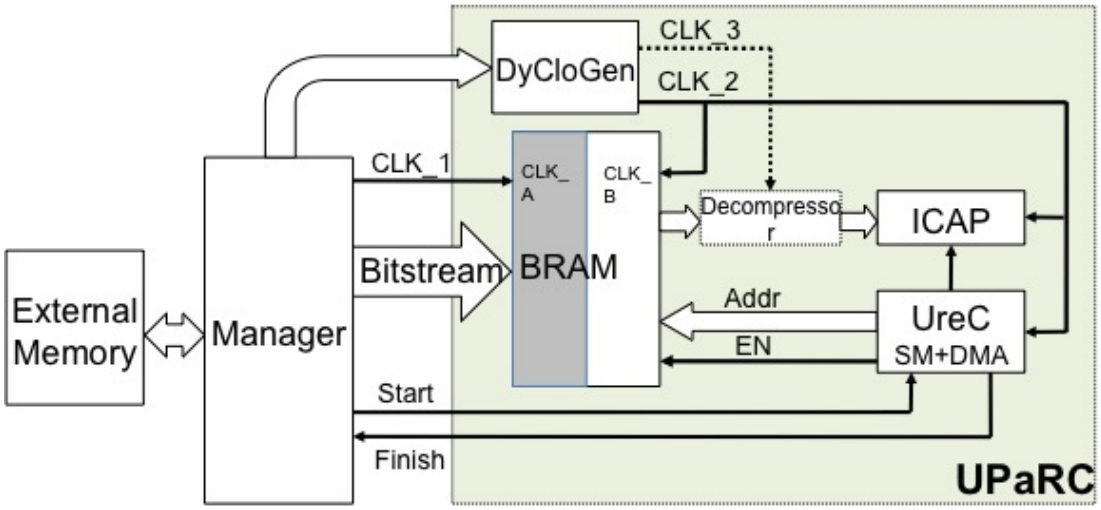

FIGURE 2.8 – structure de UPaRC [18]

Les auteurs de [57] proposent une architecture permettant d'atteindre une vitesse de re-

configuration allant jusqu'à 2200 Mega-Octets par seconde. A titre d'exemple, de telles performances permettraient de reconfigurer un processeur MicroBlaze en 100 Micro-secondes sur un Virtex-6 sur la puce. De telles performances ont pu être obtenues par une optimisation de l'architecture de base fournit par XILINX.

Dans notre cadre d'application, nous sommes soumis à des contraintes temps-réel, l'utilisation de cette fonctionnalité de reconfiguration dynamique quelque soit ses performances aura un coût temporel probablement négligeable pour de petits modèles mais devra être pris en compte dans le cadre de la supervision du support matériel global. Dans le cadre du développement d'un premier prototype nous n'envisageons pas d'utiliser immédiatement cette capacité mais plutôt d'implémenter les modèles requis par le scénario mis en place et les utiliser si cela est nécessaire pour le bon déroulement de la session de test et de simulation. Néanmoins, nous envisageons dans le cadre de nos travaux l'allocation et la migration dynamique logicielle-matérielle de tâches.

2.4 Environnements temps-réel

L'utilisation d'un environnement temps-réel dur dans notre cas d'utilisation ne se justifie pas. En effet, les architectures de test et de simulation n'étant pas critiques, elles ne requièrent qu'un environnement temps-réel mou. De plus, l'expérience passée avec l'utilisation d'un environnement temps-réel propriétaire (VxWorks) a entrainé une volonté de se tourner vers une solution Open-Source offrant des coûts de formation des développeurs inférieurs à ce qui a été pratiqué jusque lors. Le catalogue d'environnements temps-réel libres tels que FreeRTOS, Xenomai, etc. pouvant répondre à nos contraintes est important.

2.4.1 Xenomai : Une extension temps-réel Linux

Xenomai [70] est basé sur l'utilisation d'une partie d'un cœur d'un RTOS (Real Time Operating System) utile pour construire une interface temps-réel. Cela permet au développeur d'utiliser un ensemble de services d'un RTOS standard. Comme décrit dans la figure 2.9, Xenomai est un patch pour noyau Linux permettant d'obtenir des capacités temps-réel. De plus, Xenomai permet l'utilisation de routines propres à des environnements temps-réel tels que psOS, RTOS, VxWorks comme décrit dans la figure 2.9. Ainsi, il devient possible de réutiliser du code existant et développé pour un des systèmes d'exploitation cité précédemment sur ce nouvel environnement tout en bénéficiant des propriétés d'un système Linux standard. Finalement, une couche de virtualisation Adeos [37] est uti-

lisée afin d'assurer l'exécution en temps-réel des tâches souhaitées (RT tasks sur la figure) ou bien l'exécution de tâches utilisateur (User tasks sur la figure 2.9).

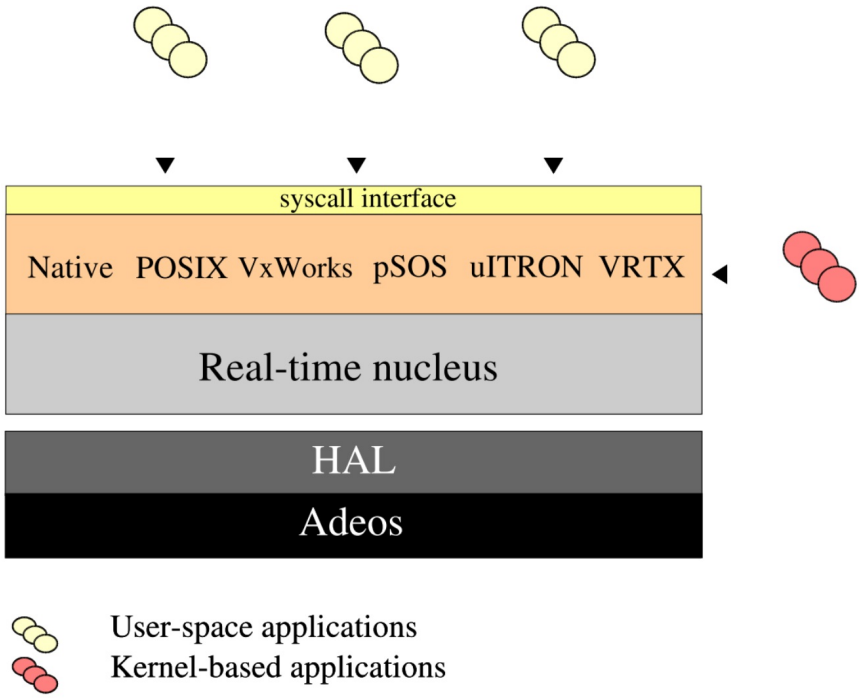

FIGURE 2.9 – Fonctionnement de Xenomai

Dans [43], les auteurs proposent l'utilisation de Xenomai dans le cadre d'exécution de tâches sur une architecture CPU/FPGA. Ici, l'environnement logiciel permet d'aboutir à des performances déterministes très intéressantes dans ce cadre d'utilisation.

Dans une première phase de nos travaux, nous avons utilisé Xenomai [6] pour satisfaire les contraintes temps-réel imposées par notre domaine d'application. Cependant, nous avons été confrontés à des limitations majeures dans le cadre d'une mise en place dans un milieu industriel et surtout dans notre cadre d'application où la stabilité du système est primordiale. Lors de nos tests, Xenomai a démontré de fortes instabilités lors des mises à jour pouvant rendre le système global instable. En effet, trois acteurs sont présents sur ce système, le noyau, le patch Xenomai ainsi que la couche ADEOS et la mise à jour de l'un d'entre eux peut mettre en péril la stabilité du système entier.

RTAI (Real Time Application Interface) [49] [41] est apparu fin des années 1990 dans le cadre d'un projet universitaire de l'école polytechnique de Milan. C'est une extension libre du noyau Linux standard qui permet d'obtenir des fonctionnalités temps-réel dures et utilise tout comme Xenomai ADEOS.

2.4.2 Linux-RT

Linux-RT [69] est une version temps-réel dure du noyau Linux. Cela est possible par l'application du patch officiel PREEMPT-RT lui apportant un comportement temps-réel dur tout en limitant le nombre de modifications à effectuer sur les codes sources.

Cette version est basée sur une version du noyau Linux dont les gestionnaires d'interruptions (entre autres) ont été modifiés afin de le rendre préemptible. De plus, une mise en place de mécanismes de protections contre le problème d'"inversion de priorités" dû à l'utilisation de sémaphores à héritage de priorité. Cette solution est d'ailleurs proposée dans le cadre agricole pour des applications de communication [9] à faibles latences.

Contrairement aux extensions concurrentes du noyau Linux tels que Xenomai ou RTAI présentés ci-dessus, il ne fait que modifier le fonctionnement du noyau standard sans ajouter un second noyau ou une couche de virtualisation temps-réel comme ADEOS, ce qui simplifie et allège le système résultant. De plus, cette solution n'impose aucune interface de programmation spécifique, utilisant l'API POSIX standard.

Dans le support logiciel que nous proposerons, nous mettrons en avant une solution basée sur l'utilisation d'un noyau Linux standard optimisé pour notre cas d'utilisation dans le but de respecter les contraintes temps-réel mou imposées. Cette solution permettra de réduire les coûts de développement et de maintenance plus élevés dans le cadre de l'utilisation des solutions présentées précédemment. En effet, un pôle d'expertise autour de ce type d'environnement Open-Source s'est créé chez Eurocopter permettant ainsi d'envisager ce type de solution à long terme. De plus, le couplage avec des composants FPGAs embarquant des accélérateurs matériels permettra de respecter les contraintes temps-réel imposées. D'ailleurs, dans le cadre du développement d'applications tel que la simulation temps-réel de réseaux neuronaux [38], les composants FPGA offrent les capacités nécessaires pour répondre aux contraintes temps-réel et de performances imposées par le domaine d'application.

2.4.3 Ordonnancement et placement de tâches

Dans la première partie de nos travaux [5], nous avons exploré une méthodologie de développement à haut niveau qui nous aurait permis de placer les tâches composant notre application de test et de simulation sur les différentes unités de traitement disponibles. Cette solution a vite démontré ses limites tant les contraintes propres au domaine d'application ne nous permettaient pas de figer le placement des tâches pour une session donnée.

C'est pour cela que nous nous sommes orientés vers des algorithmes de placement de tâches

En effet, l'utilisation d'un système temps-réel n'est pas suffisant dans le cadre du développement des futurs systèmes de test et de simulation. En effet, le support matériel proposé étant composé de nœuds de calcul hétérogènes, la supervision de l'ensemble de l'architecture devra se faire par un composant spécifique : Le superviseur.

L'allocation dynamique de tâches indépendantes et dépendantes sur des systèmes informatiques hétérogènes se fait par le biais d'heuristiques de placement.

Il faut alors distinguer deux parties, le placement qui correspond à l'attribution des tâches sur les ressources de calcul disponibles et l'ordonnancement de ces dernières qui correspond à leur ordre d'exécution dans l'application globale. D'ailleurs, dans la littérature, la majorité des travaux se réfèrent à l'ordonnancement des tâches alors qu'en réalité, leur investigation concerne leur allocation.

L'algorithme LPT (Longuest processing Time) est l'une des plus anciennes méthodes de placement. Cet algorithme trie les tâches selon un ordre décroissant de leur temps de traitement. Cette méthode permet de garder les tâches les plus courtes pour la fin du placement dans le but d'équilibrer la charges sur les différentes machines parallèles disponibles. Dès 1969, Graham [47], s'est basé sur cette méthode dans le cadre de la minimisation des temps d'exécution global d'une séquence de tâches. Cette heuristique est encore aujourd'hui très répandue.

En effet, en 2008, Koulamas and Kyparisis [20] ont démontré la robustesse de cette approche en l'utilisant pour le placement de tâches sur des machines parallèles identiques.

En 1977, Oscar H. et al [40] ont présenté des heuristique d'ordonnancement de tâches indépendantes sur des processeurs hétérogènes. En 1988, Thomas L. Casavant et al [] ont présenté une taxonomie de méthodes pour l'ordonnancement de tâches au sein d'architectures distribuées. A haut niveau, ils distinguent l'ordonnancement local et l'ordonnancement global. A un second niveau, ils distinguent deux catégories de placement, statique et dynamique distinguant ainsi les phases d'initialisation du système/application et sa phase d'exécution.

Des travaux récents ont permis la proposition des heuristiques Greedy [51] permettant l'allocation dynamique de tâches dans des systèmes hétérogènes temps-réel. Cet algorithme n'aura pas pour but de trouver la solution optimale globale dans le cas d'un système distribué mais cherchera à obtenir les solutions optimales locales dans un temps minimum.

Nos travaux diffèrent de ceux cités précédemment car notre intérêt se porte sur le placement de tâches sur des architectures hétérogènes CPU-FPGA. De plus, nous ne concentre-

rons pas uniquement sur l'allocation statique des tâches mais également leur ré-allocation dynamique dans le but de respecter les contraintes temps-réel imposée par le domaine d'application.

Bien-sûr, le placement statique doit être optimal afin d'éviter des re-placements dynamiques lors des phases de calcul. En effet, des re-placements de tâches en cours de calcul peuvent introduire des latences causées par le temps de calcul de l'algorithme lui même ainsi que par le re-placement lui même (sauvegarde du contexte, migration des données, etc.). Ce coût peut s'avérer non négligeable dans le cas de calculs haute performance, à contraintes temps-réel par exemple voire supérieur au gain obtenu par le re-placement de la tâche.

2.5 Positionnement

Comme nous l'avons décrit précédemment et comme nous pouvons l'observer dans la figure 2.10, le positionnement de nos travaux de recherche se situe à l'intersection de trois domaines, les systèmes de test et de simulation, les architectures hétérogènes dynamiquement reconfigurables et les environnements temps-réel.

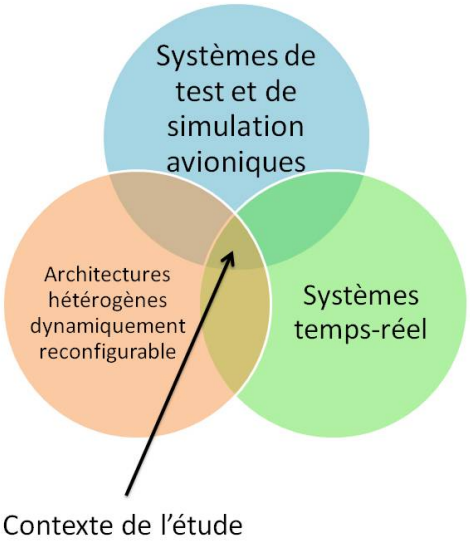

FIGURE 2.10 – Positionnement de nos travaux de recherche

Tout comme les contraintes et les exigences requises, nos propositions scientifiques et techniques seront guidés par le domaine d'application.

Tout d'abord, nous proposerons un modèle d'exécution pour la prochaine génération de systèmes de test et de simulation. Ce modèle d'exécution devra répondre aux contraintes de généricité, de performances, d'adaptabilité et temporelles liées au domaine d'application. Dans la description de ce modèle d'exécution, il sera nécessaire de définir plusieurs fonctions de surveillance, de supervision et de contrôle du support matériel déployé pour la prochaine génération de systèmes de test et de simulation avionique.

Ensuite, notre choix se portera sur un support matériel hétérogène basé sur des nœuds de calcul multi-cœur CPU et FPGA. En effet, alors que les cœurs CPU vont exécuter toute la partie logicielle propre au test et à la simulation ainsi que les modèles composant le scénario, les composants FPGA vont s'interfacer avec les équipements sous-test et exécuter les fonctions éligibles à une implémentation matérielle. Dans un même temps, il sera nécessaire d'apporter une méthodologie de développement afin d'accélérer le cycle de validation standard permettant une prise en charge du code "legacy" propre au domaine d'application. Dès lors, nous utiliserons les propriétés et capacités du support matériel afin d'apporter les capacités de reconfiguration dynamique aux systèmes de test et de simulation.

Dans un troisième temps, nous allons proposer un support d'exécution logiciel basé sur un environnement Linux standard permettant d'apporter le respect des contraintes temps-réel molles nécessaires au domaine d'application. Ce support logiciel se verra enrichi de plusieurs fonctions élémentaires décrites dans le modèle d'exécution qui permettront la surveillance, la supervision et le contrôle du support matériel afin de répondre aux contraintes liées au domaine d'application.

2.6 Conclusion

Dans ce chapitre, nous avons effectué un tout d'horizon de l'état de l'art actuel. Les trois axes de notre recherche bibliographique nous ont permis d'évaluer les solutions existantes et les tendances scientifiques et technologiques envisagées dans les années à venir. En se basant sur les résultats de nos recherches, nous proposerons un modèle d'exécution dynamique associé à un support matériel et logiciel pour la prochaine génération de systèmes de test et de simulation avioniques.

Dans le prochain chapitre, nous allons décrire le modèle d'exécution dynamique proposé s'appuyant sur un support matériel hétérogène dédié à notre domaine d'application.

Chapitre 3

Un modèle d'exécution dynamique pour les architectures hétérogènes dédié au test et à la simulation

3.1 introduction

Ce chapitre présente notre innovation dans le domaine des systèmes de test et de simulation dédié à l'avionique afin de pallier les limites décrites dans le chapitre précédent. En effet, nous proposons un modèle d'exécution dédié à la nouvelle génération de systèmes de test et de simulation. Cette innovation a fait l'objet d'un brevet [2] déposé à l'Institut National de la Propriété Intellectuelle (INPI). Ce modèle d'exécution répondra aux exigences des nouvelles fonctionnalités requises par Eurocopter dans le cycle de développement des futurs hélicoptères comme nous l'avons décrit dans [1]. Notre modèle d'exécution reposera sur des solutions standard répandues dans le monde industriel pour satisfaire les critères de performance, de généricité, d'adaptabilité, de dynamicité, d'ouverture et de réduction des coûts de développement. Le modèle d'exécution proposé s'appuiera sur une architecture matérielle hétérogène CPU-FPGA pouvant offrir des capacités de reconfiguration dynamique et de flexibilité. Ce chapitre sera organisé comme suit : Dans la section 3.2, nous décrirons les besoins et les contraintes industriels liés à ce projet de thèse. Ensuite, dans la section 3.3, nous proposerons nos contributions dans le cadre du développement d'une nouvelle génération des systèmes de test ainsi que leur apport sur le développement des nouveaux équipements embarqués. Dans la section 3.4, nous décrirons le modèle d'exécution dynamique lié aux futurs systèmes de test et de simulation.

3.2 Besoins et contraintes industriels

Dans le but de donner au lecteur une vision globale des besoins et des contraintes industriels de ce projet de thèse, nous effectuerons dans cette section un énumération de ces derniers.

3.2.1 Contraintes temps-réel

Dans un un premier temps, l'application requiert le respect de contraintes temps-réel. En effet, les systèmes de test et de simulation devront interagir avec les équipements à vérifier qui, répondent eux à des contraintes temps-réel dures requises par le domaine aéronautique. Ces contraintes sont pour les plus courtes de 10 milli-secondes.
Comme cela a été décrit précédemment, la solution actuellement déployée est basée sur un environnement temps-réel propriétaire et est de ce fait coûteuse tant par l'achat de licences que par le besoin d'un personnel spécialisé pour son développement et sa maintenance. Pour la prochaine génération de systèmes de test, Eurocopter requiert le déploiement d'une solution générique suffisamment performante pour répondre aux contraintes temporelles imposées par le domaine d'application. A l'inverse des contraintes temps-réel dures (ou strictes) imposées aux équipements embarqués qui ne tolèrent aucun dépassement car ils peuvent conduire à des situations critiques voire catastrophiques. Les systèmes de test ne requièrent que le respect de contraintes temps-réel molles (ou souple).

3.2.2 Calcul haute performance

Ensuite, le besoin se situe au niveau des performances de ces systèmes de test. En effet, l'augmentation des performances des équipements actuels ainsi que de leur nombre requiert une augmentation significative de la puissance de calcul du système de test proposé par rapport aux solutions actuelles. En effet, plusieurs modèles dors et déjà déployés ont besoin d'une grande puissance de calcul. Par exemple, l'exécution des modèles de mécanique de vol, de commande de vol, de motorisation et cinématique mécanique dans la même boucle est de six millisecondes pour une période de dix millisecondes et ce, sur un processeur cadencé à 2,1 GHZ.
Depuis quelques années, Eurocopter souhaite augmenter la complexité de ces modèles dans le but d'améliorer la précision et la pertinence des résultats des sessions de test. En effet, à titre, d'exemple, les prochaines versions de mécanique de vol seront enrichies en calculs aérodynamiques plus complexes comprenant le comportement individuel de chacune des

pales de l'appareil. En plus d'une volonté d'enrichissement des modèles de test avioniques, les futurs systèmes de tests devront être suffisamment performants dans le but de pouvoir implémenter de nouvelles fonctionnalités envisagées pour les futurs systèmes de test.

3.2.3 Déploiement aisé de la solution

Le déploiement d'une nouvelle solution devra, autant que faire ce peu, être quasi-transparent pour Eurocopter, ne nécessitant pas la mise en place de systèmes coûteux à mettre en place et à maintenir. En effet, les équipements de calcul haute performance, de bureautique, de test et de simulation sont aujourd'hui distincts. Eurocopter souhaiterait pouvoir mutualiser les différents moyens matériels dans une politique de diminution de coût. Cette contrainte est appuyée par le fait que les architectures PC actuelles basées sur des processeurs multi-cœur offrent des performances très intéressantes.
En plus de la mutualisation des moyens matériels, la solution proposée devra, si cela est possible, être basée sur un des deux environnements logiciels dors et déjà déployés, Microsoft Windows ou Debian. La mutualisation des moyens logiciels, qui, a déjà commencé chez Eurocopter a pour but de faciliter le développement des nouvelles applications ainsi que leur intégration sur les postes du bureau d'études.

3.2.4 Prise en charge du code "legacy"

Un déploiement aisé, en plus de l'architecture et de l'environnement logiciel utilisés passera également par une réutilisation efficace des modèles de test existants. En effet, dans le cadre du développement des systèmes de test actuels les différents départements d'Eurocopter livrent les modèles de tests nécessaires pour les phases de validation de leurs équipements concernés. Ces modèles sont généralement développés en C/C++.
Dans le but de tirer parti du support matériel envisagé, un profilage de ces modèles doit être effectué afin d'optimiser ces derniers lorsque cela est possible. Ce profilage devra retourner pour chaque modèle plusieurs informations. Ces informations pourront porter sur le degré de parallélisation, c'est à dire la possibilité de répartir l'exécution du modèle sur différents nœuds de calcul afin de diminuer leur temps d'exécution lorsque cela est possible. Le profilage d'architecture sera également nécessaire, en plus d'une estimation des performances de calcul, l'analyse des performances des moyens de communication utilisés est quasi nécessaire. En effet, dans le cas de la parallélisation d'un modèle, la communication est primordiale permettant à cette parallélisation d'être judicieuse. Les optimisations

effectuées devront donc tenir compte des propriétés et des capacités du support matériel proposé.

3.3 Vers une nouvelle génération de systèmes de test et de simulation

Dans cette nouvelle section, nous allons présenter notre réflexion concernant les capacités des futurs systèmes de test et de simulation.

3.3.1 Vers une mutualisation des moyens de test et de simulation

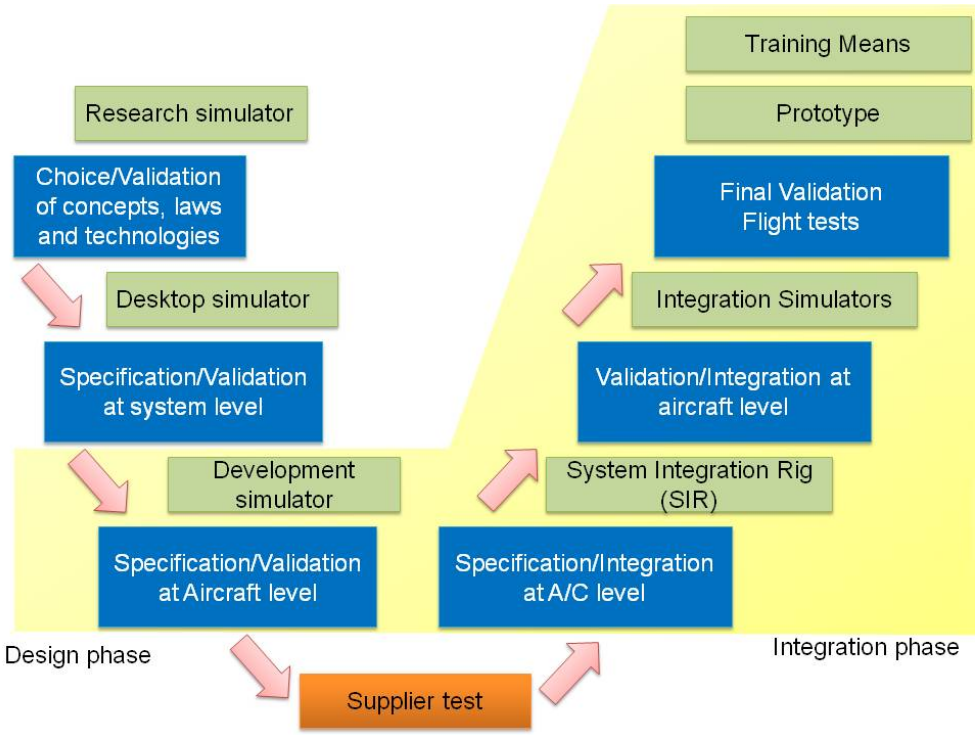

FIGURE 3.1 – Influence sur le cycle de développement d'un nouvel équipement

Dans le but d'accélérer le cycle de développement des nouveaux équipements ainsi que des hélicoptères, nous avons proposé dans le cadre de cette thèse de mutualiser les moyens de test et de simulation. Dans un premier temps, cette mutualisation a un fort impact financier. En effet, le coût d'une heure de test en vol est le même que celui de quarante

heures de simulation au sol. De plus, une mutualisation des moyens de test et de simulation permettra une accélération du cycle en V de développement des nouveaux équipements. En effet, cela impactera directement le cycle de développement d'un nouvel équipement (partie jaune de la figure 3.1). Nous avons présenté les atouts de cette solution dans notre article [4].

L'apport de ce nouveau moyen d'intégration visera à ajouter un test d'ingénierie basé au sol, c'est à dire entre les bancs d'essais des systèmes classiques d'intégration et le programme d'essais en vol tel que présenté dans le cycle de développement. Cette nouvelle solution permettra d'effectuer des tests et des simulations au niveau du système hélicoptère, anticipant ainsi la détection des problèmes de conception et de transfert de capacité d'essais de vol au niveau du sol. Les principaux systèmes de l'hélicoptère pourront ainsi être reproduit en deux blocs de construction, c'est à dire un poste de pilotage et la partie arrière du véhicule qui pourraient être couplées ou découplées en fonction de la stratégie de test et/ou de simulation. Les pilotes d'essai pourront donc intervenir avant les premiers essais en vol, permettant ainsi une meilleure cohérence dans la validation des nouveaux équipements embarqués.

3.3.2 Vers une nouvelle génération de systèmes de test et de simulation génériques et adaptatifs

Comme nous l'avons décrit précédemment, chaque équipement possède aujourd'hui son banc de test associé. La nouvelle génération de systèmes de test et de simulation devra donc être à la fois générique et adaptatif. Dans notre article [7], nous avons présenté une solution pouvant répondre aux contraintes imposées.

Pour ce faire, la mise en place d'une session de test et de simulation doit suivre plusieurs étapes, la figure 3.2 décrit la capacité d'adaptabilité de l'architecture. En effet, dans le but de pouvoir tester plusieurs équipements séparément ou bien lors d'une même session, il est nécessaire que le système de test et de simulation suive plusieurs étapes :

- Lors de la sélection de l'appareil concerné par la session de test et de simulation, une sélection sur les librairies associées à l'hélicoptère concerné est effectuée. Cela permettra aux modèles de test de s'adapter aux contraintes "appareil" (e.g type, poids, taille, etc).
- Lors de l'identification de/des équipement(s) à tester, la sélection des entrées-sorties nécessaires pour la session est effectuée.
- Les contraintes telles que le temps, l'environnement géographiques, le climat, etc.

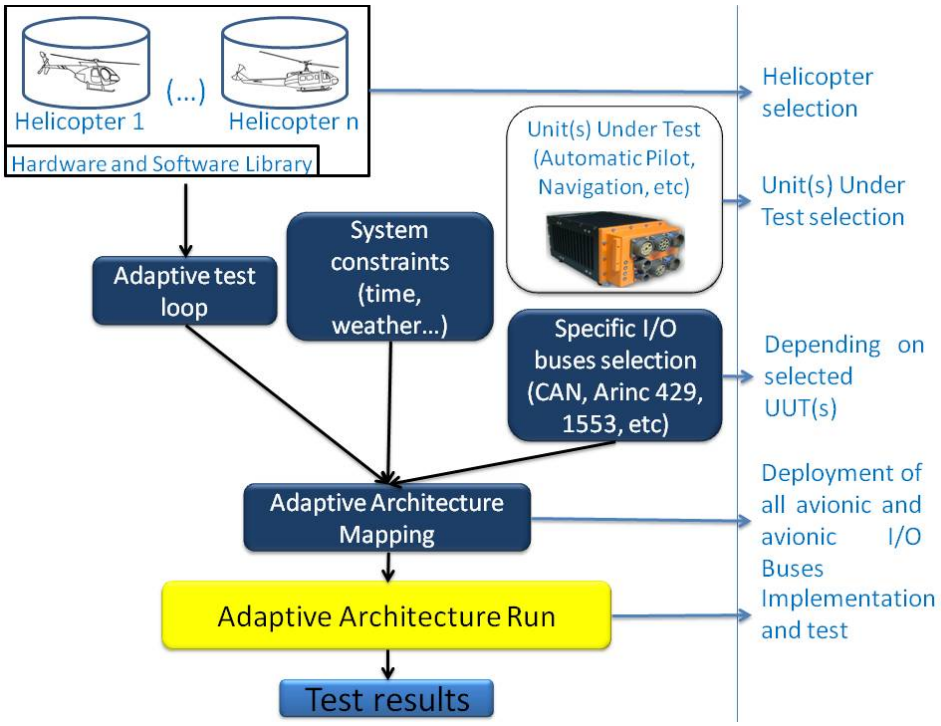

FIGURE 3.2 – Une nouvelle génération de systèmes de test adaptatifs

sont sélectionnées par l'utilisateur.

- Une fois que toutes les données liées requises sont collectées lors des étapes précédentes, le déploiement des entrées-sorties nécessaires pour le lancement de la session de test est effectué.
- Finalement, l'implémentation de tous les modèles ainsi que la liaison entre tous ces derniers est effectuée pour un lancement de la session de test et de simulation.

Ces étapes concernent la phase d'initialisation (ou configuration hors ligne) des systèmes de test et de simulation. Une fois la session lancée, le système devra respecter des contraintes temps-réel molles. Pour ce faire, nous proposons dans la prochaine section une description du modèle d'exécution proposé pour la future génération de systèmes de test et de simulation avionique.

En plus de la mutualisation des moyens de test et de simulation, nous proposons également un système entièrement collaboratif de ces moyens permettant un meilleur partage des ressources de test et de simulation mises en place.

3.3.3 Une nouvelle génération de moyens de test et de simulation collaboratifs

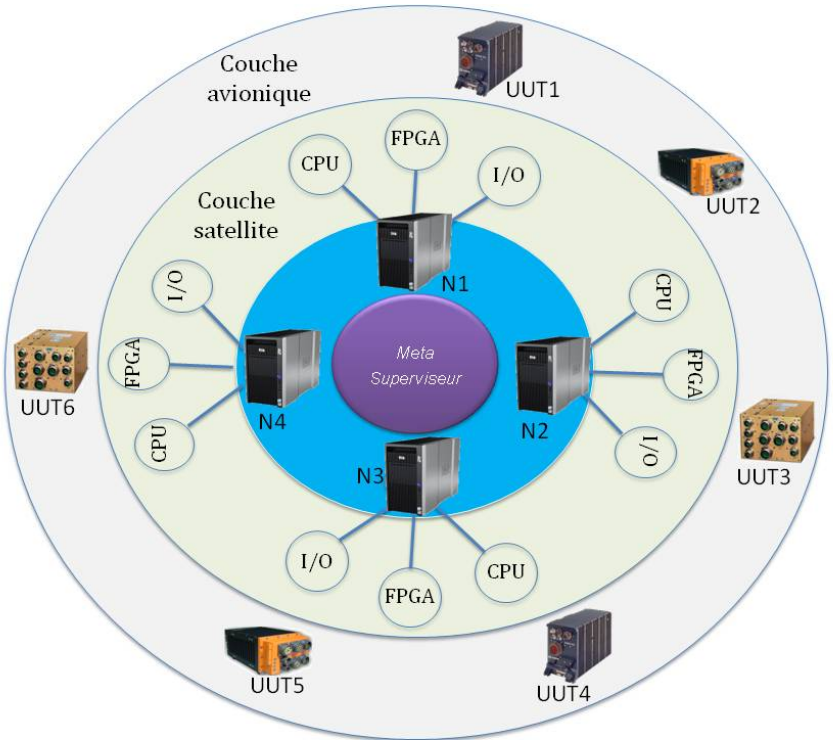

FIGURE 3.3 – Une nouvelle génération de systèmes de test et de simulation collaboratifs

La figure 3.3 illustre le dispositif versatile et générique de système de test et de simulation proposé. Les différents composants de ce dispositif sont répartis sur deux niveaux hiérarchiques.
Le premier niveau est représenté par le méta-superviseur qui permettra un contrôle à haut niveau des moyens de test et de simulation déployés au sein de l'entreprise Eurocopter. Cet élément d'abstraction des moyens de test et de simulation disponibles assurera la configuration et la supervision de l'ensemble du système. Cette supervision sera uniquement fonctionnelle permettant aux utilisateurs distants, présents sur site, ou non, de vérifier le bon fonctionnement du système utilisé.
Pendant la phase d'initialisation de ces moyens de test et de simulation, il assurera la configuration des nœuds (N sur la figure) de calcul et de communication dépendant des équipements sous-test (UUT sur la figure) concernés et des choix de l'utilisateur comme cela a été décrit précédemment. Pour ce faire, ce méta-superviseur offrira un accès complet

à une bibliothèque contenant tous les modèles de test et de simulation deployables sur l'ensemble des nœuds de calcul.

