Architectures multiprocesseurs, distribuées et reconfigurables dynamiquement pour les applications temps réel

Fabrice Muller

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

HABILITATION A DIRIGER LES RECHERCHES

Université de Nice-Sophia Antipolis

présentée par

Fabrice Muller

Architectures multiprocesseurs, distribuées et reconfigurables dynamiquement pour les applications temps réel

préparée dans l'équipe MCSOC
de l'unité de recherche UMR 6071 - LEAT/CNRS

soutenue le *7 décembre 2011*

devant le jury composé de :

Michel PAINDAVOINE
Professeur des Universités, Université de Bourgogne, Président & Rapporteur

Jean-Luc DEKEYSER
Professeur des Universités, Univ. de Lille 1, Rapporteur

Jean-Didier LEGAT
Professeur, Université Catholique de Louvain, Rapporteur

Michel AUGUIN
Directeur de Recherche CNRS, Univ. de Nice-Sophia Antipolis, Examinateur

François VERDIER
Professeur des Universités, Univ. de Nice-Sophia Antipolis, Examinateur

Résumé

Ce manuscrit résume mes activités de recherche depuis mon recrutement en 2000 en tant que Maître de Conférences à l'ESINSA [1], devenue en 2005 Polytech'Nice-Sophia, Filière Électronique (Université de Nice-Sophia Antipolis). Mes activités d'enseignement sont centrées sur l'électronique numérique, le développement de systèmes embarqués et l'informatique. Mes travaux de recherche se sont déroulés au laboratoire I3S [2] (2000-2007), puis au LEAT [3].

Mes travaux concernent les aspects temps réel dans les architectures reconfigurables intégrant un ou plusieurs processeurs supervisés par des systèmes d'exploitation pour l'embarqué. Ces architectures reconfigurables ont la capacité à se reconfigurer pour répondre ou s'adapter aux comportements et aux contraintes des applications. Ces activités de recherche interagissent selon 3 axes de recherche.

Le premier axe traite des aspects plus théoriques d'ordonnancement de tâches sur des architectures classiques monoprocesseur et multiprocesseur ou d'ordonnancement d'applications parallèles pour des architectures auto-adaptatives futuristes.

Le second axe se focalise sur la problématique de placement et d'ordonnancement de tâches purement matérielles sur des architectures reconfigurables par l'utilisation de méthodes exactes, d'heuristiques, et à l'aide de modèles de haut-niveau et de simulations tout en maintenant un effort d'implémentation de ces algorithmes sur de véritables plateformes de prototypages.

Le dernier axe s'attache plus particulièrement aux systèmes d'exploitation au sein des architectures reconfigurables et à la manière de les faire évoluer dans ce contexte. Je développe des techniques, méthodes, méthodologies permettant de répondre à des problèmes actuels dans le domaine du reconfigurable qui sont principalement liés aux systèmes d'exploitation temps réel, mais également aux systèmes d'exploitation embarqués tels que Linux. De même, je m'intéresse à des solutions pour le calcul haute performance en incluant l'aspect d'accélération matérielle reconfigurable.

1. École Supérieure d'Ingénieurs de Nice Sophia Antipolis
2. Informatique, Signaux et Systèmes de Sophia-Antipolis
3. Laboratoire d'Électronique, Antennes et Télécommunications à Sophia Antipolis

Remerciements

Je tiens tout d'abord à remercier très sincèrement Messieurs Jean-Luc DEKEYSER (Professeur des Universités de l'Université de Lille 1), Jean Didier LEGAT (Professeur, Université Catholique de Louvain) et Michel PAINDAVOINE (Professeur des Universités de l'Université de Bourgogne) d'avoir accepter de rapporter sur mon Habilitation à Diriger des Recherches.

Un grand merci à Michel PAINDAVOINE d'avoir accepter de présider le jury de mon HDR à la place de Monsieur Alain MERIGOT (Professeur des Universités, Université Paris sud) qui a eu un problème de transport et n'a pu assister à la soutenance.

Tous mes remerciements à Messieurs Michel AUGUIN (Directeur de Recherche CNRS, Université de Nice-Sophia Antipolis) et François VERDIER (Professeur des Universités, Université de Nice-Sophia Antipolis) de m'avoir fait le plaisir de participer à mon jury d'HDR.

Merci également aux doctorants, Masters que j'ai eu la chance d'encadrer et sans qui mes années de recherche n'aurait pas pu aboutir.

Enfin, je remercie l'ensemble de mes collègues du LEAT, ainsi que le personnel administratif qui, par leur travail quotidien, facilite nos travaux et dont on oublie souvent la part importante dans la vie du laboratoire.

Pour finir, une pensée particulière à toute ma famille, particulièrement mon épouse Marie-Noëlle et mes enfants Nolwenn, Victor et Olivia pour leur soutien moral.

Avant-propos

Ce manuscrit résume les activités effectuées en recherche, encadrement, animation scientifique et enseignement depuis ma thèse soutenue en janvier 2000 à l'IRESTE [4] (Université de Nantes).

J'ai été recruté comme maître de conférences en septembre 2000 à l'ESINSA devenue Polytech'Nice-Sophia, Filière Électronique en 2005. Mes activités d'enseignement se situent dans le domaine de l'électronique numérique, de la conception de systèmes embarqués, d'architectures de processeurs et de l'informatique. Mes activités de recherche se sont déroulées à l'I3S, dans l'équipe MOSARTS [5]. L'équipe travaillait principalement sur la modélisation d'applications de télécommunication et sur des méthodes de conception système dans le but d'optimiser les architectures, respectant un ensemble de contraintes temporelles et minimisant la surface de silicium ainsi que la consommation d'énergie. Mon activité de recherche concernait principalement l'exploration d'architecture au niveau système et s'inscrivait pleinement dans les thématiques de l'équipe. À partir de l'année 2008, je suis parti au LEAT où un nouveau thème s'est mis en place, appelé MCSOC [6].

Durant ces six dernières années, les activités au sein du thème MOSARTS, puis MCSOC ont évolué pour se diriger vers les problématiques de réseaux de capteurs tout en gardant la problématique de la consommation d'énergie. C'est ainsi que j'ai eu la possibilité de démarrer un axe de recherche vers les architectures reconfigurables, adaptatives pour les applications temps-réel dures ou souples. Mes travaux ont commencé avec le projet Européen ÆTHER [7] traitant des applications pervasives pour les futures architectures. Mes activités de recherche se sont accentuées vers les architectures reconfigurables et auto-adaptatives, et l'évolution des systèmes d'exploitation à l'aide de deux projets en cours (ANR FOSFOR [8] et ANR ARDMAHN [9]), dont je suis le responsable scientifique coté LEAT. Mes activités se sont

4. Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Électronique
5. MOdélisation et Synthèse d'ARchitectures pour le Traitement du Signal
6. Modélisation et Conception Système d'Objets Communicants
7. Projet européen IST-FET, FP6, `www.aether-ist.org`
8. Flexible Operating System FOr Reconfigurable platform
9. Architecture Reconfigurable Dynamiquement et Méthodologie pour l'Auto-adaptation en Home Networking

ainsi décomposées selon 3 axes : les systèmes temps-réels, le placement et l'ordonnancement des tâches matérielles sur les architectures reconfigurables et l'axe concernant les systèmes d'exploitation.

Lors de mon changement de problématique de recherche, mon activité a donc débuté par l'axe 1 qui traite des aspects plus théoriques d'ordonnancement de tâches sur des architectures classiques multiprocesseur ou d'ordonnancement d'applications parallèles pour des architectures auto-adaptatives futuristes. L'axe 2 se focalise sur les méthodes de placement et d'ordonnancement de tâches purement matérielles sur des architectures reconfigurables avec un effort d'implémentation de ces algorithmes sur de véritables plateformes de prototypages. L'axe 3 s'attache plus particulièrement aux systèmes d'exploitation au sein des architectures reconfigurables et à la manière de les faire évoluer. Ces 3 axes, pleinement intégrés dans le thème MCSOC, sont détaillés respectivement dans les chapitres 2, 3 et 4.

Avant de détailler ces axes, le chapitre 1 situe le contexte de mon activité de maître de conférences, résume mes travaux de recherche ainsi que mon activité d'enseignement et administrative. De plus, ce chapitre contient également ma bibliographie personnelle. Les travaux de recherche, décrits dans ce manuscrit, ont été publiés dans des journaux ou des conférences. D'ailleurs, une sélection de publications significatives est proposée dans l'annexe A.

Table des matières

Abbréviations

AADL	Architecture Analysis and Design Language
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
ASEDZL	Anticipating Slack Earliest Deadline first until Zero Laxity
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
CASH	CApacity SHaring
CLB	Configurable Logic Block
CPU	Central Processor Unit
DAG	Directed Acyclic Graphs
DMA	Direct Memory Access
DVFS	Dynamic Voltage and Frequency Scaling
EDF	Earliest Deadline First
EEDF	Enhanced Earliest Deadline First
ERM	Enhanced Rate Monotonic
FARM	Fast Reconfigurable Manager
FOSFOR	Flexible Operating FOr Reconfigurable platform
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HAL	Hardware Abstraction Layer
HPC	High Performance Computers
HPRC	High Performance Reconfigurable Computers
IP	Intellectual Property
LLF	Least Laxity First
LLREF	Least Local Remaining Execution First
LRU	Least Recently Used
LUT	Look-Up Table
MARTE	Modeling and Analysis of Real-Time Embedded Systems

MDA	Model-Driven Architecture
MoC	Model of Computation
MPCI	MultiProcessor Communication Interface
MPI	Message Passing Interface
MUF	Maximum Urgency First
NNUS	Nonuniform Node, Uniform System
NOC	Network On Chip
NPB	NAS Parallel Benchmark
NPZ	Non Préemption Zone
OCP	Open Core Protocol
OpenMP	Open Multi-Processing
OE	Operating Environment
O-RLE	Offset Run Length Encoding
OS	Operating System (Système d'exploitation)
PE	Processing Elements
PLB	Processor Local Bus
PR	Partial Reconfiguration
QoS	Quality of Service (Qualité de Service)
RB	Reconfigurable Bloc
RLE	Run Length Encoding
RM	Rate Monotonic
RPB	Reconfigurable Physical Bloc
RR	Reconfigurable Region
RTEMS	Real-Time Operating System for Multiprocessor Systems
RTL	Register Transfer Level
RTOS	Real Time Operating System
RUF	Real Urgency First
RZ	Reconfigurable Zone
SANE	Self Adaptive Networked Entity
SoC	System on Chip
SoPC	System on Programmable Chip
SR	Static Region
TLM	Transaction Level Modeling
VCI	Virtual Component Interface
UML	Unified Modeling Language
μT	micro Thread
WCET	Worst Case Execution Time
XML	Extensible Markup Language

CHAPITRE 1

Synthèse des travaux

Nom patronymique :	MULLER
Prénom :	Fabrice
Date et lieu de Naissance :	15 mai 1971 à Caen (14)
Nationalité :	Française
Situation de famille :	Marié, 3 enfants
Situation actuelle :	Maître de Conférences, 5ème échelon (Recruté en Septembre 2000)
E-mail :	Fabrice.Muller@unice.fr

Unité d'enseignement
Polytech'Nice-Sophia
(Filière Électronique)
1645 Route des Lucioles
06410 Biot
Téléphone : 04 92 38 85 46

Unité de recherche
Laboratoire d'Electronique, Antennes et Télécommunications UMR CNRS 6071
250, Rue Albert Einstein - Bat. 4
06560 Valbonne
Téléphone : 04 92 94 28 66

1.1 Grades et titres universitaires

1996-2000 :	**Doctorat**	Spécialité électronique
	Titre	Outil pour l'aide à la conception conjointe des systèmes matériel/logiciel.
	Directeur	Jean-Paul Calvez
	Laboratoire	IRESTE (devenue Polytech'Nantes), Équipe MCSE (Méthodologie de Conception des Systèmes Électroniques), Université de Nantes
	Soutenue le	20 Janvier 2000 à l'IRESTE
		devant le jury composé de :
	Président	Michel Corazza, ENSSAT, Université de Rennes 1, Lannion
	Rapporteurs	Michel Auguin, I3S, Université de Nice-Sophia Antipolis
		Jean-Luc Philippe, Université de Bretagne Sud, Lorient
	Examinateurs	Jean-Paul Calvez, IRESTE, Université de Nantes
		Fabrice Lemonnier, THOMSON CSF Optrosys, Guyancourt
		Olivier Pasquier, IRESTE, Université de Nantes
1995-1996 :	**D.E.A.**	Spécialité Électronique
	Titre	Démarche de génération des interfaces matériel/logiciel dans le cadre du CoDesign.
	Responsable	Jean-Paul Calvez
	Laboratoire	IRESTE, Équipe MCSE, Université de Nantes
1993-1996 :	**Diplôme d'ingénieur IRESTE**, Spécialité Systèmes Électroniques et Informatique Industrielle, à Nantes	
1991-1993 :	**D.U.T. G.E.I.I.**, Génie Électrique et Informatique Industrielle, Option automatismes et systèmes, à l'Institut Universitaire de Technologie de Nantes	

1.2 Situations successives

1996-1999 :	**Doctorant**, contrat CDD sur projet Européen Esprit CoMES (Codesign Methodology for Embedded Systems)
1999-2000 :	**ATER**, demi-poste à l'IRESTE à Nantes, en section CNU 61
2000-2011 :	**Maître de Conférences** à Polytech'Nice-Sophia, Filière Électronique (ex-école d'ingénieurs ESINSA, École Supérieure d'Ingénieurs de Nice Sophia Antipolis), en section CNU 61

1.3 Activités de recherche

1.3.1 Contexte

Mes travaux de recherche sont actuellement effectués au LEAT, qui est dirigé par le professeur Christian Pichot, dans le thème MCSOC depuis le 1er janvier 2008. Mes travaux

antérieurs (2000-2007) ont été réalisés au laboratoire I3S.

Mes travaux concernent les aspects temps réel dans les architectures reconfigurables intégrant un ou plusieurs processeurs supervisés par des systèmes d'exploitation pour l'embarqué. Ces architectures reconfigurables ont la capacité à se reconfigurer pour répondre ou s'adapter aux comportements et aux contraintes des applications. Ces projets s'inscrivent naturellement dans le thème MCSOC et plus précisément dans l'axe N°2 concernant les systèmes reconfigurables, auto-adaptatifs et virtualisation. Je suis à l'origine de la création de cet axe dont je suis actuellement le responsable scientifique.

En effet, c'est à partir de la période 2004-2006 que j'ai modifié mon thème de recherche pour mettre l'accent sur les architectures reconfigurables, adaptatives pour les applications temps-réel dures ou souples. Ceci a commencé avec le projet Européen ÆTHER traitant les applications pervasives pour les futures architectures. Mes travaux de recherche se sont accentués vers ces architectures et les systèmes d'exploitation qui devront inévitablement supporter de telles architectures dynamiques, reconfigurables. Deux autres projets en cours (ANR FOSFOR et ANR ARDMAHN), dont je suis le responsable scientifique coté LEAT, doivent répondre en partie à la problématique des architectures dynamiquement reconfigurables mais également à la manière de vérifier de telles architectures. En effet, l'un des problèmes majeur auquel je suis confronté, est de rechercher des solutions qui permettront de vérifier ou de valider le comportement de ces architectures qui fluctuent au cours du temps.

Mes activités de recherche interagissent selon 3 axes de recherche, présentées à la figure 1.1. Mes travaux au sein de ces axes sont complémentaires, et les contributions d'un axe amènent de nouvelles idées dans un autre axe. Cette organisation me parait importante car elle facilite l'interaction entre ces axes et permet, à mon sens, une ouverture d'esprit et une curiosité qui font parties de l'une des qualités d'un chercheur.

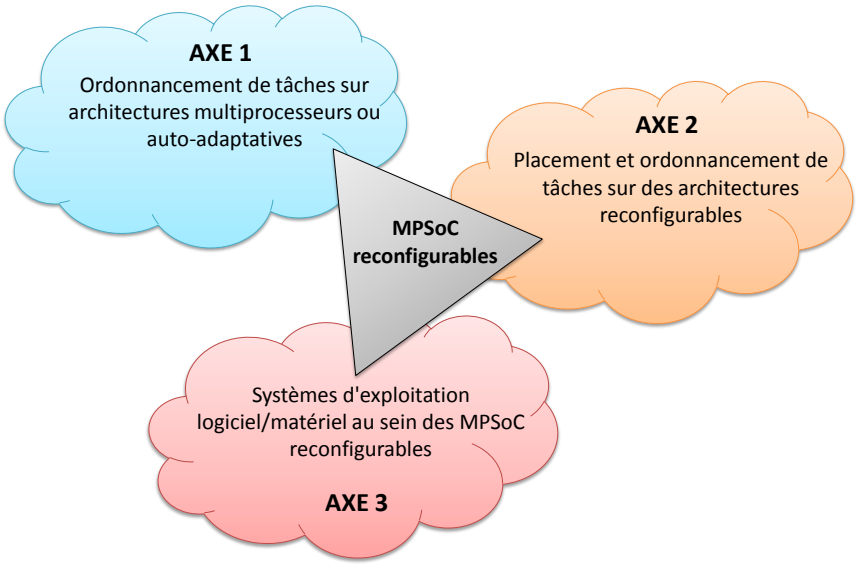


Figure 1.1 – *Positionnement des 3 axes.*

Bien sûr, ces 3 axes interfèrent entre-eux. La recherche en amont porté par l'axe 1 peut déboucher sur de l'implémentation basée sur un système d'exploitation afin d'être amélioré, ou encore de rendre l'axe 1 plus réaliste à l'aide de l'axe 2. Dans la suite, je présente brièvement

les objectifs selon les trois axes. Mon activité de recherche est résumée dans la section 1.3.3 et détaillés respectivement dans les chapitres 2, 3 et 4.

Objectifs des 3 axes

Axe 1 : Ordonnancement de tâches sur architectures multiprocesseurs ou auto-adaptatives

Mon activité de recherche a débuté par l'axe 1 qui traite des aspects plus théoriques d'ordonnancement de tâches sur des architectures classiques monoprocesseur et multiprocesseur, ou encore de l'ordonnancement d'applications parallèles pour des architectures auto-adaptatives futuristes qui rejoignent la problématique des architectures reconfigurables.

L'objectif de cet axe est de développer de nouveaux algorithmes d'ordonnancement de tâches logicielles dans un contexte monoprocesseur et multiprocesseur dont on sait que ce nombre, au sein d'un MPSoC, augmente sans cesse. Mon idée est d'étendre cette problématique d'ordonnancement à de nouvelles architectures à venir. Dans cet axe, je traite plutôt des travaux futuristes, à plus long terme, par rapport au deux axes suivants. Mes contributions sont :

- Des algorithmes d'ordonnancement monoprocesseur,
- Des algorithmes d'ordonnancement globaux/mixtes sur architectures multiprocesseur,
- Des algorithmes d'ordonnancement auto-adaptatifs pour des architectures reconfigurables futuristes.

Axe 2 : Le placement et l'ordonnancement de tâches matérielles sur des architectures reconfigurables

L'axe 2 se focalise sur les méthodes de placement et d'ordonnancement de tâches purement matérielles sur des architectures reconfigurables avec un effort d'implémentation de ces algorithmes sur de véritables plateformes de prototypages. Cet axe de recherche est un axe intermédiaire entre la recherche à plus long terme (axe 1) et l'axe 3.

Mon objectif est d'imaginer des algorithmes de placement et d'ordonnancement sur des architectures réalistes, à moyen terme, en vue d'une implémentation. Ainsi, je propose des contributions se rapportant au placement et l'ordonnancement de tâches matérielles sur des architectures reconfigurables :

- par des méthodes exactes,
- par des heuristiques,
- à l'aide de modèles de haut-niveau et par simulation.

Axe 3 : Les systèmes d'exploitation logiciel/matériel au sein des MPSoC reconfigurables

L'axe 3 s'attache plus particulièrement aux systèmes d'exploitation au sein des architectures reconfigurables et de la manière de les faire évoluer. Ainsi, ce dernier axe concerne plutôt de la recherche à moyen et court terme. En effet, je développe des techniques, méthodes, méthodologies permettant de répondre à des problèmes actuels dans le domaine du reconfigurable qui sont principalement liés aux systèmes d'exploitation temps réel, mais également aux systèmes d'exploitation embarqués tels que Linux. Plusieurs contributions ont été réalisées ou sont en cours de réflexion :

- La réduction du temps de reconfiguration qui est une nécessité afin d'obtenir des architectures reconfigurables partielles viables. Notons que j'utilise, pour le moment, la technologie Xilinx, la plus mature dans ce domaine,
- La conception d'un système d'exploitation multiprocesseur centralisé gérant des tâches logicielles,
- La conception d'un système d'exploitation FOSFOR (du nom du projet ANR FOSFOR) pouvant gérer des tâches logicielles et matérielles dans un contexte multiprocesseur et d'architecture reconfigurable partiellement,
- Le développement d'une méthodologie pour imaginer des architectures reconfigurables pour le HPC [1] et les plateformes auto-adaptatives dont certains concepts sont repris de l'axe 1.

1.3.2 Activités de recherche doctorale

1.3.2.1 DEA

Titre : **Démarche de génération des interfaces matériel/logiciel, Expérimentation sur une application de communication**

Mes travaux de DEA se sont déroulés la même année que la dernière année d'école d'ingénieur à l'IRESTE. J'ai choisi d'effectuer mon stage à l'IRESTE (Université de Nantes), dans l'équipe MCSE, sous la direction de Jean-Paul Calvez. L'objectif était de proposer une méthodologie et un outil afin de concevoir conjointement l'architecture matérielle et logicielle, appelé « CoDesign ».

Ce stage concerne le problème de la génération (semi-)automatique des interfaces nécessaires entre le matériel et le logiciel pour assurer les relations fonctionnelles de la solution. Nous partons de l'hypothèse qu'une grande partie du travail de CoDesign a été entreprise au préalable, à savoir que le partitionnement matériel-logiciel et l'allocation de la solution fonctionnelle sur l'architecture matérielle ont été décidés. La solution fonctionnelle à implanter sur chaque microprocesseur et dans chaque ASIC [2] est donc définie et sert comme point de départ pour cette étude. Comme bases de référence, l'implantation des interfaces entre fonctions logicielles est décrite dans MCSE [138] et celle des interfaces entre fonctions matérielles dans MCCE [140]. La synthèse VHDL du modèle fonctionnel a été montrée dans [139]. Notre démarche consiste dans un premier temps, à spécifier les règles de transcription des relations fonctionnelles servant de couplage entre le matériel et le logiciel en une implantation, telle que cette solution de réalisation puisse être générée si possible automatiquement par un outil. Dans un deuxième temps, la réalisation d'un outil se déduira des règles ainsi définies. Une solution de génération d'interface a été testée sur une plateforme contenant un FPGA [3] xc4010 et un microcontrôleur 68332.

1. High Performance Computers
2. Application Specific Integrated Circuit
3. Field Programmable Gate Array

1.3.2.2 Thèse de doctorat

Mes activités de recherche se sont poursuivies par la préparation de mon doctorat au sein de l'équipe MCSE, à l'IRESTE (Université de Nantes). Ma thématique portait sur le CoDesign et plus particulièrement la conception d'un outil de CoDesign, appelé MCSE ToolBox. De plus, cette thèse a aidé à la création en 2003 de la startup Cofluent Design (`http://www.cofluentdesign.com`) toujours active actuellement et achetée par la société Intel en septembre 2011.

La thèse de doctorat a été soutenue le 20 janvier 2000, à l'IRESTE et s'intitule « Outil pour l'aide à la conception conjointe des systèmes matériel/logiciel », devant le jury composé de :

Président	Michel Corazza, ENSSAT, Université de Rennes 1, Lannion
Rapporteurs	Michel Auguin, I3S, Université de Nice-Sophia Antipolis
	Jean-Luc Philippe, Université de Bretagne Sud, Lorient
Examinateurs	Jean-Paul Calvez, IRESTE, Université de Nantes
	Fabrice Lemonnier, THOMSON CSF Optrosys, Guyancourt
	Olivier Pasquier, IRESTE, Université de Nantes

Résumé

La complexité et la variété des applications nécessitant le développement rapide et la mise en œuvre de systèmes électroniques et informatiques, conduisent les concepteurs à vouloir disposer d'une aide par des outils informatiques. Ainsi, l'objectif global de cette thèse a été d'aboutir à un prototype d'outil de conception des systèmes basé sur la méthodologie MCSE (Méthodologie de Conception des Systèmes Électroniques). Pour atteindre cet objectif nous avons analysé trois points essentiels pour concevoir et réaliser ce prototype d'outil.

Le premier point a conduit à la conception d'un outil appelé MCSE ToolBox supportant la méthodologie MCSE pour la conception conjointe des systèmes matériel/logiciel temps-réel. Ainsi, l'objectif est atteint en permettant le plus rapidement possible à la vérification du comportement et des performances de l'application grâce à une technique de co-simulation du modèle. De plus, il est possible de générer du code C et VHDL pour la réalisation du prototype.

Le second point a consisté à définir une architecture d'outils pour résoudre les problèmes multi-utilisateurs, multi-projets et multi-sites. Ceci nous a conduit à une architecture en réseau permettant de supporter un nombre important d'utilisateurs simultanément, de faciliter l'intégration d'outils internes et commerciaux, et de permettre une diffusion ou un accès instantané des outils au niveau mondial. Enfin, le dernier point a été de définir une organisation permettant une bonne gestion de projets et des concepteurs à partir d'un ensemble d'outils de management. Ce concept de management permet la gestion de projets conséquents, d'assurer la collaboration de plusieurs équipes réparties géographiquement.

Tous les concepts d'architecture et d'organisation ont été validés par la mise à disposition de la plate-forme en réseau. De plus, des exemples internes et industriels ont permis de valider les outils de conception. Ces outils ont été réalisés en JAVA.

1.3.3 Activité de recherche en tant que Maître de Conférences

À partir de septembre 2000, j'ai commencé mon activité de recherche au laboratoire I3S, dans l'équipe MOSARTS dont le responsable était Michel Auguin. L'équipe travaillait principalement sur la modélisation d'applications de télécommunication et sur des méthodes de conception système dans le but d'optimiser les architectures, respectant un ensemble de contraintes temporelles et minimisant la surface de silicium et la consommation d'énergie. En effet, la consommation d'énergie est devenue au fil du temps l'activité principale de l'équipe. À partir de l'année 2008, je suis parti au LEAT où un nouveau thème s'est mis en place, appelé MCSOC, où je suis actuellement. Ce thème traite principalement de quatre sous-thèmes :

1. l'optimisation de la consommation d'énergie dans les objets communicants,
2. les systèmes reconfigurables, auto-adaptatifs et virtualisation,
3. les systèmes réactifs et coopératifs,
4. la modélisation comportementale et conception de front-end RF.

J'ai monté le second sous-thème, « systèmes reconfigurables, auto-adaptatifs et virtualisation » (à partir de 2008) qui est devenu un axe fort à partir de 2010 par l'intermédiaire de deux projets ANR FOSFOR et ARDMAHN, et à l'aide, en partie, du projet Européen ÆTHER. Ce projet ÆTHER a permis de démarrer mes travaux de recherche en 2005 pour l'élaboration de ce sous-thème dont je suis actuellement le responsable. Trois axes de recherche se sont dégagés de ce sous-thème : l'ordonnancement de tâches sur architectures multiprocesseurs ou auto-adaptatives, le placement et l'ordonnancement de tâches matérielles sur des architectures reconfigurables, et les systèmes d'exploitation logiciel/matériel au sein des MPSoC reconfigurables.

Quatre doctorants, de nombreux DEA/MASTER2 et stagiaires ont travaillé sur cette activité de recherche. Par la suite, je présente ces trois axes que nous retrouvons d'une manière détaillée dans les chapitres suivants. De plus, une dernière section 1.3.3.4 résume mes travaux connexes dans ce domaine.

1.3.3.1 Axe 1 : Ordonnancement de tâches sur architectures multiprocesseurs ou auto-adaptatives

Les processeurs embarqués représentent 98% du marché des processeurs vendus et deviennent de plus en plus intelligents, peuvent s'identifier, communiquer et interagir entre eux. Ainsi, mes premiers objectifs de recherche ont conduit à réfléchir à de nouvelles approches en tenant compte de l'intelligence des processeurs, de l'auto-adaptivité des systèmes en fonction du travail à réaliser et de la capacité à répartir des applications à travers un réseau de processeurs particuliers, appelé « processeur SANE [4] », en tenant compte de contraintes telles que le temps réel et la qualité de service. Ces architectures futuristes peuvent être considérées comme des architectures reconfigurables dynamiquement puisque mon objectif est d'exécuter des applications sur une architecture plus ou moins déterministe dont les SANEs et les réseaux inter-SANE sont reconfigurables dynamiquement. Mes travaux ont été menés à partir

4. Self Adaptive Networked Entity

de 2005 dans le cadre du projet Européen ÆTHER et ouvrent des perspectives à plus long terme.

En complément de ces activités de recherche, j'ai imaginé des algorithmes d'ordonnancement plutôt dédiés aux tâches logicielles dans le cadre d'ordonnancement mono ou multiprocesseur. En effet, les études menées ont pour dénominateur commun le constat suivant : les algorithmes d'ordonnancement temps-réel ont des bornes d'ordonnançabilité au plus égales à la capacité de traitement de l'architecture (et ce suivant certaines hypothèses). Cependant, ces algorithmes peuvent gagner en efficacité dès lors que leur impact sur les temps de gestion est réduit et ainsi augmenter la Qualité de Service (QoS [5]) des applications. Cet accroissement en efficacité peut être obtenu par une meilleure prise en compte de paramètres implicites des tâches. De plus, les bornes d'ordonnançabilité égales à la capacité de traitement de l'architecture peuvent être atteintes en relâchant certaines des hypothèses classiquement considérées. C'est à partir de ces constatations que j'ai exploré et suggéré des algorithmes d'ordonnancement. Certains de ces algorithmes sont encore trop complexes en temps de calcul pour être utilisables mais, comme je le disais précédemment, cet axe reste une activité à plus long terme.