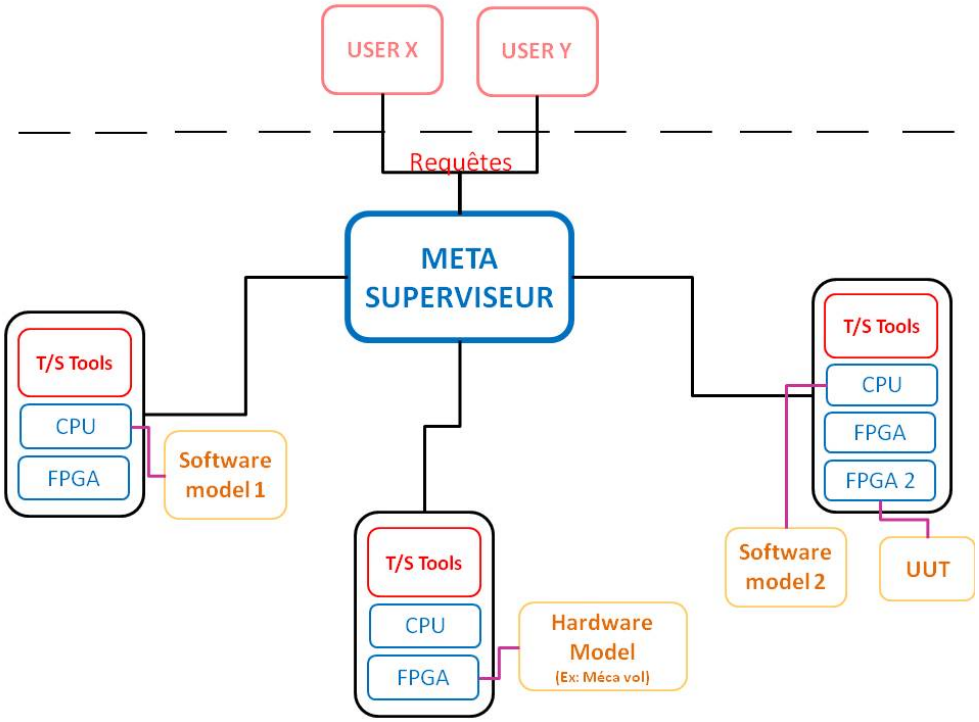

FIGURE 3.4 – Collecte de données test et simulation à distance

Pendant les différents projets de test et de simulation lancés par le méta-superviseur, ce dernier pourra collecter des données liées à ces différents projets de test et de simulation. En effet, comme cela est décrit dans la figure 3.4, chaque utilisateur pourra via des requêtes réseau collecter des données provenant d'une ou plusieurs sessions test et de simulation. Cette collecte de données permettra à ces utilisateurs de s'assurer du bon déroulement des sessions ainsi que du bon fonctionnement des nœuds de calcul et se fera bien sûr hors de la boucle temps-réel afin de ne pas perturber les sessions en cours.
Le second niveau est représenté par les nœuds de calcul et communication où chaque nœud va assurer la communication avec les équipements sous-test ainsi que les calculs nécessaires aux projets de test et de simulation associés.
Les nœuds de calculs permettront une exécution séquentielle et/ou parallèle des modèles de test et de simulation. La configuration de ces nœuds pourra être assurée hors ligne ou en ligne lors du lancement d'un nouveau projet de test et de simulation ou via une

reconfiguration dynamique lors d'un débordement dans la boucle temps-réel de test et de simulation.
Un nombre d'interfaces génériques d'entrée-sortie (différents bus industriels tels que AFDX, 1553, SPI, CAN, etc.) permettront de remplacer les cartes d'entrée-sortie spécialisées dans les architectures précédentes et par conséquence offrant une mutualisation du matériel de test. La configuration de cette interface générique peut être assurée en ligne ou hors ligne lors du lancement d'un nouveau projet de test et de simulation ou de l'intégration d'un nouvel équipement dans la boucle.

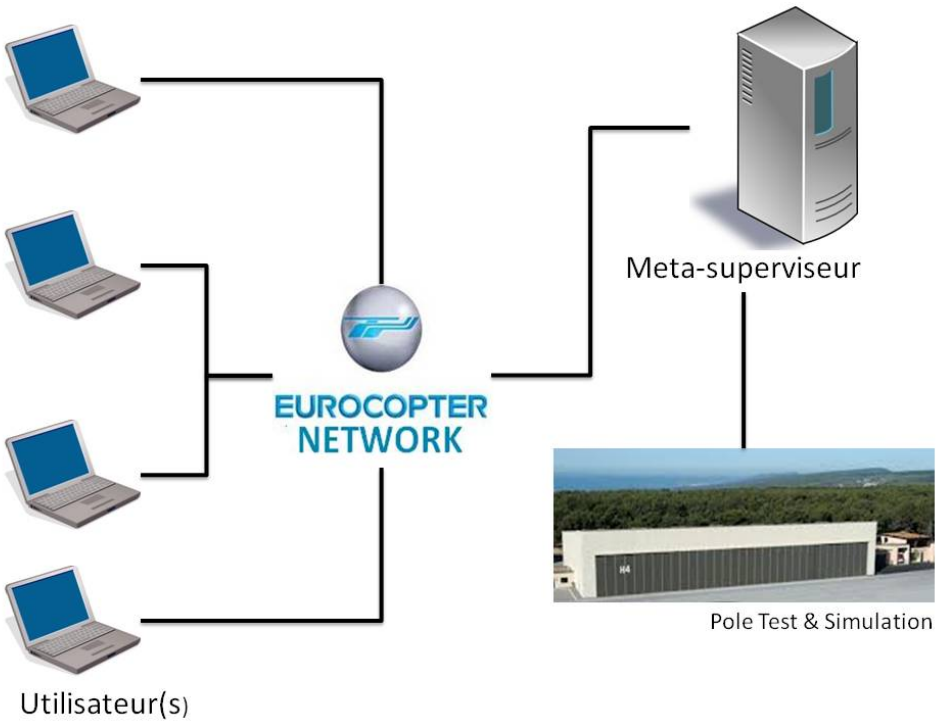

FIGURE 3.5 – Accès aux moyens de test et de simulation à distance

Le déploiement d'une telle solution permettra aux utilisateurs d'accéder à l'ensemble des systèmes de test et de simulation à l'intérieur du site et à l'extérieur par les opérateurs ayant accès au réseau Eurocopter. En effet, comme le décrit la figure 3.5, le pôle test et simulation sera visible depuis l'ensemble du réseau. Cette nouvelle fonctionnalité renforcée par la généricité des systèmes de test et de simulation apportera une grande flexibilité dans l'utilisation des ressources matérielles jusqu'à lors très limitée. En effet, le coût, l'encombrement et la non-généricité des systèmes actuels contraint les utilisateurs à mettre en

place une planification de l'accès aux ressources augmentant ainsi le temps des phases de validation des nouveaux équipements.

3.4 Un modèle d'exécution pour les futurs systèmes de test et de simulation

Dans cette section, nous allons dans un premier temps décrire brièvement le support matériel des futurs systèmes de test et de simulation pour ensuite décrire son modèle d'exécution répondant à toutes les contraintes et les besoins de l'application.

3.4.1 Description de l'architecture proposée

Afin de répondre à toutes les limites des systèmes de test actuels ainsi que les contraintes imposées, le couplage de composants FPGA reconfigurables avec les architectures multi-cœurs présentes dans les PC standard est notre proposition pour les systèmes de test et de simulation et ce, pour plusieurs raisons :

- Les architectures FPGA offrent des capacités de **calcul haute performance**. Malgré leur faible fréquence en comparaison à celle atteinte par les processeurs multi-cœur et GPUs actuels, les FPGAs peuvent offrir des performances supérieures dans certaines applications. En effet, du fait de leur structure composée de portes logiques indépendantes les unes des autres, les applications avec un haut degré de parallélisation sont éligibles pour ce type d'implémentation matérielle tel que le traitement d'image. En plus de cela, l'implémentation matérielle garantit un temps d'exécution régulier répondant ainsi aux contraintes **temps-réel** requises par l'application.
- Comme évoqué dans l'état de l'art, la disponibilité actuelle des IP Cores permet d'apporter la **généricité** à notre nouveau système. En effet, cela devient possible en remplaçant les cartes d'entrées-sorties basées sur des technologies spécifiques par des cartes génériques offrant la possibilité d'implémenter les IP Cores de communication selon la configuration souhaitée. A cela s'ajoute alors une meilleure gestion du stock de cartes d'entrées sorties, effectivement, les cartes pouvant implémenter ces IP Cores étant génériques, elles pourront assurer différentes fonctions d'interface et intervenir dans plusieurs sessions de test et de simulation distinctes les unes des autres.
- De plus, le système devient **évolutif**, les interfaces d'entrée-sortie n'étant plus assurées par des cartes physiques, la gestion de l'obsolescence en devient simplifiée. En

effet, le système proposé n'étant plus dépendant d'un type de cartes en particulier ni d'un fournisseur, Eurocopter pourra faire évoluer son catalogue d'IP Cores et ainsi apporter de nouvelles fonctionnalités matérielles à leurs systèmes de test et de simulation. Ces cartes génériques, permettront également l'exécution de modèles devenant en plus d'une interface vers les équipements, un support d'exécution supplémentaire dans les futurs systèmes de test et de simulation.

- les systèmes de test actuels ne permettent pas d'adapter l'environnement de tests lors du passage d'un modèle spécifique (ex. pilote réel vers pilote automatique) d'une gamme d'hélicoptère au même modèle d'une autre gamme. Par ailleurs, l'assemblage de composants pour démarrer une nouvelle session de test se fait hors-ligne. L'architecture du système proposé lui permettra d'être **adaptatif**, en effet, les modèles et les interfaces spécifiques pourront être déployés selon les besoins spécifiques de la session de test et de simulation.
- Le système proposé aura la capacité de replacer les tâches logicielles et/ou matérielles en cours de session. Cette capacité de reconfiguration dynamique homogène et hétérogène permettra au système entier de se reconfigurer en ligne (lors d'une session de test) implémentant ainsi une allocation dynamique des modèles ou des interfaces d'entrée-sortie. Ces reconfigurations pourront intervenir dans le but de respecter les contraintes temps-réel imposées par le domaine d'application. Pour ce faire, un environnement de supervision devra être mis en place dans le but de surveiller le comportement de système et anticiper de potentiels dépassements temporels pouvant faire échouer la session de test. Ensuite, ces reconfigurations pourront être prévues lors de la mise en place de la session de test et de simulation par l'utilisateur.

3.4.2 Description du modèle d'exécution

En plus de leur généricité, les nouveaux systèmes de test et de simulation devront être adaptatifs et dynamiquement reconfigurables. En effet, les systèmes de test et de simulation devront, dans le but de respecter les contraintes temps-réel requises, être capables de ré-allouer les modèles propres aux sessions en cours.
Le principe de la solution décrite ci-dessous consiste à équilibrer au travers d'un modèle d'exécution notre architecture hétérogène CPU-FPGA communiquant via un bus rapide. La solution consiste en la réalisation d'un superviseur du système global qui fédèrera à la fois les fonctions d'analyse des performances de l'application courante et, suite aux résultats de cette dernière, les fonctions de redistribution quasi-dynamique des modèles pertinents

à déplacer sur le système distribué. La distribution dynamique du traitement inhérent au système de test, réalisée dynamiquement par le superviseur sur l'architecture hétérogène CPU-FPGA, est une innovation notable dans le domaine des systèmes de tests d'intégration avioniques temps-réel. Dans ce type de systèmes actuels les modèles ne sont ni exécutés de façon monotone dans des composants matériels ni reconfigurables dynamiquement entre des processeurs et des composants matériels de type FPGA. Le modèle d'exécution proposé dans cette section consiste ainsi en une évolution notable pour les systèmes de test avioniques temps-réel.

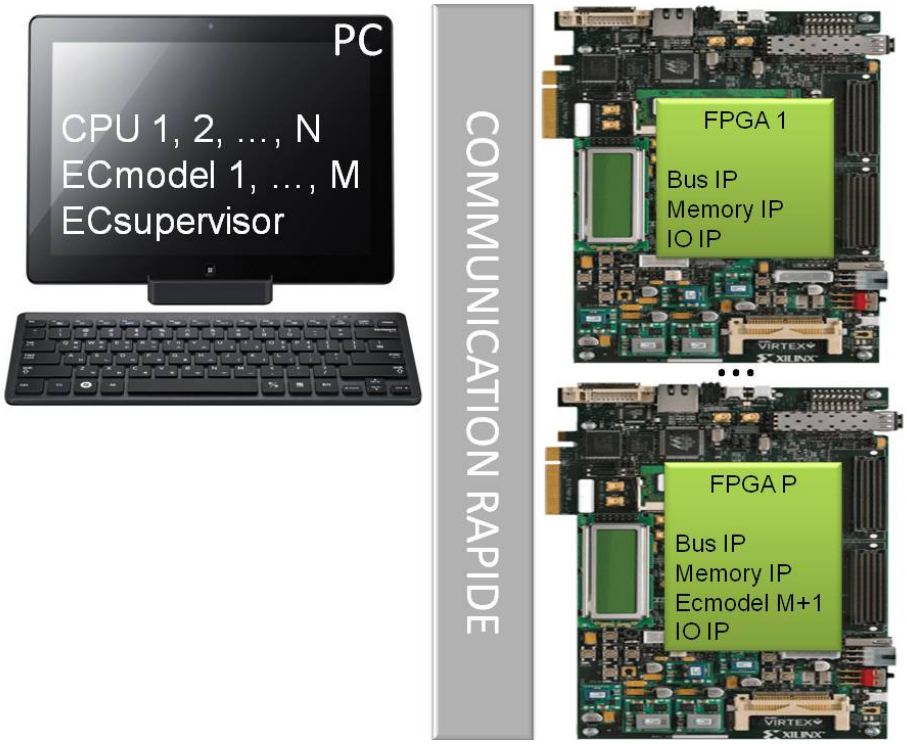

FIGURE 3.6 – Exemple d'implantation des modèles

La figure 3.6 représente l'état du système et la répartition des différents composants (Modèles et IPs) sur l'architecture du système avant intervention du superviseur, à savoir :

- Sur le PC : N CPUs ou N cœurs de processeurs exécutant M modèles avioniques (ECmodel sur la figure) ainsi que le superviseur élément indispensable à la mise en place de cette architecture dynamiquement reconfigurable.
- Sur le FPGA 1 : Les IPs mémoire et bus qui sont les 2 composants fondamentaux nécessaires à la communication entre les différents éléments de l'architecture ainsi qu'un ou plusieurs IPs d'entrées-sorties adressant le monde des entrées-sorties vers

les équipements sous-test.
- Sur le FPGA P : Les mêmes éléments élémentaires que sur le FPGA 1 ainsi qu'un modèle de test numéroté M+1 sur la figure.

Lorsqu'un modèle dépasse le temps d'exécution qui lui a été attribué, une ré-allocation dynamique est effectuée afin de relancer la phase de test. Pour ce faire plusieurs étapes sont nécessaires :

1. Le superviseur collecte en permanence les paramètres pertinents sur l'état général du système : par exemple le temps d'exécution de tous les modèles de test avionique.
2. Sur détection de débordement de ECmodel 2, le superviseur doit être capable d'analyser rapidement la place disponible sur tous les composants disponibles, d'identifier le code pré-synthétisé du modèle ECmodel 2 pour vérifier sa possible implémentation sur le FPGA 2 si cela est la solution la plus efficace.
3. Ensuite, le superviseur lance alors la reconfiguration de FPGA 2 incluant maintenant l'IP ECmodel 2 dans la boucle de test et prépare le nouveau flux de données résultant de la migration de ECmodel 2 dans le FPGA 2, tout cela sans arrêter la configuration en cours d'exécution (même dégradé par la suspension du modèle fautif).
4. Quand la nouvelle répartition est faite et que la nouvelle distribution peut être exécutée, le superviseur peut alors arrêter l'ancienne configuration et lancer la nouvelle.

Le superviseur centralisé hébergé sur un seul CPU est particulièrement adapté à un système contenant un nombre limité de FPGAs et CPUs. Au delà d'un certain nombre, il sera difficile de superviser des variables en temps-réel au vue des délais de communication entre le superviseur et les autres composants. Par ailleurs, le temps de la prise de décision augmente en fonction du nombre de nœuds (CPU et FPGA) ce qui risque de pénaliser fortement le temps d'exécution final des modèles. De plus, l'utilisation d'un superviseur centralisé rend le système non tolérant aux pannes. En effet, un dysfonctionnement du processus de supervision ou une anomalie sur la machine hôte correspondante engendre l'arrêt du système entier. Dans ce cas, il convient de concevoir un environnement de supervision distribué. De ce fait, la prise de décision sera répartie sur les différents CPUs. Une telle architecture permettra ainsi un meilleur équilibrage de la charge "load-balancing" induite par l'exécution de l'algorithme de supervision. Les échanges d'informations relatives aux débordements ou débordements anticipés ainsi que les ressources matérielles disponibles sur chaque nœud, se basent sur des protocoles de communication. Cette proposition est en cours d'étude pour le dépôt d'un brevet [3].

3.5 Vision conceptuelle du modèle d'exécution

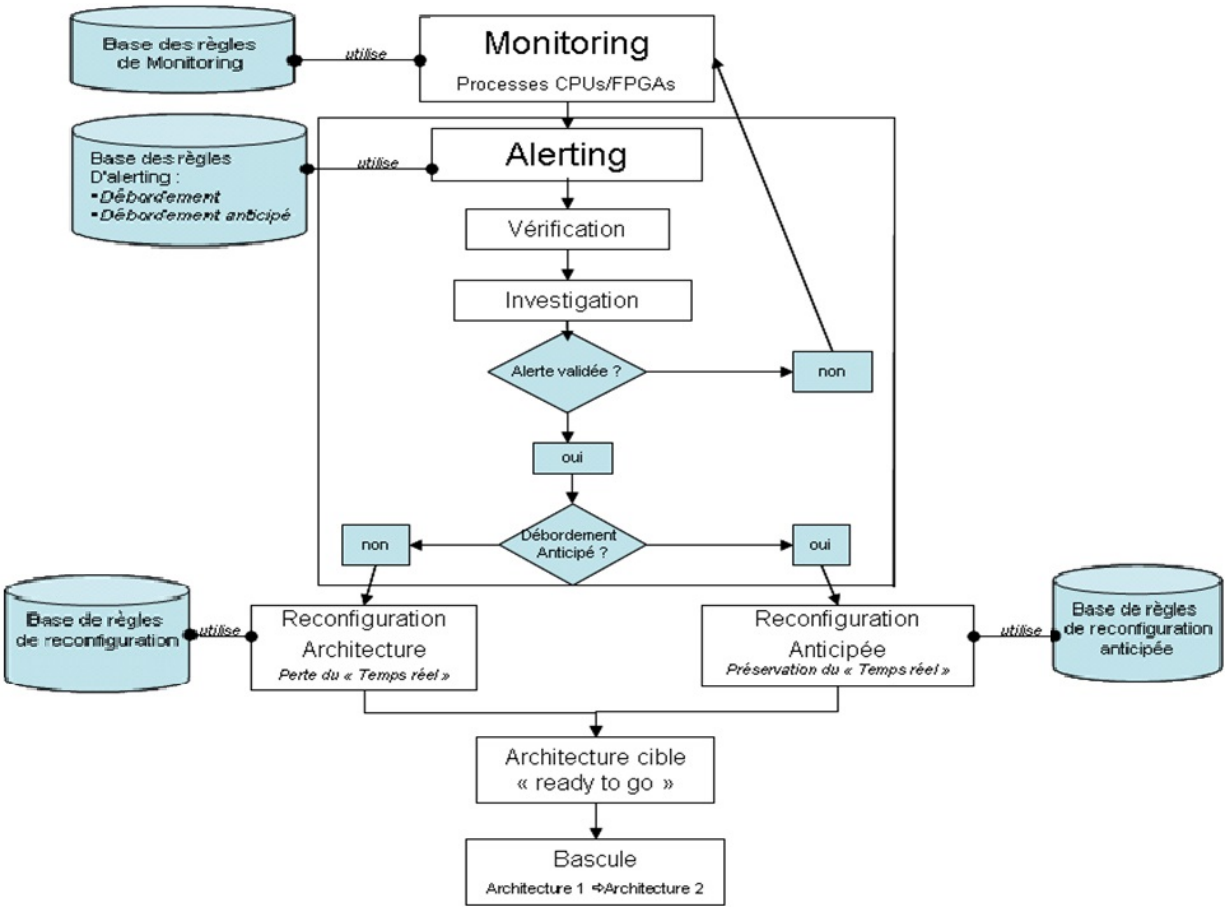

FIGURE 3.7 – Suivi de l'exécution d'une session de test et de simulation

Dans le but d'assurer le bon déroulement des sessions de test et de simulation, notre architecture est capable d'assurer la reconfiguration dynamique hétérogène de ces modèles. Pour ce faire, comme cela est décrit dans la figure 3.7 plusieurs fonctions doivent être assurées dans le but de détecter un besoin et de déclencher une ré-allocation dynamique des tâches liées au test et à la simulation :

3.5.1 la fonction de monitoring

L'environnement de supervision réalise via cette fonction un ensemble de règles de monitoring des modèles exécutés sur les FPGAs et CPUs composant l'architecture. Certaines variables peuvent être traitées en temps-réel, d'autres peuvent être construites de

façon composites et faire l'objet de statistiques sur des périodes de temps. L'objectif de la fonction de monitoring est d'alimenter la base de règles d'alertes.

3.5.2 Les alertes

La fonction d'alertes repose sur une base de règles (règles d'alerting sur la figure). Celle-ci consiste à pouvoir lancer des actions de reconfiguration dynamique de l'architecture. Lorsqu'une variable monitorée est identifiée comme une alerte potentielle, plusieurs actions peuvent être réalisées :

- Une Vérification : Une action de vérification de l'alerte est lancée avant de la valider. On analyse dans ce cas une nouvelle occurrence de la variable remontée en alerte.
- Une Validation : Des règles d'investigation de l'alerte peuvent être lancées comme l'analyse d'une variable composite construite pour l'investigation, l'analyse d'une variable précédemment non monitorée, l'analyse de valeurs statistiques, etc. L'objectif est de parvenir à une validation ou une invalidation de l'alerte.
- Une Migration/Reconfiguration : La fonction de migration/reconfiguration inclut l'ensemble des règles concluant à la validation de l'alerte remontée en première instance. Elle déclenche la reconfiguration de l'architecture via la migration d'un ou plusieurs modèles critiques pour le bon déroulement du test.

Il convient désormais d'apporter des précisions sur le terme de reconfiguration dynamique d'un système distribué respectant les contraintes temps-réel. Ici, il s'agit de minimiser le temps de passage de l'ancienne configuration à la nouvelle pilotée par le Superviseur. A savoir que l'on ne basculera d'une configuration à l'autre que lorsque les modèles déplacés par la reconfiguration se seront déclarés au superviseur "ready to go".

Précisons alors, dans quelles conditions le superviseur intervient :

- Cas périodique (ou cyclique) : Une tâche temps-réel est un ensemble de modèles avioniques à exécuter pour une période donnée. On ajoute de manière dynamique un modèle dans cette liste et la tâche concernée déborde. Dans ce cas, la tâche elle se trouve dans un état "en cours d'exécution" alors qu'elle devrait être "suspendue" jusqu'à la prochaine occurrence du cycle.
- Cas apériodique (ou asynchrone) : Un flot d'évènements entrainant le traitement prioritaire de procédures associées à ces évènements peut perturber l'exécution des tâches périodiques en cours.
- L'instrumentation pour la mesure du temps d'exécution des modèles sur laquelle repose le fonctionnement des superviseurs traditionnels peut générer un niveau d'in-

trusion susceptible de provoquer un débordement.

3.6 Reconfiguration du système

Les systèmes de test et de simulation sont soumis à plusieurs types de contraintes. Dans un premier temps, les contraintes les plus importantes sont celles pouvant entrainer un débordement temporel des modèles. Pour ce faire un ensemble de règles doit être établi dans le but de détecter les conditions nécessaires au déclenchement d'une reconfiguration. Ensuite, certains évènements, prévus par le scénario ou par un évènement non prévu par ce dernier devront nécessiter une reconfiguration partielle du système pour assurer la continuité de la session de test et de simulation.

3.6.1 Restriction au domaine d'application

La capacité de reconfiguration de ce système apporte un grand nombre de contraintes et d'exigences. Nous allons réduire ces contraintes en nous limitant à celles imposées à notre domaine d'application.
Nous allons également restreindre le type de données échangées entre les modèles. En effet, cette restriction nous permettra une compatibilité entre les modèles et les différents outils dors et déjà mis en place. Notre choix s'est porté sur le standard IEEE 754.
Ces données devront rester disponibles pour une utilisation depuis des outils annexes Eurocopter.
La grande disponibilité des composants FPGAs déployables ainsi que la faible occupation des IP Cores de communication avioniques nous permet de nous abstenir d'implémenter une gestion de la reconfiguration dynamique des composants FPGAs. Ainsi, nous pouvons envisager que les composants matériels embarqueront des modèles "dormants" qui seront utilisés si le superviseur estime leur(s) utilisation(s) nécessaire.

3.6.2 Reconfiguration sur débordement anticipé

La fonction de supervision prend en compte des règles de prédiction de débordement qui pourront permettre une reconfiguration dynamique d'un composant FPGA (ou CPU) pendant que l'autre poursuivra l'exécution des modèles. Lorsqu'un élément ou un ensemble d'éléments susceptibles d'entrainer à terme un débordement est isolé, on procède alors à une reconfiguration partielle du FPGA (ou CPU) le moins chargé aussi bien en termes

d'espace que de temps disponible.
Lorsque la reconfiguration partielle de l'ensemble FPGA-CPU est terminée, on peut alors entrer dans l'état "ready to go" de la nouvelle distribution. Cette reconfiguration prédictive qui anticipe un événement de débordement, peut permettre d'assurer la continuité de l'exécution temps-réel de la session de test.
A cela peut s'ajouter l'augmentation du temps d'exécution d'un modèle, en effet, ce temps dépend de la couverture de toutes les boucles ainsi que de toutes les conditions. Par exemple, un modèle de climat, n'aura pas le même temps d'exécution selon les besoins requis par la session de test et de simulation. En effet, la couverture de code sera beaucoup plus importante dans le cas d'un climat complexe (pluie, vent, etc) que lors d'un climat clément (ciel dégagé sans vent).
Comme cela a été décrit précédemment, la ré-allocation des modèles sera assurée par la fonction de supervision pour un respect des contraintes temps-réel.

3.6.3 Reconfiguration prévue par le scénario

Lors de la mise en place du scénario, il est possible de prévoir l'utilisation d'un équipement sous-test au cours de la session. Si l'on prend l'exemple d'une session de test et de simulation dirigée par un pilote, ce dernier pourra choisir de déclencher le pilote automatique dans le but de vérifier son bon fonctionnement. Afin d'assurer le bon déroulement de la session, il est possible qu'un ou plusieurs modèles doivent être réaffectés au cours de cette dernière.

Comme décrit dans la figure 3.8, lors de la description du scénario à T0 (sur la figure), l'utilisateur peut choisir l'implémentation dynamique d'un modèle pendant la session de test et de simulation à un moment précis (TN + x sur la figure). Cette implémentation prévisionnelle déclenchera une allocation dynamique d'un ou plusieurs modèles (M10 et M11 sur la figure à TN).
L'implantation de ces nouveaux modèles peut également entrainer une modification de l'arbre d'exécution des modèles. Cette modification de l'arbre d'exécution nécessitera également la création ou la réorientation de flux de communication ou la création de nouveaux flux de communication. Pour ce faire, cette implémentation de ces nouveaux modèles devra être accompagnée des communications associées, qu'elles soient logicielles ou matérielles comme désigné sur la figure 3.8 par l'allocation d'un (ou plusieurs) IP d'entrée-sortie (IP E/S 3 sur la figure).
L'allocation de ces nouveaux modèles avioniques sur des nœuds de calcul précis ne peut

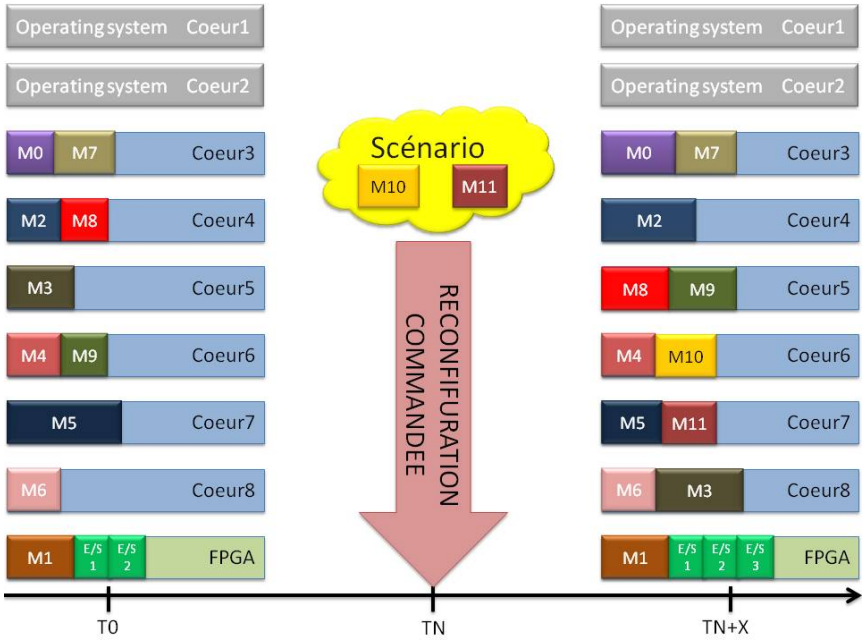

FIGURE 3.8 – Session soumise à une reconfiguration commandée par le scénario

être prédictive. En effet, lors de la session, les modèles verront probablement leur temps d'exécution modifié comme cela l'a été précisé précédemment et donc, l'allocation de ces derniers ne sera pas la même qu'à T0 (lors de la mise en place de la session). L'allocation des modèles M10 et M11 sera donc effectuée par le superviseur à TN pour une première exécution à TN + x.
Dans l'exemple décrit dans la figure 3.8, les deux modèles sont alloués à un nœud de calcul à un même moment. L'utilisateur pourra choisir plusieurs reconfigurations indépendantes les unes des autres temporellement.

3.7 Attraits et limites de ce modèle d'exécution

Plusieurs avantages peuvent être décrits immédiatement par l'exploration des potentialités de ce type de système en y ajoutant le superviseur :

– Dans le monde des entrées-sorties où une réduction substantielle du coût d'un point de connexion (un bus ARINC 429, par exemple) au travers des technologies IP Cores d'entrées-sorties a d'ores et déjà été constatée et évoquée dans l'état de l'art de ce document.

– Dans le cadre de la conception d'équipements embarqués, la méthodologie qui est abordée ici peut permettre de caractériser les modèles qui peuvent être délocalisés dans un FPGA en tenant compte bien sur des contraintes de certification. Certains systèmes de servitude distribués pour l'instant dans l'appareil au travers de boitiers spécifiques pourraient être ainsi hébergés dans un FPGA dédié (IP ECmodel + IP communication + IP memory) localisé et relié au processeur par un bus de communication rapide dans le calculateur principal.
– Dans le nombre de configurations potentielles à tester et par le gain de mise en œuvre en termes de productivité et de rentabilité de chacune de celles-ci résultant du caractère automatique du procédé utilisé.

Néanmoins, il est clair que tous les modèles ne seront pas forcément éligibles pour une implémentation matérielle sur FPGA, particulièrement les modèles éminemment procéduraux ayant beaucoup de dépendances internes nécessaires à leur exécution : c'est par conséquent non seulement le gain de temps d'exécution des modèles (CPU vs FPGA) qui nous intéresse ici mais surtout leur possible délocalisation dans un FPGA afin de bénéficier des avantages abordés précédemment. De plus, il est primordial de proposer une méthodologie de placement de modèles adapté au support matériel hétérogène décrit dans le prochain chapitre.

3.8 Conclusion

Dans ce chapitre, nous avons mis en avant le modèle d'exécution proposé pour les futurs systèmes de test et de simulation dynamiquement reconfigurables. La rencontre des métiers du test et de la simulation par la mutualisation des moyens permis par notre système permettra d'accélérer le cycle de développement des nouveaux équipements embarqués, et par extension, des futurs hélicoptères. Le système proposé aura la capacité de se configurer lors de la phase d'initialisation (configuration hors ligne) en fonction des besoins requis par la session de test et simulation mise en place par l'utilisateur. Ensuite, ces systèmes pourront se reconfigurer dynamiquement (configuration en ligne) dans le but de respecter les contraintes temps-réel requises par le domaine d'application ainsi que les besoins émis lors de la mise en place de la session de test et de simulation.
Dans le prochain chapitre, nous décrirons le support d'exécution matériel proposé dans le cadre de ce projet de thèse pour la prochaine génération de systèmes de test et de simulation.

3. UN MODÈLE D'EXÉCUTION DYNAMIQUE POUR LES ARCHITECTURES HÉTÉROGÈNES DÉDIÉ AU TEST ET À LA SIMULATION

Chapitre 4

Un support matériel pour la prochaine génération de systèmes de test et de simulation

4.1 Introduction

Comme cela a été évoqué dans le chapitre d'introduction de cette thèse, la validation des nouveaux équipements embarqués est une phase essentielle dans le cycle de développement d'un nouvel hélicoptère. Les limites de la solution actuelle imposent une refonte complète du support matériel existant. Ce chapitre a pour but essentiel d'étudier la problématique des architectures de test à plusieurs niveaux. Dans un premier temps, nous l'étudierons au niveau matériel, mettant en avant chacun des nœuds de calculs hétérogènes ainsi que le mode de communication primordial dans ce type d'architecture. Ensuite, nous mettrons en avant la capacité d'allocation dynamique hétérogène de notre architecture permettant ainsi le placement et le replacement de modèles de test. Ces nouvelles fonctionnalités auront pour but de réduire le "time-to-market" des nouveaux appareils en accélérant le cycle de développement tout en réduisant les coûts liés au test et à la simulation. Ce chapitre est organisé comme suit. La section 4.2 détaille l'architecture choisie en détaillant chacun des nœuds de calcul ainsi que leurs capacités. Dans la section 4.3, nous étudierons les bus rapides éligibles pour notre support matériel. Enfin dans les sections 4.4 et 4.5 nous étudierons les optimisations nécessaires à effectuer sur les modèles afin d'obtenir les meilleures performances possibles en prévision d'une exécution sur le support matériel proposé.