Afin de mener à bien mes travaux de recherche, un doctorant [93] a participé à cet axe, ainsi qu'un ingénieur d'étude et un stagiaire de Master. L'ensemble des travaux menés au sein de cet axe de recherche ont conduit à 1 brevet [3], 10 publications dans des conférences internationales [19, 22, 23, 24, 25, 26, 27, 29, 30, 37] et 2 publications dans des conférences nationales [49, 51]. Ces travaux ont été principalement menés dans le cadre du projet Européen ÆTHER.

1.3.3.2 Axe 2 : Le placement et l'ordonnancement de tâches matérielles sur des architectures reconfigurables

J'ai également traité le cas de l'ordonnancement des tâches mais qui seront matérielles, c'est-à-dire conçues en une description matérielle comme par exemple en langage VHDL. Ces tâches matérielles sont sur des zones reconfigurables assimilées à des processeurs matériels hétérogènes. Je rappelle que dans un système monoprocesseur, l'ordonnancement consiste essentiellement à déterminer la date d'exécution des tâches. Dans le contexte qui m'intéresse, l'hétérogénéité et la multiplicité des ressources posent directement la question de la ressource de calcul où doivent s'exécuter les tâches. Implicitement, ce choix amène à déterminer la meilleure position possible d'une tâche compte tenu de l'occupation des ressources de calcul à l'instant d'ordonnancement. Pour ce qui concerne la zone reconfigurable, un axe supplémentaire apparaît, il s'agit de l'axe temporel puisque cette ressource peut accepter plusieurs tâches séquentiellement. Ceci induit un problème de placement et d'ordonnancement de tâches matérielles sur un ensemble de zones reconfigurables hétérogènes. Je présente différentes approches permettant d'appréhender cette complexité de type NP-complet : heuristiques, programmation dynamique par solveur, modèle de simulation.

Dans le cas du modèle de simulation, mon objectif est de développer une méthodologie pour la modélisation de haut niveau des architectures reconfigurables dynamiquement. Je

5. Quality of Service

désire introduire de l'exploration d'architectures dans ma méthodologie en vue de fournir au futur développeur un flot de conception qui pourrait être intégré, par exemple, dans un flot de reconfiguration partielle. Je base mon approche sur un modèle de conception en Y qui fusionne mon modèle applicatif et mon modèle de plateforme en un modèle architectural qui fournit au concepteur une implémentation en conformité avec une stratégie d'ordonnancement. Cette approche repose sur un simulateur décrit en SystemC. En bref, la finalité de ces travaux est de fournir au concepteur un moyen de trouver le nombre de zones reconfigurables la plus proche de l'optimal en fonction de l'application et de l'algorithme d'ordonnancement (modifiable) tout en respectant les contraintes temps-réel.

Certains de mes travaux de recherche seront probablement utilisables à moyen terme en fonction de l'évolution des architectures reconfigurables.

À la contribution de cet axe, deux doctorants y ont participé ainsi que des stagiaires de Master. L'ensemble de ces travaux menés au sein de cet axe de recherche a conduit à 1 chapitre de livre [4], 2 revues [7, 9], 8 publications dans des conférences internationales [12, 13, 15, 16, 18, 20, 21, 35] et 5 publications dans des conférences nationales [42, 44, 46, 48, 50]. Ces travaux sont menés dans le cadre de deux projets ANR FOSFOR et ARDMAHN en cours.

1.3.3.3 Axe 3 : Les systèmes d'exploitation logiciel/matériel au sein des MPSoC reconfigurables

Depuis quelques années, notamment avec l'apparition des systèmes sur puce et des architectures reconfigurables dynamiquement, l'évolution de la densité d'intégration a proposé de répondre aux contraintes de conception par la parallélisation au niveau tâches ou données, augmentant ainsi le nombre des unités de calcul, notamment dans les systèmes embarqués. Cette tendance semble se confirmer aujourd'hui et risque de devenir un véritable « mur » de complexité lorsque, à moyen terme, les systèmes intégreront plusieurs dizaines de blocs (processeurs et/ou coprocesseurs). Le caractère multiprocesseur, ou plutôt multi-ressources d'exécution de la plateforme lié à la distribution de l'OS [6] requiert des mécanismes particuliers pour parvenir à assurer une cohérence globale du système. En effet, il est nécessaire que chaque tâche puisse accéder à l'ensemble des services du système d'exploitation et ceci sans distinction de type de tâches et/ou de localisation. C'est dans le cadre du projet FOSFOR que je suis en charge de concevoir le cœur de l'OS FOSFOR coté matériel, afin de fournir des services aux tâches matérielles.

Le système d'exploitation (OS) a clairement un rôle important à jouer dans ces nouvelles architectures reconfigurables dynamiquement. Un OS est classiquement organisé autour de services fournis aux tâches logicielles ou matérielles. Ces systèmes d'exploitation, qui doivent exploiter au mieux les architectures reconfigurables, doivent avoir les moyens de gérer de manière efficace le reconfigurable. Par exemple, les FPGAs récents, comme ceux de la société Xilinx, fournissent cette fonctionnalité via le port de configuration interne (ICAP [124]) mais engendrent des temps et la gestion du reconfigurable loin de l'optimal. J'essaye ainsi, dans le cadre du projet ARDMAHN, d'apporter des solutions pour diminuer le temps de

6. Operating System (Système d'exploitation)

reconfiguration des zones où s'exécutent les tâches matérielles. Je propose également des modèles de coût pour estimer les temps de reconfiguration à haut niveau, utilisables dans les outils de niveau système comme notre simulateur SystemC ou d'autres outils industriels comme Cofluent Design.

Les architectures de calcul parallèle commencent progressivement à intégrer des unités matériellement reconfigurables. Ces unités peuvent accueillir différents circuits logiques, et donc implémenter des accélérateurs matériels en lien direct avec l'application exécutée. Mon objectif est de développer une nouvelle approche permettant une plus grande mixité entre les éléments de calcul logiciels et matériels afin de tirer parti de toute la puissance que peut apporter un accélérateur matériel pour l'exécution d'une tâche spécifique. À cette fin, je propose un flot destiné à la conception d'applications parallèles (pouvant être de type HPRC [7]) tout en gardant une compatibilité avec les applications déjà existantes. Mes travaux en cours doivent également aboutir à une plateforme d'exécution pour ces applications qui utilisent la reconfiguration dynamique partielle afin de réduire la granularité de la reconfiguration, actuellement souvent sous-exploitée.

Cet axe de recherche a pu se renforcer grâce aux projets ANR FOSFOR et ARDMAHN, ainsi qu'à l'aide de deux doctorants actuellement en 3^{e} année de thèse, et également des stagiaires. Ces travaux ont conduit à 3 revues internationales [5, 6, 8], 2 revues internationales en première soumission [11, 10], 1 revue nationale en première soumission [41], 6 conférences internationales [12, 14, 17, 19, 36, 40], et 5 conférences nationales [42, 43, 45, 47, 54].

1.3.3.4 Autres implications

Modélisation à haut niveau d'un capteur intelligent à ultrason [31]

La première implication, durant l'année 2001, concerne des capteurs intelligents qui diffèrent des capteurs conventionnels. Ces capteurs intelligents permettent de traiter de grandes quantités de données à proximité de la source et également de communiquer de façon bi-directionnelle. Dans ces travaux, j'ai passé en revue la conception d'une perception distribuée en utilisant l'outil MCSEToolbox (qui deviendra Cofluent Design en 2003). Ce système de perception est fait d'un ensemble de capteurs intelligents reliés par un bus de terrain. J'ai particulièrement axé mes travaux sur la conception de haut niveau d'un tel système ou plutôt en me focalisant sur la partie du capteur intelligent à ultrason. Cette conception a abouti à une structure fonctionnelle décrivant la solution interne d'une manière indépendante de la technologie. Cette solution peut être validée par simulation avant de considérer les aspects de conception architecturale menant à l'étape de prototypage.

Réseaux de neurones multi-couches sur FPGA en vue de l'utilisation de la reconfiguration partielle [28]

Cette seconde implication en 2006 a été réalisée en collaboration avec deux industriels, la société NodBox et la société Xilinx. L'objectif majeur de Nodbox est de prédire les risques de perte de contrôle du véhicule pendant la phase de conduite normale. Ceci semble possible

7. High Performance Reconfigurable Computers

à l'aide d'algorithmes basés sur du réseau neuronal multi-couches. Ainsi, mon objectif était de définir une architecture générique pour la phase d'extraction d'un algorithme de réseau neuronal multi-couches pouvant être mis en œuvre sur un FPGA. Mon but est d'avoir une architecture qui peut être appliquée à n'importe quels réseaux de neurones multi-couches composés d'un nombre donné de couches et d'un nombre donné de neurones dans chaque couche. En plus, cette architecture améliore la densité du FPGA en proposant des solutions de multiplexage temporelle qui permettrait d'utiliser la reconfiguration dynamique et partielle. Plusieurs réseaux de tailles différentes ont été mis en œuvre basés sur cette architecture générique. J'ai évalué les performances à partir d'un FPGA Virtex-4, via un réseau de neurones multi-couches en analysant la variation de la durée minimale et le nombre de ressources occupées.

1.3.4 Encadrement de travaux de recherche doctoral

1.3.4.1 Encadrement ou co-encadrements de thèses

Le tableau 1.1 résume les thèses en encadrement ou co-encadrement. Le détail de chaque thèse est ensuite présenté. Pour le moment, je suis co-encadrant à 70% de 2 thèses en cours.

Tableau 1.1 – *Résumé du nombre d'encadrement ou co-encadrements de thèses.*

État de la thèse et % d'encadrement	Encadrement	Nombre	Période
Soutenue, 33% d'encadrement	co-directeur	1	2005 → 2009
Soutenue, 100% d'encadrement	directeur	1	2008 → 2011
À soutenir fin 2012, 70% d'encadrement	co-encadrant	2	2009 → 20xx

A - Thèse de Farroq Muhammad

Années :	2005/2009
Titre :	Ordonnancement de Tâches Efficace et à Complexité Maitrisée pour des Systèmes Temps Réel
État :	Soutenue le 9 avril 2009, à l'Université de Nice-Sophia Antipolis
Encadrement :	Michel Auguin (Directeur de thèse), Fabrice Muller (Co-directeur, 33%)
Situation :	R&D à SUPARCO (Space & Upper Atmosphere Research Commission) au Pakistan

Résumé :

Les performances des algorithmes d'ordonnancement ont un impact direct sur les performances du système complet. Les algorithmes d'ordonnancement temps réel possèdent des bornes théoriques d'ordonnançabilité optimales mais cette optimalité est souvent atteinte au prix d'un nombre élevé d'événements d'ordonnancement à considérer (préemptions et migrations de tâches) et d'une complexité algorithmique importante. Notre opinion est qu'en exploitant plus efficacement les paramètres des tâches il est possible de rendre ces algorithmes plus efficaces et à coût maitrisé, et afin d'améliorer la Qualité de Service (QoS) des

applications. Nous proposons dans un premier temps des algorithmes d'ordonnancement monoprocesseur qui augmentent la qualité de service d'applications hybrides c'est-à-dire qu'en situation de surcharge, les tâches à contraintes souples ont leur exécution maximisée et les échéances des tâches à contraintes strictes sont garanties. Le coût d'ordonnancement de ces algorithmes est aussi réduit (nombre de préemptions) par une meilleure exploitation des paramètres implicites et explicites des tâches. Cette réduction est bénéfique non seulement pour les performances du système mais elle agit aussi positivement sur la consommation d'énergie. Aussi nous proposons une technique associée à celle de DVFS [8] afin de minimiser le nombre de changements de points de fonctionnement étant donné qu'un changement de fréquence implique un temps d'inactivité du processeur et une consommation d'énergie.

Les algorithmes d'ordonnancement multiprocesseur basés sur le modèle d'ordonnancement fluide (notion d'équité) atteignent des bornes d'ordonnançabilité optimales. Cependant cette équité n'est garantie qu'au prix d'hypothèses irréalistes en pratique du fait des nombres très élevés de préemptions et de migrations de tâches qu'ils induisent. Dans cette thèse un algorithme est proposé (ASEDZL [9]) qui n'est pas basé sur le modèle d'ordonnancement fluide. Il permet non seulement de réduire les préemptions et les migrations de tâches mais aussi de relâcher les hypothèses imposées par ce modèle d'ordonnancement. Enfin, nous proposons d'utiliser ASEDZL dans une approche d'ordonnancement hiérarchique ce qui permet d'obtenir de meilleurs résultats que les techniques classiques.

Composition du jury :

M. Pascal Richard (Professeur, Université de Poitiers, rapporteur), M. Yvon Trinquet (Professeur, Université de Nantes, rapporteur), M. Joël Goossens (Professeur, Université Libre of Brussels, examinateur), M. Robert de Simone (Directeur de recherches INRIA, Sophia Antipolis, examinateur), M. Michel Auguin (Directeur de recherches CNRS LEAT, Sophia Antipolis, directeur de thèse), M. Fabrice Muller (Maître de conférences, Université de Nice-Sophia Antipolis, co-directeur de thèse)

B - Thèse de Ikbel Belaid

Années :	2008/2011
Titre :	Placement et Ordonnancement statique et dynamique de tâches matérielles temps-réel sur plateformes reconfigurables dynamiquement
État :	Soutenue le 11 juillet 2011, à l'Université de Nice-Sophia Antipolis
Encadrement :	Fabrice Muller (Directeur de thèse, 100%)
Situation :	Post-Doct au TIMA, équipe SLS, à Grenoble à partir d'octobre 2011

Résumé :

Le placement et l'ordonnancement des tâches matérielles sont les éléments clés du système d'exploitation temps-réel. Ces deux problèmes doivent être traités efficacement afin d'améliorer la qualité du placement exprimée par le taux de fragmentation de ressources, la latence de

8. Dynamic Voltage and Frequency Scaling
9. Anticipating Slack Earliest Deadline first until Zero Laxity

reconfiguration, ainsi que la qualité d'ordonnancement représentée par la durée d'exécution de l'application et la garantie des échéances. En utilisant les systèmes sur puce programmable, nous proposons d'exploiter les caractéristiques physiques de ces puces, en particulier la reconfiguration partielle dynamique.

Nous traitons, dans un premier temps, les tâches indépendantes. Nous suggérons une résolution analytique par des solveurs de programmation en nombres entiers mixtes qui se basent sur la méthode de séparation et évaluation pour réaliser le placement hors-ligne de ces tâches sur puce. La métaheuristique des abeilles est aussi proposée pour traiter ce problème. Nous proposons d'employer l'algorithme EDF [10] pour construire l'ordonnancement temps-réel en-ligne.

Nous nous intéressons ensuite aux tâches dépendantes. En se basant également sur la programmation en nombres entiers mixtes, le placement et l'ordonnancement statiques des tâches matérielles périodiques, constituant un graphe acyclique orienté, sont élaborés. Quatre approches dynamiques sont proposées pour effectuer le placement et l'ordonnancement dynamique de plusieurs graphes sur différentes puces. Par les techniques de réutilisation et de prédiction, ces approches visent la réduction des temps d'exécution des graphes, la garantie des échéances et l'efficacité des ressources. Ces travaux sont effectués dans le cadre du projet ANR FOSFOR.

Composition du jury :

M. Loïc Lagadec (Maître de conférences HDR, Université de Bretagne occidentale, rapporteur), M. Gilles Sassatelli (Directeur de recherche CNRS / LIRMM, rapporteur), M. Bertrand Granado (Professeur, Université de Cergy Pontoise, examinateur), M. François Verdier (Professeur, Université de Nice-Sophia Antipolis, examinateur), M. Fabrice Muller (Maître de conférences, Université Nice-Sophia Antipolis, directeur de thèse)

C - Thèse de Clément Foucher

Années :	2009/2012
Titre :	Déploiement d'un flot de conception pour des applications parallèles sur des architectures distribuées et reconfigurables dynamiquement
État :	En cours
Encadrement :	Alain Giulieri (Directeur de thèse, 30%), Fabrice Muller (co-encadrant, 70%)

Résumé :

L'objectif global de la thèse est la mise au point d'un flot de conception pour les applications décrites à divers degré de parallélisme et tirant parti des circuits reconfigurables dynamiquement. Ce flot de conception est imaginé dans une optique d'architecture auto-adaptative distribuée gérant de manière efficace le parallélisme des applications. De plus, afin de prouver ces nouveaux concepts de gestion des ressources pour le reconfigurable, plusieurs implémentations sont réalisées.

10. Earliest Deadline First

Dans un premier temps, les travaux portent sur la réalisation d'une plateforme de type High Performance Computer (HPC) sur la base d'un réseau de circuits reconfigurables. L'objectif est de valider de manière préliminaire l'approche d'une architecture générique auto-adaptative (reconfigurable) à l'aide d'une première plateforme distribuée.

Dans un second temps, les travaux se focalisent sur la mise au point du flot de conception d'applications, ainsi que sur la réalisation d'une seconde plateforme introduisant des aspects de reconfiguration dynamique partielle. Concernant ce flot de conception, divers objectifs ont été spécifiés afin de fixer les hypothèses. Parmi l'un des principaux objectifs se trouve l'idée de virtualisation des noyaux de calcul (portion d'application contenant des calculs spécifiques) afin de s'abstraire de leur type d'exécution réel. Ainsi, chaque noyau peut disposer de plusieurs implémentations, matérielles et/ou logicielles. Le choix de l'implémentation utilisée est réalisé en-ligne en fonction des ressources disponibles.

Conjointement à la description de ces travaux, les caractéristiques de la plateforme générique auto-adaptative ont été spécifiées sous le nom de Simple Parallel platfOrm for Reconfigurable Environment (SPORE). La plateforme reprend ainsi les grandes caractéristiques des HPCs, à savoir une architecture mémoire globalement distribuée, localement partagée. À ces caractéristiques sont ajoutées des contraintes imposant la possibilité de reconfiguration dynamique pour les unités d'exécution.

Enfin, une seconde implémentation de la plateforme générique d'exécution a été réalisée afin de prendre en considération l'aspect dynamique imposé par les spécifications de SPORE. Cette plateforme introduit le caractère reconfigurable des noyaux, l'accès à l'interface des noyaux de manière générique, et supporte une implémentation du flot applicatif via des descriptions au format XML [11]. Ceux-ci permettent de décrire la structure de l'application, des noyaux la composant ainsi que divers éléments nécessaires au dialogue avec les noyaux.

L'ensemble de ces plateformes est géré à l'aide d'un hôte (système d'exploitation Linux embarqué), qui permet la gestion de l'ordonnancement, l'interprétation des fichiers XML ainsi que la gestion de la reconfiguration via des pilotes spécifiquement développés.

Cet ensemble plateforme/environnement d'exécution permet l'automatisation complète de l'exécution de l'application qui est distribuée et utilise au mieux les ressources reconfigurables. Le choix de l'implémentation à utiliser ainsi que la gestion de la reconfiguration en ligne sont gérés à la volée par un ordonnanceur s'exécutant sur le système d'exploitation Linux embarqué.

D - Thèse de François Duhem

Années :	2009/2012
Titre :	Modélisation haut niveau et amélioration des performances pour la gestion d'architectures reconfigurables dynamiquement
État :	En cours
Encadrement :	Philippe Lorenzini (Directeur de thèse, 30%), Fabrice Muller (co-encadrant, 70%)

Résumé :

11. Extensible Markup Language

Les travaux de thèses sont organisés autour de deux axes majeurs. Tout d'abord, la conception et l'optimisation en performance d'un contrôleur de reconfiguration dynamique. Cette technique introduite dans les derniers FPGA de Xilinx permet de changer la fonctionnalité d'une partie définie d'un système pendant que le reste du système continue de fonctionner normalement, permettant ainsi un gain de ressources et/ou une baisse de la consommation d'énergie. Néanmoins, le contrôleur de gestion du reconfigurable de Xilinx n'est pas assez performant en terme de temps de reconfiguration pour permettre une utilisation efficiente de la reconfiguration dynamique et ainsi de limiter l'overhead. Ces travaux ont abouti à un module FaRM [12] optimisant et atteignant le débit théorique de reconfiguration dans certaines conditions. De plus, la réduction de l'overhead a été également facilitée par la mise au point d'un algorithme de compression du bitstream partiel allant jusqu'à 90%. L'évaluation des performances a été réalisée à partir d'une application de cryptographie (AES [13]) et d'une application vidéo H264 qui montrent que l'utilisation de FaRM permet d'atteindre les contraintes temps-réel contrairement au contrôleur de Xilinx.

La seconde partie du travail consiste en la modélisation du concept de reconfiguration dynamique à un haut niveau d'abstraction. En effet, la technologie est encore assez jeune et manque de tels modèles. Ceux-ci permettraient aux développeurs d'applications matérielles de valider leur description en utilisant la reconfiguration dynamique dès les premières étapes du développement. La contribution consiste donc en un simulateur écrit en SystemC permettant d'obtenir le nombre et le placement des zones reconfigurables dynamiques et partielles en fonction d'un algorithme d'ordonnancement qui s'exécutera ensuite sur la plateforme. La description se fait au niveau architectural en termes de zones reconfigurables ainsi qu'au niveau applicatif. Le cas d'étude est une application de test consistant en une chaîne de transcodage qui permet de changer l'encodage d'une vidéo (par exemple H.264 vers MPEG-2). Le simulateur permet notamment de tester différents algorithmes d'ordonnancement afin d'optimiser la gestion de la reconfiguration dynamique. De plus, il utilise une description des communications au niveau transactionnel (TLM [14]). Cela permet une abstraction des communications permettant de s'affranchir de certains détails d'implémentation ainsi que de considérablement réduire le temps de simulation tout en gardant une précision suffisante. Ces travaux sont effectués dans le cadre du projet ANR ARDMAHN.

1.3.4.2 Encadrements de stages de DEA, Master, ingénieur ou technicien

Nous accueillons différents étudiants pour des stages de recherche. Ces stages ont pour but, soit de réaliser des études amonts, soit d'aider les doctorants à creuser l'un des points de recherche en cours, ou bien encore de réaliser des développements ou des expérimentations afin de valider mes approches à des fin de publications.

- **Bassem Ouni :** MASTER (ENIS, École Nationale d'Ingénieurs de Sfax), Année 2009,

12. Fast Reconfigurable Manager
13. Advanced Encryption Standard
14. Transaction Level Modeling

Titre : *Placement et ordonnancement des tâches matérielles sur des zones reconfigurables en utilisant le Bees Algorithm*

- **Clément Foucher :** Stage Ingénieur (Polytech'Nice-Sophia, Filière Électronique),
  Année universitaire 2008-2009,
  Titre : *Conception d'un OS matériel compatible RTEMS pour l'exécution de tâches matérielles*
- **Emmanuel Vallois & Thomas Lebrun :** Projet Ingénieur (Polytech'Nice-Sophia, Filière Informatique),
  Année universitaire 2008-2009,
  Titre : *Développement d'un outil d'analyse de traces pour un noyau temps-réel matériel*
- **Laurent Rodriguez :** Ingénieur d'Étude, financé sur le projet Européen ÆTHER,
  Année universitaire 2007-2008,
  Titre : *Implementation of a SystemC simulator allowing validation of functionalities in an « execution environment » for the ÆTHER project*
- **Nguyen Anh Dung :** MASTER2, Université de Nice-Sophia-Antipolis, dépt. EEA,
  Année universitaire 2007-2008,
  Titre : *Study and evaluation the performances of dynamic reconfiguration in SoPC (Virtex-5)*
- **Ikbel Belaid :** MASTER2, (ENIT, École Nationale d'Ingénieurs de Tunis),
  Année universitaire 2007-2008,
  Titre : *Dynamic Partial Reconfiguration in SoPC*
- **XianXian Niu :** MASTER2, Université de Nice-Sophia-Antipolis, dépt. EEA,
  Année universitaire 2006-2007, Encadrement à 50%
  Titre : *SystemC modeling of an operating environment in self-adaptive system*
- **Geoffrey Austrate :** MASTER2, Université de Nice-Sophia-Antipolis, dépt. EEA,
  Année universitaire 2005-2006, Encadrement à 50%
  Titre : *Modélisation d'architecture de communication temps réel déterministe pour MPSOC*
- **Nicole Chalhoub :** MASTER2, Université de Nice-Sophia-Antipolis, dépt. EEA,
  Année universitaire 2005-2006,
  Titre : *Étude de l'implémentation d'un réseau de neurones sur FPGA et détermination des axes de recherche sur la reconfiguration partielle*
- **Farooq Muhammad :** MASTER2, Université de Nice-Sophia-Antipolis, dépt. EEA,
  Année universitaire 2005-2006,
  Titre : *Contention-Conscious Dynamic But Deterministic Scheduling Of Computational And Communication Tasks*
- **Éric Charpentier :** Projet Ingénieur (ESIEE, Sophia-Antipolis),
  Année universitaire 2004-2005,
  Titre : *Development of an hardware, multiprocessor Real Time Operating System*
- **Florian Broekaert :** Projet Ingénieur (ESIEE, Sophia-Antipolis),
  Année universitaire 2004-2005,
  Titre : *echOS Embedded and Configurable Hardware-Software Operating System, Modélisation Multiprocesseur & TLM*

- **Émilien Perrault :** Projet Ingénieur (ESIEE, Sophia-Antipolis),
  Année universitaire 2003-2004,
  Titre : *Embedded and Configurable Hardware-software Operating System*

1.3.5 Relations scientifiques

1.3.5.1 Participation à des jurys de thèse

- Ludovic Devaux : Thèse de doctorat
  Prévue le 24 novembre 2011, à l'Université de Rennes 1
  Intitulé : *Flexible interconnexion networks for dynamically reconfigurable architectures*
  Jury :
  - Fernando Moraes, PrU à PUCRS (Brésil), Rapporteur
  - Jean-Philippe DIGUET, Directeur de Recherche CNRS, Rapporteur
  - Frédéric Pétrot, PrU à l'ENSIMAG, Examinateur
  - Fabrice Muller, MdC à l'Université de Nice Sophia Antipolis, Examinateur
  - Daniel Chillet, MdC HDR à l'Université de Rennes 1, Examinateur
  - Didier Demigny, PrU à l'Université de Rennes 1, Directeur de thèse
  - Sébastien Pillement, MdC HDR à l'Université de Rennes 1, Co-encadrant

- Ikbel Belaid : Thèse de doctorat
  Soutenue le 11 juillet 2011, à l'Université de Nice-Sophia Antipolis
  Intitulé : *Placement et Ordonnancement statique et dynamique de tâches matérielles temps-réel sur plateformes reconfigurables dynamiquement*

- Farroq Muhammad : Thèse de doctorat
  Soutenue le 9 avril 2009, à l'Université de Nice-Sophia Antipolis
  Intitulé : *Ordonnancement de Tâches Efficace et à Complexité Maitrisée pour des Systèmes Temps Réel*

1.3.5.2 Participation à des projets nationaux

- **Projet ARDMAHN** (Octobre 2009 - Novembre 2012)
  « Architecture Reconfigurable Dynamiquement et Méthodologie pour l'Auto-adaptation en Home Networking »
  Type de contrat : Projet ANR/ARPEGE, Coordinateur du projet : LIEN
  Budget total/LEAT : 1M€ / 150K€
  Responsable scientifique pour le LEAT : Fabrice Muller
  Partenaires : IMS (Laboratoire de l'Intégration du Matériau au Système, Bordeaux), LaBRI (Laboratoire Bordelais de Recherche en Informatique, Bordeaux), LEAT/UNS, LIEN (Laboratoire d'Instrumentation Electronique de Nancy), THOMSON Video Network (Rennes), Viotech (Montigny le Bretonneux)
  site web : `http://ardmahn.org`

- **Projet ANR FOSFOR** (Janvier 2008 - Décembre 2011)

Labellisé par le Pôle de Compétitivité ScS [15]
« Flexible Operating System FOr Reconfigurable platform »
Type de contrat : Projet ANR/ARPEGE, Coordinateur du projet : LEAT
Budget total/LEAT : 723K€ 163K€ dont 13K€ du pôle ScS
Responsable scientifique pour le LEAT : Fabrice Muller
Partenaires : ETIS (Équipes Traitement des Images et du Signal), IRISA-CAIRN (Institut de Recherche en Informatique et Systèmes Aléatoires), UNS/LEAT, TRT [16]
site web : `http://www.polytech.unice.fr/~fmuller/fosfor/`

- **Projet Sys2RTL** (Septembre 2006 - Septembre 2008)
  Dans le cadre de la plateforme Conception de CIMPACA [17]
  « Plateforme de validation de SoC[18] *décrits par interconnexion d'IP inter-opérables »*
  Type de contrat : Bourse Doctorale cofinancée Région-entreprise
  Responsable scientifique pour l'I3S : Michel Auguin
  Partenaires : SYNOPSYS, UNS (Université de Nice-Sophia Antipolis)
  La thèse associée à ce projet a été arrêtée par l'étudiant pour raisons personnelles. Les travaux ont donné lieu à une publication [39]
  Directeur de thèse : Michel Auguin, Co-directeur : Fabrice Muller