4.2 Architecture hétérogène CPU-FPGA

La figure 4.1 présente l'architecture choisie pour les futurs systèmes de test et de simulation. La solution proposée offre la généricité ainsi qu'une réponse aux contraintes du domaine d'application. Dans les prochaines sous-sections, nous allons décrire les différents nœuds de calcul ainsi que le lien de communication entre ces derniers.

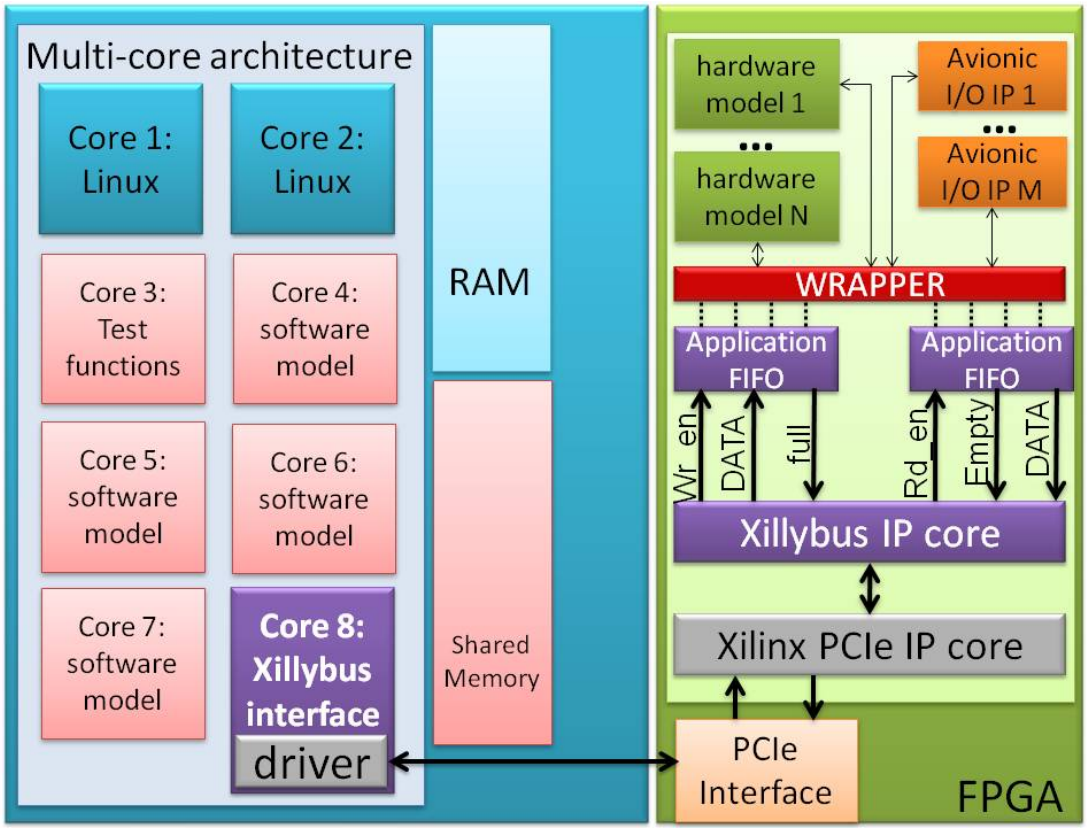

FIGURE 4.1 – Architecture CPU-FPGA pour les bancs de test et de simulation

4.2.1 nœud de calcul multi-cœur

Les architectures multi-cœur actuelles offrent des performances compatibles avec celles requises par le domaine d'application. De plus, les environnements actuels fournissent les éléments nécessaires pour le développement d'applications tirant profit des capacités et des propriétés de ces architectures.

Depuis le lancement des premiers ordinateurs sur le marché, les demandes croissantes des capacités de calcul par les applications ont poussé les constructeurs à proposer des

architectures répondant aux contraintes de performances requises. L'augmentation des fréquences de calcul a entrainé des problèmes de surchauffe.

En effet, l'augmentation des capacités de calcul est tributaire de la consommation de puissance, et le fait d'augmenter cette consommation nécessite alors de gérer les niveaux de dissipation thermique. Pour remédier à ces problèmes, des solutions telles que le refroidissement par eau ou bien la ventilation voient le jour. Une autre approche est de multiplier les cœurs de calcul plutôt que leur fréquence. De plus, le multitraitement symétrique (SMP) a longtemps été utilisé pour améliorer les performances en répartissant les charges sur plusieurs processeurs. Le SMP est particulièrement efficace dans les environnements multi-tâches où plusieurs tâches doivent être gérées simultanément. L'appellation multi-cœur est alors employée pour décrire un processeur composé de plusieurs unités de calcul placées sur la même puce. C'est en 2001 qu'est commercialisé le premier processeur multi-cœur, il s'agit du POWER4 [45] par IBM.

A l'heure actuelle, plusieurs architectures multi-cœur sont disponibles. Intel et AMD se partagent un marché grand public. La demande croissante de l'industrie amène les grands constructeurs à multiplier les cœurs. Dans ce cadre, le projet Tera-Scale [50] [66] dirigé par Intel propose plus de 80 cœurs de calcul sur une même puce. De plus d'une architecture, Intel propose également un modèle de programmation Ct [28] basé sur une extension du langage C++ permettant d'exploiter les propriétés parallèles de l'architecture proposée en assurant la parallélisation automatique de code source exploitant les caractéristiques de l'architecture SMP proposée.

Les architectures processeur actuelles contiennent des mémoires caches, cette propriété est également vraie pour les architectures multi-cœur. Toutefois, il n'y a deux possibilités dans l'accès à ces mémoires caches. En effet, une première possibilité est le partage d'une mémoire cache visible depuis tous les cœurs et une seconde verra une mémoire cache dédiée à chaque cœur.

Une architecture basée sur une spécialisation des caches sur chacun des cœurs permettra qu'il n'y ait aucune interférence dans les accès à ces mémoires. De plus, le temps d'accès aux mémoires est généralement plus faible que dans le cas d'un cache partagé.

Une architecture basée sur l'utilisation de caches partagés présente également plusieurs avantages. Tous les programmes devront alors se partager l'accès à ces mémoires et ce, dynamiquement. Il pourront ainsi se répartir l'occupation de la mémoire de manière assez souple avec des mécanismes de cohérence. Dès lors, il devient possible de partager les données entre plusieurs threads ne nécessitant pas alors une réplication des données comme

cela peut être nécessaire dans le cas de caches dédiés. Néanmoins, ce type d'architecture présente quelques inconvénients. En effet, la mémoire cache devra partager ses accès entre plusieurs cœurs, ce qui introduira des latences supplémentaires. Les accès simultanés devant être possibles afin d'éviter l'introduction de mécanismes d'attentes entre les cœurs de calcul ayant besoin d'accéder une même zone. De ce fait, ces mémoires caches sont des mémoires multi-ports.

Dans la réalité, compte tenu des contraintes décrites ci-dessus, tous les caches du processeur multi-cœur ne sont pas partagés. En effet, ceux qui ont besoin d'une faible latence (cache L1) sont des caches dédiés, tandis que les autres sont partagés (L2, L3).

Néanmoins, ce type d'architecture peut présenter un désavantage. En effet, les applications doivent être conçues pour exploiter les caractéristiques de ces processeurs, la fréquence de chacun des cœurs de calcul étant inférieure à la fréquence des anciens processeurs mono-cœur, un programme non optimisé verra par conséquent ses performances diminuées. Dans le cadre de notre projet, nous allons optimiser les tâches composant notre application dans le but de tirer parti des propriétés de ces unités de traitement.

Comme cela est décrit sur la figure 4.1, notre stratégie est de "caractériser" l'utilisation des cœurs selon trois niveaux hiérarchiques.

- Un premier niveau sera consacré au système d'exploitation (cœurs bleus sur la figure 4.1). Cette spécialisation aura pour but de rendre les autres cœurs non disponibles vis à vis des interruptions systèmes qui pourraient perturber le bon fonctionnement de la session de test. Ce niveau est hors contrainte temps-réel.
- Un second niveau (cœurs roses sur la figure 4.1) est attribué aux fonctions et aux modèles de test et de simulation, c'est à dire soumis aux contraintes temps-réel requises par le domaine d'application. Sur ces cœurs de calcul, seront exécutées les fonctions propres à la session de test et de simulation en cours.
- Un troisième niveau (cœur violet sur la figure 4.1) est attribué aux fonctions de communication avec les accélérateurs matériels hébergés au sein du composant FPGA.

Cette hiérarchisation des cœurs processeurs est également valable pour la mémoire RAM disponible. En effet, cette dernière peut être répartie selon son utilisation. Par exemple, dans la figure 4.1, la partie bleue sera utilisée par le système et la partie rose sera utilisée par les cœurs roses. Ainsi, il devient possible d'effectuer les communications inter-modèles via des sections de mémoires partagées. En effet, les données produites lors de sessions de test et de simulation sont émises par un producteur unique vers un ou plusieurs lecteurs. Dès lors les coûts de communication inter-modèles deviennent constants indépendamment

de leur taille.

4.2.2 nœud de calcul FPGA

Les réseaux de logique programmables sont des circuits composés de nombreuses cellules logiques élémentaires pouvant être assemblées sur mesure. Les composants FPGA sont basés sur ces cellules SRAM aussi bien pour le routage que pour les blocs logiques interconnectables. Chacun de ces blocs logiques est généralement constitué d'une ou plusieurs table(s) de correspondance (Look-Up-Table ou LUT) et d'une ou plusieurs bascule(s) (Flip-Flop). La LUT permet d'implémenter des fonctions logiques et peut être également considérée comme une mémoire de petite capacité, un registre à décalage ou un multiplexeur. Si l'on prend le cas du Virtex 7 de chez Xilinx [23] [21], chaque "slice" contient 4 LUTs et 8 bascules. D'ailleurs, certains de ces slices peuvent utiliser leurs LUTs comme des mémoires RAM distribuée ou bien des registres à décalage.

La configuration de ces composants est aujourd'hui facilitée par la possibilité d'implémentation d'IP Cores (Intellectual Property) pouvant assurer des fonctions spécifiques. Un catalogue varié d'IP Cores est proposé par les constructeurs eux-mêmes comme Xilinx, Altera, ainsi que par des sociétés extérieures. Ces IP Cores peuvent être implantés aisément sur les composants FPGAs via les environnements de développement disponibles sur le marché et proposés par les constructeurs.

Aujourd'hui, plusieurs types d'IP Cores sont proposés, assurant des fonctions les plus élémentaires aux plus complexes :

- Les Blocs élémentaires tels que les FIFOs, les mémoires, les opérateurs arithmétiques à virgule flottante utiles dans le cadre de développement de modèles test et de simulation parallélisables complexes.
- Les IPs de test permettant la vérification dynamique des fonctions implémentées sur le composant dans le but d'assurer leur bon fonctionnement.
- Les éléments architecturaux tels que les interfaces mémoire, horloges et d'entrées/-sorties standard nécessaires pour l'utilisation optimale de la carte où le composant FPGA est implémenté.
- Les BUS interfaces rapides tels que le PCI, PCIe, Ethernet pour la liaison rapide du composant FPGA vers d'autres nœuds de calcul. Ces interfaces rapides sont nécessaires pour le développement d'architectures hétérogènes pour le calcul haute performance.
- Les fonctions DSP telles que du traitement de signal (FIR, FFT, etc.). L'apport des

architectures FPGAs permettra alors une implémentation aisée de fonctions purement matérielles sur les futurs systèmes de test et de simulation.
- Les fonctions plus spécifiques à certains domaines tels que les bus avioniques (Arinc 429, CAN, 1553, etc) primordiaux dans le cadre du développement des futurs systèmes de test et de simulation.

Dans le cadre de notre projet, nous proposons de définir plusieurs familles d'IPs propres à notre application :

- Les **Entrées-Sorties** qui permettront la communication avec les équipements sous tests, composées de bus utilisés dans le domaine de l'aéronautique (CAN, 1553, ARINC 429, etc.) sont aujourd'hui disponibles sur le marché et pourront donc être facilement intégrées aux composants présents sur le support matériel.
- Les **fonctions matérielles** telles que le traitement de signal, aisées à intégrer sur ce type d'architecture comme nous l'avons vu précédemment. Ces fonctions sont nécessaires dans le cadre de scénarios de test et de simulation afin d'introduire des signaux ou perturbations matérielles. En effet, un scénario peut prévoir ce type de perturbations lors de la session de test.
- Les **fonctions de traitement de données** nécessaires pour la gestion des différents types transitant au sein du/des composants hétérogènes du support matériel.
- Les **modèles de test et de simulation** seront les mêmes que ceux composant les sessions décrites dans le chapitre d'état de l'art. Ces derniers devront nécessiter un développement spécifique décrit dans la suite de ce chapitre. En effet, afin de tirer parti des propriétés des composants FPGA, la mise en place d'un environnement de développement est nécessaire. D'ailleurs, nous allons proposer dans le chapitre suivant une description de cet environnement.

Il devient alors possible lors de la conception de notre support matériel spécifique d'obtenir des solutions "sur étagère" pour accélérer le cycle de développement. Dans le cadre de notre projet, notre intérêt s'est porté dans un premier temps sur cette technologie car l'offre d'IP Core de bus avioniques répond désormais à la demande des futurs architectures de test et de simulation pour la communication avec les équipements sous test.
De plus de ces facilités de développement, les FPGAs offrent une capacité de configuration et de reconfiguration dynamique intéressantes. En effet, il est possible de remplacer une configuration par une autre en rechargeant l'intégralité du composant ou bien de créer des régions reconfigurables dynamiquement comme cela se fait usuellement sur les composants FPGA dotés de cette capacité.

Malgré des fréquences de fonctionnement inférieures, les composants FPGA bénéficient grâce à leur structure interne composée d'éléments indépendants les uns des autres de capacités de calcul haute performance intéressantes. En effet, dans le cas d'algorithmes composés de fonctions pouvant s'exécuter indépendamment les unes des autres les composants FPGA peuvent offrir des performances supérieures aux CPUs ainsi qu'aux GPUs dans le cas de traitement d'image par exemple. En effet, dans [12], les auteurs présentent la comparaison de temps d'exécution d'application spécifiques entre CPU, GPU et FPGA. Il est alors prouvé que les architectures FPGA offrent de meilleurs performances dans le cadre de certaines applications telles que le traitement d'image en comparaison des performances offertes par les CPUs et GPUs.

Comme nous l'avons décrit dans la sous-section précédente sur l'utilisation des cœurs processeurs, un second niveau de hiérarchisation est visible pour les régions reconfigurables sur la partie droite de la figure 4.1 :

- Un premier niveau hiérarchique correspond à l'ensemble des modèles de test et de simulation (**Hardware model** sur la figure 4.1). Ces derniers pourront communiquer avec d'autres modèles présents sur les cœurs, sur le composant FPGA ou avec un équipement sous test via un IP d'entrées-sorties.
- Un second niveau hiérarchique correspond à l'ensemble des IPs d'entrées-sorties vers les équipements (**AVIONIC I/O IP** sur la figure 4.1).
- Un troisième niveau hiérarchique correspond à un **wrapper** qui fera le lien entre les données entrantes et sortantes vers les IPs (modèles, fonctions ou entrées-sorties).

S'il devenait nécessaire d'implémenter des mécanismes de reconfiguration dynamiques pour des raisons d'optimisation de la consommation ou des ressources matérielles sur les composants FPGA, ces trois niveaux peuvent correspondre à trois différentes régions reconfigurables. Dès lors, il sera nécessaire que le wrapper soit mis à jour dès qu'un modèle ou un IP soit implanté/remplacé par un autre.

La définition des nœuds de calcul n'est pas suffisante dans ce type d'architecture, un lien de communication performant entre les nœuds doit être établi. En effet, aussi performants les nœuds de calcul soient-ils indépendamment les uns des autres, le lien permettant de rendre l'ensemble du système cohérent devra permettre un coût de communication inter-nœud très faible voire négligeable. Dans la prochaine section, nous allons décrire la solution mise en place dans le cadre du développement des futurs systèmes de test et de simulation.

4.3 Un bus de communication rapide pour les systèmes de test et de simulation

L'implémentation des modèles sur les nœuds de calcul hétérogènes ne pourra être optimale qu'après mise en place d'un bus rapide répondant à plusieurs critères. Le premier concerne la performance de ce bus, point essentiel dans le cadre de recherche de hautes performances. Le second concerne la compatibilité avec les machines actuelles, en effet, les nœuds de calcul matériels devront pouvoir s'interfacer avec les bus disponibles.

4.3.1 Ethernet

Le standard de transmission de données Ethernet (également connu sous le nom de norme IEEE 802.3) est basé sur le principe de membres (pairs) sur le réseau. Chaque pair y est identifié par une clef appelé adresse MAC unique dans le but de s'assurer qu'il n'y ait pas de doublon sur le réseau.

Les pairs d'un réseau Ethernet sont tous reliés à une même ligne de transmission, la communication se faisant par le biais d'un protocole CSMA/CD (Carrier Sense Multiple Access with Collision Detect) signifiant que ce dernier est à accès multiple avec surveillance de porteuse et à détection de collision.

Avec le protocole décrit ci-dessus, chaque pair est autorisé à émettre sur la ligne de transmission unique à n'importe quel moment et ce, sans aucune notion de priorité entre les pairs.

Les paquets des données transmis doivent avoir une taille maximale et un temps d'attente peut apparaître entre deux transmissions. Ce temps est variable selon le nombre de collisions, c'est à dire que plusieurs pairs émettent simultanément obligeant chaque pair à attendre une unité de temps lors de chacune des collisions. Cette contrainte est problématique dans le cadre du développement d'un système avec des contraintes temps-réel.

4.3.2 PCIe

Le standard PCIe (pour Peripheral Component Interconnect express) fût introduit en 2004 par Intel spécifiant un bus local série dérivé de la norme PCI existante et un connecteur permettant de connecter des cartes d'extension sur les carte-mère actuelles. Le PCIe utilise une interface série basée sur des lignes bidirectionnelles.

A l'heure actuelle, le PCIe est disponible au niveau des deux puces du chipset des

carte-mères, le northbridge et le southbridge voire également intégré directement à certains microprocesseurs.

C'est en 2010 que le PCI-SIG (Peripheral Component Interconnect Special Interest Group) publie le cahier des charges du PCIe gen 3.0 proposant une multiplication du débit par deux. En effet, avec une fréquence de fonctionnement passant de 5 GHz sur le gen 2.0 à 8 GHz, cette nouvelle génération offre des débits encore supérieurs à la génération actuelle.

4.3.3 Test des différents bus proposés

Dans cette section, nous allons évaluer les performances des bus Ethernet et PCIe tant au niveau des performances que des latences sur ces supports. Le but de ces essais est de déterminer le meilleur moyen de communication avec un FPGA hébergeant des accélérateurs matériels sur notre architecture hétérogène.
Dans le but d'obtenir les meilleures performances possibles, nous avons utilisé une API de communication spécifique fournie par la société Dolphin : SuperSockets. Cette API utilise des fonctions standard en langage C pour créer une communication sur la base de modèle client/serveur. Dès lors, l'utilisation de SuperSockets est totalement transparente pour les développeurs. Initialement, cette API a été créée pour une communication PCIe vers PCIe. Toutefois, il peut être utilisé sur un réseau Ethernet. Dans le but d'obtenir des résultats de tests pertinents, nous utiliserons les mêmes protocoles de mesures dans les deux cas.

4.3.3.1 Évaluation des performances du bus Ethernet

Comme nous pouvons l'observer sur la figure 4.2, l'Ethernet 1 Gb/s présente deux parties linéaires. La première est comprise pour des tailles de mémoire tampon comprises entre 1 et 8192 Octets. La seconde partie linéaire présente une croissance inférieure à la première pour des valeurs supérieures à 16 Kilo-Octets de taille du tampon. La bande passante maximale est atteinte avec une taille tampon égale à 64 Kilo-Octets avec une valeur de 99 Mega-Octet/s. Cette bande passante représente 80% de l'occupation du bus.

La figure 4.3 nous permet d'observer les latences introduites par l'utilisation de l'Ethernet 1Gb en fonction de tailles de mémoire tampon. Nous pouvons observer que les latences moyennes présentent un profil linéaire. De plus, les latences minimales étant proches des latences moyennes, nous pourrons nous baser sur ces dernières pour une estimation du coût de ces latences. La courbe RTT représente le temps de transmission total pour chaque taille de mémoire. Une fois de plus, nous observons un profil linéaire pour cette courbe

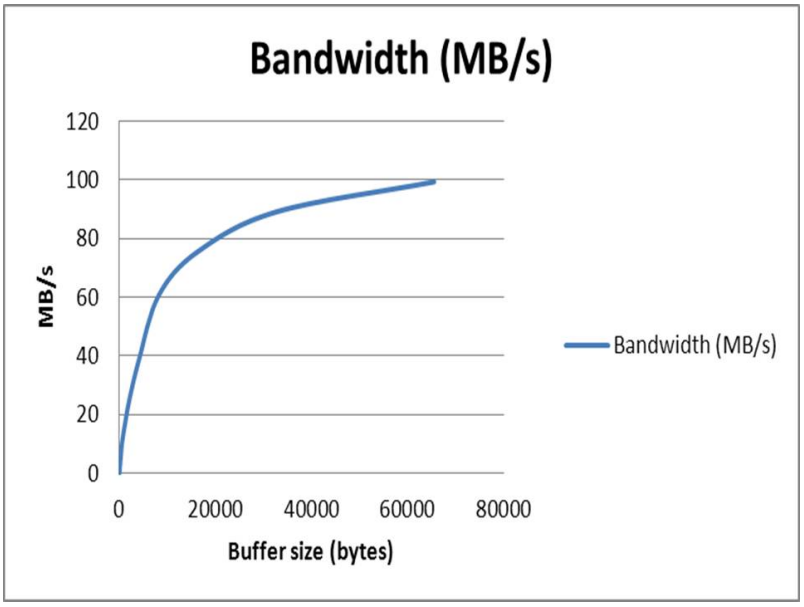

FIGURE 4.2 – SuperSockets API - Ethernet 1Gb/s - Test des performances

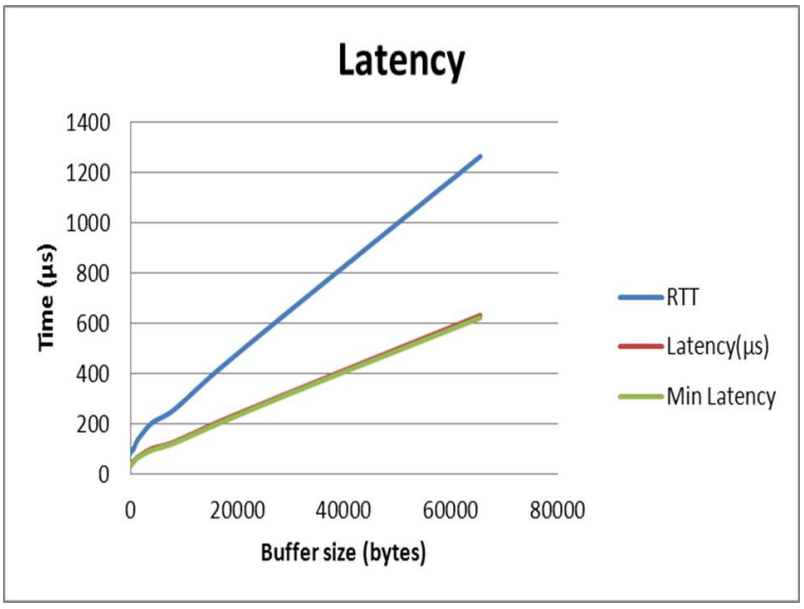

FIGURE 4.3 – SuperSockets API - Ethernet 1Gb/s - Test des latences

nous permettant encore une fois une estimation aisée des coûts de communication.

4.3.3.2 Évaluation des performances du bus PCIe en mode mémoire à mémoire

Dolphin fournit également l'API SISCI qui nous servira au test de performances du bus PCIe en mode mémoire à mémoire. Dans les tests qui suivent nous nous sommes basés sur un modèle client/serveur basé sur deux nœuds distants et dont les résultats seront fournis par le client.

Le premier test consiste en un transfert mémoire à mémoire entre deux nœuds de calcul. Les mesures et les résultats présentés ci-dessous porteront sur la latence moyenne et la bande passante pour des données allant de 1 à 64 kilo-octets. Pour chaque taille de mémoire tampon, nous avons effectué 20 000 boucles.

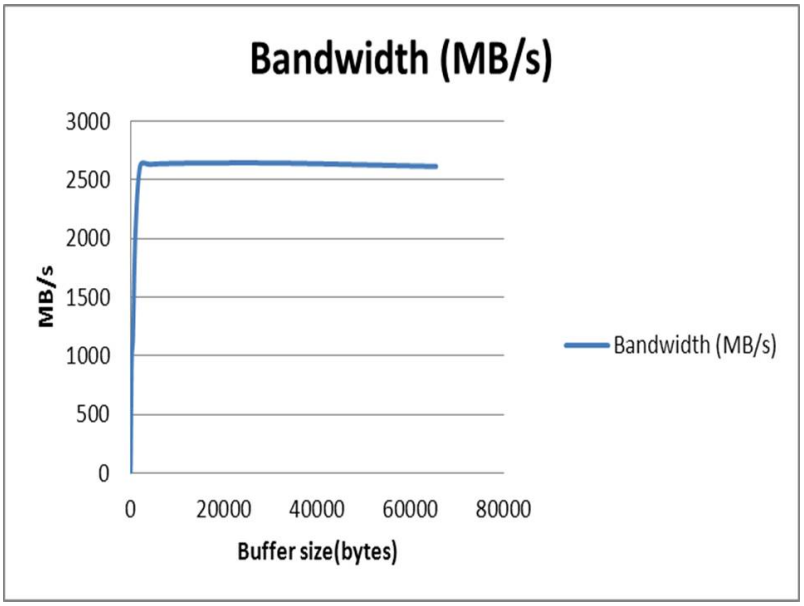

FIGURE 4.4 – SuperSockets API - PCIe - Test en Performances

Le résultat de ce premier test présenté sur la figure 5.9 montre que ces transferts mémoire à mémoire offrent une bande passante fortement croissante pour des valeurs de données comprises entre 1 et 2048 Octets pour se stabiliser à 2,5 Giga-Octets par seconde.

La figure 5.10, résultat du profilage des latences moyennes sur des transferts mémoire à mémoire présente un profil linéaire. La latence maximale de 6,19 micro-secondes correspond à un transfert de 64 Kilo-Octets.

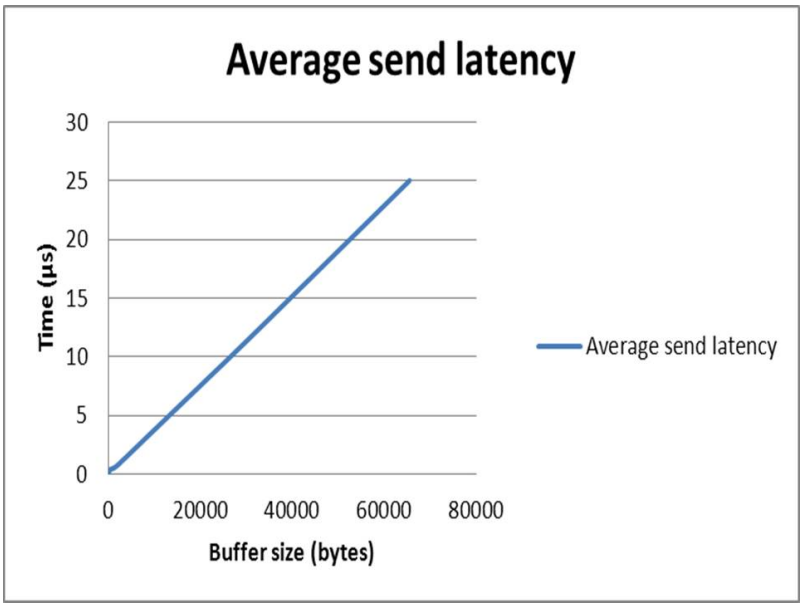

FIGURE 4.5 – SuperSockets API - PCIe - Test des latences

4.3.3.3 synthèse

Les résultats présentés ci-dessus nous permettent d'orienter notre choix concernant le bus de communication à utiliser dans notre support matériel. En effet, les latences introduites par le bus Ethernet ne sont pas compatibles (surtout avec une utilisation avec le protocole TCP/IP) avec les contraintes temps-réel requises. De plus, les architectures PC actuelles embarquent toutes des emplacements PCIe permettant d'obtenir les performances décrites ci-dessus.

4.3.4 Xillybus : Solution de prototypage pour les architectures CPU-FPGA

La mise en place de ce type de communication est généralement long et contraignant tant l'obtention de hautes performances est complexe. En effet, une telle mise en place nécessite le développement d'un driver ainsi que d'un IP Core PCIe. Dans le cadre d'une première validation fonctionnelle, peu de solutions "clefs en main" existent et permettent de mettre en place un premier prototype. Dans le cadre de ce projet de thèse, nous avons proposé la solution Xillybus pour le prototypage de notre architecture CPU-FPGA.

Dans le cadre du développement de notre prototype, nous nous sommes intéressés à la

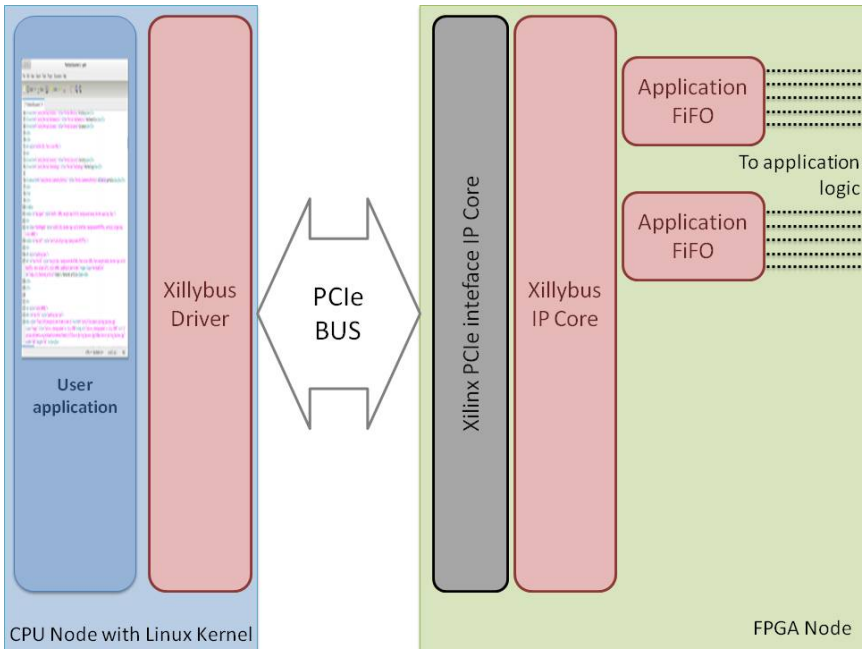

FIGURE 4.6 – Solution de prototypage Xillybus

communication entre nos nœuds de calcul hétérogènes. Nous nous sommes alors adressés à une jeune entreprise, Xillybus [76] créée en 2012 par Eli Bilauer, qui offrait un "kit" de démarrage permettant aux développeurs d'implémenter facilement des accélérateurs sur des architectures CPU-FPGA reliées par un bus PCIe. Dès lors, nous avons collaboré afin de stabiliser, tester et valider le bon fonctionnement du produit proposé. Cette contribution a été mise en avant par une co-écriture d'un article de conférence [8].

Ainsi, nous avons utilisé cette solution pour une première validation fonctionnelle du flux de données à travers notre architecture CPU-FPGA évitant de nombreux bogues liés au développement d'un nouvel IP ainsi qu'un driver PCIe associé. De plus, Xillybus fournit aujourd'hui des solutions supplémentaires telles qu'un DMA PCIe, plusieurs tubes de flux de données permettant un lien direct entre les applications logicielles et matérielles.

Comme cela est décrit dans la figure 4.6, la solution Xillybus est composée de deux interfaces. La première permettra de connecter une application logicielle communicant avec un accélérateur matériel FPGA en fournissant un driver simple offrant un accès direct à l'accélérateur matériel embarqué sur le composant FPGA. La seconde, directement connectée à l'IP Core PCIe de base intégré sur le composant. Deux FIFOs viendront s'interfacer directement avec l'application matérielle implémentée sur le FPGA.

Bien sûr, une telle solution n'offrira pas les mêmes performances qu'une solution orientée

haut débit, cela est dû aux fréquences (250 Méga-Hertz) de fonctionnement du composant FPGA. En effet, les FIFOs alimentant les accélérateurs matériels implémentés dans le composant sont cadencées à cette même fréquence. Dès lors, toute sérialisation du flux de données réduit le débit global de transmission.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	537	301440	0%
Number of Slice LUTs	738	150720	0%
Number of fully used LUT-FF pairs	469	806	58%
Number of bonded IOBs	23	600	3%
Number of Block RAM/FIFO	9	416	2%
Number of BUFG/BUFGCTRLs	5	32	15%

FIGURE 4.7 – Ressources utilisées par Xillybus

Dans la figure 4.7, nous présentons l'espace occupé sur un composant FPGA Xilinx Virtex 6 (XC6VLX240T [74]). Nous pouvons constater que l'espace occupé est très négligeable par rapport à l'espace disponible permettant ainsi d'implémenter aisément un accélérateur matériel à valider.

4.4 Optimisations sur le code legacy des modèles avioniques

Comme nous l'avons décrit précédemment, le support du code legacy est un besoin industriel essentiel. En effet, les modèles avioniques développés sont à la base des sessions de test et donc nécessaires pour leur mise en place. Dès lors, un profilage de ces modèles et leur optimisation pour l'obtention de hautes performances sont proposés.

4.4.1 Profilage et parallélisation du code legacy des modèles avioniques

L'architecture proposée et décrite précédemment permettra de répondre aux contraintes imposées par le domaine du test et de simulation. Toutefois, il est nécessaire d'apporter un environnement complet d'outils permettant d'effectuer toutes les étapes nécessaires depuis

le profilage d'un modèle jusqu'à l'implémentation sur sa cible d'exécution matérielle. En effet, pour surmonter les difficultés liées à ce type de conception, nous avons proposé une méthodologie [6] décrite dans la figure 4.8.