1.3.5.3 Participation à des projets internationaux

- **Projet DGRS Tunisie/CNRS** (2010 - 2013)
  « Conception et exploration d'architectures faible consommation pour les applications multimédia »
  Type de contrat : DGRST/CNRS
  Budget LEAT : 6K€
  Responsable scientifique pour le LEAT : Fabrice Muller
  Partenaires : École Nationale d'Ingénieurs de Sfax/ Université de Sfax, Équipe ReD-CAD, LEAT/CNRS

- **Projet Européen ÆTHER** (Janvier 2006 - Juin 2009)
  « Self-Adaptive Embedded Technologies for Pervasive Computing Architectures »
  Type de contrat : Projet européen IST-FET (4^{th} call ACA / FP6)
  Budget Total/CNRS : 4M€ / 251K€
  Responsable scientifique pour l'I3S/LEAT : Michel Auguin
  Partenaires : CEA/LIST (Leader), CNRS (I3S/LESTER, FR), Univ. Amsterdam (UVA, NL), Univ. Karlsruhe (ITIV, DE), Imperial College of London (UK), Univ. Politècnica de Catalunya (UPC, ES), TRT (Thales Research & Technology, FR), VTT (FI), ATMEL (GR), INTRACOM (GR), Univ. Prague (UTIA, CZ), ACIES (administration, FR), Università della

15. Solutions Communicantes Sécurisées, `www.pole-scs.org`
16. THALES Research and Technology
17. Centre Intégré de Microélectronique Provence-Alpes-Côte d'Azur, `www.arcsis.org/cimpaca.htm`
18. System on Chip

Italiana (USI/ALARI, CH), University of Hertfordshire (UK)

- **Projet Européen CoMES** (Juillet 1998 - Janvier 2000)
  « Codesign Methodology for Embedded Systems »
  Type de contrat : projet Européen Esprit 26971
  Responsable scientifique pour l'IRESTE, Équipe MCSE : Jean-Paul Calvez
  Partenaires : Matra Bae Dynamics (Velizy), Space division of INDRA (Barcelona, SP) with University of Cantabria, SIDSA (Madrid), IRESTE Équipe MCSE (Nantes)

1.3.5.4 Activités d'expertise et animation de la recherche

- Membre du comité de lecture pour les conférences
  - SympA'2011, Symposium en Architectures nouvelles de machines
  - IEEE NEWCAS'2010
  - DASIP'2009, Design & Architectures for Signal & Image Processing
  - IEEE IES'2006, Industrial Embedded Systems
- Membre du comité de lecture pour les revues
  - EURASIP Journal of Parallel and Distributed Computing (2011)
  - EURASIP Journal on Embedded Systems (2008)
  - TSI, Technique et Science Informatiques (2006)
- Membre du comité d'organisation et Président de la session « Work in Progress » pour la conférence IES (2006)
- Membre du comité d'organisation de la conférence ETFA (2001)
- Membre du comité de lecture pour d'évaluation de projet pour l'ANR (2008)
- Membre du comité de lecture pour l'appel d'offre projet Européen CATRENE (2008)
- Commissions
  - Membre externe du Comité de Sélection (CNU 61) à ENSEIR-MATMECA, Bordeaux (2011)
  - Membre élu de la Commission de spécialistes (CNU 61) de l'Université de Nice Sophia Antipolis depuis 2001 jusqu'à sa dissolution en 2008.

De plus, je participe régulièrement au GDR SoC/SiP, AS Architecture reconfigurable dynamiquement.

1.3.5.5 Valorisation de la recherche et transfert technologique

Dans le cadre de ma thèse de doctorat et en plus des concepts de recherche illustrés dans mon manuscrit de thèse, j'ai en partie développé un outil MCSE ToolBox au sein de l'équipe MCSE à l'IRESTE. Cet outil a fait l'objet d'un transfert technologique et a donné naissance en 2003 à une startup Cofluent Design [94] qui a été acquis par Intel en septembre 2011.

J'ai également participé à rechercher de grands industriels pour Cofluent Design. Ceci s'est concrétisé par un contrat de licences avec la société TI (Texas Instrument, Villeneuve Loubet) au niveau mondial entre 2004-2006. De plus, un contrat de 3 ans a été signé dans le cadre de CIM PACA où un ensemble de PME peuvent accéder à ces outils (2006).

J'ai également des activités de recherche avec TRT (Thales Research & Technology) dans

le cadre d'un projet national (ANR FOSFOR) mais également Européen (Projet ÆTHER) afin de réaliser du transfert technologique. Par exemple, j'ai travaillé et conçu les services de base de l'OS FOSFOR pour que TRT puisse, en autre, l'évaluer et peut être l'intégrer dans de futures architectures.

De même, j'ai des relations privilégiées depuis 2006 avec Xilinx afin d'évaluer et d'apporter des solutions plus efficaces pour la technique de reconfiguration partielle de FPGA. De plus, j'essaye d'intégrer Xilinx dans les projets de recherche comme par exemple dans le projet FOSFOR où Xilinx est partenaire technologique. Un autre exemple concerne une collaboration avec la société NodBox et la société Xilinx dont l'objectif était de vérifier que le réseau de neurone spécifié par NodBox et que j'ai évalué et implémenté, répondait à leur problématique de prédiction des risques de perte de contrôle du véhicule pendant la phase de conduite normale.

les sociétés Thomson Video Network (Rennes) et Viotech (Montigny le Bretonneux) sont également intéressées pour intégrer encore plus de reconfigurable dans les futurs set top box grâce au projet ANR ARDMAHN en cours et dont j'ai été moteur lors de la phase de montage du projet et de recherche de partenaires.

Depuis septembre 2011, je suis en train de mettre en place une collaboration avec TUS (Thales Underwater System, Sophia Antipolis) sur la problématique de l'intégration du reconfigurable (FPGA) et du GPU pour des applications hautes performances en proposant des solutions de langages uniformes. Ceci se traduira par un stage de MASTER2 débutant en février 2012 et une thèse CIFRE qui démarrera en septembre 2012.

1.3.6 Production scientifique

Le tableau 1.2 résume la production scientifique [19]. La bibliographie détaillée est présentée en fin de chapitre, section 1.5. En outre, une sélection des publications significatives est présentée en annexe A.

Tableau 1.2 – *Résumé du nombre de production scientifique.*

Type d'article	DEA/Thèse				Maître de Conférences												
	96	97	98	99	00	01	02	03	04	05	06	07	08	09	10	11	Σ
Brevet													1				1
Chap. Ouvrages																1	1
Revues int.															2	3	5
Conférences int.			1	2		1					3	4	6	4	2	6	29
Conférences nat.	1			1								1	2	3	2	2	12
Papiers invités												1	1				2
Journaux et Colloques Industriel/Pédagogique							1				1		3	1			6

J'ai actuellement deux revues internationales en première soumission [10, 11] et une revue nationale en première soumission [41].

19. Les premières années ont été plutôt consacrées à la création de modules d'enseignement comme l'explique la section 1.4.2 et comme l'illustre la figure 1.2

1.3.7 Perspectives de Recherche

Je souhaite continuer mes recherches dans les architectures reconfigurables, auto-adaptatives. En effet, ces systèmes hétérogènes intègrent de plus en plus de réseaux de capteurs intelligents ou de réseaux ad hoc d'objets communicants mobiles. D'ailleurs, ceci va dans le sens du projet Européen ÆTHER. Le reconfigurable a clairement un rôle à jouer grâce à sa dynamicité. De plus, il est évident que des calculs réalisés en matériel consomment moins d'énergie qu'en logiciel. Et le problème principal de ces réseaux est justement cette consommation d'énergie excessive. Quelle est le bon compromis qui dépend probablement de l'environnement à l'instant présent, de la qualité de service associée et d'autres paramètres à analyser ? Mon objectif est de proposer des idées et démonstrateurs afin de contribuer à répondre à ces questions.

La technologie évolue plus vite que le domaine de la conception, c'est-à-dire que la technologie proposée n'est pas exploitée correctement. Prenons l'exemple de la technologie 3D qui promet des avantages considérables. Cependant, le défi est l'exploitation de l'espace de conception qui est incontournable pour concevoir des circuits efficaces. Mon objectif serait de proposer des algorithmes en avance de phase sur cette technologie 3D.

Depuis de nombreuses années, l'effort s'est principalement focalisé sur des flots en Y afin de concevoir des systèmes embarqués, de complexité croissante, à partir de descriptions de plus haut niveau. Mon défi serait de prendre en compte au sein de ces flots d'autres types de spécifications que des spécifications structurelles et fonctionnelles comme les contraintes de mobilité, les réseaux de capteurs, la flexibilité du reconfigurable, la tolérances aux fautes. D'ailleurs, les réseaux de capteurs et les réseaux ad hoc font partie d'un Labex [20] en cours de montage où je serais impliqué.

De même, les systèmes d'exploitation ne demandent qu'à évoluer pour répondre aux applications futures. Cela concerne aussi bien les OS embarqués que les OS plus lourds permettant de faire du calcul intensif comme du HPC. L'un des problèmes majeurs du HPC est également la consommation d'énergie. Le HPRC peut répondre en partie à ce problème. Je travaille actuellement en ce sens sur la mixité des modules de calcul matériel/logiciel.

Concernant les architectures hautes performances basées sur un ensemble de FPGAs, GPUs et CPUs [21], un autre problème que la consommation est la description d'applications. Dans le cadre d'une thèse CIFRE avec TUS que je suis en train de négocier, mon objectif sera d'évaluer et de proposer des solutions afin d'exécuter dynamiquement des algorithmes de traitements gourmands en puissance de calcul sur ces architectures hétérogènes.

1.3.8 Réflexion sur le métier de chercheur

Depuis ces 10 dernières années, j'ai eu l'occasion d'encadrer des doctorants, des étudiants de DEA/MASTER2, un ingénieur d'étude et des stagiaires de niveau technicien provenant d'IUT ou d'école d'ingénieur 2^{e} année. De même, j'ai participé et monté des projets comme ÆTHER, FOSFOR, ARDMAHN. J'ai également participé à des conférences de diverses manières.

20. Laboratoires d'excellence
21. Central Processor Units

Je pense que c'est de part ces activités que nous sommes considérés comme des chercheurs. Les étudiants sont une ressource fondamentale pour que nous puissions réussir nos projets de recherche et que ces derniers puissent devenir eux-même de bons chercheurs. Cependant, la pénurie d'étudiants se fait sentir depuis ces dernières années, m'obligeant, et cela doit être vue comme une opportunité, à rechercher encore plus de collaborations internationales. J'ai par exemple eu des étudiants Pakistanais, Tunisiens, Chinois, Libanais.

L'animation scientifique est également primordiale. Celle-ci permet d'échanger, d'ouvrir l'esprit des doctorants mais également des chercheurs qui sont souvent dans leurs projets. Depuis plusieurs années, mensuellement, au sein de notre thème MCSOC, 2 ou 3 doctorants présentent leurs travaux aux autres. De plus, depuis cette année, nous avons mis en place des journées afin que chaque doctorant, MASTER2, stagiaires puissent exposer leurs travaux. De toute manière, ces échanges sont très valorisant pour l'étudiant et permettent de mieux appréhender son travail avec notre aide.

La dissémination des résultats reste toujours un problème délicat. Pour ma part, les conférences et les journaux/revues sont une première étape qui permet ensuite de valoriser notre recherche. Par exemple, les travaux que je mène actuellement sur les architectures reconfigurables va permettre de décrocher ma première thèse en CIFRE. C'est par ces publications et en instaurant un climat de confiance et de compétences que ceci est possible. Dans notre domaine, je pense qu'il est essentiel de coopérer étroitement avec les industriels. De même, pour le montage de projets nationaux ou Européens, c'est en rencontrant d'autres chercheurs lors de conférences ou autre, et en utilisant comme support nos disséminations que ces projets peuvent émerger, comme cela a été le cas pour mes projets de recherche.

L'un des points fondamental pour la recherche est l'espace de liberté. En effet et contrairement à l'industrie, le chercheur doit passer du temps à comprendre les problèmes, à suivre différentes pistes de recherche sans avoir, dans une certaine mesure, de contraintes de performances. De plus, la recherche se dessine de plus en plus multi-disciplinaire ce qui demande un recul encore plus important.

Pour finir, l'un de nos rôles est également de préparer des étudiants voulant devenir de futurs chercheurs (au sens large) et à découvrir les différentes facettes de la recherche et du métier d'enseignant/chercheur. Pour ma part, j'embauche généralement pendant la période d'été un ou deux étudiants de 2^{e} année pour travailler avec un ou plusieurs doctorants sur un point particulier. C'est très enrichissant pour l'étudiant stagiaire qui apprend ce qu'est la vie d'un laboratoire de recherche.

1.4 Activités d'enseignement et administratives

Lors de mon recrutement en tant que Maître de Conférences à l'ESINSA devenu Polytech'-Nice-Sophia - Filière Électronique, l'un des objectifs fût d'analyser les cours existants en informatique industrielle. Cette étude a abouti à des adaptations et à des créations de modules complets (Cours/TD/TP). L'un des plus importants travaux a été d'imaginer une nouvelle option GSE [22] et en collaboration avec des industriels. Durant la même période, une forma-

22. Génie des Systèmes Embarqués

tion par alternance ITII [23] sera également créée. La partie d'électronique numérique de cette formation m'a été également confiée. Dans le même sens, j'ai participé à des enseignements au CNAM [24] de Nice pendant quelques années.

Un autre point important concerne la relation pédagogique vis-à-vis de l'extérieur. Je m'efforce à faire en sorte que notre école soit présente au sein de colloques, conférences afin de mettre en valeur notre enseignement. Je participe notamment régulièrement aux journées pédagogiques du GIP CNFM [25] [59, 60, 61] ou autres [58, 38].

1.4.1 Création de l'option GSE

La première rentrée de l'option GSE dont j'ai été co-responsable pendant 3 ans, a démarré en février 2004 (Semestres 8 et 9 du cursus ingénieur). Cette option, comme toutes les options de la filière électronique, permet de donner une « coloration » scientifique aux futurs ingénieurs. En effet, la formation d'ingénieur dans la filière électronique reste dans l'ensemble généraliste. En d'autres termes, certains modules de l'option sont communs avec les autres options et d'autres modules sont spécifiques à l'option. L'objectif de cette option est de former des concepteurs de systèmes temps-réel embarqués intégrant une double compétence matériel/logiciel partant du niveau RTL [26] jusqu'au niveau système.

Cette option, ayant toujours du succès, est une source de doctorants. En effet, certains étudiants, formés à notre domaine des systèmes embarqués, font leur stage d'été dans notre laboratoire. C'est ensuite qu'ils accomplissent leur stage d'ingénieur également dans un laboratoire de recherche ou dans l'industrie. Certains d'entre eux sont recrutés en thèse. C'est le cas de mes deux doctorants Clément Foucher (stage ingénieur sur FOSFOR), François Duhem (stage ingénieur en entreprise), etc...