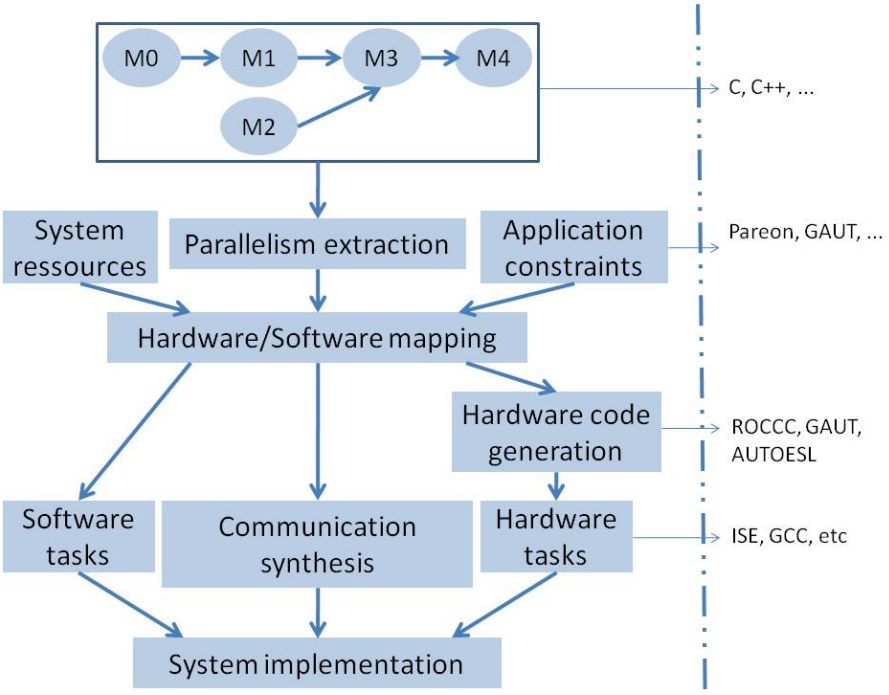

FIGURE 4.8 – Méthodologie de développement

Tout d'abord, il est nécessaire de spécifier un projet de simulation présenté comme un graphe de tâches contenant différents modèles communicants (M0, M1, etc.). Tous les projets n'étant pas adaptés à une exécution performante sur architecture CPU-FPGA, il est nécessaire d'effectuer une analyse complète du code source afin de déterminer son potentiel de parallélisation.
Pour ce faire, des outils d'analyse tels que Pareon [67] proposé par la société Vector Fabrics, Intel Parallel Studio [33] ainsi que GAUT [19] sont aujourd'hui disponibles. Ces outils effectuent généralement une compilation de ce code source pour une analyse bas niveau de ce dernier. Ainsi, il devient possible de situer les dépendances de données et ainsi d'extraire le degré de parallélisation comme nous pouvons le constater sur la figure 4.9. Sur cette figure, nous pouvons observer le résultat d'une analyse d'un code par l'outil Pareon présentant plusieurs boucles. Deux de ces boucles présentent une faible dépendance de données entre les différentes itérations, dès lors en répartissant leur exécution sur plusieurs threads, il est possible de réduire leur temps d'exécution. Dans l'exemple présent sur cette

figure, nous pouvons constater que l'utilisation de quatre threads permettra une réduction du temps d'exécution par un facteur de 3,6. De plus, Pareon offrira une représentation visuelle du nouveau graphe d'exécution du code parallélisé permettant ainsi de visualiser les mécanismes de communication et de synchronisation à mettre en place.

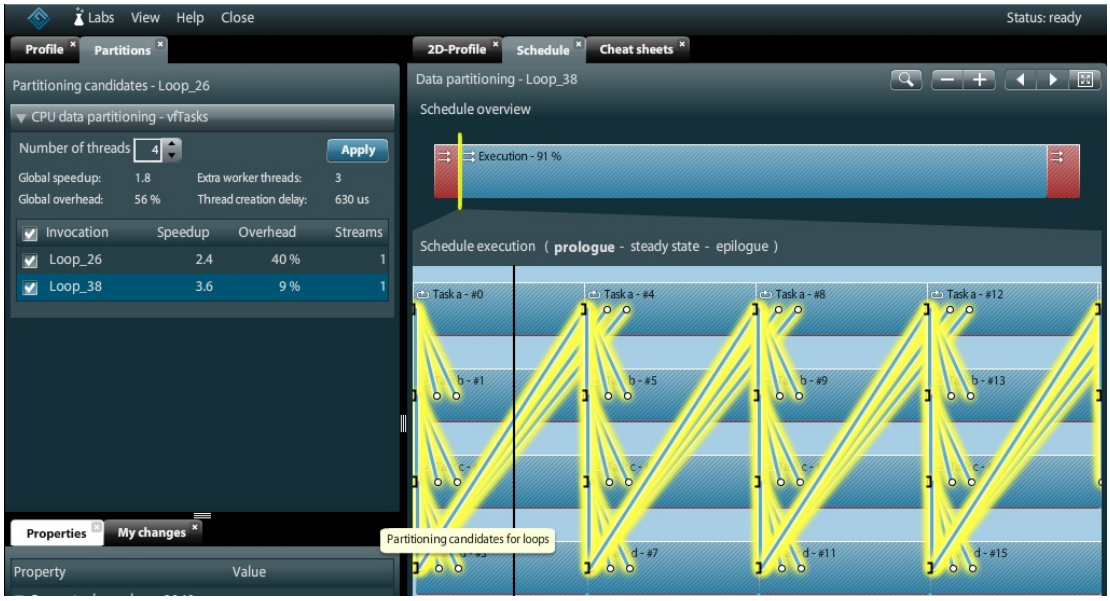

FIGURE 4.9 – Résultat d'une analyse sous Pareon

C'est par une étape de compilation que GAUT offrira une représentation de l'arbre d'exécution au niveau instruction. Cette représentation de bas niveau permettra une visualisation de l'implémentation possible sur une cible matérielle. En effet, les blocs mathématiques proposés par les fournisseurs d'IP Cores pourront venir remplacer ces instructions. D'ailleurs, l'outil GAUT propose en complément de sa fonction de profilage, une fonction de génération de code VHDL depuis une source codée en langage C. Le code généré sera sous la forme d'une structure pipeline avec une exécution parallèle de fonctions à chaque étage avant synchronisation des données.

Pour effectuer l'allocation des différents modèles du scénario, il est nécessaire de connaitre les ressources matérielles disponibles ainsi que les contraintes de l'application. Cette étape nécessite une méthode approchée (heuristique ou méta-heuristique) pour résoudre ce problème d'exploration multi-objectif tel qu'il sera décrit dans la section suivante. Le but sera ici d'obtenir une solution efficace dans des délais respectant les contraintes de l'application. Une méthode exacte sera utilisée dans le cas du placement initial (hors ligne) du projet de simulation, le temps d'exécution n'ayant pas d'impact sur le bon déroulement de la session

de test sur cette phase.
Après l'étape d'allocation des différents modèles sur les ressources matérielles disponibles, il sera nécessaire de développer les fonctions matérielles correspondant aux modèles constituant le scénario de test. Dans le but d'accélérer cette étape, nous proposons l'utilisation de générateurs de code tels que Riverside Optimizing Compiler for Configurable Computing (ROCCC) [68] et Vivado HLS (High Level Synthesis) [73]. ROCCC, tout comme GAUT effectue une compilation du code source du scénario pour une analyse précise des dépendances de données des modèles composant ce dernier. En effet, cet outil ne se contente pas de traduire le code C vers du VHDL, il optimise le nombre total de cycles d'horloge nécessaires en parallélisant l'algorithme à implémenter dans une structure pipeline. Xilinx, propose depuis peu l'outil HLS qui permettra également la génération de code avec de nouvelles fonctionnalités telles que l'utilisation de virgules flottantes tout en ajoutant les interfaces nécessaires pour la mise en place des mécanismes de communication entre les modèles générées vers d'autres cibles matérielles en utilisant les interfaces AXI [10] conformes aux spécifications AMBA AXI version 4 provenant de chez ARM. Dès lors, le code généré devient un accélérateur matériel disponible et utilisable depuis l'ensemble de la suite logicielle disponible.
Une fois l'ensemble des codes sources de l'application développés et/ou générés, une dernière étape de compilation permettra l'implémentation finale des modèles composant l'application sur le support matériel.
La dernière étape consistera à implémenter la communication entre tous les modèles afin de rendre l'ensemble cohérent. Pour ce faire, les dépendances de données entre les modèles décrites lors de la mise en place du scénario de test permettront dé générer les médiums de communication.

4.4.2 Allocation des modèles sur le support matériel

L'étape d'allocation nécessite une méthode approchée qui prendra en entrée le graphe d'exécution et un ensemble de paramètres associés à chaque modèle. Son but sera axé sur la recherche des meilleures configurations qui satisfont les exigences de performances ainsi que l'utilisation des ressources disponibles. Le tableau 4.4.4 résume les différents paramètres ainsi que les fonctions coût.

Pour chaque modèle, l'algorithme proposé explorera les différentes implémentations matérielles logicielles en fonction du temps d'exécution et des ressources matérielles disponibles. De plus, il tient compte de la surcharge de communication en fonction de la cible de

TABLE 4.1 – Paramètres et fonctions coût

Paramètres	Définitions
$T_i^s(v)$	Temps d'exécution d'une implémentation logicielle d'un modèle avionique i en fonction de la version
$T_i^h(v)$	Temps d'exécution d'une implémentation matérielle d'un modèle avionique i en fonction de la version
δ_i	différences entre les temps d'exécution des version matérielles et logicielles
$C_i^s(m)$	Coût de la communication d'une version logicielle d'un modèle en fonction du "mapping" des données
$C_i^h(m)$	Coût de la communication d'une version matérielle d'un modèle en fonction du "mapping" des données
$Rt_i(s)$	Temps de reconfiguration en fonction de la taille du bitstream
$M^s t_i$	Temps de re-affectation d'un modèle logicielle i
$M^h t_i$	Temps de re-affectation d'une modèle matérielle i

Fonctions de coût	Définitions
T_{time}	Temps d'exécution total
T_{area}	Surface totale utilisée

traitement (CPU ou FPGA) et l'allocation des données. Aujourd'hui, avec l'apport de la reconfiguration dynamique partielle (PDR) disponible sur les composants FPGA actuels, la reconfiguration du système peut se faire à la volée lors de l'exécution de l'application. Notre algorithme d'allocation des modèles considère le temps de reconfiguration pour une implémentation matérielle. Cet aspect est très intéressant pour les systèmes qui impliquent des contraintes temps-réel. En raison des ressources matérielles limitées, les migrations de modèles à partir du logiciel vers matériel ou vice-versa peuvent être nécessaires, ce que

nous considérons par un surcoût lié ($M^s t_i$ et $M^h t_i$).

4.4.3 Optimisation des modèles avioniques

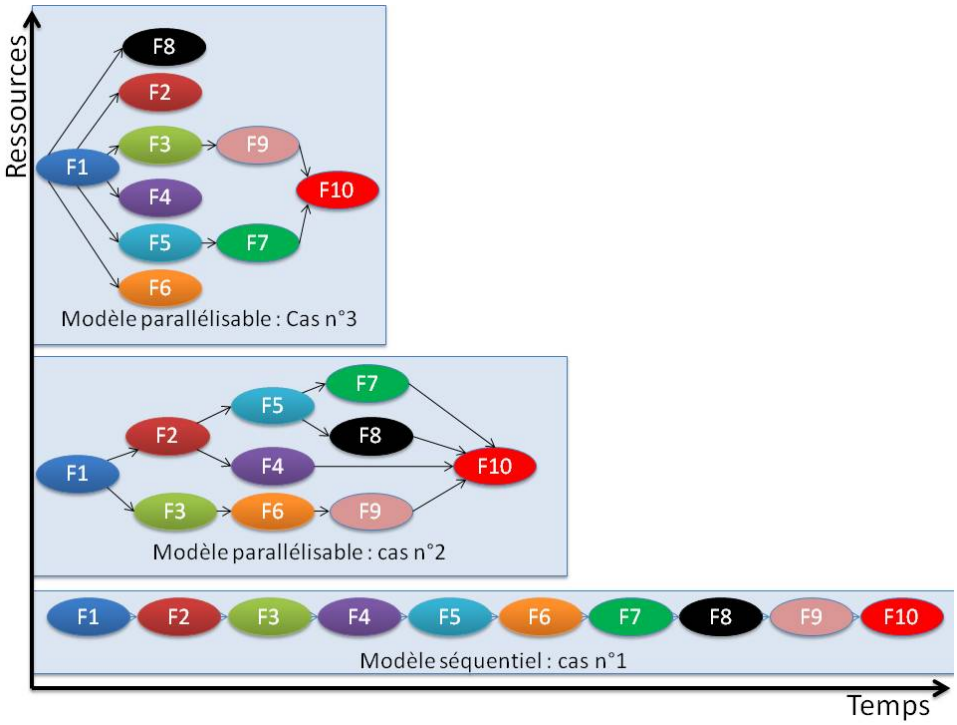

FIGURE 4.10 – Effets de la parallélisation d'un modèle avionique

Les optimisations logicielles d'un modèle de test et de simulation dépendront des propriétés de ce dernier. Comme nous pouvons le constater dans la figure 4.10, plusieurs cas sont possibles.
Si le résultat du profilage de ce modèle met en évidence un faible degré de parallélisation (cas numéro 1 sur la figure 4.10), une exécution purement séquentielle est justifiée nécessitant peu de ressources mais offrant un temps d'exécution non réductible à moins d'augmenter la fréquence de calcul.
Si le résultat du profilage met en évidence une possibilité de diminution du temps d'exécution du modèle par une parallélisation de ce dernier et ce, avec l'utilisation de peu de ressources de calcul, l'éclatement du modèle se fera par une exécution parallèle en plusieurs fonctions réparties sur les différents cœurs processeur (cas numéro 2 sur la figure 4.10). La principale difficulté réside alors dans la mise en place des communication inter-fonctions.

En effet, le gain obtenu par l'éclatement de l'application en plusieurs fonctions ne devra pas être annulé par un coût trop important des communications inter-fonctions.
Si le résultat du profilage met en évidence un haut degré de parallélisation (cas numéro 3 sur la figure 4.10), l'éclatement du modèle permettra une diminution du temps d'exécution en l'effectuant sur un composant matériel. Néanmoins, l'utilisation des ressources augmentera avec l'éclatement des fonctions.

La figure 4.10 met également en évidence l'influence des dépendances de données. En effet, nous pouvons remarquer que les dépendances de données entre les fonctions ont une influence sur leur implémentation et le temps d'exécution final une fois le modèle parallélisé. En effet, les dépendances de données inter-fonctions vont influer sur l'implémentation des fonctions au sein du modèle global et donc, de leur exécution parallèle les unes par rapport aux autres. Finalement, les architectures multi-cœur présentent une limite dans le cas d'un modèle de test et de simulation ayant un haut degré de parallélisation. En effet, le nombre limité d'unités de traitement limitera de ce fait la parallélisation du modèle. Dès lors, le développement d'une version matérielle est envisagée comme dans le cas numéro 3.

4.4.4 Profilage de modèles avioniques

Comme nous l'avons décrit précédemment, il est nécessaire de paralléliser les modèles de test et de simulation afin d'obtenir de meilleures performances sur les nœuds de calcul proposés. Pour ce faire, il est nécessaire de déterminer leur degré de parallélisation. Ci-dessous, nous proposons le profilage de plusieurs modèles et fonctions de test avioniques avec l'utilisation de l'outil Pareon décrit précédemment. Ce profilage nous permettra d'obtenir plusieurs résultats :

1. L'accélération possible par une parallélisation de l'application. Cette accélération dépendra elle même du nombre de nœuds de calculs disponibles.
2. Le nombre maximum de threads utilisables. En effet, comme le prévoient les lois sur le parallélisme, au delà d'un certain nombre dépendant de l'application, l'ajout de nœuds de calcul (ou de threads ici) n'aura aucun effet sur la réduction du temps d'exécution de l'application.
3. Le coût dans la création de tous les modèles et de leur synchronisation. En effet, la mise en place des mécanismes de création des modèles ainsi que leur synchronisation aura un temps, parfois non négligeable par rapport au gain obtenu par une parallélisation de l'application.

TABLE 4.2 – Analyse de modèles avec Pareon

Résultats	accélération	Max. threads utiles	Coût de la synchronisation
Model A	1.2	2	1%
Model B	1	1	0%
Model C	2.4	3	0%
Model D	3.5	6	39%
Model E	3	4	29%
Model F	486.7	10000	95%

L'analyse du modèle A (Modèle de mécanique du vol) se conclue sur une diminution du temps d'exécution du modèle avec un facteur 1,2 avec un coût de synchronisation faible et ce avec seulement deux fils d'exécution. Le modèle B (Modèle de test) offre un degré de parallélisation nul, dans ce cas, une exécution sur une architecture mono-cœur sera suffisante.
Les modèles C, D et E, tous les trois effectuant des calculs aérodynamiques spécifiques offrent un plus haut degré de parallélisation avec un coût de synchronisation nul pour le modèle C. Ces modèles sont appropriés à une implémentation parallèles sur architecture multi-cœur ou hybride multi-cœur CPU-FPGA par la mise en œuvre d'un fractionnement des modèles dans différentes fonctions exécutées sur des nœuds de calcul hétérogènes. Cela est possible en raison du faible coût de synchronisation.
Enfin, le modèle F montre un degré de parallélisation élevé avec une étape de synchronisation importante, ce modèle est très adapté pour une implémentation matérielle, dans la section 4.5, nous allons générer du code VHDL utilisant le cadre compilation ROCCC.

4.4.5 Développement pour une exécution matérielle des modèles avioniques

Le développement de la version matérielle d'un modèle de test avionique sera également différente selon les propriétés de ce dernier.
En effet, dans le cas d'un modèle présentant un fort degré de parallélisation, une implémentation purement matérielle tirant parti des propriétés du modèle permettra d'obtenir des performances supérieures aux versions logicielles malgré des fréquences de calcul inférieures. Pour ce faire, l'implémentation devra "éclater" le modèle en plusieurs entités de calcul indépendantes les unes des autres tout en tenant compte des phases de synchroni-

sation nécessaires au déroulement du calcul global ainsi que des dépendances de données internes aux modèles contraignant la mise en place de courtes entités séquentielles dans cette implémentation. De telles implémentations, pourront alors prendre la forme de structures pipeline permettant une exécution indépendante des différents étages. Ainsi, une telle implémentation permettra de réduire le temps d'exécution global à celui de l'étage le plus long.
Ensuite, dans le cas d'un modèle présentant un fort degré de parallélisation uniquement sur certaines de ces portions, une implémentation hétérogène logicielle-matérielle peut être envisagée si le coût de synchronisation n'est pas trop important. En effet, les composante FPGAs ne peuvent offrir des performances supérieures à celles de cœurs de calcul du fait de leur faible fréquence de fonctionnement. Alors, une implémentation mêlant les performances des deux nœuds de calcul permettra d'apporter des performances intéressantes. Dans le cadre d'une telle implémentation, l'utilisation d'un bus performant entre les nœuds de calcul est primordial. Dans le cas contraire, le gain obtenu sur les phases de calcul se verra annulé par le coût de la communication.
C'est en partant de ce constat que des sociétés telles que Xilinx proposent aujourd'hui une nouvelle génération de systèmes sur puce composée de deux cœurs de calcul ARM ayant un accès direct à une région reconfigurable. Ces EPP (Extensible Processing Platform) permettent alors de développer des applications mêlant calculs logiciels et matériels connectés par des interfaces rapides.
La figure 4.10 montre la dépendance entre le gain de performances et le niveau auquel nous placerons la granularité de notre analyse. En effet, nous nous étions placés précédemment au niveau fonction en divisant l'application à optimiser. Lors d'une implémentation sur composant FPGA, il sera possible d'augmenter le niveau de granularité de la parallélisation en éclatant au niveau des calculs le modèle. En effet, une application ne présentant pas des boucles avec peu ou pas de dépendances de données entre les itérations pourra présenter peu de dépendances entre les différents calculs effectués. Ainsi une implémentation parallèle des différents calculs apportera une diminution des temps d'exécution globaux de l'application.

4.5 Implémentation des modèles de test et de simulation

Comme cela l'a été décrit précédemment, le résultat du profilage des modèles legacy pourra si cela peut apporter de meilleurs temps d'exécution entrainer le développement

d'une nouvelle version de ce modèle qu'elle soit logicielle ou matérielle.

4.5.1 Implémentation parallèle d'un modèle logiciel de test et de simulation

Dans le cadre de la parallélisation purement logicielle, l'utilisation de fils d'exécutions communément appelés threads permet d'exécuter un ensemble d'instructions du langage machine sur un cœur du processeur utilisé . Du point de vue de l'utilisateur, l'exécution des différents threads peut sembler se faire en parallèle, néanmoins les threads partagent la mémoire virtuelle du processus qui leur est associé contrairement aux processus qui possèdent leur propre mémoire virtuelle. D'ailleurs, nous distinguons les threads du multi-processus du fait que ces derniers sont généralement indépendamment les uns des autres et peuvent interagir à travers une interface de programmation (ou API) fournie par le système telle que les communications inter-processus (ou IPC). De plus, les threads partagent une information sur l'état du processus qui leur est associé et les zones mémoires. De ce fait, il est peu coûteux d'effectuer la ré-allocation d'un thread.

Comprenant l'intérêt de l'utilisation de threads dans le cadre de développement d'applications parallèles, la société Intel a proposé son implémentation du Simultaneous Multi Threading (SMT) : Hyper-Threading. Cela consiste à implémenter deux processeurs logiques dotés de leurs registres de données et de contrôle ainsi que d'un contrôleur d'interruptions et ce, sur une seule puce. Ces deux processeurs partagent leur cache, le bus système permettant à l'utilisateur de lancer deux processus sur un même processeur dans le but d'utiliser au mieux les ressources du composant.
Finalement, cette implémentation permet une gestion simultanée de plusieurs fils d'exécutions, l'utilisateur pourra aisément exécuté ses programmes multi-threadés avec de meilleurs temps de réactions et de réponse que sur des architectures standard.

La programmation basée sur l'utilisation des threads reste complexe, en effet, la gestion des ressources partagées par le programme doit être gérée par ce dernier dans le but d'éviter des incohérences dans l'exécution globale du programme concerné. Il devient alors incontournable de mettre en place des mécanismes de communication dans le but de synchroniser les données provenant des différents fils d'exécutions dans le but de continuer l'exécution du programme global. Toutefois, il est nécessaire de tenir compte que ces synchronisations peuvent entrainer des situations de blocages des threads entre eux et donc du processus global. Nous étudierons dans la prochaine section les différentes solutions de communication inter-threads possibles dans le cadre de notre projet de thèse.

Finalement, l'utilisation de threads reste complexe, en effet, leur utilisation pourra aboutir à plusieurs bogues dus à l'implémentation de ces derniers mais nous permettra de paralléliser les modèles logiciels de test et de simulation.

4.5.2 Implémentation matérielle

Lorsqu'un modèle dont le profilage a démontré un fort degré de parallélisation, une implémentation purement matérielle est envisagée.

4.5.2.1 Implémentation hybride d'un modèle de test et de simulation

Dans un premier temps, afin de mettre en évidence les performances des IP Cores en comparaison à une exécution purement logicielle, nous avons implémenté notre mécanique de vol, modèle central des sessions de test et de simulation dans une version hétérogène. En effet, nous implémenterons une partie du code sur un softcore et l'autre sur des accélérateurs matériels. Nous avons effectués les tests qui suivent sur une Xilinx Virtex 4 (XC4VFX12) présente sur la carte de développement ML403.

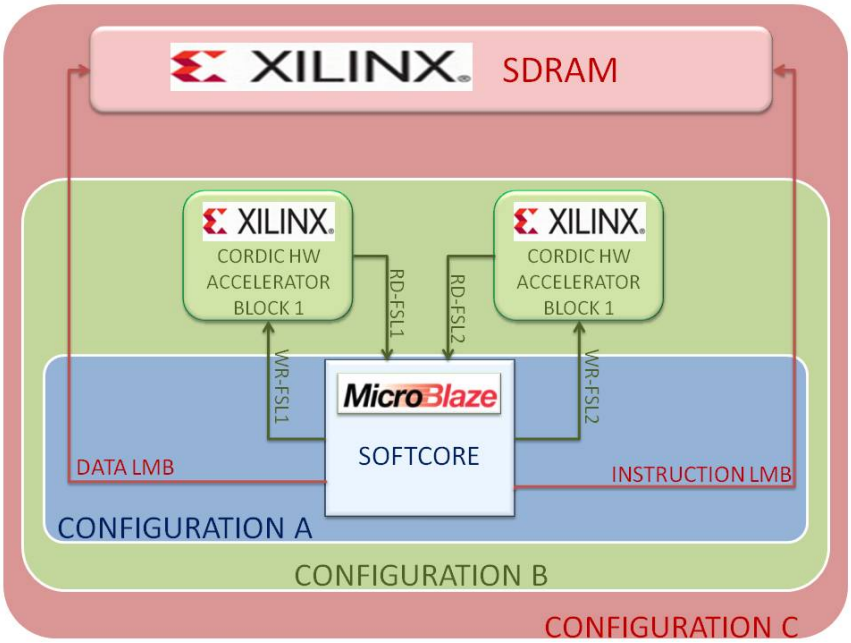

FIGURE 4.11 – Version hétérogène de la mécanique de vol

Comme cela est décrit dans la figure 4.11, nous avons réalisé trois implémentations

différentes de notre mécanique du vol afin de démontrer l'apport des accélérateurs matériels dans notre cas d'utilisation.

Le système de base représenté par la configuration A en bleu sur la figure, tous les blocs fonctionnels du modèle sont exécutés sur un processeur unique Microblaze cadencé à 100 MHz. Le processeur communique avec le mémoire BRAM (Block RAM) locale où les données locales et les instructions des applications sont stockées sur puce. Nous utilisons d'ailleurs également les caches de données et d'instructions dans cette configuration. Ensuite, dans la seconde configuration notée B et présente en vert sur la figure, nous utiliserons les accélérateurs CORDIC (Coordinate Rotation Digital Computer) reliés au Microblaze via le bus FSL (Fast Simplex Link) fourni par Xilinx. Le but ici est de pouvoir accélérer le temps d'exécution des fonctions trigonométriques majoritairement présentes sur le modèle de mécanique du vol. Dans la troisième configuration, notée C et présente en rouge sur la figure nous avons utilisé la mémoire SDRAM externe pour le stockage des données et des instructions.

L'objectif de cette étude sera d'obtenir différentes configurations avec différents compromis en termes de performance, d'occupation de ressources sur le composant FPGA.

Dans la configuration A (en bleu sur la figure), nous rappelons que toutes les fonctions sont exécutées sur un processeur unique Microblaze cadencé à 100 MHz ou les données locales et les instructions sont stockées sur la puce. Cette configuration nous délivre un temps d'exécution de 30 ms.

Dans la configuration B, en vert sur la figure qui fait appel à des accélérateurs matériels offre un temps d'exécution égal à 0,9 ms, trente fois inférieur à la configuration précédente. Néanmoins, cette configuration utilise énormément de ressources : 100% des bloc RAM disponibles et 86% des Slices disponibles.

La configuration C, en rouge sur la figure offre un bon compromis entre temps d'exécution et espace occupé sur le composant par rapport aux deux configurations précédentes. En effet, offrant un temps d'exécution de 0,94 ms, 13% des ressources Block RAM utilisées et 81% des Slices occupés, cette configuration sera la plus éligible à une possible implémentation. De plus, cette configuration a permis d'obtenir un temps d'exécution correct sans réécriture importante du code source.

4.5.2.2 Implémentation matérielle d'une fonction via la génération de code

Dans le but de mettre en avant les possibilités offertes par les générateurs code actuels, nous proposons l'évaluation de ROCCC dans le cas d'une fonction présentant un fort degré

de parallélisation. Notre choix s'est porté sur un un filtre à Réponse Impulsionnelle Finie (RIF) pour ses caractéristiques et sa parallélisation possible.

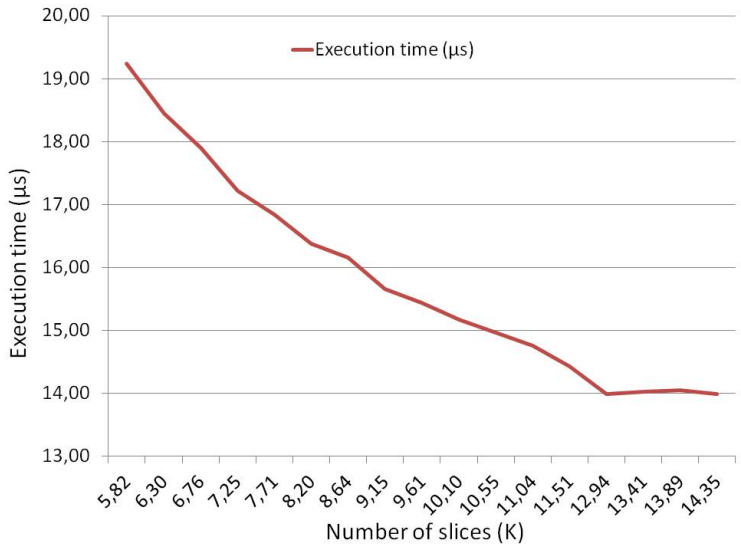

FIGURE 4.12 – Génération de code en fonction du déroulement d'une boucle

Avec la gestion du parallélisme intrinsèque à cette fonction, plusieurs compromis entre le temps d'exécution et la surface d'utilisation peuvent être obtenus. Pour ce faire, nous avons analysé cette fonction avec Pareon pour extraire les partitions qui peuvent bénéficier de la mise en œuvre parallèle et l'accélération matérielle. La boucle principale de la fonction RIF s'adapte parfaitement à une telle transformation. Avec l'aide de l'outil ROCCC, une étape de compilation de C à VHDL est effectuée. De plus, nous faisons profiter de la fonctionnalité de déroulage de boucles fournies par ROCCC afin d'obtenir des implémentations variées pour la boucle principale RIF. Nous avons généré 17 échantillons avec une parallélisation croissante de la boucle principale. Les résultats sont présentés dans la figure 6.15. Nous avons obtenu un temps d'exécution variant de 19 à 14 micro-secondes avec une utilisation de ressources allant de respectivement 5800 à 14300 Slices. En effet, plus nous paralléliserons la boucle principale RIF, plus nous réduirons le temps d'exécution et plus nous augmenterons l'espace occupé sur le composant matériel.

La génération de code VHDL présente encore aujourd'hui des limitations. En effet, malgré la prise en charge des codes utilisant des données à virgule flottantes dans certains outils tels que HLS (High Level Synthesis) [73] proposé par Xilinx, certaines fonctions mathématiques telles que les fonctions trigonométriques ne sont pas encore disponibles

limitant ainsi la portée d'utilisation des ces outils. Dès lors, une étape de développement manuelle est nécessaire.

4.5.2.3 Développement matériel d'un modèle de test et de simulation

Les résultats précédents ont mis en évidence les performances possibles d'une implémentation purement matérielle. De plus, une analyse poussée du code source du modèle de mécanique de vol a mis en évidence une faible dépendance de données entre plusieurs calculs composant ce modèle.
Afin de tirer profit de l'importante granularité du modèle concerné, nous avons conçu sa version matérielle comme un circuit électronique. En effet, la forte disponibilité d'IP Cores mathématiques compatibles prenant en charge des contraintes d'utilisation de virgule flottante nous a permis d'implémenter toutes les équations mécaniques en associant les composants les uns aux autres.

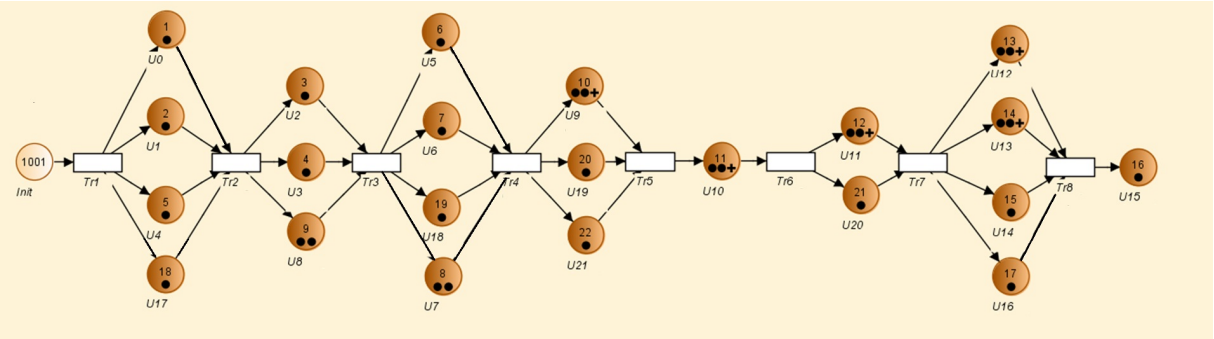

FIGURE 4.13 – Version matérielle de la mécanique de vol

Comme le montre la figure 4.13, nous avons implémenté ce modèle dans une structure en pipeline. Cette structure nous a permis de paralléliser les calculs à chaque étage composant cette implémentation.

De plus, cela nous permettra de réduire significativement le temps d'exécution final. En effet, le premier calcul fournira le délais le plus long. Ensuite, chaque livraison de résultat sera égal au temps de l'étage le plus long de cette structure. Ici, le temps de passage global est égal à 8 micro-secondes et le temps de propagation dans l'étage le plus long est égal à 2 micro-secondes. Cette implémentation offre alors un temps d'exécution supérieur à celui offert par un processeur standard.

4.6 Conclusion

Dans ce chapitre, nous avons mis en avant le support matériel des futurs systèmes de test et de simulation. Ce support matériel répond à toutes les contraintes imposées par le domaine d'application et l'industriel tant au niveau des performances attendues qu'en termes de flexibilité et de généricité. A cela, nous avons proposé une étude des bus de communication disponibles et répondant aux contraintes imposées par le domaine d'application. Enfin, nous avons mis en avant les optimisations nécessaires pour une implémentation et une exécution optimale des modèles de test et de simulation sur le support d'exécution proposé.
Dans le prochain chapitre, nous décrirons le support d'exécution logiciel proposé dans le cadre de ce projet de thèse pour la prochaine génération de systèmes de test et de simulation.