1.4.2 Récapitulatif des Enseignements

La figure 1.2 résume mon activité d'enseignement à partir de mon année de recrutement comme Maître de Conférences par cycle de formation. Ces enseignements ont été soit modifiés (couleur orangée), soit créés en totalité (couleur bleue). De plus, je participe à quelques enseignements déjà créés dont je ne suis d'ailleurs pas responsable.

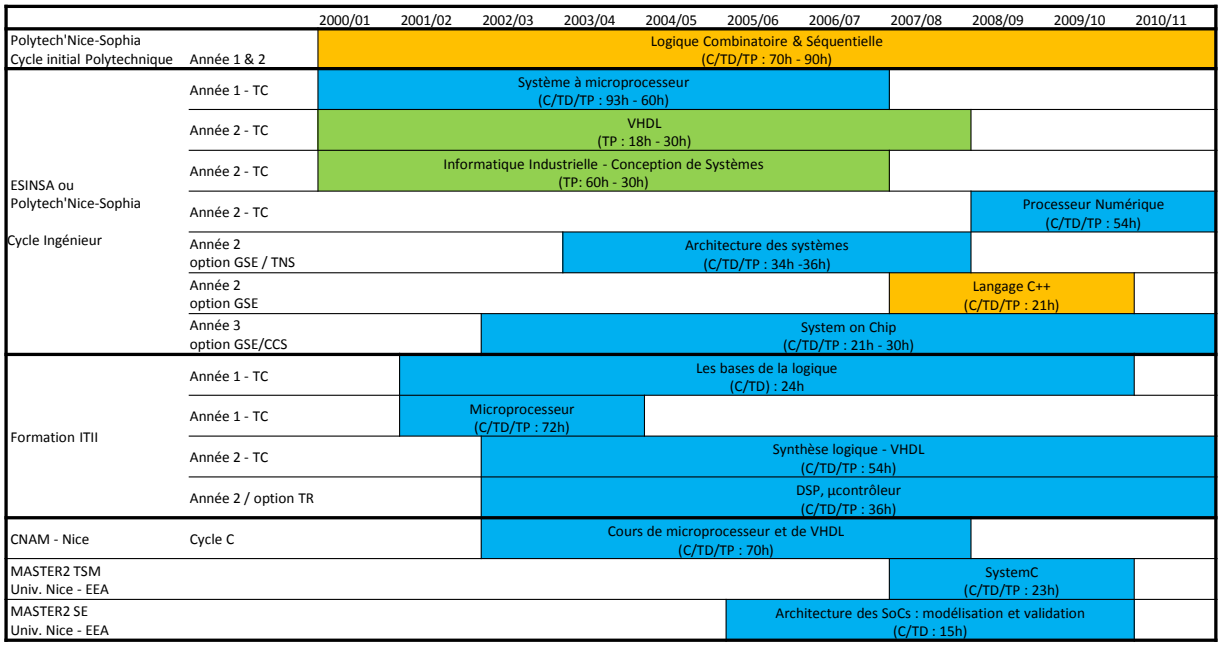
Depuis le début de mon activité, j'ai toujours enseigné dans le cycle préparatoire intégré à l'ESINSA, puis à Polytech'Nice-Sophia. Cette activité est très importante car elle prépare partiellement à la filière Électronique comme à d'autres filière (informatique, etc.). De plus, la présence d'enseignants/chercheurs permet d'essayer de donner un certain recul aux étudiants dans le domaine de l'électronique. J'essaye de modifier, si nécessaire, les travaux pratiques. Par exemple, durant ces trois dernières années, j'ai renouvelé le matériel de TP, modifier les TPs en conséquence et, l'année dernière, ajouté un TP pour que les étudiants puissent utiliser des outils industriel de CAO (QuartusII d'Altera).

23. Institut des Techniques d'Ingénieur de l'Industrie
24. Conservatoire National des Arts et Métiers
25. groupement d'intérêt public pour la Coordination Nationale de la Formation en Microélectronique et en nanotechnologies
26. Register Tranfert Level

Concernant le cycle ingénieur, filière Électronique, il a fallu refaire complètement le cours de microprocesseur. J'ai conçu entièrement de nouvelles cartes dédiées au TPs. Lors de la création de l'option GSE, des modules complets cours/TD/TP ont été créés : architectures, System on Chip, SystemC.

Certains de ces modules créés m'ont servi de base pour la formation ITII et CNAM. Pour ma part, il est très important d'enseigner en formation continue ou alternance. Ces étudiants n'ont pas la même vue de l'enseignement car ils sont déjà intégrés dans l'entreprise. Cela permet à l'enseignant de modifier son approche pédagogique. En effet, ces enseignements, bien que partiellement repris, ne sont pas identiques à la formation cycle ingénieur initiale.

De plus, l'enseignement en MASTER2 est également important pour le recrutement des futurs doctorants, même si nous recrutons aussi des étudiants de Polytech'Nice-Sophia comme précisé dans la section 1.4.1.



↪ Les heures sont en équivalent TD
↪ Légende : Matière (C/TD/TP : heure en début d'activité de la matière / heure en 2010-2011)
TC : Tronc Commun
↪ Couleur : Verte → Réalisée par un autre enseignant
Orangée → Reprise & Modification de la matière (responsable de la matière)
Bleue → Création de la matière (responsable de la matière)

Figure 1.2 – *Résumé de l'activité d'enseignement.*

Dans le cursus ingénieur, nous organisons également des projets avec des industriels lors du second semestre de la deuxième année d'école d'ingénieur. Le principe est d'avoir un industriel et un enseignant qui encadre un binôme ou trinôme d'étudiant le vendredi après-midi sur un sujet défini principalement par l'industriel. Cette relation est importante pour l'école et permet aux étudiants d'avoir une première expérience un peu industrielle et facilite,

dans certain cas, la recherche de stages.

Un autre point important concerne le suivi des étudiants où j'interviens en stage de technicien et d'ingénieur (environ 3 à 5 suivis d'étudiant par an). Ce suivi permet d'avoir également une relation privilégiée avec l'industriel lors de la visite de stage. Cela permet de discuter de l'évolution de l'enseignement que l'on adapte au niveau des options en dernière année d'école d'ingénieur.

Pour finir, je fais parti régulièrement de jurys de mémoires CNAM (1 jury par an en moyenne).

1.4.3 Responsabilités pédagogiques et administratives

Responsabilités actuelles

- Directeur des études, Polytech'Nice Sophia, Filière Électronique en 2e et 3e année du cycle ingénieur (depuis septembre 2011)
- Membre élu du conseil de département, Polytech'Nice-Sophia, Filière Électronique, depuis juillet 2008, ré-élu en juillet 2011
- Responsable de l'électronique numérique (Cours / TD / TP) en Cycle initial Polytechnique (CiP1 et CiP2)
- Pôle CNFM PACA, site de Nice-Sophia Antipolis : responsable de la salle dédiée à l'enseignement Systèmes Embarqués à Polytech'Nice-Sophia et des demandes de licences des logiciels correspondants.

Responsabilités antérieures

- Co-Responsable de l'option Génie des Systèmes Embarqués (Dept Elec) jusqu'à août 2007
- Responsable de la communication pour le département Électronique à Polytech'Nice-Sophia (uniquement en 2008). Une personne a été embauchée à Polytech'Nice-Sophia pour la communication.

À partir de la rentrée septembre 2011, je prends, comme énuméré ci-avant, la responsabilité de la direction des études à Polytech'Nice Sophia, Filière Électronique en 2e et 3e année du cycle ingénieur. Cette responsabilité est importante car son rôle est de gérer les emploi du temps, l'évaluations des enseignements mais également avoir une relation particulière avec les étudiants. En effet, il faudra être encore plus à l'écoute des étudiants pour essayer de les conseiller le mieux possible dans leur cursus.

1.4.4 Réflexion sur le métier d'enseignant

En tant qu'enseignant et chercheur, je suis persuadé qu'il est essentiel d'utiliser ses compétences de chercheur pour l'enseignement. J'ai la chance d'avoir un poste où ceci est tout à fait possible. Par exemple, l'option GSE a été montée grâce au recul en recherche et aux besoins industriels identifiés à court terme, ce qui maintenant permet de former des étudiants compétents et performants dans le domaines des systèmes embarqués. L'adaptation et la remise en question de certains modules d'enseignement, même aujourd'hui, est toujours

nécessaires. Ainsi, depuis environ 4-5 ans, nous avons maintenant des étudiants qui réalisent leur stage dans des entreprises qui, auparavant, ne les prenaient pas.

J'ai la chance d'enseigner au niveau cycle préparatoire, ingénieur, formation par alternance ITII, et, pendant un certain temps au CNAM. Dans chacune de ces formations, le type de public n'est pas le même ce qui nous demande une adaptation de la pédagogie. Je pense que ceci fait parti de notre métier d'enseignant. À titre d'exemple, le même module en formation initiale et en formation par alternance ITII n'est pas réalisé de la même manière, aussi bien au niveau du contenu que du rythme du module.

Un autre point à prendre en compte est l'évolution de l'enseignement et de la manière d'enseigner comme par exemple la documentation électronique. Depuis 2002, tous mes documents (Cours/TD/TP) sont à disposition sur mon site web (`www.polytech.unice.fr/~fmuller/`).

L'évaluation des enseignements est importante dans une certaine mesure. L'objectif est d'évaluer la qualité du cours, TD et TP de manière objective. Avant que cette procédure devenue obligatoire approximativement en 2007, je faisais systématiquement mes propres évaluations ce qui a permis d'améliorer notamment les nouveaux modules. Ce moyen est donc plutôt positif à condition de ne pas utiliser cet outil à d'autres fins que l'amélioration de la qualité de l'enseignement.

De plus, et je pense que c'est un point essentiel, c'est de participer à la vie de l'école ou de l'Université. Depuis quelques années, je participe à des forums, des présentations dans les lycées pour Polytech, mais également pour promouvoir les métiers de l'électronique. Par exemple, j'aide l'association SAME [27] à la promotion des métiers de l'Ingénieur dans les lycées depuis l'année dernière. Cet exemple montre bien le lien fort enseignement/recherche. C'est cette action qui fera, je l'espère, reprendre goût aux étudiants de venir dans le domaine scientifique qu'est le miens.

Dans le même sens, je prend à partir de septembre 2011 la direction des études à Polytech'Nice-Sophia, Filière Électronique en 2^{e} et 3^{e} année du cycle ingénieur. C'est également l'une des facettes de notre métier d'enseignant qui permet le bon fonctionnement de mon école.

1.5 Bibliographie personnelle

1.5.1 DEA

[1] Fabrice Muller. Démarche de génération des interfaces matériel/logiciel, expérimentation sur une application de communication. Master's thesis, Equipe Modélisation et Conception des Systèmes Electroniques, Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Electronique (IRESTE), Juillet 1996.

1.5.2 Thèse de doctorat

[2] Fabrice Muller. Démarche de génération des interfaces matériel/logiciel, expérimentation sur une application de communication. Master's thesis, Equipe Modélisation

27. Sophia Antipolis MicroElectronics, `www.same-conference.org/`

et Conception des Systèmes Electroniques, Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Electronique (IRESTE), Juillet 1996.

1.5.3 Brevet

[3] Farroq Muhammad, Michel Auguin, and Fabrice Muller. Procédé de gestion des préemptions dans un système d'exploitation temps réel, n° d'application wo/2009/087317, international application no. : Pct/fr2008/001481, european patent office n° 088699004-2211, 22.10.2008.

1.5.4 Ouvrages ou chapitres d'ouvrages

[4] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. *Algorithm-Architecture Matching for Signal and Image Processing : "A New Three-Level Strategy for Off-line Placement of Hardware Tasks on Partially and Dynamically Reconfigurable Hardware"*, volume 73. Springer, 2011.

1.5.5 Revues internationales

[5] Clément Foucher, Fabrice Muller, and Alain Giulieri. Online codesign on reconfigurable platform for parallel computing. *Microprocessors and Microsystems*, Accepted in December 2011.

[6] François Duhem, Fabrice Muller, and Philippe Lorenzini. Reconfiguration time overhead on fpga : Reduction and cost model (accepted). *IET Computers and Digital Techniques*, Accepted in August 2011.

[7] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Static scheduling of periodic hardware tasks with precedence and deadline constraints on reconfigurable hardware devices. *Special Issue in EURASIP International Journal of Reconfigurable Computing (IJRC)*, February 2011.

[8] Fabrice Muller, Jimmy Le Rhun, Fabrice Lemonnier, Benoît Miramond, and Ludovic Devaux. A flexible operating system for dynamic applications. *XCell Journal, Issue 73*, Fourth Quarter 2010 2010.

[9] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. New three-level resource management enhancing quality of off-line hardware task placement on fpga. *EURASIP International Journal of Reconfigurable Computing (IJRC)*, April 2010.

En première soumission

[10] Clément Foucher, Fabrice Muller, and Alain Giulieri. Fast integration of hardware accelerators for dynamically reconfigurable architecture. *IEEE Embedded Systems Letters*, Submitted in November 2011.

[11] François Duhem, Nicolas Marques, Fabrice Muller, Hassan Rabah, Serge Weber, and Philippe Lorenzini. Dynamically reconfigurable multi-standard video adaptation using farm. *Microprocessors and Microsystems*, Submitted in November 2011.

1.5.6 Publications internationales avec actes et comités de lecture

[12] François Duhem, Fabrice Muller, and Philippe Lorenzini. Methodology for designing partially reconfigurable systems using transaction-level modeling. In *IEEE Design and Architectures for Signal and Image Processing (DASIP) (Accepted)*, Tampere, Finland, November 2-4 2011. IEEE.

[13] I. Belaid, F. Muller, and M. Benjemaa. Schedulers-driven approach for dynamic placement/scheduling of multiple dags onto sopcs. In *IEEE International Symposium on Rapid System Prototyping (RSP)*. IEEE, May 2011 2011.

[14] François Duhem, Fabrice Muller, and Philippe Lorenzini. Farm : fast reconfiguration manager for reducing reconfiguration time overhead on fpga. In *Proceedings of the 7th international conference on Reconfigurable computing : architectures, tools and applications*, ARC'11, pages 253–260, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] B. Ouni, I. Belaid, F. Muller, and M. Benjemaa. Placement of hardware tasks on fpga using the bees algorithm. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011)*, Vilamoura, Algarve, Portugal, March 2011.

[16] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Optimal static scheduling of real-time dependent tasks on reconfigurable hardware devices. In *IEEE International Conference on Communications, Computing and Control Applications (CCCA 2011)*, Hammamet, Tunisia, March 2011. IEEE.

[17] Clément Foucher, Fabrice Muller, and Alain Giulieri. Exploring FPGAs capability to host a HPC design. In *IEEE CAS 28th Norchip Conference (Norchip 2010)*, pages 1–4, Tampere Finlande, November 2010.