Chapitre 5

Un support logiciel pour la prochaine génération de systèmes de test et de simulation

5.1 Introduction

Ce chapitre présente le support logiciel respectant les contraintes temps-réel mou livré à Eurocopter pour la prochaine génération de systèmes de test et de simulation. Cet environnement embarque plusieurs fonctions telles que la surveillance des nœuds de calcul, le lancement et le placement de tâches sur architecture hétérogène dans le but de respecter les contraintes temps-réel imposées par le domaine d'application. Dans un premier temps, nous allons décrire l'architecture logicielle qui va analyser le comportement et l'état du système lors de la session de test et de simulation. Cette analyse devra répondre à plusieurs critères comme la rapidité, la légèreté et la généricité dans le but de répondre aux contraintes temps-réel requises par notre application. Nous introduirons l'heuristique de placement des tâches sur notre architecture hétérogène, cette dernière a été développée dans le cadre de ce projet de thèse. L'intérêt de ce type de solution a été motivé par le fait que le mauvais placement de tâches était la cause principale de débordements temporels lors des sessions de tests. Une méthode exacte a été proposée pour effectuer la répartition des tâches hors ligne. Par ailleurs, nous proposerons dans la suite de ce chapitre le concept de modèle avionique virtuel permettant un niveau d'abstraction suffisant pour réduire la complexité introduite par la gestion simultanée et dynamique des modèles logiciels et matériels.

Ce chapitre est structuré comme suit. Dans la section 5.2, nous décrirons l'architecture

logicielle proposée. Ensuite, dans la section 5.3 nous nous intéresserons aux heuristiques de placement des modèles sur les nœuds de calcul hétérogènes disponibles. Enfin, dans la section 5.4, nous décrirons les gestion des scénarios de test et de simulation via le concept de modèle avionique virtuel. Dans chacune des sections, nous mettrons en avant les résultats issus de l'implémentation des concepts proposés.

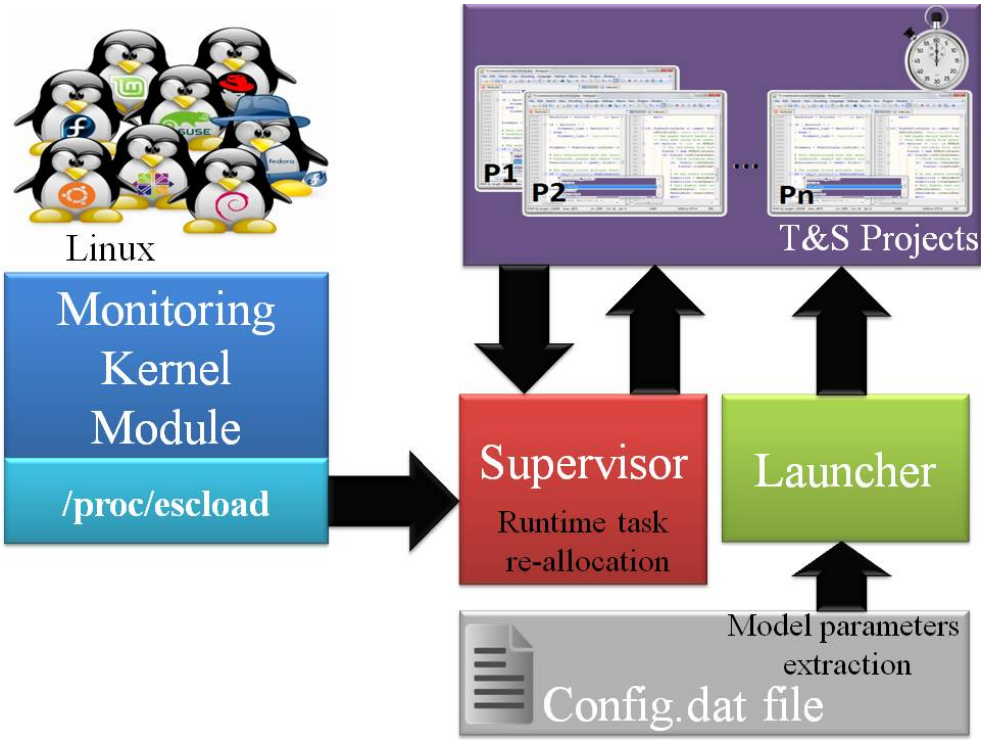

FIGURE 5.1 – Architecture logicielle proposée

5.2 Architecture logicielle proposée

Dans cette section, nous allons décrire les différentes fonctions présentées composant notre architecture logicielle proposée dans le cadre de cette thèse. Cette dernière répondra aux critères imposés par l'industriel tels que la généricité, l'évolutivité, la performance et le respect des contraintes temps-réel nécessaires au domaine d'application. Les fonctions assurées par notre environnement présentées dans la figure 5.1 sont les suivantes :

– **Le superviseur** : Cette fonction permet la ré-allocation dynamique des modèles composant les scénarios de test et de simulation. Elle sera sollicitée dès qu'un modèle attendra un pourcentage seuil de sa période temps-réel afin d'éviter tout débordement

temporel. Pour ce faire, une méthode approchée (heuristique ou méta-heuristique) sera intégrée à cette fonction afin d'apporter un placement efficace dans des délais très courts face aux périodes temps-réel imposées.

- **Le moniteur** : Cette fonction surveille les ressources de calcul présentes sur le système proposé, nombre de cœur(s), nombre de FPGA(s) ainsi que leur taux d'utilisation périodiquement.
- **Le lanceur** : Cette fonction assure le lancement d'une nouvelle session de test et de simulation ainsi que des modèles qui la composent. Le placement de ces modèles sera assuré par une méthode exacte afin d'exploiter efficacement les ressources disponibles.
- **Le simulateur** : Cette fonction assure l'ordonnancement de l'exécution de l'ensemble des modèles en se basant sur un graphe de tâches. Par ailleurs, elle permettra l'exécution périodique des boucles de test et de simulation décrites précédemment ainsi que la communication entre les modèles.

5.2.1 Fonction superviseur

La fonction de supervision représentée en rouge sur la figure 5.1 permettra d'effectuer le choix d'une nouvelle configuration de placement des tâches sur les unités de calculs. En effet, basée sur une heuristique de placement ayant accès aux caractéristiques des différentes versions de modèles disponibles, elle pourra lors d'une sollicitation provenant d'un acteur de l'environnement proposer une nouvelle allocation des tâches. Dans cette partie, nous proposerons deux heuristiques mises en place et testées lors de notre projet de thèse.

Pour la suite de ce chapitre, nous définissons deux seuils (seuil 1 et seuil 2) qui représentent un pourcentage de période temps-réel définie pour l'exécution des modèles de test et de simulation. L'atteinte du seuil 1 (ou S1) entrainera l'activation du superviseur par le modèle concerné afin d'effectuer une ré-allocation en ligne de ce modèle. L'atteinte du seuil 2 (ou S2) entrainera, si cela est possible, une ré-allocation hors ligne ou un arrêt de la simulation si les contraintes temps-réel ne peuvent plus être respectées dans la suite de la session. Nous définissons également L_{max}, la charge processeur maximale à ne pas dépasser dans la suite.

Le réseau de Pétri modélisant le superviseur est présentée en figure 5.2 ; ses places et transitions sont énumérées dans la table 5.1. Dès que ce dernier reçoit une demande de ré-allocation de modèle, il récupère les charges processeurs à cet instant donné via une lecture des informations fournies par le moniteur. Dès lors, l'heuristique est lancée et propose une nouvelle ré-allocation du modèle concerné selon les ressources hétérogènes disponibles. Dans

5. UN SUPPORT LOGICIEL POUR LA PROCHAINE GÉNÉRATION DE SYSTÈMES DE TEST ET DE SIMULATION

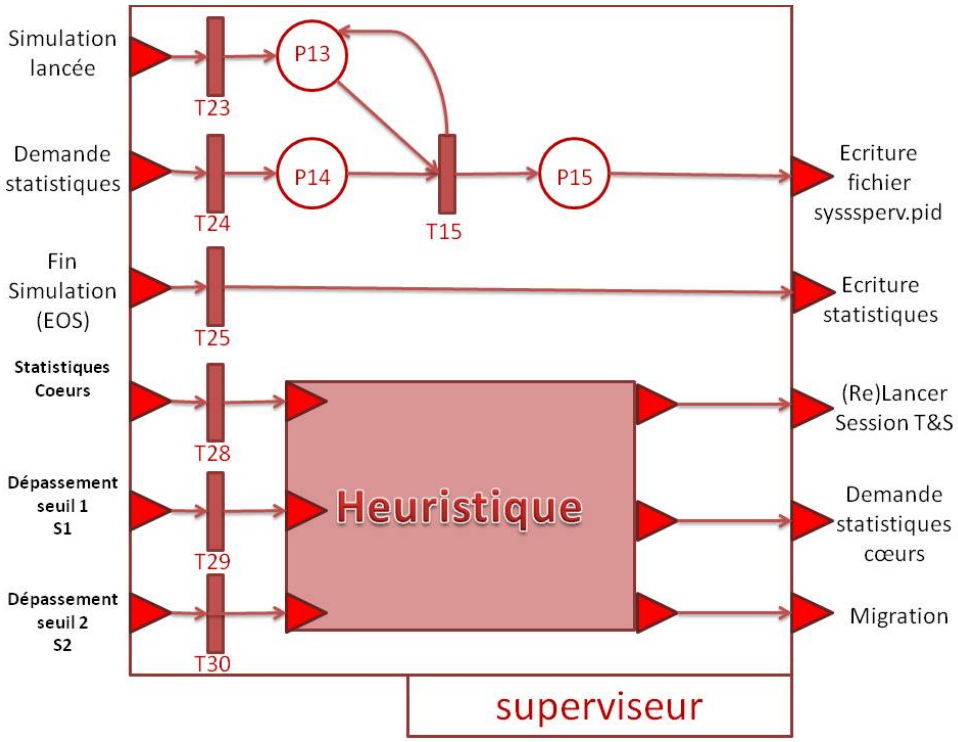

FIGURE 5.2 – Représentation du superviseur

TABLE 5.1 – Places et transitions du réseau de Pétri "Superviseur"

Nom	Description
P13	Architecture lancée et active, pas de demande de statistique en cours
P14	Demande de statistiques cœurs via un accès à */proc/escload* mis à jour par le moniteur
P15	Place de sortie, lecture des statistiques dans */proc/escload*.

Nom	Description
T15	Transition interne, bloque les demandes de statistiques cœurs si l'architecture logicielle n'est pas lancée
T23	Transition d'entrée, réception du signal indiquant que l'architecture logicielle est bien lancée
T24	Transition d'entrée, demande de calcul de statistiques cœurs
T25	Transition d'entrée, transmet l'information de fin de session de test et de simulation
T28	Transition d'entrée, réception des statistiques cœurs pour un traitement par l'heuristique
T29	Transition d'entrée, dépassement du seuil S1 signalé
T30	Transition d'entrée, dépassement du seuil S2 signalé

notre mode de fonctionnement actuel, le superviseur connait via le fichier de configuration les accélérateurs matériels disponibles et implantés sur les composants FPGAs. Toutefois, dans certains cas, lorsqu'aucune solution n'est envisageable, le superviseur enverra un ordre d'arrêt de la simulation et écrira les statistiques de comportement de la session afin de permettre une investigation ultérieure.

La complexité du superviseur réside dans l'heuristique embarquée. C'est elle qui décide de l'instant de calcul des statistiques. C'est elle qui provoque une migration de modèle(s) à l'issue d'un dépassement du seuil S1. Suite à un dépassement du seuil S2, c'est également elle qui pourra décider de relancer une simulation avec un nouveau fichier de configuration. Dès lors, le choix reviendra à l'utilisateur de relancer la simulation avec ce nouveau fichier de configuration.

5.2.2 Fonction Moniteur

Les systèmes d'exploitation actuels tels que Linux proposent des outils comme top ou htop [32] permettant de récupérer les charges des différents cœurs présents sur la machine avec des fréquences de rafraichissement de l'ordre de la seconde. Ainsi, ces solutions ne permettent pas d'obtenir des périodes de rafraichissement inférieures aux périodes minimales utilisées dans le domaine avionique (10 milli-secondes). Cela est dû en partie à la configuration standard des noyaux Linux dans leurs systèmes d'exploitation. Dès lors, le défit de développement de cette fonction réside dans sa fréquence de rafraichissement ainsi que dans sa légèreté en terme de consommation de ressources.

Pour l'implémentation globale de notre système, cette fonction sera assurée par un module noyau. Cette implémentation permet un calcul autonome de la charge des cœurs. Les éléments nécessaires pour le calcul de la charge sont mémorisés périodiquement dans une file circulaire. Cette mémorisation périodique est provoquée par un timer interne au moniteur dont la période égale à une milli-seconde est fixée par un paramètre "Pt". En effet, avec un noyau dont l'ordonnanceur est basé sur une fréquence de 1000 Hertz (par une modification de la fréquence de rafraichissement noyau) nous pourrons obtenir les charges des cœurs toutes les milli-secondes. Pour diminuer la charge (CPU) induite par le moniteur, il est possible d'augmenter la valeur Pt, au détriment de la vitesse de mise à jour des statistiques relatives à la charge.

Dans l'exemple de la figure 5.3, la taille de la file circulaire est Ne = 7. Les curseurs "Head" et "Tail" contiennent respectivement la valeur la plus récente V2 et la valeur la plus ancienne V1. À chaque mise à jour des valeurs via le timer interne du moniteur, les

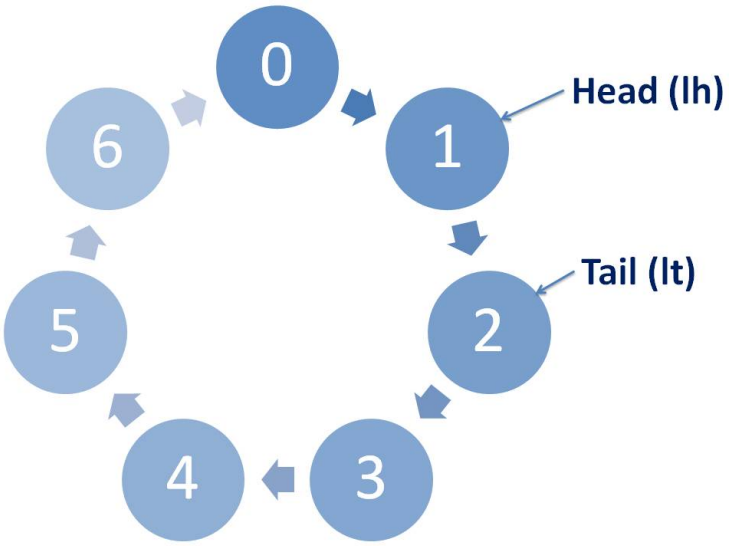

FIGURE 5.3 – Structure pipeline du calcul des charges processeur

curseurs sont "avancés" d'une case, modulo la taille de la file.

Néanmoins, l'utilisation d'un pipeline de mesure dans le moniteur provoque un effet de bord non désirable. En effet tant que la file circulaire n'est pas complètement alimentée, les mesures de charges retournées par le moniteur sont incorrectes. La solution proposée est de lancer le moniteur au moins $Ne * Pt/F$ secondes avant de lancer une simulation. Pour Ne = 7, Pt = 1 et F = 1000 Hertz, la phase d'initialisation dure 7 milli-secondes, ce qui est acceptable dans notre cadre de travail. Dès lors, nous pouvons envisager d'instancier le module moniteur dès le démarrage du système d'exploitation afin qu'il soit dans un état "stable" pour le lancement de sessions de test et de simulation.

À l'intérieur de cette fonction, le moniteur récupère la structure kstat_cpu(i), ainsi, la charge de chaque cœur est calculée à partir des valeurs de cpustat.user, cpustat.system et cpustat.nice. Le moniteur délivrera les charges processeur et les rendra disponibles sur le pseudo système de fichiers */proc/escload* permettant ainsi au superviseur d'accéder à ces données rapidement.

Le réseau de Pétri modélisant le moniteur est présenté en figure 5.4 ; ses places et transitions sont énumérées dans la table 5.2.

Dans sa version actuelle, sans préjuger des détails de son implémentation en tant que module noyau, le principe de fonctionnement du moniteur est décrit dans la figure 5.4 par un réseau de Pétri. Dès qu'il reçoit une demande de statistiques en entrée, le moniteur calcule ces statistiques et les rend disponibles au superviseur.

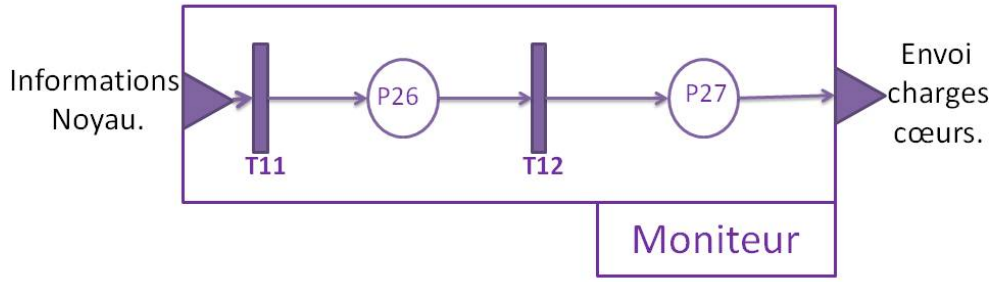

FIGURE 5.4 – Représentation du moniteur

TABLE 5.2 – Places et transitions du réseau de Pétri "Moniteur"

Nom	Description
P26	Réception des informations processeurs sur l'état des cœurs
P27	Place de sortie, mise à jour des informations sur les cœurs

Nom	Description
T11	Transition d'entrée, mise à jour des statistiques sur les cœurs
T12	Transition temporisé selon la durée de calcul des statistiques sur les cœurs par le moniteur

5.2.3 Fonction Lanceur

Dans un premier temps, il récupère les paramètres de lancement du modèle depuis le fichier de configuration. À partir de ces paramètres, il lance les modèles composant la session de test et de simulation. Le paramètre utilisé pour le moment est l'affectation c'est-à-dire à quel cœur ou FPGA nous affecterons le modèle. Le lanceur inscrit le modèle au niveau du superviseur en écrivant son pid (identifiant) dans un le fichier modele.pid. Lors de l'accès à ce fichier, le superviseur pourra identifier le modèle concerné. Dans l'implémentation du système, la fonction lanceur sera un processus Linux.
En plus du lancement des modèles, la fonction Lanceur génèrera également les moyens de communication entre les différents modèles. En effet, le graphe de tâches définissant la session spécifie également les dépendances de données entre les modèles.

5.2.4 Représentation de l'environnement global

La représentation globale modélisant le support proposé dans son ensemble est présenté en figure 5.5 ; ses places et transitions sont énumérées dans les tables présentées dans les sections précédentes.

Cette représentation globale est un assemblage des réseaux présentés dans les sections précédentes. Les liaisons entre les réseaux ont été colorées pour faciliter leur identification :

- Les liens colorés en rouge représentent les sorties du superviseur. Parmi celles-ci figurent les communications avec le lanceur, l'instruction de lancement d'une simulation ("(Re)lancer/stopper simulation" sur la figure 5.5) ordonnant le (re)lancement d'une simulation. Ensuite, une sortie ("CONF FILE" dans la figure 5.5) permet de garder une trace des reconfigurations effectuées lors de la session, vers un fichier. Une sortie ("STAT RQ" sur la figure 5.5) informe que cette fonction accède aux statistiques des cœurs disponibles dans */proc/escload*. La dernière sortie ("Migration" sur la figure 5.5) sera l'ordre de migration du modèle vers un autre nœud de calcul.
- Les liens colorés en vert représentent les sorties du bloc modèle. Parmi celles-ci figurent deux alertes de seuil d'atteinte de périodes temps-réel ("S1" et "S2" sur la figure 5.5). Une ordre sortie ("EOF" sur la figure 5.5) signalera l'arrêt de la simulation au superviseur.
- Le lien coloré en violet représente la sortie "Statistiques Cœurs" sur la figure 5.5 du bloc moniteur, ce dernier délivrera toutes les milli-secondes l'ensemble des charges des cœurs processeur disponibles.

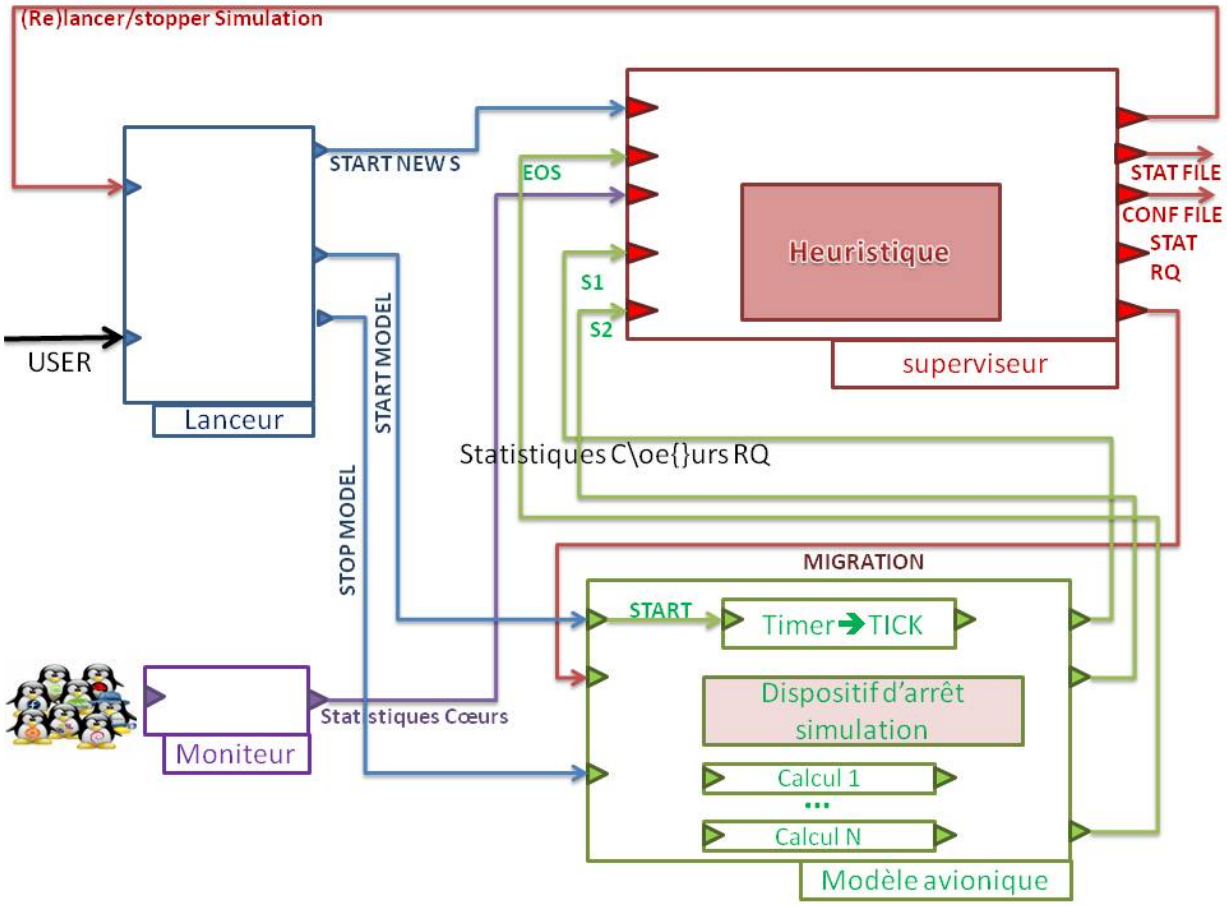

FIGURE 5.5 – Représentation globale de l'environnement

– Les liens colorés en bleu représentent les sorties du bloc lanceur. Parmi ces dernières figurent deux commandes du bloc modèle ("START MODEL" et "STOP MODEL" sur la figure 5.5) qui permettront respectivement de démarrer et d'arrêter le modèle. Ensuite, la sortie "START NEW S" permettra le lancement d'une nouvelle simulation par une requête au superviseur.
– Le lien coloré en noir sur la figure représente l'action utilisateur pour démarrer une nouvelle simulation (USER sur la figure 5.5)

Il est clair que ce réseau de Pétri permet d'illustrer aisément le fonctionnement du support logiciel proposé. Cependant une telle modélisation atteint rapidement ses limites lorsqu'il s'agit de considérer la gestion de plusieurs modèles avioniques. En effet, le cas où plusieurs modèles provoquent simultanément des dépassements demande un traitement particulier qu'il peut s'avérer complexe de modéliser sous forme de réseaux de Pétri.

5.3 Heuristique de placement

Dans cette section nous allons décrire l'élément essentiel de la fonction superviseur, l'heuristique de placement de tâches. Dans le cadre de notre système à contraintes temps-réel, le placement optimal en phase d'initialisation des projets de test est primordial. En effet, cette phase permettra de garantir le respect des contraintes temps-réel au lancement de la session de test et de simulation. Par ailleurs, cette heuristique aura pour but de ré-allouer les modèles afin de permettre le bon déroulement de la session. Notre objectif ici n'est pas d'obtenir la solution optimale mais d'apporter une solution efficace dans de courts délais afin de respecter les contraintes temps-réel imposées par le domaine d'application.

Lors de la phase d'initialisation, l'heuristique de placement aura deux buts. Premièrement, dans le but d'éviter les surcharges sur les cœurs de calcul, situations critiques dans le cadre d'applications à contraintes temps-réel. Le second objectif est de minimiser le coût, nous supposerons dans le cadre de cette étude que l'attribution des tâches au composant FPGA aura potentiellement un coût supérieur compte tenu du temps de reconfiguration et le coût des communications entre le processeur et le FPGA.

Comme cela est décrit dans la figure 5.6, nous définissons Z, la différence entre la charge maximale et la charge minimale sur l'ensemble des cœurs de calcul. Dans le but d'éviter les surcharges processeur, nous allouerons les modèles sur les cœurs disponibles afin de minimiser la valeur de Z. Dans le but de trouver le meilleur compromis, l'heuristique placera les tâches selon trois niveaux hiérarchiques :

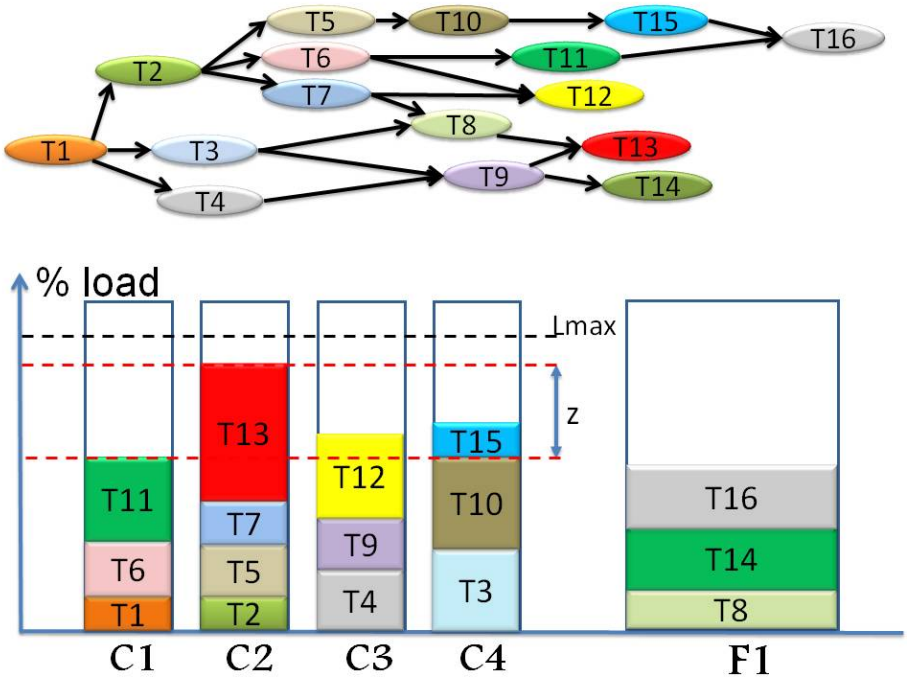

FIGURE 5.6 – Exemple d'allocation de modèles sur un système CPU-FPGA

– Option 1 : Placer la tâche entière sur les cœurs de calcul.
– Option 2 : Répartir l'exécution de la tâche sur le CPU et le FPGA.
– Option 3 : Placer la tâche entière sur le composant FPGA.

5.3.1 MLF

Dans un premier temps, nous avons développé une heuristique efficace afin de valider notre environnement : MLF (Minimum Loaded First). Comme sont nom l'indique, cette heuristique simple a été conçue pour affecter les modèles vers l'unité de traitement la moins chargée. Cette implémentation se justifie pour une validation avec peu de modèles.

Comme cela est prévu dans notre environnement, l'affectation initiale est lue à partir du fichier "config.dat" configuration. MLF nous a permis de valider notre environnement avec peu de modèles pour les essais préliminaires. La description de MLF nous contraint d'introduire certaines hypothèses et variables pour une meilleure compréhension de sa description.

Tout d'abord, nous définissons P, le nombre d'unités de traitement disponibles, N le nombre de cœurs processeurs disponibles et M, le nombre de composants FPGA connectés et utilisables. Ensuite, nous définissons W_k, la charge du modèle k supposée inconnue avant

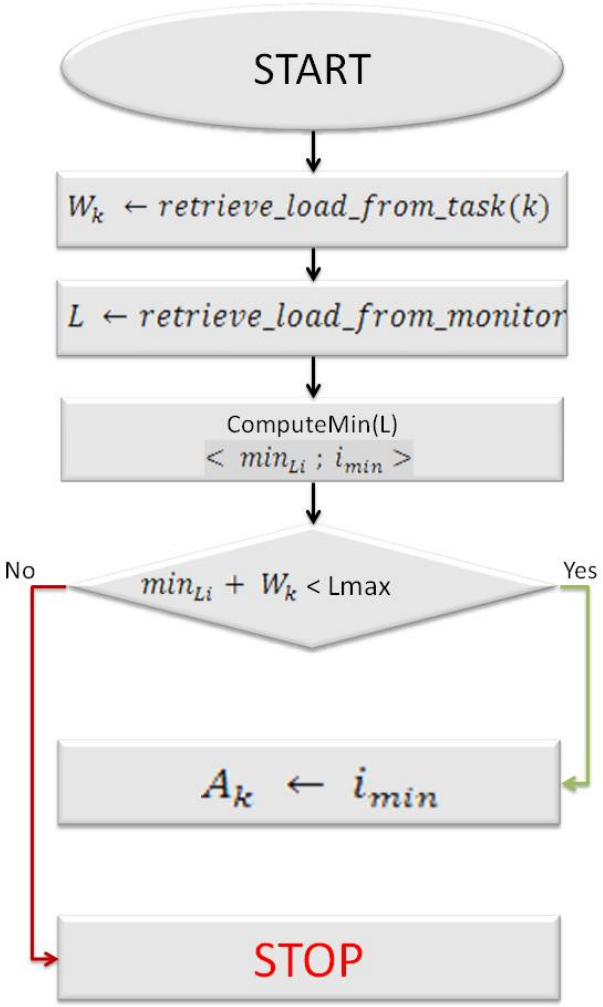

FIGURE 5.7 – L'heuristique MLF

le lancement de la session de test et de simulation et connue après la première période de la phase d'initialisation du système.

Nous définissons "i" comme étant l'index d'une unité de traitement tel que :
i=1 lorsque nous adressons le cœur 1,
i=2 lorsque nous adressons le cœur 2,
...
i=N lorsque nous adressons le cœur N (le dernier cœur disponible),
i=N+1 lorsque nous adressons le FPGA 1,
i=N+2 lorsque nous adressons le FPGA 2,
...
i=N+M lorsque nous adressons le FPGA M (le dernier composant FPGA disponible).

Nous définirons pour la suite L_i comme étant la charge de l'unité de traitement "i" exprimée en pourcentage. Ainsi $minL_i$ est la charge minimale détectée sur l'ensemble des unités de traitement disponibles et i_{min} est l'index de la première (dans l'ordre croissant de l'index "i") unité de traitement ayant cette charge $minL_i$. Enfin, T_k désigne le modèle k qui sera alloué à une unité de traitement et A_k son affectation. La figure 5.7 présente l'Heuristique MLF. Comme nous l'avons décrit précédemment, MLF attribuera à chaque modèle k l'unité de traitement la moins chargée à chaque fois qu'un modèle excédera un pourcentage (ici seuil 1) de sa période définie dans le scénario de test et de simulation.

5.3.2 MLPT

Dans cette seconde partie, nous allons décrire une heuristique de placement de modèles de test et de simulation. En guise d'exemple, nous nous placerons dans le cas d'un projet contenant plusieurs modèles (ou tâches) à placer hors et en cours de session sur notre architecture CPU-FPGA dynamiquement reconfigurable.

La figure 5.6 illustre cet exemple d'un projet composé de 16 tâches à exécuter sur une architecture à 4 cœurs CPU et un FPGA. Dans ce projet, les modèles sont exécutés de façon séquentielle en respectant le graphe de tâches représenté sur la figure 5.6. Dans l'environnement temps-réel requis, nous considérons que les tâches occupent au moins leur charge mesurée lors de la phase d'initialisation pendant une partie de la durée d'exécution du projet.

Dans le cadre de cet exemple, nous définissons C, l'ensemble des cœurs $[C1, C2, C3, C4]$ et F, l'ensemble des composants FPGA disponibles. Dans notre exemple, un seul compo-

sant FPGA est présent et sera noté [$F1$]. Ainsi, $CardC$, le cardinal de C sera égal à quatre et $CardF$, le cardinal de F sera égal à 1. Nous allons définir S1 et S2 comme étant respectivement les seuils 1 et 2.

5.3.2.1 Heuristique MLPT

L'Heuristique LPT est l'une des anciennes méthodes pour l'allocation de tâches. Déjà en 1969, Graham [47] livra des résultats concernant l'utilisation de cette heuristique sur l'ordonnancement de tâches sur des machines parallèles. L'utilisation des règles LPT et le TCLSF (version à jour de la LPT) est encore très répandue. D'ailleurs, comme nous l'avons déjà évoqué dans l'état de l'art, en 2008, Koulamas et Kyparisis [20] ont démontré la robustesse de cette approche pour l'ordonnancement de deux machines identiques parallèles.

Dans le cadre de notre projet, nous utilisons l'Heuristique LPT avec les options 1, 2 et 3 afin de garder la même stratégie que dans la méthode exacte évoquée précédemment.

L'algorithme proposé est présenté dans la figure 1. Cette heuristique s'effectue selon trois étapes principales. La première étape chargera l'ensemble des tâches composant l'application. Ensuite, une étape de "pré-calcul" permettra de trier toutes ces tâches selon leur charge sur les cœurs processeur p_1(i,j) selon un ordre décroissant. Enfin, l'Heuristique LPT allouera chaque tâche dans l'ordre définit précédemment au cœur de calcul le moins chargé en respectant la limite définie du seuil 2. Les n1 tâches seront affectées lors de l'étape 1 (notée STEP 1). Pour les n-n1 tâches restantes, l'algorithme continuera l'allocation dans l'étape 2 (notée STEP 2) mais cette fois-ci avec l'option 2. Finalement, dans l'étape 3, nous assignerons les tâches restantes au composant matériel.

5.3.3 Reconfiguration causée par l'augmentation de charge de tâches

Dans le cas de la ré-allocation d'une tâche due à une surcharge d'un processeur, deux scénarios sont possibles :

Dans le premier scénario décrit dans la figure 5.8, une surcharge intervient dans le cœur le moins chargé. Dans ce cas, nous ré-allouerons la tâche ayant provoqué cette surcharge vers le composant FPGA si cela est possible.

Le second scénario décrit dans la figure 5.8 implique une surcharge n'intervenant pas sur le cœur le moins chargé comme précédemment. Dans le cadre de l'étude de scénario, nous

Algorithm 1 Algorithm MLPT

(1) Initialize T the list of tasks of the project P_1.
(2) Peprocessing step : Rank the N Tasks of "T" in the order of decreasing CPU occupancy rate P_1(i,j).
(3) STEP 1 :
while (**do**
$(L_{c1}<L_{max})||(L_{c2}<L_{max})||(L_{c3}<L_{max})||(L_{c4}<L_{max}))$
for each task i in N **do**
if (**then**
$>L_{max}$) store T_{N-i} on new list "T_{S1}" ; N1 ++
else
assign T_{N-i} to the least loaded core under option 1
end if
end for
end while
store the remaining tasks on the list "T_{S1}" with N1, the size of T_{S1}
(4) STEP 2 :
while (**do**
$(L_{c1}<L_{max})||(L_{c2}<L_{max})||(L_{c3}<L_{max})||(L_{c4}<L_{max}))$
for each task i in N1) **do** split T_{N-i} between the least loaded core and the FPGA under option 2 store the remaining tasks on the list "T_{S2}" with N2, the size of T_{S2}
end for
end while
(5) STEP 3 : store the N2 tasks on the FPGA under option 3

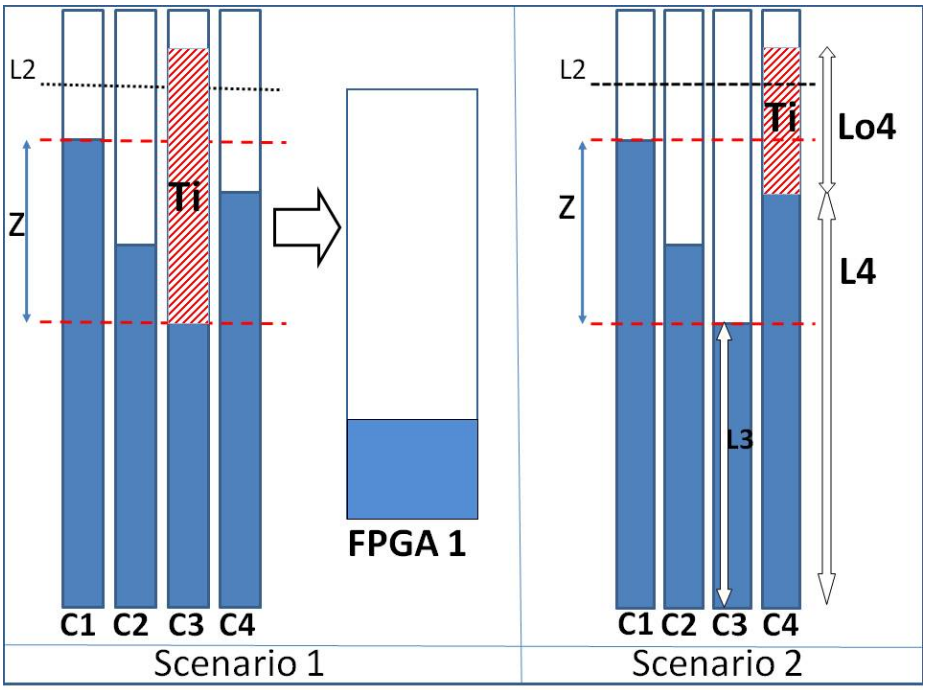

FIGURE 5.8 – Scénarios 1 et 2

supposerons que la tâche Ti localisée sur le cœur C4 est responsable de cette surcharge. Nous notons L3, la charge du cœur le moins chargé (ici C3) et L4 et Lo4 respectivement la charge initiale et la valeur de la surcharge sur le cœur C4. Finalement, nous considèrerons que la charge initiale de Ti sera de 20%.
Nous procéderons ainsi :
Si "L3 + Lo4 + 20% $\leq L_{max}$%" nous assignerons la tâche "Ti" au cœur "C3". Sinon, nous assignerons "Ti" au composant matériel si cela est possible bien sûr.

5.3.4 Perspectives d'améliorations

Dans une optique d'amélioration, nous pourrons alimenter une méthode exacte avec des résultats de profilage du coût de la communication entre les nœuds hétérogènes du support matériel proposé.

En effet, les heuristiques décrites précédemment présentent des limitations au niveau de la précision des coûts de communication. Pour ce faire, il est nécessaire de profiler le support matériel ainsi que les mécanismes mis en place dans le cadre de l'application.
Si l'on reprend le profilage de médiums de communication effectué dans le chapitre décrivant le support matériel, nous pouvons déduire des équations qui nous permettrons d'estimer

le coût de la commutation.

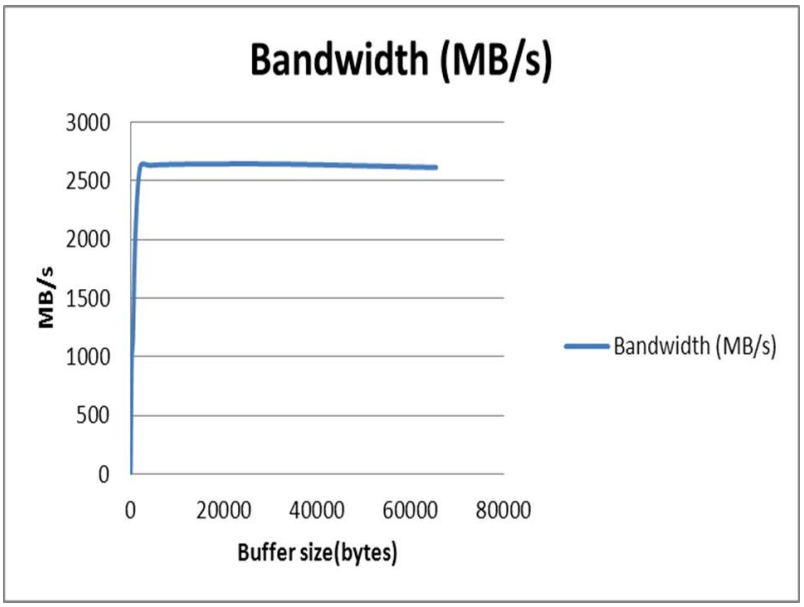

FIGURE 5.9 – PCIe - Test en performances

La figure 5.9 présente le débit du bus PCIe en fonction de la taille de buffer. De cette courbe, nous pouvons déduire le débit en fonction de la taille de buffer. Ici, nous pouvons observer que pour des tailles supérieures à 2 Kilo-Octets (ce qui est le cas dans notre application) sera égale à 2,5 Giga-Octets par seconde.

La figure 5.10 présente la latence introduite par le bus PCIe. Nous pouvons en déduire l'équation 5.1 avec y la latence et x la taille du buffer. Cette équation présente une précision inférieure à 10% par rapport au cas réel.

$$y = 0,0004x + 0,1358 \tag{5.1}$$

Comme nous l'avons démontré, le profilage des moyens de communication permettra de développer une méthode exacte adaptée au système global proposé dans le cadre de cette thèse. En effet, les approximations imposées par l'utilisation d'heuristiques "standard" pourront être éliminées par un profilage très précis de l'architecture finale déployée chez Eurocopter. D'ailleurs, conscients de cette problématique, nous avons démarré un autre projet en parallèle du notre afin de répondre à ce type de problématiques [58].

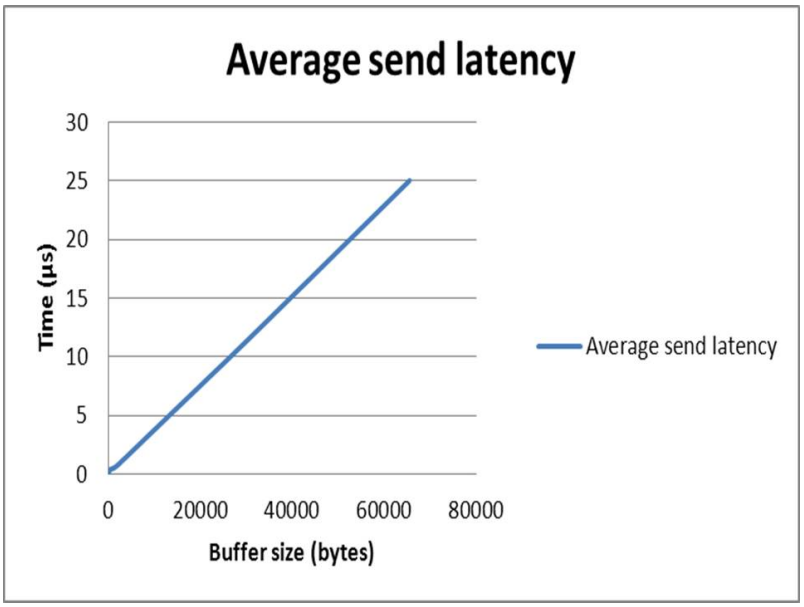

FIGURE 5.10 – PCIe - Test des latences

5.4 Gestion des scénarios de test et de simulation

5.4.1 Représentation des modèles avioniques

Le réseau de Pétri modélisant un modèle avionique est présenté en figure 5.11 ; ses places et transitions sont énumérées dans la table 5.3. Le modèle avionique est principalement organisé autour d'une boucle (finie) de calcul, modélisée par le cycle P20, T17, P17 et T19. Le nombre d'itérations est contrôlé par la place P19 qui se comporte comme un "réservoir de jetons". Le nombre de jetons initialement présents en cette place détermine le nombre d'itérations de calcul effectuées par le modèle avionique avant arrêt automatique de ce dernier. Pour obtenir une boucle infinie, il suffit de supprimer l'arc P19 → T19, voire de supprimer P19.

La place P17 joue un rôle essentiel dans ce réseau de Pétri. Elle correspond à la fin d'exécution du code de calcul du modèle avionique (*i.e.* une itération). Plusieurs arcs "sortent" de cette place, ce qui signifie qu'un tirage aléatoire détermine la prochaine transition à franchir (T19, T20 ou T21). Ceci permet de simuler – respectivement – aucun débordement temporel, dépassement du seuil 1 ou dépassement du seuil 2. Par défaut, la probabilité de franchir l'une des trois transitions (T19, T20 ou T21) est de l'ordre d'un tiers (équi-probabilité).

TABLE 5.3 – Places et transitions du réseau de Pétri "Modèle avionique"

Nom	Description
P17	Fin d'exécution du code de calcul du modèle avionique, un tirage aléatoire détermine la prochaine transition à franchir (T19, T20 ou T21) pour simuler – respectivement – aucun débordement temporel, dépassement du seuil 1 ou dépassement du seuil 2
P19	Réservoir de jetons. Le nombre de jetons initialement présents en cette place détermine le nombre d'itérations de calcul effectuées par le modèle avionique avant arrêt automatique de ce dernier.
P20	Modèle avionique en cours de calcul
P21	Demande d'arrêt du modèle avionique en cours de traitement
P22	Place de sortie, le modèle signale un dépassement du temps de calcul du seuil 1
P23	Place de sortie, le modèle signale un dépassement du temps de calcul du seuil 2
P34	Non utilisée dans cette version
P38	Place de sortie, écriture des statistiques calculées par le modèle

Nom	Description
T7	Transition d'entrée, demande de lancement du modèle
T8	Transition d'entrée, demande de migration du modèle
T9	Transition d'entrée, demande d'arrêt du modèle
T17	Transition temporisée selon la durée de calcul d'une itération du modèle avionique
T19	Transition interne, bloque le modèle lorsque le nombre d'itérations attendu (déterminé par le nombre de jetons présents dans la place P19) a été effectué
T20	Transition temporisée selon la fraction de temps à ajouter à la durée normale d'une itération du modèle avionique (*cf.* T17) pour correspondre à un dépassement du seuil 1 de la durée normale
T21	Transition temporisée selon la fraction de temps à ajouter à la durée normale d'une itération du modèle avionique (*cf.* T17) pour correspondre à un dépassement du seuil 2 de la durée normale

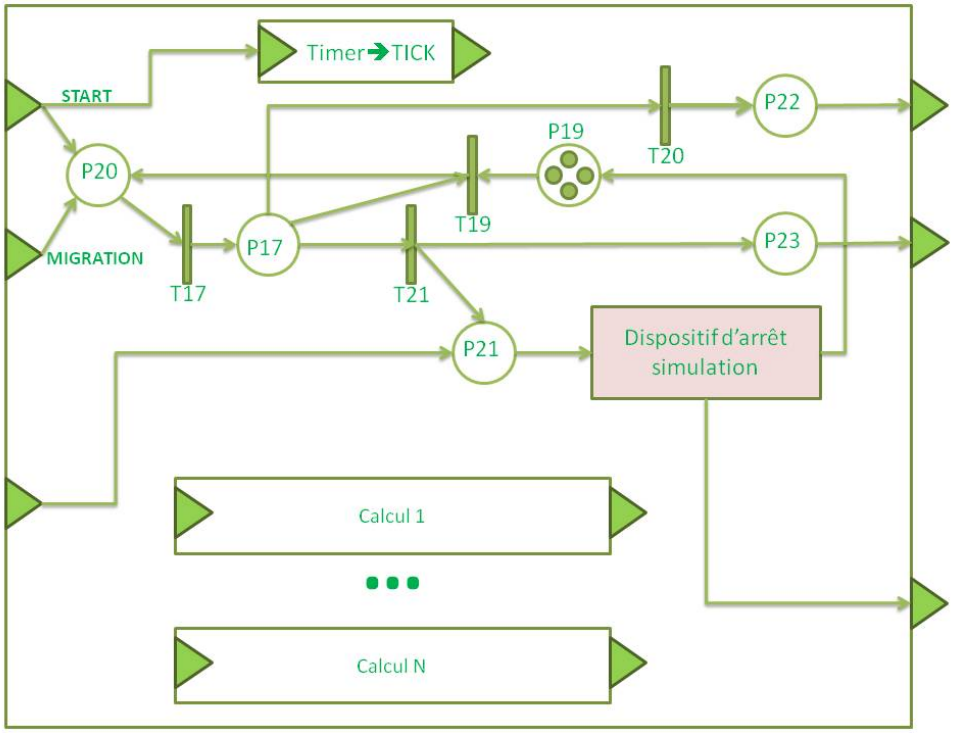

FIGURE 5.11 – Modèle avionique

Dans tous les cas, la durée effective d'une itération du modèle avionique correspond à la durée de T17 lorsqu'il n'y a pas de débordement. Elle correspond au cumul des durées des transitions T17 et T20 en cas de dépassement du seuil 1. Elle correspond au cumul des durées des transitions T17 et T21 en cas de dépassement du seuil 2.

La durée de calcul d'une itération du modèle avionique est déterminée par la durée de la transition temporisé T17. Cette durée est fixe dans l'exemple fourni, mais il est possible de la faire varier selon une loi déterminée ou un fichier contenant un historique (*i.e.* une trace issue d'un autre simulateur ou de mesures effectuées en situation réelle) en utilisant par exemple le simulateur VLE [46].

La "place de sortie" P38 correspond à l'écriture des statistiques calculées par le modèle. Dans la version présentée (en mode mise au point) les statistiques sont calculées à l'issue de chaque itération (*i.e.* phase de calcul) du modèle avionique. Dans d'autres versions, les statistiques sont écrites uniquement à l'issue de la première itération, plusieurs scénarios sont alors envisageables. La place P34 peut être utilisée pour modifier ce comportement. Dans une seconde version, P34 comporte un jeton pour demander une seule fois les statistiques du modèle après le premier calcul (voir place P38).

5.4.2 Modèles virtuels de test et de simulation

Dans le but de rendre l'implémentation de modèles plus simple depuis les outils de test et de simulation ainsi que de limiter les interlocuteurs vers le superviseur, nous proposons le concept de modèles virtuels comme cela est décrit sur la figure 5.12.

En effet, ce modèle sera composé de plusieurs fonctions représentant chacune une version spécifique d'un modèle de test et de simulation. Ainsi, le superviseur n'aura qu'un "interlocuteur" pour une pluralité de versions. De plus, cette implémentation permettra également de partager plus efficacement les données de contexte du modèle dans le but d'effectuer les ré-allocations plus efficacement.
Comme nous pouvons l'observer dans l'exemple présenté sur la figure 5.12, plusieurs implémentations sont représentées par des fonctions partageant le contexte d'exécution du modèle. Nous définissons par "contexte", l'ensemble des données propres au modèle nécessaires pour reproduire une exécution identique, c'est à dire un même jeu d'entrée produisant systématiquement les mêmes sorties.

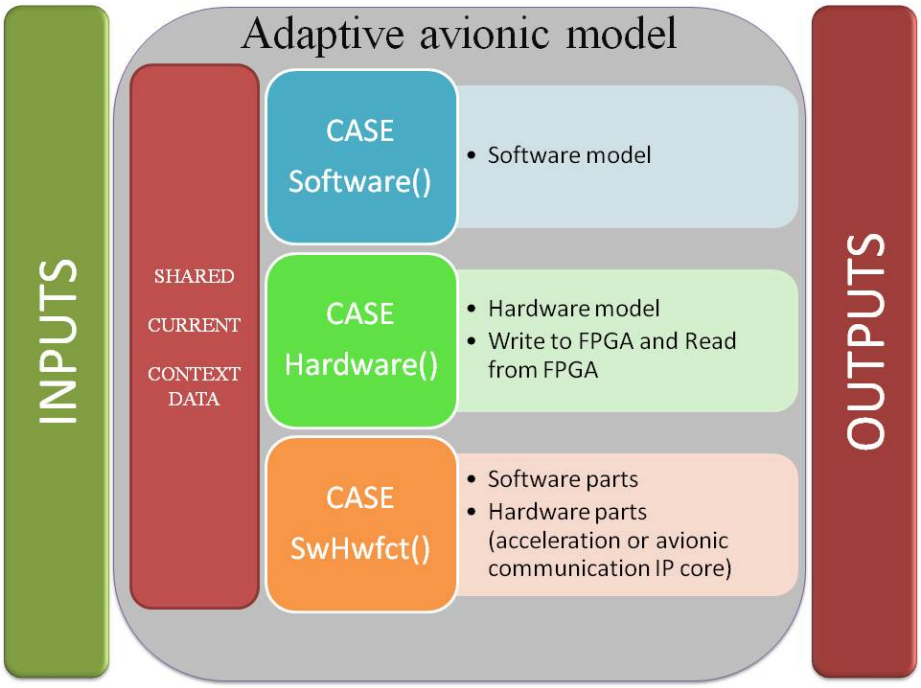

FIGURE 5.12 – Modèle de test et de simulation virtuel

Finalement, cette proposition permettra aux outils de test et de simulation de gérer les modèles composant les scénarios depuis leurs interfaces d'entrées-sorties. Ainsi, ce niveau d'abstraction permettra aux utilisateurs des outils de ne pas être impliqués dans l'allocation

statique et dynamique des différents modèles sur les unités de traitement disponibles.

5.4.3 Ré-allocation dynamique des modèles de test et de simulation

L'implémentation précédente permet également de faciliter les étapes de ré-allocation dynamique des tâches. En effet, les données de contexte étant globales au modèle virtuel, chacune des fonctions aura accès à ces dernières.
En effet, comme nous l'avons décrit précédemment, les modèles avertiront le superviseur dans le cas où son temps d'exécution attendra un pourcentage défini de leur période attribuée plusieurs cas se présenteront :

- Si le superviseur conclue que l'un des cœurs processeur peut prendre en charge l'exécution du modèle que ce soit en terme de temps d'exécution qu'en terme de "charge disponible". Le superviseur enverra au modèle concerné sa nouvelle allocation et ce dernier continuera de façon transparente son exécution sur l'unité de calcul qui lui a été attribué. Dans la figure 5.12, cette exécution sera assurée par la fonction **Software()**.
- Si le superviseur conclue qu'aucun des cœurs processeur ne peut prendre en charge l'exécution du modèle et que ce dernier possède une version matérielle présentant un temps d'exécution suffisant pour le bon déroulement de la session, le choix d'une implémentation matérielle sera effectué. Dès lors, l'implémentation des modèles avioniques proposée précédemment prend tout son intérêt. En effet, le partage des données de contexte permettra au modèle virtuel de les transmettre à l'accélérateur matériel afin qu'il puisse continuer l'exécution du scénario en toute transparence pour l'outil. Dans ce cas, ce sera la fonction **Hardware()** qui sera exécutée par le modèle virtuel permettant une fois de plus de limiter les "interlocuteurs" avec les outils de test et les fonctions liées à l'architecture de supervision.
- Un autre cas similaire au précédent est envisageable, en effet, comme nous l'avons décrit précédemment, certains modèles pourront être développés de façon à avoir des parties logicielles localisées sur les cœurs de la machine hôte et des parties matérielles exécutées par les accélérateurs matériels disponibles. Ainsi, au sein de la fonction **HwSwfct()** (sur la figure 5.12) il y aura des envoi(s)-réception(s) de données vers l'accélérateur matériel.
- Si le superviseur ne parvient pas à trouver une solution convenable, la session de test et de simulation est interrompue, une trace de l'exécution sera générée permettant

ainsi de pouvoir définir ce qui a pu entraîner cet échec.

5.5 Conclusion

Dans ce chapitre, nous avons présenté le support logiciel des futurs systèmes de test et de simulation proposé. Ce support logiciel permettra de répondre aux contraintes imposées par le domaine d'application et d'apporter des capacités hautes performances. De plus, ce support logiciel permettra d'exploiter au mieux le potentiel de performances offert par le support matériel décrit dans le chapitre précédent.
Dans le prochain chapitre, nous mettrons en avant les performances du système proposé dans le cadre de ce projet pour la prochaine génération de systèmes de test et de simulation. De plus, nous proposerons ce système en tant que support d'applications qui pourra être déployé dans d'autres domaines que l'aéronautique.

5. UN SUPPORT LOGICIEL POUR LA PROCHAINE GÉNÉRATION DE SYSTÈMES DE TEST ET DE SIMULATION

Chapitre 6

Étude de cas et validation expérimentale

6.1 Introduction

Ce chapitre offre l'occasion au lecteur d'avoir une idée plus précise des performances offertes par le nouveau système de test et de simulation. Dans la section 6.2, nous allons décrire la configuration matérielle utilisée dans le cadre de ce chapitre expérimental. Ensuite, dans la section 6.3, nous allons décrire la configuration logicielle ainsi que les optimisations de l'environnement Linux standard effectuées afin d'obtenir les performances attendues et le respect des contraintes imposées par le domaine d'application. Ensuite, dans la section 6.4, nous allons définir le scénario mis en place pour la validation de notre environnement ainsi que les résultats liés à l'exécution de ce scénario. Dans la section 6.5, nous proposerons l'utilisation de notre système dans plusieurs cadres d'applications autres que le test et la simulation avionique.

6.2 Configuration matérielle

Avant de présenter les résultats dans la suite de ce chapitre expérimental, nous introduisons le support matériel et logiciel utilisés afin de valider notre système. Ici, le support matériel utilisé est composé de deux processeurs Intel 5520 embarquant en tout huit cœurs cadencés à 2.26 Giga-Hertz, de 16 Giga-Octet de mémoire RAM DDR3 et d'une carte de développement ML605 [75] de chez Xilinx embarquant un Virtex 6 reliée par un lien PCIe Gen 2 (x4) comme cela est présenté sur la figure 6.1.

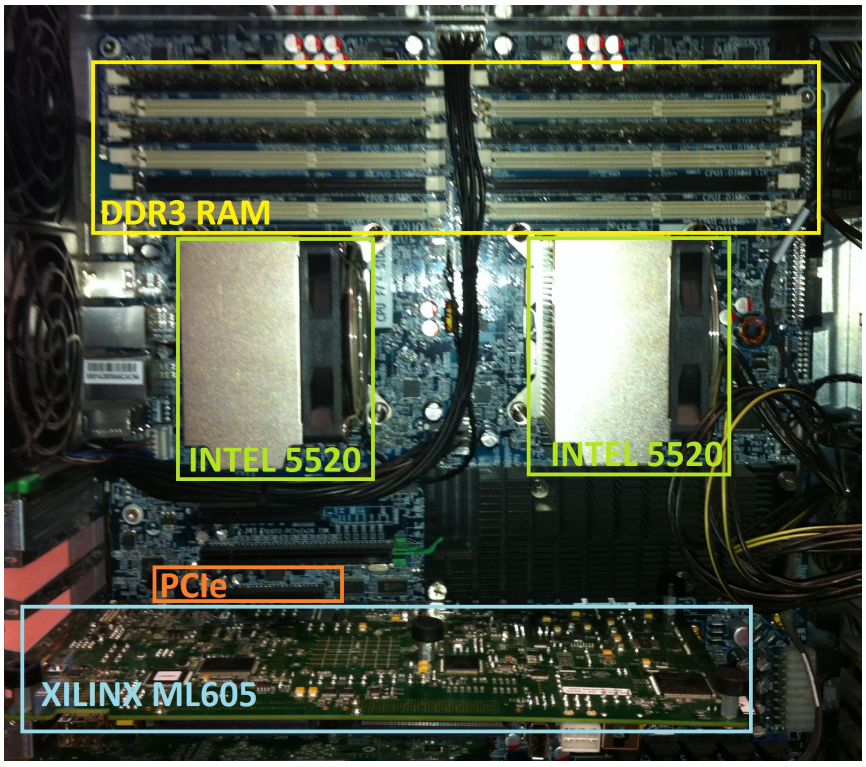

FIGURE 6.1 – Configuration matérielle

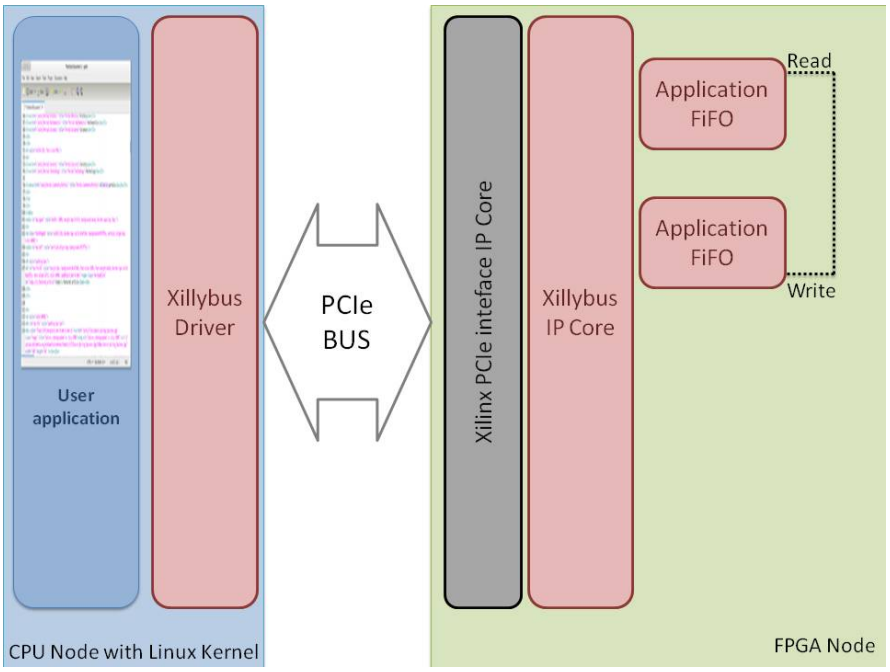

FIGURE 6.2 – Test des latences entre les modèles logiciels et matériels

Une première étape de validation consiste à vérifier que la communication entre les modèles logiciels et matériels fonctionne. Pour ce faire, nous avons implanté la solution Xillybus et nous avons mesuré la durée d'un aller-retour de données depuis l'environnement Linux. Pour ce faire, nous avons "rebouclé" les FIFOs de lecture et d'écriture au sein du composant matériel. Ainsi, nous pourrons mesurer effectivement le passage des données sur le bus PCIe ainsi que sur les deux FIFOs d'écriture et de lecture sur le composant matériel. Par ailleurs, ces mesures vont nous permettre de caractériser ce moyen de communication afin d'enrichir notre fichier de configuration de session de test permettant ainsi au superviseur de prendre en considération les latences correspondantes.

Du coté de l'application logicielle, nous avons utilisé les fonctions **write()** et **read()** dans un programme en langage C mesurant ces latences comme cela est montré dans le pseudo code 6.1. En guise de conclusion, la simplicité de la solution utilisée permettra ainsi une abstraction et un prototypage rapide de la communication hétérogène CPU-FPGA.

```
1 // Send data to ML605
2 // bufwr : Write buffer
3 IO_WriteAllToFPGA(fdrdwr, bufwr, sizeof(bufwr));
4
```

```
5  // For write performance measure
6  #ifdef ESM_FPGADETAILEDSTATS
7    SU_endperiod(&sptrwfpgatime);
8    SU_addperiod(&sudtwfpgatime, &sptrwfpgatime);
9  #endif
10
11 // Let's recieve some data from the ML605
12 // bufrd : read buffer
13 IO_ReadAllFromFPGA(fdrdwr, bufrd, sizeof(bufrd), &rc);
14
15 // For read performance measure
16 #ifdef ESM_FPGADETAILEDSTATS
17 SU_endperiodcons(&sptrwfpgatime);
18 SU_addperiod(&sudtrfpgatime, &sptrwfpgatime);
19 #endif
```

Listing 6.1 – Communication PCIe et mesure du coût

Le modèle de mécanique de vol implanté sur le composant matériel nécessite l'envoi et la réception de 33 données réelles sur 32 bits comprenant les valeurs d'entrées-sorties ainsi que les données de contexte. Le temps d'écriture offert par la solution Xillybus est de 14 micro-secondes pour l'écriture et de 11 micro-secondes pour la lecture. Ce coût est très négligeable face à la période temps-réel minimale de 10 milli-secondes requise. De plus, le temps d'exécution de l'accélérateur matériel est fixe comme nous l'avons décrit précédemment. Ainsi, un passage d'un modèle logiciel vers un modèle matériel ne permettra peut être pas un gain en performances mais permettra un respect de la contrainte temps-réel.

Dans une prochaine version, il sera possible, afin de réduire encore les coûts de communication, de mettre en place un protocole de communication qui permettra d'initialiser les modèles avec les valeurs de contexte et de ne communiquer que les valeurs d'entrées-sorties par la suite.

Dans la prochaine section, nous allons décrire la configuration logicielle de l'environnement proposé dans le chapitre précédent. Ainsi, le lecteur pourra comprendre l'ensemble des étapes effectuées qui ont permis l'obtention des résultats présentés dans ce chapitre.

6.3 Configuration logicielle

Notre environnement logiciel est basé sur un système d'exploitation Linux avec une version 2.6 du noyau. Dans le but de respecter les contraintes logicielles imposées par le domaine d'application, plusieurs optimisations doivent être apportées. Nous nous intéresserons dans un premier temps à la configuration spécifique du noyau, puis à la spécialisation des cœurs processeur aux calculs propres à notre application.

6.3.1 Spécialisation des cœurs processeur

Les systèmes d'exploitations Linux actuels sont développés de façon à utiliser toutes les ressources disponibles dont les cœurs processeur. Ces systèmes n'étant pas temps-réel, toute application déployée sur les unités de calculs pourra être interrompue à tout moment par le système d'exploitation pour l'exécution des tâches à priorité supérieure.

Notre solution consiste à "priver" l'accès au système d'exploitation à certains cœurs processeur. Pour ce faire, plusieurs étapes sont nécessaires.

```
title Fedora (2.6.35.6-45.fc14.i686)
        root (hd0,2)
        kernel /boot/vmlinuz-2.6.35.6-45.fc14.i686 ro root=UUID=9050c9cb-8c26-4c49-b0f8-
b5df8d7bdcf2 rd_NO_LUKS rd_NO_LVM rd_NO_MD rd_NO_DM LANG=fr_FR.UTF-8 SYSFONT=latarcyrheb-
sun16 KEYTABLE=fr-latin9 rhgb quiet
        initrd /boot/initramfs-2.6.35.6-45.fc14.i686.img

title Fedora shielded (2.6.35.6-45.fc14.i686)
        root (hd0,2)
        kernel /boot/vmlinuz-2.6.35.6-45.fc14.i686 ro root=UUID=9050c9cb-8c26-4c49-b0f8-
b5df8d7bdcf2 rd_NO_LUKS rd_NO_LVM rd_NO_MD rd_NO_DM LANG=fr_FR.UTF-8 SYSFONT=latarcyrheb-
sun16 KEYTABLE=fr-latin9 rhgb quiet isolcpus=1 noirqbalance
        initrd /boot/initramfs-2.6.35.6-45.fc14.i686.img
```

FIGURE 6.3 – Exemple de modification du fichier grub.conf

Dans un premier temps, nous modifions le fichier grub.conf comme cela est montré dans l'exemple présenté dans la figure 6.3 afin de démarrer sur un nombre limité d'unités de calcul. Pour ce faire, il suffit de copier la ligne de démarrage du noyau Linux (ici pour l'exemple Fedora) situé dans le rectangle noir de la figure 6.3 et la coller en apportant quelques modifications apparaissant dans le rectangle bleu, avec le titre "Fedora shielded". Après cette modification, au démarrage d'une nouvelle session, une nouvelle ligne apparaît dans le menu de sélection du grub : "Fedora shielded". Il permet de démarrer sur le même noyau, avec quelques options différentes sans modifier ce dernier. Ainsi, l'option "isolc-

pus=1" permettra de démarrer uniquement sur le premier cœur. Pour démarrer sur les n premiers cœurs, l'option sera "isolcpus=1-n".