[18] I. Belaid, F. Muller, and M. Benjemaa. New three-level resource management for off-line placement of hardware tasks on reconfigurable devices. In *Reconfigurable Communication-centric Systems on Chip (ReCoSoC 2010)*, Karlsruhe, Germany, May 2010.

[19] Farooq Muhammad Fabrice Muller. An embedded, generic and multiprocessor hardware operating system. In *Design and Architectures for Signal and Image Processing (DASIP)*, Sophia-Antipolis, France, 22-24 September 2009.

[20] I. Belaid, F. Muller, and M. Benjemaa. Off-line placement of reconfigurable zones and off-line mapping of hardware tasks on fpga. In *Design and Architectures for Signal and Image Processing (DASIP)*, Sophia-Antipolis, France, 22-24 September 2009.

[21] I. Belaid, F. Muller, and M. Benjemaa. Off-line placement of hardware tasks on fpga. In *19th IEEE International Conference on Field Programmable Logic and Application (FPL'09)*, pages 591–595, Prague, Czech Republic, September 2009. IEEE.

[22] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Slack-conserving based scheduling of periodic real-time tasks. In *Proceedings of the 2008 Fifth IEEE International Symposium on Embedded Computing*, pages 37–42, Beijing, China, October 6-8 2008. IEEE Computer Society.

[23] Farooq Muhammad, Khurram Bhatti, Fabrice Muller, and Michel Auguin. Precognitive dvfs : Minimizing switching points to further reduce the energy consumption. In *14th IEEE Real-Time and Embedded Technology and Applications Symposium, WIP session*, St. Louis, MO, USA, 22-24 april 2008. IEEE.

[24] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Weight bound limits in supertasking approach for guaranteed timeline constraints. In *Proceedings of the 2008 International Conference on Parallel Processing - Workshops (ICPPW'08)*, pages 9–16, Krakow, Poland, 1-5 july 2008. IEEE Computer Society.

[25] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Hierarchical scheduling approach for real time tasks. In *In Proc. International Conference on Parallel Processing, Workshop on Scheduling Ressource Management for Parallel and Distributed Systems*, Portland (Oregon), USA, 8-12 september 2008.

[26] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Aether : dynamic and self-adaptive middleware. In *in Proc. IEEE International Multitopic Conference*, Lahore, Pakistan, 28-30 december 2007. IEEE.

[27] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Dynamic and self adaptive resource management : Aether operating environment. In *in Proc. 3rd IEEE Int. Conference on Emerging Technologies*, Islamabad, Pakistan, 12-13 november 2007. IEEE.

[28] Nicole Chalhoub, Fabrice Muller, and Michel Auguin. Fpga-based generic neural network architecture. In *IEEE Symposium on Industrial Embedded Systems (IES'2006)*, Juan les pins, France, October 18-20 2006. IEEE.

[29] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Self balancing computational load on multiprocessor architecture. In *IEEE International Conference on Self-Organization and Autonomous Systems in Computing and Communications - SOAS'2006*, Erfurt, Germany, sept. 2006.

[30] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Contentions-conscious dynamic but deterministic scheduling of computational and communication tasks. In *Annual ACM Symposium on Applied Computing*, Dijon, France, Apr. 2006. ACM Press.

[31] Éric Dekneuvel, Fabrice Muller, and Thierry Pitarque. Ultrasonic smart sensor design for a distributed perception system. In *IEEE 8th Conference on Emerging Technologies and Factory Automation (ETFA'2001)*, Juan les pins, France, October 15-18 2001. IEEE.

[32] Fabrice Muller, Jean-Paul Calvez, Dominique Heller, and Olivier Pasquier. An interactive modeling and generation tool for the design of hw/sw systems. In *EUROMICRO'99, Workshop on Digital System Design*, Milan, Italy, Sept 8-10 1999.

[33] Olivier Pasquier, Fabrice Muller, Jean-Paul Calvez, Dominique Heller, and Éric Chenard. The mcse approach for system-level design & design languages. In *FDL'99, Workshop on System Specification*, ENS Lyon, France, 30 August-3 September 1999.

[34] Jean-Paul Calvez, Dominique Heller, Fabrice Muller, and Olivier Pasquier. A programmable multi-language generator for codesign. In *DATE'98*, Paris, France, February 23-26 1998.

1.5.7 Autres publications internationales

[35] François Duhem, Fabrice Muller, and Philippe Lorenzini. Transaction-level modeling of dynamically reconfigurable systems using systemc. In *Sophia Antipolis MicroElectronics, SAME'2011, Poster session)*, Sophia Antipolis, France, October 12-13 2011.

[36] Fabrice Muller and Farooq Muhammad. Virtual platform for hw rtos - multiprocessor hardware rtos. In *IEEE in Proc. Design Automation and Test in Europe, (DATE 2009), University Booth*, Nice, France, 21-23 april 2009. IEEE.

[37] J-Ph Diguet, M.El Khodary, G. Gogniat, F. Muller, and M. Auguin. On simulating operating environment decisions in a sane network. In *AMWAS'08 (2nd AETHER - MORPHEUS Workshop- Autumn School From Reconfigurable to Self-Adaptive Computing)*, Lugano, Switzerland, October 7-9 2008.

[38] Fabrice Muller and Marc Vieillot. Veloce solo emulator. In *Sophia Antipolis MicroElectronics, SAME'2008, University Booth)*, Sophia Antipolis, France, October 1-2 2008.

[39] Geoffrey Austrate, Michel Auguin, Fabrice Muller, and Pierre Bricaud. Standardized multi-level ip interconnection. In *Sophia Antipolis MicroElectronics, SAME'2007, Poster session*, Sophia Antipolis, France, October 3-4 2007.

[40] Fabrice Muller, Farooq Muhammad, and Michel Auguin. Design of a hardware multiprocessor real-time operating system. In *IEEE in Proc. Design Automation and Test in Europe (DATE 2007), University Booth*, Nice, France, 17-19 april 2007. IEEE.

1.5.8 Revues nationales avec actes et comité de lecture

En première soumission

[41] Clément Foucher, Fabrice Muller, and Alain Giulieri. Méthodologie dédiée aux applications parallèles sur plateforme reconfigurable dynamiquement. *Technique et Science Informatiques*, Soumise en Septembre 2011.

1.5.9 Publications nationales avec actes et comité de lecture

[42] François Duhem, Fabrice Muller, and Philippe Lorenzini. Dynamic and partial reconfiguration transaction-level modeling in systemc. In *Colloque GDR SoC/SiP*, Lyon, France, 15-17 juin 2011.

[43] Clément Foucher, Fabrice Muller, and Alain Giulieri. Flot de conception d'applications parallèles sur plateforme reconfigurable dynamiquement. In *Symposium en Architectures nouvelles de machines (Sympa'14)*, St Malo, France, 10-13 mai 2011.

[44] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Off-line placement/scheduling of hardware tasks on reconfigurable devices. In *Colloque GDR SoC/SiP*, Cergy-Pontoise, France, 9-11 juin 2010.

[45] François Duhem, Fabrice Muller, and Philippe Lorenzini. Services pour des systèmes reconfigurables dynamiquement. In *Colloque GDR SoC/SiP*, Cergy-Pontoise, France, 9-11 juin 2010.

[46] Ikbel Belaid, Fabrice Muller, Maher Benjemaa, and Alain Giulieri. Off-line placement of hardware tasks on fpga. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[47] Clément Foucher, Fabrice Muller, and Alain Giulieri. Implémentation d'un système d'exploitation matériel compatible rtems. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[48] Bassem Ouni, Fabrice Muller, and Maher Benjemaa. Placement et ordonnancement des tâches matérielles sur des zones reconfigurables en utilisant le bees algorithm. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[49] Farooq Muhammad, Khurram Bhatti Muhammad, Fabrice Muller, and Michel Auguin. Efficient and optimal multiprocessor scheduling for real time tasks. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

[50] Ikbel Belaid, Fabrice Muller, and Alain Giulieri. Virtualisation de l'ordonnancement matériel/logiciel sur plateforme reconfigurable dynamiquement. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

[51] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Proportionate scheduling of hard and soft real time tasks. In *Colloque GDR SoC/SiP*, Paris-Jussieu, France, 13-14-15 juin 2007.

[52] Fabrice Muller and Jean-Paul Calvez. Mcse toolbox : Prototype d'outil pour la conception des systèmes matériels/logiciels. In *Colloque CAO de circuits intégrés et systèmes*, Aix-en-Provence, France, 10-12 mai 1999.

[53] Fabrice Muller, Jean-Paul Calvez, and R. Gasmann. Génération d'interfaces matériel/logiciel, expérimentation pour un bus à microprocesseur. In *Séminaire CoDesign*, CNET Meylan, France, 25 juin 1996.

1.5.10 Conférences invitées

[54] Fabrice Muller. Os & reconfigurable. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

[55] Fabrice Muller. Rtos pour ar. In *Journée Thématique Architecture Reconfigurable, GDR SoC/SiP*, Paris-Jussieu, France, 11-12 octobre 2007.

1.5.11 Colloques à caractère recherche et industriel

[56] Fabrice Muller, Farooq Muhammad, and Michel Auguin. Embedded full hardware multiprocessor real-time operating system. In *Journée Industriel*, Sophia-Antipolis, France, 29 mai 2008.

[57] Ikbel Belaid and Fabrice Muller. Dynamic partial reconfiguration in sopc. In *Journée Industriel*, Sophia-Antipolis, France, 29 mai 2008.

1.5.12 Journaux et Colloques à caractère pédagogique

[58] Fabrice Muller and Gilles Jacquemod. Validation de soc : Les étudiants se forment à l'émulation sous veloce. In *Journal La Puce à l'Oreille*, N°29, Mars 2009, page 4, 2009.

[59] Fabrice Muller, Gilles Jacquemod, and Rachid Bouchakour. Vérification de soc sous veloce. In *Journées Pédagogiques CNFM*, St Malo, France, 26-28 novembre 2008.

[60] Muller Fabrice, B. Fontaine, E. Meneceur, and R. Meyer. Développement d'un simulateur automobile pour la validation d'un rtos matériel. In *Les Neuvièmes Journées pédagogiques du CNFM*, St Malo, France, 22-24 novembre 2006.

[61] Fabrice Muller, Éric Dekneuvel, and Michel Auguin. Intégration de modules ip sur une carte de prototypage excalibur-nios. In *Les Septièmes Journées pédagogiques du CNFM*, St Malo, France, 27-29 novembre 2002.

1.5.13 Rapports de contrats et rapports internes

[62] Rapport d'avancement du projet anr ardmahn, analyse de l'existant pour les méthodologies pour architectures auto-adaptatives, Septembre 2010.

[63] Rapport d'avancement du projet anr fosfor, liste des fonctionnalités des services à implémenter, Mars 2009.

[64] Rapport d'avancement du projet anr fosfor, définition de l'ordonnancement matériel, Mars 2009.

[65] Rapport annuel du projet anr fosfor, évaluation du management du projet, Janvier 2009.

[66] Rapport d'avancement du projet anr fosfor, analyse de l'existant, Juin 2008.

[67] Rapport annuel du projet européen æther, second annual research report on the sane processor and its system environment - enabling interaction and adaptation, Janvier 2008.

[68] Rapport d'avancement du projet européen æther, « first demonstration of the oe systemc simulator based on the sane hardware, Octobre 2007.

[69] Rapport annuel du projet européen æther, « first annual research report on abstract sane processors and system environment (t2.1.1) and sane interactions and adaptation (t2.1.2), décembre 2006.

CHAPITRE 2

Ordonnancement de tâches sur architectures multiprocesseurs ou auto-adaptatives

2.1 Contexte

La différence principale entre un système temps réel par rapport à un système transactionnel ou même hautes performances est l'introduction de conditions temporelles dans ses spécifications. En d'autres termes, un système temps réel est correct non seulement si le code applicatif est logiquement correct et produit des résultats logiquement corrects, mais également si ces résultats sont produits dans des intervalles de temps précis. Les systèmes de contrôle de processus qui multiplexent plusieurs calculs de lois de commande tels du traitement de signal radar et de suivi de cibles dans un système de contrôle de trafic aérien est un exemple de systèmes temps réel.

Les conditions et les contraintes temporelles dans les systèmes temps réel sont généralement spécifiées sous forme d'échéances avant lesquelles les activités du système doivent terminer leurs exécutions. Considérons comme exemple un système de suivi radar comme nous le proposons dans le projet ANR FOSFOR (page 101, section 4.4). Pour suivre des cibles d'intérêt, le système radar effectue les activités ou tâches suivantes : envoi des impulsions par radio vers les cibles, réception et traitement des signaux d'écho reçus pour déterminer la position et la vitesse des objets ou des sources qui ont renvoyé les impulsions, et en association des sources aux cibles et mise à jour de leurs trajectoires. Pour un suivi efficace, chacune des tâches ci-dessus doit être exécutée périodiquement à une fréquence qui dépend de la distance, de la vitesse, et de l'importance des cibles et, à chaque invocation, doit exécuter la tâche dans un délai ou suivant une échéance spécifique. Une autre caractéristique d'un système temps réel est qu'il devrait être prévisible. La prévisibilité signifie qu'il devrait être possible de montrer ou de prouver que les exigences des tâches sont toujours satisfaites quelles que soient les hypothèses faites, comme la charge de calcul sur le ou les processeurs ou bien sur le ou les zones reconfigurables exécutant les tâches matérielles.

En fonction des conséquences induites par un non respect des exigences des tâches, les contraintes de temps sont généralement classées comme dures ou souples. Une contrainte de temps dure correspond par exemple à l'échéance dont la violation peut entrainer des conséquences désastreuses telles que des risques pour des vies humaines ou des dégâts matériels importants. Des systèmes de contrôle de procédés industriels, des contrôleurs embarqués dans l'automobile ou des systèmes de contrôle aérien en aéronautique sont autant d'exemples de systèmes soumis à des contraintes de temps réel dur. Par comparaison, un système temps réel souple possède des contraintes moins critiques, le dépassement de contraintes souples est ainsi possible. Cependant, ces dépassements ne sont en général pas souhaitables du fait que la qualité de service du système se dégrade avec le nombre de dépassements observés et par conséquent, ce nombre est généralement borné. Les systèmes multimédia ou de réalité virtuelle sont des exemples de systèmes à contraintes de temps souples.