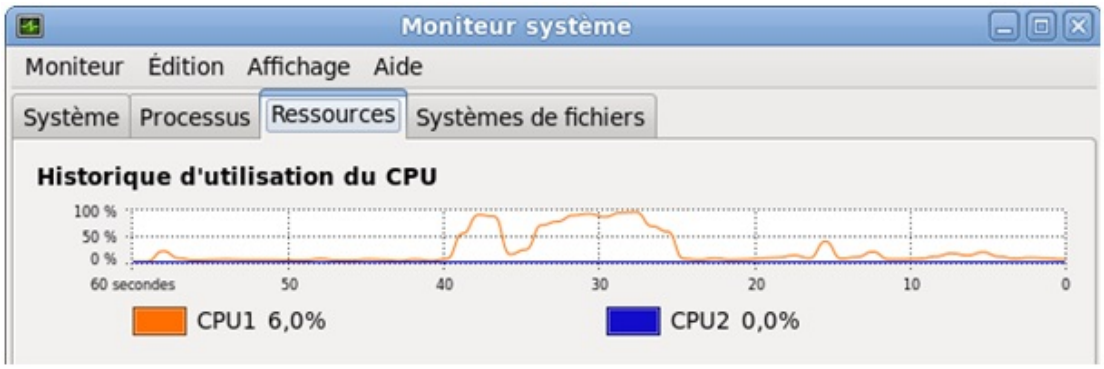

FIGURE 6.4 – Exemple de redirection des interruptions sur le cœur 1

6.3.2 Utilisation des affinités processeur

Les affinités processeurs permettent à notre système de lier un processus spécifique et les interruptions matérielles à une liste de cœurs en particulier. Comme nous pouvons le constater sur l'exemple décrit dans la figure 6.4, la redirection de toutes les interruptions sur le premier cœur nous permet un accès "complet" à l'autre cœur disponible. En effet, ce dernier ne peut plus être utilisé par le système d'exploitation nous permettant alors de lancer les modèles avioniques sans risque de perturbations ni d'interruptions par l'OS.

Pour ce faire, nous avons interrompu le démon Linux "Irqbalance" qui permet la distribution des interruptions sur les différents cœurs processeur disponibles dans le but d'éliminer les différentes interruptions pouvant avoir lieu sur les cœurs autres que ceux attribués au système d'exploitation. En effet, dans le cadre du développement de notre application, ce service d'interruptions est à proscrire dans le but de ne pas perturber l'exécution des modèles déployés sur les cœurs processeur.

Dans la figure 6.5, nous décrivons l'allocation des cœurs dans le cadre de nos tests pour les différents types de calculs pouvant avoir lieu sur ces derniers.

1. Les cœurs 1 et 2 sont alloués au système d'exploitation, comme nous l'avons décrit précédemment, nous avons contraint le système à démarrer sur les deux premiers cœurs. De plus, nous avons redirigé toutes les interruptions vers ces deux unités de calcul afin de laisser "libre" les autres cœurs.

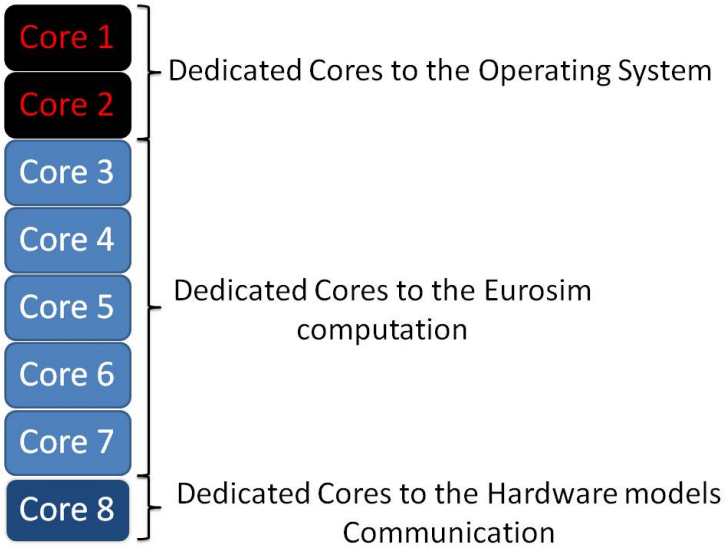

FIGURE 6.5 – Allocation des cœurs processeur

2. Les cœurs 3 à 8 sont alloués aux calculs propres à la session de test et de simulation lancée. Ces cœurs ne sont plus alors accessibles par les interruptions provenant du système d'exploitation. Ainsi, ces unités de calcul ne seront plus perturbées par ce dernier.
3. Le cœur 8 est un cas particulier parmi le cas décrit précédemment. En effet, il est alloué à un type de calcul particulier, c'est à dire l'exécution de modèles utilisant des accélérateurs matériels. Cette allocation permet de simplifier l'allocation des modèles sur les unités de calcul. En effet, le superviseur alloue les modèles virtuels au cœur 8 et ces derniers commenceront immédiatement à utiliser les fonctions communicant avec les accélérateurs matériels implantés dans le composant FPGA.

Finalement, dans le cadre de nos tests, le superviseur allouera les tâches logicielles sur les cœurs 3 à 7 et les tâches nécessitant un accélérateur matériel sur le 8.

6.3.3 Configuration adaptée du noyau

Il est également nécessaire d'adapter la configuration du noyau Linux aux contraintes de notre application.
Pour ce faire, quelques modifications sont à effectuer sur la configuration standard du noyau. Dans un premier temps, il est nécessaire de désactiver le swap (**S**upport for paging of anonymous memory (swap)) afin de ne pas perturber l'exécution des applications. En

effet, ceci évitera que les données d'un modèle avionique dormant se retrouvent sur l'espace de stockage de masse pouvant introduire des latences indésirables lors du réveil du processus concerné. De même, nous avons désactivé l'option **Optimize for size** qui permet, lorsqu'elle est activée de réduire la taille du noyau au détriment de sa vitesse. Enfin, l'activation de **Profiling support** nous permettra d'évaluer les performances apportées par nos modifications de notre système.

L'activation de l'option **IO Schedulers** permet d'accélérer les entrées-sorties, dont l'utilisation est primordiale dans le cadre du développement de notre environnement logiciel de test et de simulation.

Ensuite dans la section **Processor type and features**, l'activation des options **High Resolution Timer Support** ainsi que l'affectation de 1000 Hertz comme valeur de **Timer frequency** nous permettra de mettre à jour les données du moniteur à cette fréquence et d'obtenir un timer précis pour le cadencement des périodes temps-réel des sessions de test et de simulation. De plus, il est nécessaire de sélectionner le processeur embarqué sur la machine hôte (option **Processor family**) ainsi qu'activer l'option **Symmetric multiprocessing support** adaptée au type d'architecture utilisée dans le cadre de notre projet. Ensuite, l'activation de l'option de **Preemption Model** permettra d'accélérer la réactivité du noyau Linux lorsque ce dernier sera sollicité. Enfin, il est nécessaire de désactiver l'option **Paravirtualized guest support** qui pourrait venir perturber le déroulement de l'exécution des modèles de test et de simulation.

Enfin, la désactivation de l'option **CPU Frequency Scaling** dans **Power management and ACPI options** permettra de fixer les fréquences des cœurs processeur à leur maximum. En effet, leur activation permet de faire varier ces fréquences en fonction de paramètres de température et/ou de consommation.

Il est à noter que plusieurs de ces options sont également à modifier dans le cadre de l'utilisation d'un patch temps-réel tel que Xenomai. En effet, ce type de solution est également basé sur l'utilisation d'un noyau Linux standard, la modification des options décrites précédemment permettra d'obtenir des résultats satisfaisant les contraintes du domaine d'application.

6.3.4 Modification de la politique d'ordonnancement

Après avoir modifié la configuration du noyau, nous avons changé la politique d'ordonnancement des modèles sur les cœurs processeur pour une meilleure gestion de ces derniers durant les sessions de test et de simulation. Nous avons opté pour l'ordonnanceur "FIFO",

en effet, l'ordonnanceur "Round Robin", par défaut de Linux, ne permet pas à un processus d'être exécuté pendant une durée supérieure à un quantum de temps.

Dès lors, avec l'utilisation de l'ordonnanceur FIFO, un processus en cours d'exécution ne peut être interrompu que par une entrée/sortie ou par un processus avec une priorité plus élevée. De ce fait, nous avons pu obtenir des résultats très satisfaisants lors des essais avec cette nouvelle politique d'ordonnancement. De plus, la politique "FIFO", présente plusieurs avantages tels que sa simplicité de mise en place ainsi qu'une grande efficacité dans son utilisation avec des processus fréquemment bloqués par la gestion des entrées-sorties sur des systèmes multi-processeurs ce qui correspond à notre cas d'utilisation.

Dans la prochaine section, nous allons décrire l'implémentation du support logiciel déployé, de la description du scénario jusqu'à la présentation des résultats liés à l'exécution de ce dernier.

6.4 Implémentation du support logiciel déployé

Dans cette section, nous allons décrire dans un premier temps le scénario mis en place pour la validation du système proposé. Ensuite, nous évaluerons les performances de notre environnement lors du lancement de ce scénario.

6.4.1 Description du scénario

Dans cette partie, nous allons décrire le scénario mis en place pour le profilage du système proposé. Nous allons reprendre le scénario décrit précédemment dans ce document et décrit dans la figure 6.6. Ce scénario prévoit l'utilisation de trois modèles, le **pilote automatique**, la **mécanique du vol** et la **navigation**.

Comme nous l'avons décrit dans le chapitre proposant le support matériel, les liens de communication entre les différents modèles peuvent être aisément générés. En effet, comme cela est décrit dans la figure 6.6 les modèles partagent leurs données d'entrées et de sorties entre eux. Ainsi, il est simple à partir des dépendances de données inter-modèles de définir des régions de mémoires partagées entre ces derniers. A noter, qu'il serait tout aussi simple de faire de même avec des tubes nommés. D'ailleurs dans le cadre de ces expérimentations, nous avons utilisés des tubes nommés entre les différents modèles. Cependant, cette solution reste efficace jusqu'à un certain nombre de modèles et un certain volume de données.

Dans le cadre de ces tests, le modèle virtuel **mécanique de vol** embarquera une fonction logicielle et une fonction matérielle de façon transparente comme nous l'avons décrit

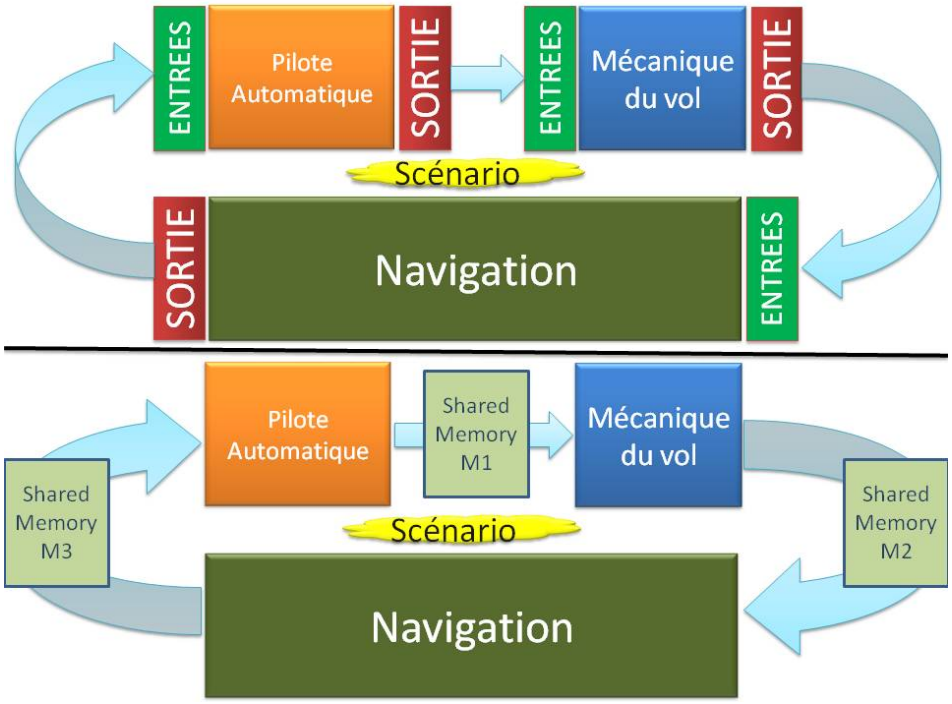

FIGURE 6.6 – Implémentation d'une boucle de test

précédemment. Ces informations vont figurer dans le fichier de configuration généré par l'heuristique de placement à partir d'un graphe d'exécution. Plusieurs groupes d'informations figurent dans ce fichier de configuration.

```
1  # List of models
2  #
3  # WARNING : "MODELS" MUST BE THE SECOND SECTION TO APPEAR
4  # IF/WHEN PROVIDED (i.e. fixed graph)
5  #
6  MODELS
7  #Nr-Task Type    HW        SW        DurHW DurMinSW DurMaxSW
8   1         SW      0       100     0           14         16
9   2       SW/HW    50        50     2           45         50
10  3         SW      0       100     0           9          12
```

Listing 6.2 – Modèles composant le scénario

Dans le pseudo code 6.2 apparaît l'ensemble des modèles composant le scénario de test et de simulation ainsi que la période temps-réel à respecter (ici 10 milli-secondes, cf. ligne

1 du pseudo code

Comme nous pouvons le remarquer, dans l'exemple de la figure 6.2, ce dernier, est composé de deux modèles logiciels ainsi que d'un modèle ayant une version logicielle et une version matérielle implantée sur le composant FPGA.

```
1  SCENARIO
2  #   Nr  Period (milli-seconds)
3  1       10        10
4
5  # List of links
6  #
7  # WARNING : "LINKS" MUST BE THE THIRD SECTION TO APPEAR IF/WHEN
8  # PROVIDED (i.e. fixed graph)
9  # LINKS
10 #
11 # WARNING
12 # DurCom1 defines the communication time between two software tasks
13 # DurCom2 defines the communication time between a software and a
14 # Hardware task or vice-versa
15 #Source Dest  DurCom1   DurCom2
16 1            2      10        11
17 2            3      10        15
18 3            1      10        0
```

Listing 6.3 – Déroulement du scénario

Les dépendances entre les modèles sont décrites dans le pseudo code 6.3 et proviennent de la description du scénario de modèles de test et de simulation. Cette partie du fichier de configuration offre plusieurs informations sur le scénario mis en place pour la validation de notre système.

```
1  # List of (initial) affectations (models to CPU)
2  # WARNING : "AFFECTATION" MUST BE THE FOURTH SECTION
3  # TO APPEAR IF/WHEN PROVIDED (i.e. fixed graph)
4  # THEY MUST ALSO BE GIVEN IN ASCENDING ORDER (1, 2, 3...)
5  #
6  # AFFECTATIONS
7  Nr  AFF
8  1    3
```

```
9  2      8
10 3      4
```

Listing 6.4 – Affectation des modèles sur les cœurs

Le pseudo code 6.4 présente l'affectation des modèles composant le scénario de test et de simulation. Ces affectations pourront être modifiées au cours de l'exécution de scénario si cela est nécessaire afin de préserver le respect des contraintes temps-réel imposées. Dans ce cas, nou pouvons observer que la tâche numéro 2 (la mécanique de vol) est affectée sur l'accélérateur matériel par défaut.

6.4.2 Implémentation des fonctions

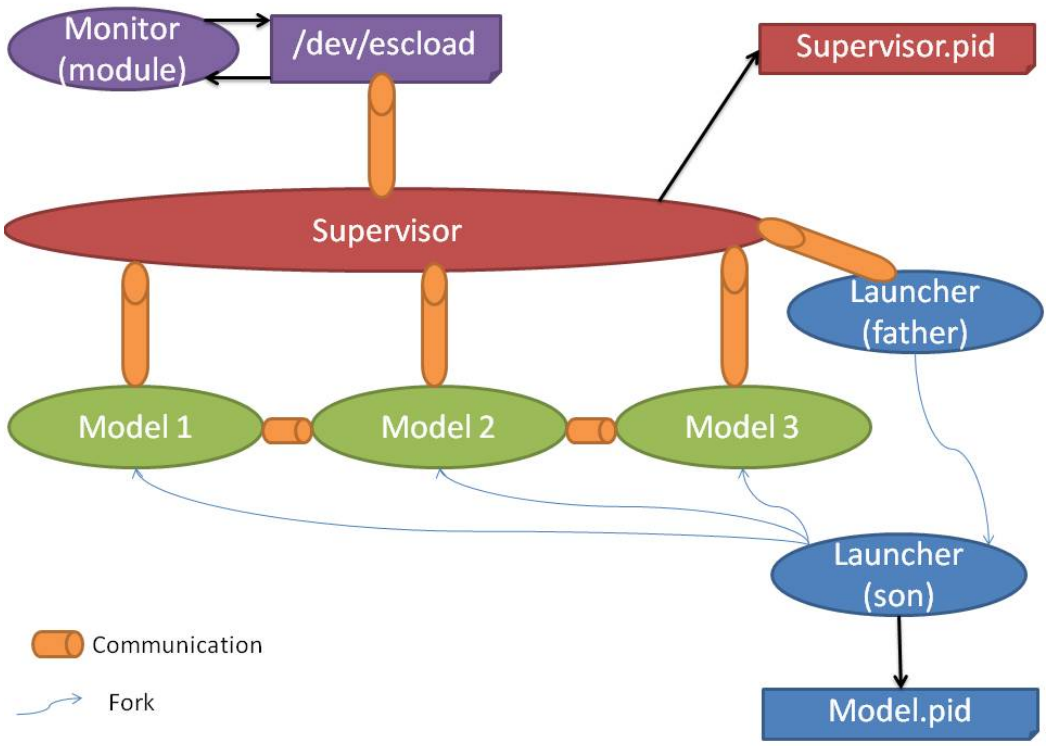

FIGURE 6.7 – Implémentation du support logiciel proposé

La figure 6.7 présente l'implémentation de l'environnement. Nous retrouvons ici les différentes fonctions décrites précédemment implémentées au sein de notre environnement. Comme cela a été décrit dans la section précédente, le modèle numéro 2 est la mécanique de vol ayant à la fois une version matérielle et logicielle.

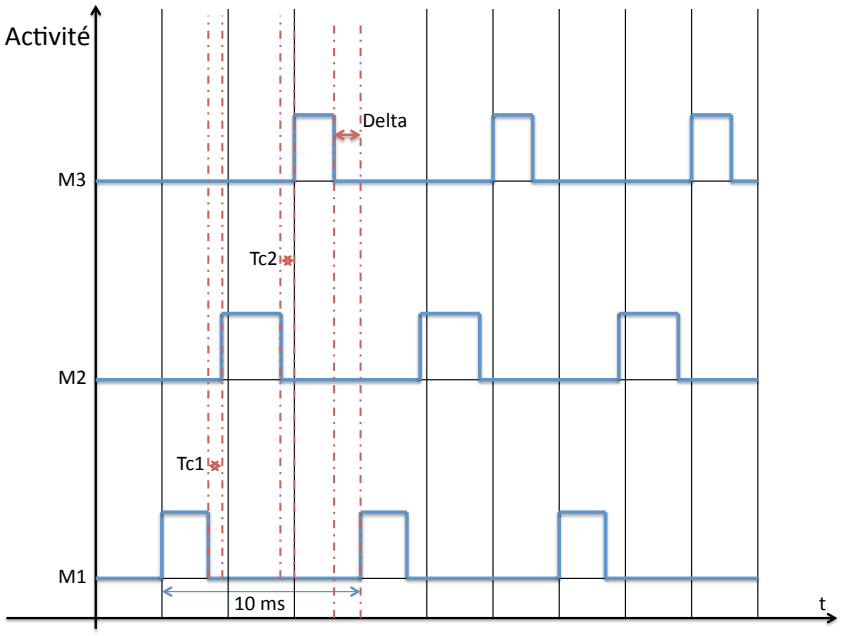

FIGURE 6.8 – Synchronisation et communication entre les modèles d'une session

Dans la figure 6.8, nous décrivons une session composée de trois modèles. Dans notre cas, le modèle source dispose d'un timer qui permettra le lancement périodique du graphe de tâches et ainsi vérifier du respect de la contrainte temps-réel définie pour la session donnée.

L'ensemble des modèles s'exécutera pendant un nombre donné d'itérations qui sera soit définit lors de la mise en place de la session soit dépendant de la durée de cette dernière si l'utilisateur souhaite l'interrompre lui même. Chaque modèle attend la mise à jour de ses entrées pour s'exécuter et attendra de même dès la mise à jour de ses sorties.

La valeur Delta définie dans la figure 6.8 est la différence entre la période, ici égale à 10 milli-secondes et la date de fin d'exécution du dernier modèle. Son signe permettra de définir si la période d'exécution de l'ensemble des modèles est bien inférieure à la contrainte définie. En effet, si Delta est positif la période est respectée, la session pourra alors continuer. La durée de communication de données entre deux modèles consécutifs doit être négligeable. Si ce n'est pas le cas, comme cela est montré dans la figure 6.8, un temps (Tc dans la figure 6.8) est alors introduit entre les modèles. L'utilisation de mécanismes de mémoire partagée est privilégié dans le cadre du développement de notre système afin de réduire au maximum ce temps et surtout de le rendre constant et non plus dépendant du volume de données à transmettre. Dès lors, les temps Tc1 et Tc2 seront égaux.

6.4.3 Évaluation de l'environnement proposé

Dans cette partie, nous allons mettre en avant les performances du système proposé via une présentation de plusieurs résultats issus de mesures effectuées lors d'une session ayant duré six jours complets. Cette durée n'est bien-sûr pas représentative d'une session standard (une demi-journée au maximum) mais cela nous a permis de vérifier et de valider au niveau fonctionnel le système proposé.

Dans le cadre de ce scénario, les modèles logiciels ont été conçus afin que leur temps d'exécution augmente au cours du scénario. En effet, si ce n'était pas le cas, nous n'aurions pas eut de garantie de requêtes de ré-allocation des modèles, les premiers tests sur des modèles significatifs ont démontré une grande stabilité de l'environnement lors des exécutions des premiers scénarios.

Dans la suite de cette section expérimentale, il sera nécessaire de récupérer quelques informations clefs afin d'évaluer les performances de notre système global.

1. Les performances du superviseur sont primordiales, en effet, elles vont nous permettre d'évaluer l'efficacité des choix de placement ainsi que les coûts réels de reconfiguration.
2. Le temps de ré-allocation d'une tâche logicielle d'un cœur processeur vers un autre considéré comme plus approprié par l'heuristique de placement.
3. Le temps de ré-allocation d'une tâche logicielle vers une tâche matérielle, dans le cas de notre étude, le modèle recevra l'ordre de se délocaliser sur le cœur 8 et exécutera en interne l'envoi et la réception de données vers l'accélérateur matériel implanté sur le composant FPGA.
4. La cohérence de données entre l'accélérateur matériel et le modèle logiciel exécutant la même fonction. Cette vérification permettra de valider l'accélérateur matériel implanté sur le composant FPGA dans un cas réel.
5. Le temps d'accès aux accélérateurs matériels implantés sur le composant FPGA, ce temps est très important dans le cadre du développement d'un système de test et de simulation performant.
6. Le temps de récupération des charges processeurs, cette étape doit être très courte permettant au superviseur de lancer l'heuristique avec les informations nécessaires à jour.
7. La répartition du temps au sein du superviseur lors de la session de test et de simulation. Comme nous l'avons décrit précédemment, ce dernier n'est lancé que lors

qu'il est sollicité par un des modèles. De plus, les seuils S1 et S2 sont définis en phase hors ligne, en déclenchant plusieurs reconfigurations, le superviseur possède les paramètres nécessaires afin d'effectuer la reconfiguration en ligne. Dès lors, il est possible de définir le seuil S1 comme étant le temps nécessaire pour effectuer la reconfiguration en ligne et l'exécution du modèle concerné et de ceux qui le suivent dans le fil d'exécution. Ensuite, S2 sera le temps au delà duquel il ne sera plus possible d'effectuer une seule reconfiguration ni de finir la boucle en cours d'exécution.

6.4.4 Résultats expérimentaux

Comme nous l'avons décrit précédemment, nous avons lancé plusieurs scénarios avec les mêmes modèles et mêmes périodes temps-réel. Ainsi ce test a permis l'exécution de plus de 50 millions de boucles. Nous avons constaté que le système a requis 46 reconfigurations et la simulation n'a été à aucun moment intérompue par un débordement de la période temps-réel définie.

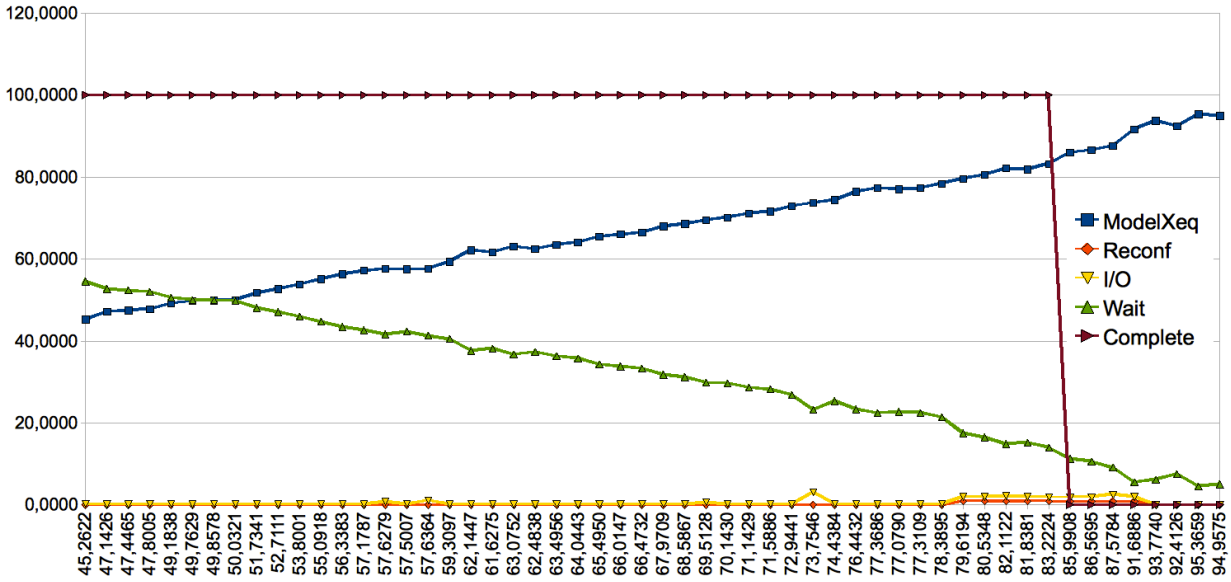

FIGURE 6.9 – Répartition du temps au sein du modèle de mécanique du vol

Dans la figure 6.9, nous décrivons la répartition du temps au sein du modèle de mécanique de vol au cours de l'exécution du scénario. Comme nous l'avons énoncé précédemment, le modèle est conçu de façon à augmenter son temps d'exécution à chaque itération du scénario. Dans notre cas, l'augmentation du temps d'exécution est proportionnelle à la charge processeur et donc inversement proportionnelle au

temps d'attente (*Wait* sur la figure 6.9) correspondant à la variable Delta décrite précédemment. La courbe *Complete* correspond au pourcentage du modèle exécuté en fonction de la charge. Enfin, nous pouvons remarquer que les temps occupés par les entrées-sorties et les reconfigurations sont négligeables mêmes lors du dépassement du seuil S1 correspondant au décroissement brutal de la courbe *Complete*.

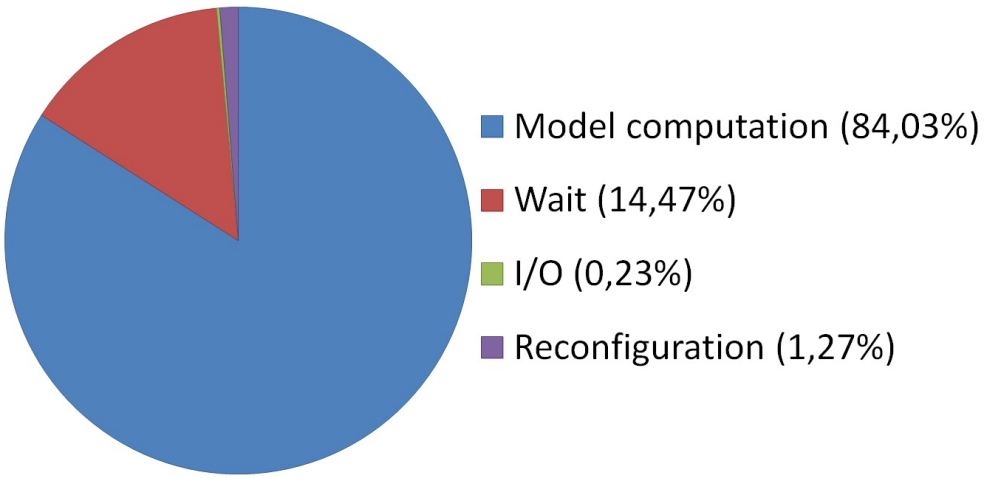

FIGURE 6.10 – Répartition du temps en fonction du temps d'exécution du scénario

Dans la figure 6.10, nous nous plaçons dans le cas d'une charge modèle de 84.03%, c'est à dire autour du seuil S1. Comme nous pouvons le constater sur cette période d'exécution, le modèle occupe effectivement 84.03% de la période alors que la majeure partie du temps restant est de l'attente de la fin de période (*Wait* sur la figure 6.10). Dans ce cas, le temps occupé respectivement par les reconfigurations et les entrées-sorties est de 1,27% et 0,23% de la période de 10 milli-secondes ce qui est négligeable vis à vis de la période définie.

Dans la figure 6.11, nous présentons la répartition moyenne du temps au sein du superviseur lors de l'exécution du scénario de test et de simulation. La majorité du temps correspond aux phases d'attentes du superviseur (*Wait* sur la figure 6.11), en effet, comme nous l'avons précisé précédemment, ce dernier n'est exécuté que lorsqu'il est sollicité par un des modèles de test et de simulation.
Ensuite, les pourcentages liés aux entrées-sorties et à l'exécution de l'heuristique représentent respectivement 1,08% et 0,17% du temps global. Enfin, le temps passé à récupérer les charges auprès du moniteur (*Monitor Access* sur la figure 6.11) est négligeable, en effet, ces données sont récupérées par une lecture du pseudo-système

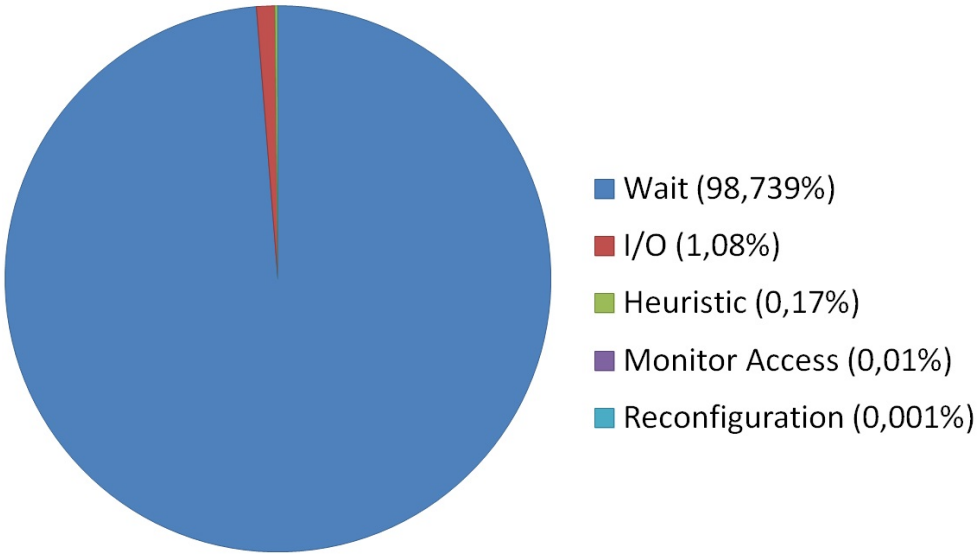

FIGURE 6.11 – Répartition du temps au sein du superviseur

de fichier */proc/escload*. Le temps consacré aux reconfigurations n'est quasi pas mesurable car ces actions sont effectuées au sein du modèle.

Les temps de sauvegarde du contexte d'exécution au sein du modèle n'est pas mesurable, en effet, ces données sont partagées par les fonctions logicielles et matérielles.

TABLE 6.1 – Temps et données clefs

Intitulé	Temps-donnée
Temps de reconfiguration	100 micro-secondes
Temps d'accès (écriture et lecture) vers le FPGA pour la mécanique de vol	30 micro-secondes
Temps d'exécution de la mécanique du vol (version logicielle)	40 micro-secondes
Temps d'exécution de la mécanique du vol (version matérielle)	2 micro-secondes
Charge processeur d'un modèle exécutant un accélérateur matériel	< 1 %

Dans le tableau 6.1, nous mettons en avant les temps d'exécution de certaines fonctions clefs au sein de notre système. Nous pouvons remarquer qu'il ne figure qu'un seul type de temps de reconfiguration. En effet, les reconfiguration logicielles et les reconfiguration hétérogènes ont le même coût. Cela est dû à l'utilisation de modèles virtuels décrits dans le chapitre précédent. Finalement, ces temps sont bien-sûr dépendants du support matériel déployé et décrit au début de ce chapitre.

6.4.5 Synthèse

Nous venons de démontrer dans ce chapitre la stabilité et le bon fonctionnement de notre système. Notre projet porte au sein d'EADS le nom CHARTS (Collaborative High performance Architecture for Reconfigurable Test and Simulation systems).

Nous allons dans cette dernière sous-section situer notre solution par rapport aux solutions existantes et déployées au sein du groupe EADS. Pour ce faire, nous allons effectuer une comparaison multi-critères présentée dans la figure 6.12. Les critères exposés ici sont les mêmes que ceux définis dans l'introduction et l'état de l'art de ce document. A cela, nous avons ajouté un critère important, l'interface utilisateur.

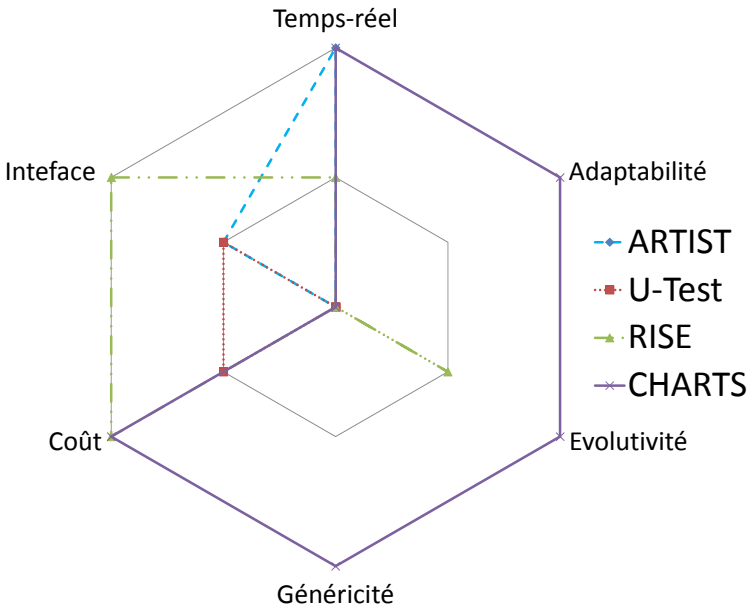

FIGURE 6.12 – Analyse multi-critères des solutions aujourd'hui existantes

La figure 6.12 permet d'évaluer l'éligibilité de chacune des solutions.

– **ARTIST**, la solution de test interne à la société Eurocopter ne répond qu'au seul critère de contraintes temps-réel imposées par le domaine d'application avec l'utilisation d'un système temps-réel dur VxWorks. Son support matériel considéré comme obsolète depuis 2008 est basé sur un fond de panier VME qui ne permet pas de remplir les critères d'adaptabilité, d'évolutivité, de généricité et de réduction de coût. En effet, les cartes CPU et d'entrés sorties ont vu leur coût augmenter

significativement au cours des dix dernières années et deviennent de plus en plus complexes à maintenir.

- **U-Test**, développé par la société EADS Test and Services est basé sur une architecture hétérogène multi-cœur-VME. Son seul attrait dans notre domaine d'application est le coût réduit du développement sur le système d'exploitation Linux qui n'a subit aucune modification et donc ne peut répondre aux contraintes temps-réel. Son support matériel dédié aux entrées-sorties est similaire à celui déployé avec ARTIST et ne répond donc pas, par extension, à tous les autres critères de notre analyse.
- **RISE**, la solution de simulation interne à la société Eurocopter favorablement à deux des cinq critères établis. En effet, cette solution est basée sur un environnement Linux utilisant tout comme CHARTS, les affinités processeurs. Toutefois, Cette solution ne peut répondre en tout point aux contraintes temps-réel, en effet, ce système restera passif face au dépassement des périodes définies et laisse à l'utilisateur le choix du placement des modèles sur les cœurs disponibles. De plus, cette solution n'ayant pas de dispositif d'entrées-sorties vers les équipements ne permet d'offrir qu'une partie visualisation décrite précédemment. Néanmoins, cette solution est la seule offrant un environnement de simulation complet permettant la mise en place d'un scénario ainsi que la visualisation d'un environnement virtuel.
- **CHARTS**, notre solution proposée dans le cadre de cette thèse répond à la majorité des critères établis par la société Eurocopter. Toutefois, CHARTS fait appel à des modèles à la fois logiciels et matériels ce qui engendre un surcoût pour le développement des IPs ainsi que leur intégration dans l'environnement proposé.

En guise de conclusion de cette analyse, nous pouvons envisager la rencontre des solutions CHARTS et RISE. En effet, comme cela est décrit dans la figure 6.12, CHARTS ne dispose pas d'interface homme-machine comme le propose RISE. Ainsi ce simulateur pourra bénéficier des toutes les capacités offertes par notre solution et devenir l'outil de référence pour le test et la simulation chez Eurocopter.

6.5 Vers un support d'applications étendu

Dans cette section, nous allons décrire dans quels domaines d'applications nous pourrons étendre les concepts décrits dans ce projet de thèse. En effet, le modèle d'exécution, le support matériel et/ou le support logiciel proposés peuvent être adaptés

pour une utilisation autre que le test et la simulation avionique.

6.5.1 Utilisation dans le cadre du domaine des transports

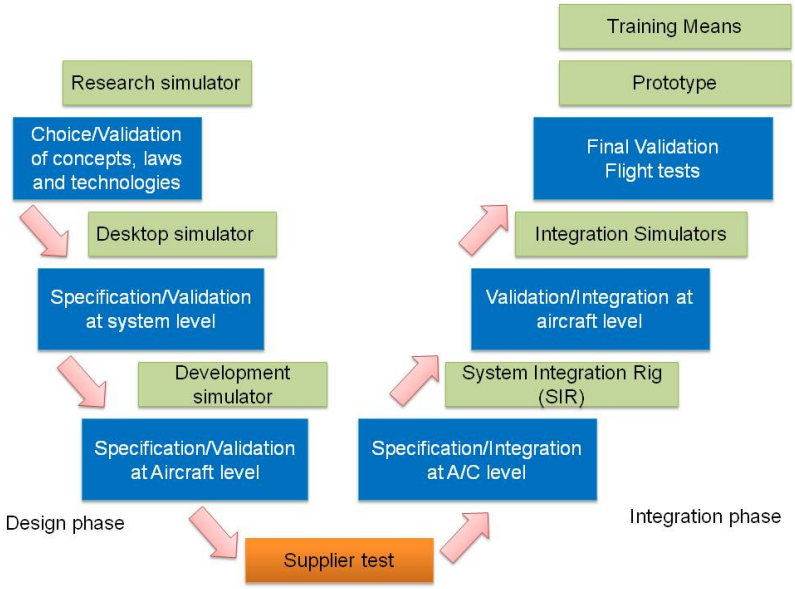

FIGURE 6.13 – Cycle de développement en V d'un nouvel équipement

Nous avons présenté dans le cadre de ce mémoire de thèse une solution adaptée au domaine aéronautique. Cette solution pourra également être utilisée dans un cadre plus étendu des transports. En effet, les systèmes embarqués critiques (ou non) sont de plus en plus présents dans les modes de transports existants. De ce fait, les cycles de développement de ces équipements peuvent être semblables à ceux décris dans la figure 6.13.

Par exemple, dans l'automobile, au même titre que dans le domaine aéronautique, plusieurs types de données circulent entre les équipements utilisant des bus particuliers. Dans le secteur automobile, le bus CAN est répandu, et le respect de contraintes temps-réel peut être nécessaire pour la gestion d'équipements de sécurité. Ainsi, un système de vérification et de validation similaire au notre pourra être déployé.

De plus, ce type d'architecture peut également voir le jour dans le domaine ferroviaire, les contraintes temps-réel y étant également présentes et ces systèmes nécessitent le même type de cycles de vérification et de validation.

6.5.2 Du test et de la simulation vers l'embarqué

Récemment, des circuits embarqués intégrant des processeurs multi-cœurs et des régions reconfigurables ont vu le jour.
A titre d'exemple, la société Xilinx a mis sur le marché un nouveau type de produit appelé EPP (Extensible Programmable Platform) composé de processeurs ARM inter-connectés avec une région reconfigurable comme cela est présenté sur la figure 6.14.

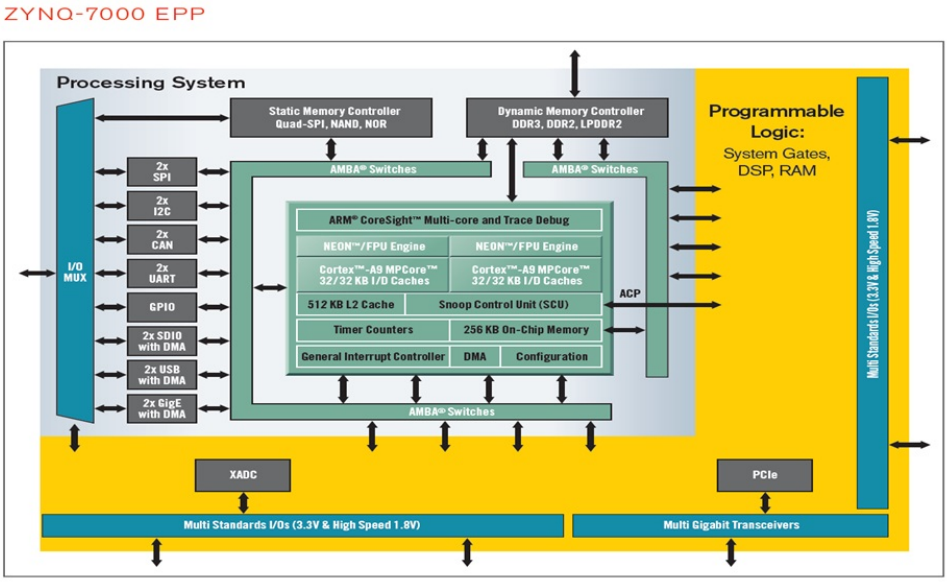

FIGURE 6.14 – Support des EPP par notre solution, cas du Zynq

Ce nouveau type d'architecture embarqué permet la "rencontre" des performances processeurs multi-cœurs aux réseaux de portes programmables permettant une implémentation plus efficaces d'applications "hétérogènes" mêlant parties logicielles et matérielles.

Tout d'abord, l'ensemble des concepts abordés dans cette thèse ciblant le domaine de calcul hétérogène à hautes performances pour le test et la simulation retrouvera son équivalent dans le domaine des systèmes embarqués. En effet, sur ce type de circuit, il est possible de gérer les régions reconfigurables (voire dynamiquement) depuis les processeurs de calcul. En plus, les moyens de communication sont multiples et peuvent s'adapter selon la quantité de données à transmettre ainsi que les fréquences de fonctionnement.

Dans le cas des applications embarquées, une implémentation de notre système permettra une utilisation efficace des nœuds de calculs. De plus, l'utilisation de modèles virtuels tels que nous l'avons proposé dans le cadre de notre projet est justifiée. De plus, les communications "hétérogènes" (de l'application logicielle vers les accélérateurs matériels implantés dans la région reconfigurable et vive-versa) deviennent simplifiées par l'utilisation du bus Axi proposé par Xilinx dans le cadre de l'exemple des EPP.

6.5.3 Nouveaux défis pour les systèmes embarqués

Dans le cadre de cette thèse, nous avons présenté une nouvelle génération de systèmes de test et de simulation. Les concepts ainsi que l'implémentation proposés précédemment pourront être utilisés dans plusieurs domaines et dans d'autres types d'architectures comme nous l'avons décrit précédemment.

De plus, différentes heuristiques pourront être développées selon les contraintes imposées par le domaine d'application.

Par exemple, dans le cadre d'un système embarqué ayant des contraintes de consommation, de respect de contraintes temps-réel, de fiabilité, de fréquences de fonctionnement réduites, etc. L'heuristique pourra prendre à ces paramètres afin de placer les tâches composant l'application en ciblant une optimisation multi-critères. Ces algorithmes fortement complexes devront être à l'image de leur domaine d'application.

A titre d'exemple, comme nous l'avons décrit dans le chapitre support logiciel, nous avons décrit la génération de plusieurs modèles en prenant pour paramètre le degré de parallélisation de ce dernier comme cela est décrit dans la figure 6.15. Dès lors, les différents modèles générés auront différents temps d'exécution pour différentes quantités de ressources matérielles utilisées et donc par extension, différentes consommations. Alors, dans ce cas, l'heuristique pourra implémenter l'une des versions selon les contraintes de consommation dynamiques imposées.

Ces contraintes pourront également concerner l'espace disponible sur les composants FPGA déployés, effet, en reprenant l'exemple cité précédemment, l'heuristique pourra implémenter l'une des versions du modèle selon l'espace utilisable.

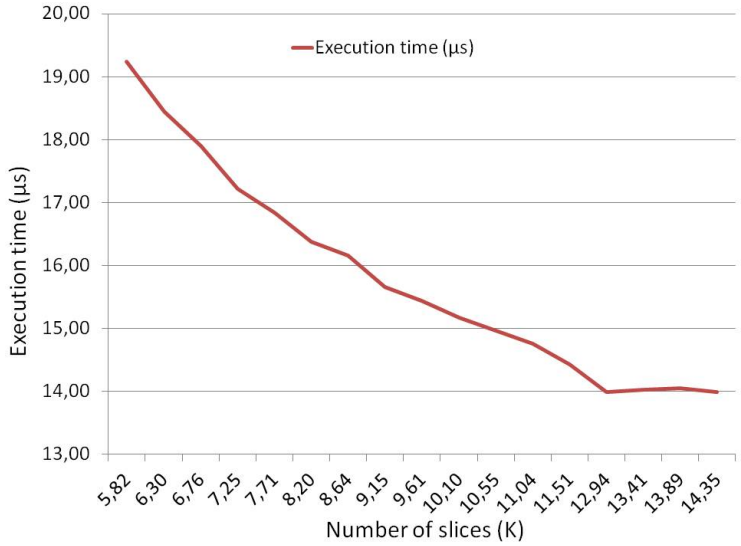

FIGURE 6.15 – Génération de code en fonction du déroulement d'une boucle

6.6 Conclusion

Dans ce chapitre, nous avons mis en avant les performances de notre architecture et ce, via une étude de cas présentant l'exemple d'un scénario de test et de simulation type. Les résultats présentés ont été possible par l'optimisation de l'environnement, une configuration noyau spécifique,l'utilisation des capacités offertes par le système d'exploitation ainsi que du matériel disponible.

Dans le prochain chapitre, nous conclurons ce mémoire et nous présenterons les perspectives liées à ce projet de thèse.

6. ÉTUDE DE CAS ET VALIDATION EXPÉRIMENTALE

Chapitre 7

Conclusion et perspectives

7.1 Bilan

Les travaux que nous venons de présenter s'inscrivent dans le cadre du développement de systèmes hétérogènes CPU-FPGA dédiés au test et à la simulation avionique chez Eurocopter. Ces systèmes ont un rôle primordial dans le cadre du développement des nouveaux équipements embarqués dans les futurs aéronefs. Nous avons dans un premier temps exposé les difficultés imposées par les systèmes actuels ainsi que leur limites technologiques.

Nos travaux ont débuté par l'étude des besoins propres au domaine du test et de la simulation avionique. En nous basant sur l'existant et l'expérience des concepteurs et utilisateurs des outils déployés chez Eurocopter, nous avons pu déduire les contraintes imposées ainsi que les besoins et les fonctionnalités pour la prochaine génération de systèmes de test et de simulation. Le respect des contraintes temps-réel mou et les capacités de communication vers les équipements sous-test composent les principales contraintes du système. Ensuite, nos investigations ont permis de mettre en évidence des besoins en termes de capacités croissantes de performance de calcul, de généricité du système ainsi que son adaptabilité aux contraintes statiques (hors session) et dynamiques (en cours de session) exprimés par l'industriel. Enfin, Eurocopter nous a imposé quelques critères supplémentaires tels qu'un déploiement aisé de la solution proposée et la prise en charge, lorsque cela est possible, du code "legacy".

Dans une première contribution, nous avons proposé un modèle d'exécution dynamique qui permettra aux futurs systèmes de test et de simulation de répondre aux contraintes et besoins industriels exprimés précédemment. En effet, notre proposi-

tion de rencontre des deux métiers du test et de la simulation sur un seul système permettra de réduire significativement le cycle en V du développement des nouveaux équipements embarqués. Ensuite, la généricité des futurs systèmes de test et de simulation rendue possible par notre modèle d'exécution permettra deux réductions de coût majeures. La première sera la possibilité d'adresser différents équipements à tester permettant ainsi de réduire les temps de validation de ces derniers. La seconde sera d'augmenter la productivités des équipes liées au développement des nouveaux équipements en leur permettant de faire intervenir en avance de phase de test en vol les pilotes d'essais pour une validation au sol. Ce modèle d'exécution nous a également permis d'apporter à Eurocopter une nouvelle génération de moyens de test et de simulation collaboratifs. En effet, le partage des données à travers le réseau d'entreprise permettra aux équipes de partager les ressources matérielles disponibles sur le site permettant ainsi une meilleure gestion de ces derniers. Enfin, nous avons mis en avant les capacités de reconfiguration dynamique de notre système permettant ainsi une adaptation aux contraintes imposées lors des sessions de test et de simulation dans le but de préserver le bon déroulement de cette dernière.

Dans un second temps, nous avons proposé un support matériel pour la prochaine génération de systèmes de test et de simulation composé de nœuds de calculs multi-cœurs liés par un bus rapide à des composants matériels FPGAs. Ce support matériel permettra d'apporter les performances nécessaires et suffisantes aux systèmes de test et de simulation tout en prenant en compte le critère de mise en place rapide et transparente sur le site industriel. Ainsi, la gestion du parc de systèmes de test et de simulation ne sera plus dissocié du parc informatique standard. De plus, les capacités de reconfiguration dynamiques offerts par les nœds de calculs composant ce support matériel permettront d'apporter généricité et adaptabilité au système global. Ensuite, le choix du bus PCIe permettra une utilisation et une parfaite cohésion de l'architecture globale dans la recherche de performances. Enfin, nous avons mis en avant les optimisations possibles et nécessaires pour un gain en performances des modèles de test et de simulation avioniques en mettant à contribution les propriétés de parallélisation du support matériel proposé.

Dans un troisième temps, nous avons proposé un support logiciel pour la prochaine génération de systèmes de test et de simulation. Nous avons opté pour une solution basée sur un environnement Linux standard et ce pour plusieurs raisons. La première réside dans la diffusion de ce système ainsi que le pôle créé autour de celui-ci chez

Eurocopter. La seconde est la flexibilité offerte par ce système, c'est ce critère qui nous a permis d'obtenir l'environnement temps-réel qui sera intégré dans les futurs systèmes de test et de simulation. La troisième est son faible coût de mise en service et de maintenance réduit par rapport à un système propriétaire ou un système temps-réel spécifique. Nous avons proposé une implémentation des différentes fonctions composant l'environnement de supervision proposées dans le premier chapitre afin de construire le support logiciel des futurs systèmes de test et de simulation.

Dans notre chapitre expérimental, nous avons présenté les performances offertes de notre système : Eurosim. Enfin, nous avons également proposé Eurosim en tant que support générique d'applications.

7.2 Perspectives

Le travail effectué dans cette thèse peut être poursuivi suivant plusieurs directions, nous en présenterons ici quelques-unes :

Industrialisation

Une phase d'industrialisation verra le jour chez Eurocopter dans le but d'évaluer le déploiement du système proposé. Ensuite, la couche logicielle développée pour les prochains outils de test pourra s'interfacer avec les fonctions développées dans le cadre de cette thèse dans le but d'obtenir le système de test et de simulation complet et cohérent. Enfin, pour que cette industrialisation soit complète, une collaboration avec des fournisseurs d'IP avioniques devra se faire. Le but ici sera d'obtenir une bibliothèque d'IPs à la fois logiciels et matériels cohérente permettant à notre système de pouvoir implanter ces dernières selon les scénarios à exécuter. Dès lors, tout comme, nous l'avons décrit dans ce document de thèse il sera nécessaire de définir une stratégie d'intégration de ces IPs dans l'environnement CHARTS. Aujourd'hui, plusieurs outils tels que IP-XACT [24] proposé par le consortium SPIRIT permettent la gestion et l'interface automatique de ces IPs. De nos jours, l'utilisation des méthodologies basées sur l'Ingénierie dirigée par les modèles (IDM) [5] sont très répandues dans le monde industriel.

Intégration au sein d'un système collaboratif

Dans cette thèse nous avons proposé une nouvelle génération de systèmes de test et de simulation ainsi qu'un prototype embarquant les fonctionnalités majeures de ce dernier. Cependant, la validation de notre système n'est intervenue que sur un seul nœud de calcul hétérogène (CPU-FPGA) répondant à la problématique initiale. Toutefois, nous avons également proposé au cours de cette thèse la mise en place d'un système collaboratif composé de plusieurs nœuds hétérogènes inter-connectés. Des futurs travaux au sein d'Eurocopter permettront la mise en place de ce système collaboratif. Les défis à relever couvrent la synchronisation et la communication d'un système distribué soumis à des contraintes temps-réel.
Par ailleurs, dans le cadre du développement d'un système collaboratif similaire à celui que nous avons proposé, il deviendra nécessaire de développer des heuristiques spécifiques adaptées à ce nouveau type d'environnement en vue d'un placement de tâches optimisé répondant à plusieurs critères.

Dynamicité, reconfiguration dynamique du système proposé.

Afin d'obtenir un système de test et de simulation complet, des fonctions matérielles pourront être développées. En effet, dans le cadre de la mise en place d'un nouveau scénario de test, une fonction de placement dynamique des différents IP matériels nécessaires (entrées-sorties et calcul) afin d'ajouter de la dynamicité et de la généricité dans cette phase. Dès lors, il sera possible d'implanter des mécanismes de reconfiguration dynamique rapides comme cela a été décrit dans l'état de l'art. En effet, dans le cadre de notre thèse, nous avons proposé l'utilisation de bibliothèques de configurations statiques propres à une ou plusieurs familles de scénarios. Le développement d'un tel type de fonction permettra une accélération de la conception de nouveaux types de scénarios ainsi qu'une mise en place d'un niveau d'abstractions permettant à tous les utilisateurs de manipuler les IPs.

Adaptation aux systèmes enfouis

L'environnement proposé pourra être étendu à une implémentation sur des systèmes sur puces dans le cadre de développement de systèmes embarqués. En effet, ce type de système hétérogène pourra voir les tâches composant l'application embarquée être

allouées et ré-allouées dynamiquement sur les nœuds de calculs hétérogènes. Dès lors, une adaptation des concepts proposés tels que celui de modèle virtuel pourra être adapté à ce nouveau cadre d'application. Ensuite, l'heuristique de placement proposée devra tenir compte des contraintes imposées par le domaine d'application (consommation, performance, contraintes temps-réel dur, etc).

7. CONCLUSION ET PERSPECTIVES

Bibliographie personnelle

[1] George Afonso, Nicolas Belanger. CHARTS : An innovation Matrix. In *SAE AeroTech Industrial Congress Exhibition*, Toulouse, France, Oct 2011.

[2] George Afonso, Nicolas Belanger, Jean Luc Dekeyser, Rabie Ben Atitallah. Conception d'une architecture hybride CPUs/FPGAs supervisee et dynamiquement reconfigurable pour optimiser l'execution des traitements de fonctions Test Systems sur bancs de tests temps reel. In *Eurocopter Patent*, 2011.

[3] George Afonso, Nicolas Belanger, Jean Luc Dekeyser, Rabie Ben Atitallah. Dispositif unifie et generique de test et de simulation d'equipements embarques . In *Eurocopter Patent*, 2013.

[4] George Afonso, Nicolas Damiani, Nicolas Belanger, Rabie Ben Atitallah, Martial Rubio. Hybrid and Multi-Core optimized architectures for test and simulation systems. In *Simutools*, Nice, France, Mar 2013.

[5] George Afonso, Rabie Ben Atitallah, Nicolas Belanger, Martial Rubio, Jean-Luc Dekeyser. An Efficient Design Methodology for Hybrid Avionic Test Systems. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, sep 2010.

[6] George Afonso, Rabie Ben Atitallah, Nicolas Belanger, Martial Rubio, Jean-Luc Dekeyser, Alexandre Loyer. A prototyping environment for high performance reconfigurable computing. In *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, Montpellier, France, June 2011.

[7] George Afonso, Rabie Ben Atitallah, Nicolas Belanger, Martial Rubio, Jean-Luc Dekeyser, Stephan Stilkerich. Toward Generic and

BIBLIOGRAPHIE PERSONNELLE

Adaptive Avionic Test Systems. In *NASA/ESA Conference on Adaptive Hardware and Systems*, San Diego, USA, June 2011.

[8] George Afonso, Zeineb Baklouti, Rabie Ben Atitallah, David Duviver, Eli Billauer,Stephan Stillkerich. Heterogeneous CPU/FPGA reconfigurable computing system for avionic test application. In *20th Reconfigurable Architectures Workshop*, BOSTON ,USA, May 2013.

Bibliographie

[9] Alfredo Revenaz, Massimiliano Ruggeri, Velio Tralli. Low Latency WI-FI Real-Time Protocol for Agricultural Machines Synchronization Using Linux RT Kernel. In *Industrial Electronics (ISIE)*, June 2011.

[10] AMBA AXI4 Interface Protocol. http://www.xilinx.com/ipcenter/axi4.htm.

[11] Armando Astarloa, Jesus Lazaro, Unai Bidarte, Aitzol Zuloaga, Jaime Jimenez. PCIREX : A Fast Prototyping Platform for TMR Dynamically Reconfigurable Systems. In *International Conference on Reconfigurable Computing and FPGAs*, 2009.

[12] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. Performance Comparison of FPGA, GPU AND CPU in Image Processing. In *19th IEEE International Conference on Field Programmable Logic and Applications, FPL*, Prague, Czech Republic, August 2009.

[13] Nicolas Belanger, Joel Bovier, Jean-Francois Gilot, Jean-Pierre Lebailly, and Martial Rubio. Multi-Core computers and PCI Express The future of data acquisition and control systems. In *ETTC International Conference*, Toulouse, France, 2009.

[14] Nicolas Belanger, Nicolas Favarcq, and Yann Fusero. An open real time test system approach. In *IEEE International Confererence on Advances in System Testing and Validation Lifecycle*, Porto, Portugal, September 2009.

[15] Frederic Rousseau Frederic Petrot Benaoumeur Senouci, Abdellah.M Kouadri.M. Multi-cpu/fpga platform based heterogeneous multiproces-

sor prototyping : New challenges for embedded software designers. In *The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, 2009.

[16] Benkrid, K., Erdogan, A.T., Arslan, T.,Azkarate, M., Martinez, I., Perez, A. R3TOS : A reliable reconfigurable real-time operating system. In *NASA/ESA Conference on Adaptive Hardware and Systems*, Anaheim, CA, USA, June 2010.

[17] Benkrid, K., Erdogan, A.T., Arslan, T.,Azkarate, M., Martinez, I., Perez, A. R3TOS : A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient and Dependable Computing on FPGAs. In *IEEE Transactions on computers*, April 2013.

[18] Pillement Sebastien Chillet D Bonamy Robin, Hung-Manh Pham. Uparc ultra fast power aware reconfiguration controller. In *Design, Automation Test in Europe Conference Exhibition*, 2012.

[19] Chavet Cyrille, Andriamisaina Caaliph, Coussy Philippe, Casseau Emmanuel, Juin Emmanuel, Urard Pascal and Martin Eric. A design flow dedicated to multi-mode architectures for DSP applications. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design (ICCAD '07)*, San Jose, California, 2007.

[20] Christos Koulamas , George J. Kyparisis. A modified LPT algorithm for the two uniform parallel machine makespan minimization problem, 2008.

[21] The ALTERA company. http://www.altera.com/corporate/about_us/history/optical/abt-optical-interconnects.html.

[22] The KONTRON company. x86 meets fpga - endless possibilities. In *White paper*, 2010.

[23] The XILINX company. http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[24] Sean Boyan David Murray. Addressing hw/sw interface quality through standards. 2012. http://www.duolog.com/wp-content/uploads/Increasing-HW_SW_interface_quality_through_standards.pdf.

[25] Enno LÃijbbers, Marco Platzner. ACM Transactions on Embedded Computing Systems . In *NASA/ESA Conference on Adaptive Hardware and Systems*, Oct 2009.

[26] FOSFOR : Flexible Operating System FOr Reconfigurable platform. http://users.polytech.unice.fr/~fmuller/fosfor/.

[27] Philippe Lorenzini FranÃğois Duhem, Fabrice Muller. Farm : Fast reconfiguration manager for reducing reconfiguration time overhead on fpga. In *Reconfigurable Computing : Architectures, Tools and Applications*, 2011.

[28] Anwar Ghuloum, Eric Sprangle, Jesse Fang, Gansha Wu, Xin Zhou, et al. Ct : A flexible parallel programming model for tera-scale architectures. *Intel White Paper*, 2007.

[29] Gordon Brebner. An investigation of virtual hardware using FPGA technology,. In *Honours year dissertation, Department of Computer Science, University of Edinburgh*, 1996.

[30] H. Plankl. Embedded solutions for development tests, component tests and system integration in the test centre. In *Aerospace Testing*, Munich, Germany, Oct 2009.

[31] Frederic Petrot Hao Shen. Novel task migration framework on configurable heterogeneous mpsoc platforms. In *Design Automation Conference*, 2009.

[32] htop - an interactive process viewer for Linux. http://htop.sourceforge.net/.

[33] Intel Corporation : Intel Parallel Studio. http://software.intel.com/en-us/intel-parallel-studio-home.

[34] IOxOS tehcnology. PCI Express to VME64x Bridge VME64x IO Interface. In *IOxOS Technical documentation*.

[35] IOxOS tehcnology. VCC 1104 AROLLA VME64x COM Express Carrier Board. In *IOxOS Technical documentation*.

BIBLIOGRAPHIE

[36] Kwok. A virtual hardware operating system for the Xilinx XC6200. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 1996.

[37] Le Projet Adeos. http://home.gna.org/adeos/index.fr.html.

[38] SW Moore, PJ Fox, SJT Marsh, AT Markettos, and A Mujumdar. Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *FCCM*, 2012.

[39] Nicolas Belanger, Jean Pierre Lebailly. Promoting avionic test systems as productivity enablers. In *Aerospace Testing*, Les Menuires, France, Oct 2010.

[40] OSCAR H. IBARRA, CHUL E. KIM. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors, January 1977.

[41] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, D. Beal. RTAI : Real-Time Application Interface. In *Linux Journal*, April 2000.

[42] Patrick S. Ostler, Michael J. Wirthlin, Joshua E. Jensen. FPGA Bootstrapping on PCIe Using Partial Reconfiguration. In *International Conference on Reconfigurable Computing and FPGAs*, 2011.

[43] Pierre Olivier, Jalil Boukhobza. A Hardware Time Manager Implementation for the Xenomai Real-Time Kernel of Embedded Linux. In *Embed With Linux (EWiLi) workshop*, Lorient, France, June 2012.

[44] PlanAhead Design and Analysis Tool. http://www.xilinx.com/tools/planahead.htm/.

[45] 1GHz Processor Power 4 : The First Multi-Core. http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/power4/.

[46] Gauthier Quesnel, Raphaël Duboz, and Éric Ramat. The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, **17** :641–653, April 2009.

[47] R. L. Graham. Bounds on Multiprocessing Timing Anomalies, March 1969.

BIBLIOGRAPHIE

[48] ReconOS : A programming model and operating system for reconfigurable hardware. http://reconos.de/.

[49] RTAI - Real Time Application Interface Official Website. https://www.rtai.org/.

[50] Tera scale Computing Architectural Overview. http://www.intel.com/content/www/us/en/research/intel-labs-terascale-computing-demo.html.

[51] Shoukat Ali, Jong-Kook Kim, Howard Jay Siegel, Anthony A. Maciejewski, Yang Yu, Shriram B. Gundala, Sethavidth Gertphol, Viktor Prasanna. Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems, 2007.

[52] SHROFF technology. http://schroff.fr/.

[53] SILKAN technology. 6U VME Intel i7 Core Air Cooled Processor Board. In *ACROMAG Technical documentation*.

[54] SILKAN tehcnology. http://www.silkan.com/sectors/aerospace/.

[55] SILKAN tehcnology. Arinc 429 ER (x8) optically isolated. In *SILKAN Technical documentation*.

[56] SILKAN tehcnology. Arion-100 Real-time Technology. In *SILKAN Technical documentation*.

[57] Jim Torresen Simen Gimle Hansen, Dirk Koch. High speed partial run-time reconfiguration using enhanced icap hard macro. In *International Parallel Distributed Processing Symposium*, 2011.

[58] Omar Souissi, Rabie Ben Atitallah, Abdelhakim Artiba, and Salah E. Elmaghraby. Optimization of run-time mapping on heterogeneous cpu/fpga architectures. In *9th International Conference of Modeling, Optimization and Simulation - MOSIMâĂŹ12*, Bordeaux, France, 2012.

[59] Techsat company. hhttp://www.techsat.com.

BIBLIOGRAPHIE

[60] The ALTERA company. 1553 DO-254 Compliant. In *ALTERA Technical documentation*.

[61] The KONTRON company. Kontron PCIe/104 MICROSPACE MSMST : First SBC with configurable Intel Atom E600C processor series. In *http ://us.kontron.com/about-kontron/news-events/kontron-pcie104-microspace-msmst-first-sbc-with-configurable-intel-atom-e600c-processor-series.4208.html*.

[62] The XILINX company. ARINC-429 Rx/Tx DO-254 compliant. In *XILINX Technical documentation*.

[63] The XILINX company. Designing Software Applications with an Extensible Virtual Platform. In *http ://www.xilinx.com/products/zynq-7000/extensible-virtual-platform.htm*.

[64] The XILINX company. LogiCORE IP CAN. In *XILINX Technical documentation*.

[65] The XILINX company. Xilinx Extends Leadership in Avionics with Certifiable, All Programmable Design Solutions and Dedicated Support. In *http ://press.xilinx.com/2013-03-25-Xilinx-Extends-Leadership-in-Avionics-with-Certifiable-All-Programmable-Design-Solutions-and-Dedicated-Support*.

[66] Timothy G. Mattson, Rob Van der Wijngaart, Michael Frumkin. Programming the Intel 80-core network-on-a-chip terascale processor. In *ACM/IEEE conference on Supercomputing*, 2008.

[67] VectorFabrics : Pareon. Analyze your sequential C code to create an optimized parallel implementation. http://www.vectorfabrics.com/.

[68] Najjar Walid Villarreal Jason, Park Adrian and Halstead Robert. Designing Modular Hardware Accelerators in C with ROCCC 2.0. In *Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10)*, Washington, DC, USA, May 2010.

[69] Wolfgang Betz, Marco Cereia, Ivan Cibrario Bertolotti. Experimental Evaluation of the Linux RT Patch for Real-Time Applications. In *Emerging Tehnologies and Factory Automation*, Mallorca, September 2009.

BIBLIOGRAPHIE

[70] Xenomai : Real-Time Framework for Linux. http://www.xenomai.org.

[71] Khaled Benkrid Xiang Tian. High-performance quasi-monte carlo financial simulation : Fpga vs. gpp vs. gpu. In *ACM Transactions on Reconfigurable Technology and Systems*, 2010.

[72] XILINX. AXI Hardware ICAP. In *http ://www.xilinx.com/products/intellectual-property/axi$_h$wicap.htm*.

[73] Xilinx Electronic System Level Design. http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/index.htm.

[74] Xilinx Technical Documentation. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[75] Xilinx Virtex-6 FPGA ML605 Evaluation Kit. http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm.

[76] Xillybus : An FPGA IP core for easy DMA over PCIe with Windows and Linux. AnFPGAIPcoreforeasyDMAoverPCIewithWindowsandLinux.