Il existe des applications émergentes à la fois temps réel et de grande complexité algorithmique. Des exemples sont les systèmes de suivi automatique de cibles et les systèmes de télésurveillance. Ces applications ont des contraintes temporelles qui visent à produire une grande réactivité du système et peuvent aussi être nécessaires pour assurer un comportement correct voire crucial dans certaines applications telles que la télé-chirurgie. En outre, leurs besoins de traitement peuvent facilement dépasser la capacité de calcul d'un simple processeur et dans ce cas, un système multiprocesseur peut être nécessaire pour réaliser une exécution efficace de l'application. Par ailleurs, les architectures multiprocesseurs peuvent être plus rentables qu'un processeur unique de même capacité de calcul du fait que le coût d'un système à k-processeurs peut être moins élevé que celui d'un processeur k fois plus rapide (dans l'hypothèse où un processeur de cette performance est en effet disponible). D'ailleurs, la tendance est à l'augmentation du nombre de cœurs de processeur dans les années futures [153]. Une autre solution consiste également à déporter des fonctions de calculs en matériel appelé également des accélérateurs matériels dont le coût de conception est plus important et dont la flexibilité n'est pas mature en comparaison à une fonction de calcul logicielle. Cependant, cette flexibilité a émergé par l'arrivée des architectures reconfigurables. Notons également que le GPU [1] est une solution intermédiaire qui fait d'ailleurs partie de mes perspectives de recherche au même titre que le reconfigurable.

En effet, depuis ces dernières années, les architectures reconfigurables ont progressé à une vitesse phénoménale. L'objectif majeur de ces architectures est de satisfaire toutes les exigences des systèmes embarqués dont principalement les contraintes temps-réel. Au sein de ces architectures reconfigurables s'exécutent des fonctions de calcul décrites matériellement. En fait, l'introduction de calculs reconfigurables dans les systèmes embarqués permet d'optimiser et d'adapter le matériel pour mieux répondre à des besoins immédiats. De plus, les concepteurs cherchent à rendre ces architectures de plus en plus flexibles pour élargir le spectre d'application. Pour répondre à ce compromis entre contraintes spécifiques (débit, encombrement, consommation ...) et flexibilité, il sera nécessaire de faire cohabiter des processeurs et des zones matérielles dynamiquement et partiellement reconfigurables. Ce couplage engendre une complexité accrue en termes de placement des tâches matérielles et d'ordonnancement global de l'application sur l'architecture reconfigurable. Cette problématique placement/or-

1. Graphics Processing Unit

donnancement de tâches purement matérielles sera plutôt abordée dans le chapitre 3.

Toutes ces observations soulignent l'importance croissante de la problématique d'ordonnancement de tâches pour des systèmes temps-réel devant être adaptée à ces architectures. Dans ces travaux, nous nous concentrons sur plusieurs points centraux relatifs à cette problématique.

Nous allons tout d'abord présenté les contributions (section 2.2) de cet axe de recherche. Nous détaillerons ensuite deux principales contributions : l'algorithme EEDF [2]/ERM [3] (section 2.3) qui a fait l'objet d'un brevet et l'intergiciel auto-adaptatif pour de futures architectures reconfigurables (section 2.4) qui fait le lien avec le chapitre 3.

2.2 Contributions

Les études menées ont pour dénominateur commun le constat suivant : *les algorithmes d'ordonnancement temps-réel ont des bornes d'ordonnançabilité au plus égales à la capacité de traitement de l'architecture (et ce suivant certaines hypothèses). Cependant, ces algorithmes peuvent gagner en efficacité dès lors que leur impact sur les temps de gestion est réduit et ainsi augmenter la Qualité de Service (QoS) des applications. Cet accroissement en efficacité peut être obtenu par une meilleure prise en compte de paramètres implicites des tâches.*

Dans la suite nous décrivons nos contributions de manière résumée et pour certaines d'entres elles plus détaillées. Les bornes d'ordonnançabilité sont de toute évidence importantes à identifier mais l'efficacité est aussi essentielle dans un grand nombre d'applications émergentes et temps réel et ce afin d'améliorer la QoS de l'application et de minimiser la consommation d'énergie du système. L'objectif est de proposer des algorithmes d'ordonnancement qui possèdent un temps d'exécution faible du fait d'une complexité algorithmique réduite liée en particulier à une diminution du nombre d'invocations de l'ordonnanceur, c'est-à-dire la réduction du nombre d'événements d'ordonnancement. Par ailleurs, des algorithmes d'ordonnancement qui augmentent le nombre de tâches exécutées dans un temps imparti en exploitant des paramètres dynamiques des tâches en cours d'exécution, sont à priori efficaces puisque la QoS de l'application est directement liée au nombre de tâches exécutées pendant ce temps. Nous abordons les problèmes d'efficacité d'ordonnancement en apportant des modifications significatives aux algorithmes d'ordonnancement existants et ce en cherchant à minimiser le nombre d'événements d'ordonnancement et en diminuant le coût de gestion d'un événement d'ordonnancement ce qui a pour résultat d'améliorer la QoS de l'application (partie décrite brièvement dans la section 2.2.1). Nous proposons également des algorithmes pour minimiser le nombre de préemptions sans compromettre les bornes d'ordonnançabilité (section 2.2.2). Dans le cas de systèmes multiprocesseurs, la borne d'ordonnançabilité égale au nombre de processeurs peut être atteinte mais au prix d'hypothèses souvent irréalisables vis-à-vis de l'application du fait des coûts élevés induits par la complexité de gestion en ligne et les nombres de préemptions et migrations de tâches. Nous apportons une réponse à ces problèmes dans le cas d'architectures multiprocesseurs en proposant des techniques qui relâchent les hypothèses et permettent ainsi de réduire le nombre de préemptions de manière

2. Enhanced Earliest Deadline First
3. Enhanced Rate Monotonic

importante (section 2.2.3). Nous avons également étudié des algorithmes d'ordonnancement hybrides et présentons trois approches ayant des bornes optimales d'ordonnançabilité avec un coût maîtrisé (section 2.2.4). Pour finir, nous proposons également un modèle permettant de prendre en compte les communications inter-processeur (section 2.2.5).

2.2.1 Algorithmes d'ordonnancement RUF

Considérons un ensemble de tâches composé de tâches critiques et non critiques tel que la charge totale de cet ensemble est supérieure à 100%. Il se pose alors le problème de définir une technique d'ordonnancement qui garantit une terminaison de toutes les tâches critiques avant leurs échéances tout en maximisant les exécutions des tâches non critiques. Des algorithmes existants, tels que EDF [78], MUF [4] [152], CASH [5] [151], n'apportent pas de garanties vis-à-vis des tâches critiques lors de situations transitoires de surcharge ou bien ne maximisent pas l'exécution des tâches non critiques. Dans ce contexte, nous proposons d'exploiter les paramètres dynamiques (en ligne) des tâches, c'est-à-dire les intervalles de temps $C_i - AET_i$ (représente le temps d'exécution actuel de la tâche T_i, Actual Execution Time) où le processeur est disponible à chaque exécution d'une tâche lorsque le temps d'exécution effectif de la tâche est inférieur à son temps d'exécution pire-cas. L'objectif n'est pas seulement de fournir des garanties aux tâches critiques mais aussi de maximiser l'exécution des tâches non critiques. Pour répondre à ce problème, un nouvel algorithme d'ordonnancement appelé RUF [6] [198] est proposé. Nous définissons un mécanisme de contrôle des tâches non critiques afin de maximiser leurs exécutions et ceci sans compromettre les échéances des tâches critiques. Ce contrôle des tâches non critiques à un instant t dépend des paramètres dynamiques des tâches qui ont terminé leurs exécutions avant l'instant t.

2.2.2 Algorithmes d'ordonnancement EEDF/ERM

Dans le cas de systèmes monoprocesseur, des algorithmes d'ordonnancement temps réel ont des bornes d'ordonnançabilité optimales mais au prix d'une complexité en ligne qui peut être élevée, avec pour conséquence un nombre important de préemptions de tâches (ou d'événements d'ordonnancement). Ce nombre élevé de préemptions diminue en pratique l'ordonnançabilité des tâches, mais augmente également la consommation d'énergie du système. Ces algorithmes n'exploitent pas en général tous les paramètres implicites et dynamiques (en ligne) des tâches, aussi des optimisations sont possibles afin de minimiser les coûts de gestion par l'ordonnanceur et l'énergie du système. La laxité est par exemple un paramètre implicite d'une tâche qui fournit une certaine flexibilité à l'ordonnanceur. L'ordonnanceur peut exploiter cette flexibilité pour relâcher certaines règles de l'algorithme d'ordonnancement et ainsi aider à réduire le coût lié aux préemptions des tâches. Nous proposons d'utiliser ce paramètre implicite dans un algorithme d'ordonnancement afin de réduire très significativement le nombre de préemptions des tâches. Deux variantes de cette approche sont proposées, l'une

4. Maximum Urgency First
5. CApacity SHaring
6. Real Urgency First

est statique, l'autre dynamique. Toutes deux sont appliquées aux politiques d'ordonnancement EDF et RM [7] [78]. On peut remarquer également que les préemptions des tâches sont en général liées au taux de charge du processeur par les tâches. Ces travaux sont détaillés dans la section 2.3.

2.2.3 Algorithme d'ordonnancement ASEDZL

Dans le cas de systèmes multiprocesseurs, des algorithmes d'ordonnancement optimaux ont été proposés, basés sur un modèle d'ordonnancement fluide [156, 154, 155] avec une idée d'équité sous-jacente. Les algorithmes basés sur cette approche d'équité impliquent un coût de gestion élevé dû aux préemptions et aux invocations de l'ordonnanceur ce qui les rend parfois inutilisables. Pour minimiser ce coût de gestion, nous proposons un algorithme d'ordonnancement qui n'est justement pas basé sur un ordonnancement équitable. Dans cet algorithme appelé ASEDZL (Anticipating Slack Earliest Deadline first until Zero Laxity), les tâches ne sont pas allouées à un processeur pour un temps fonction de leurs poids, mais elles sont sélectionnées pour s'exécuter entre deux requêtes consécutives de tâches de telle sorte que les tâches ayant les échéances les plus proches sont assurées de s'exécuter jusqu'à la prochaine requête. Cet algorithme fournit de meilleurs résultats en termes de nombre de préemptions et de migrations de tâches par rapport aux algorithmes à ordonnancement fluide. Nous avons montré également par simulation que les résultats obtenus sont meilleurs (en préemptions et migrations) que les algorithmes classiques qui ont également une borne d'ordonnançabilité égale à la capacité de traitement de l'architecture.

2.2.4 Algorithme d'Ordonnancement Hiérarchique

Les algorithmes d'ordonnancement hybrides ou hiérarchiques garantissent en général de meilleurs résultats que les algorithmes d'ordonnancement globaux ou partitionnés dans le cas d'architectures à mémoire partagée et distribuée. Cependant, un très petit nombre d'algorithmes d'ordonnancement hybrides sont proposés ayant une borne d'ordonnançabilité égale au nombre de processeurs (en considérant les hypothèses précisées dans [93], chapitre 5). Une technique de supertâche est proposée par Moir et al. [92] dans un algorithme d'ordonnancement hybride où l'algorithme Pfair est utilisé comme un ordonnanceur global. Cependant, il est nécessaire de définir une condition limite pondérée entre les tâches locales (celles partitionnées) et les supertâches correspondantes pour atteindre la borne maximum d'ordonnançabilité. Nous établissons cette condition dans [93], chapitre 5. Nous proposons d'utiliser l'algorithme ASEDZL comme ordonnancement global avec l'intérêt qu'il n'impose aucune condition sur les poids respectifs des supertâches et des tâches locales. Nous comparons l'algorithme proposé avec des approches existantes pour illustrer l'obtention de meilleurs résultats. Ainsi, nous montrons que le nombre de préemptions est moindre que tout autres algorithmes d'ordonnancement dans ce contexte. De plus, un nouvel algorithme d'ordonnancement hybride est également proposé où des intervalles de temps sont réservés pour l'exécution des tâches locales ou globales et qui permet de garantir les échéances des tâches.

7. Rate Monotonic

2.2.5 Modèle déterministe pour MPSoCs tenant compte des communications inter-processeur

En temps réel, la qualité de service est synonyme d'exactitude temporelle. Une propriété d'un système, dit d'exactitude temporelle, signifie que toutes les opérations sont dans les limites de temps connues. Mais comment cette exactitude temporelle peut-elle être assurée dans un système multiprocesseur ? Cette exactitude signifie que la communication et l'exécution des tâches sont effectuées au sein d'intervalles de temps bornés connus. Dans le cas d'un système mono-processeur, la communication inter-tâches peut être assurée en vérifiant le taux d'utilisation du processeur et des communications inter-tâches (mais intra-processeur). Cependant, dans le cas de systèmes multiprocesseur, les communications inter-tâches peuvent être de deux types : les échanges de communications intra-processeur et inter-processeurs. Ainsi, comment s'assurer que la communication inter-processeur est dans les limites de temps connues ? Comment obtenir le comportement déterministe du système dans une architecture multiprocesseur ?

Pour cela, nous voulons concevoir un modèle où le système entier est déterministe, en particulier dans le cas d'un système multiprocesseurs. L'objectif est de rendre les communications inter-processeurs déterministes (avec des limites de temps connues) et de pouvoir embarquer ce modèle dans un modèle de calcul classique afin d'utiliser un ordonnancement classique comme EDF.

Cette contribution étudie tous les conflits possibles (conflits sur les bus, les nœuds et les mémoires partagées) subies par les tâches exécutées sur une architecture multiprocesseurs à mémoire partagée. Ici, nous ne nous sommes pas axés sur l'ordonnancement ou le regroupement de tâches sur les processeurs, mais plutôt nous nous sommes concentrés sur l'ordonnancement des communications de manière dynamique entre les processeurs. Nous avons assuré une solution afin d'isoler les tâches dépendantes localisées sur différents processeurs. Nous avons proposé un schéma d'ordonnancement pour bien maîtriser la charge de communication entre les tâches en tenant compte des conflits sur les liens de communication ainsi que les nœuds. Pour cela nous avons étroitement couplé l'ordonnanceur et l'arbitre. Nous avons divisé la mémoire partagée distribuée à l'intérieur de la mémoire locale des processeurs, couplée à une mémoire de type boîte aux lettres et jouant le rôle de mémoire partagée. De plus, nous avons introduit un algorithme dans les services du noyau du RTOS [8] pour assurer une isolation complète de l'espace d'adressage, rendant virtuellement les tâches indépendantes. Nous avons mis en œuvre les services du noyau RTOS, à savoir l'ordonnanceur et l'arbitre, en matériel pour maîtriser la communication inter-processeur sur le bus de communication partagé de manière plus efficace.

2.2.6 Intergiciel auto-adaptatif pour de futures architectures reconfigurables

Les algorithmes d'ordonnancement auto-adaptatifs considèrent généralement des tâches avec un degré de parallélisme déterminé statiquement et une allocation de ressources dyna-

8. Real Time Operating System

mique afin d'une part, de tenir compte de la variation du parallélisme apparaissant lors de l'exécution de la tâche suivant son flot de contrôle et d'autre part, de réduire le gaspillage en ressources. Ces algorithmes peuvent prendre en compte la création ou la suppression dynamique de tâches mais ils ne sont pas capables de traiter le cas de tâches ayant des capacités à s'auto-optimiser à l'exécution ce qui conduit à un degré de parallélisme effectif pendant l'exécution différent de celui calculé statiquement. Par ailleurs, l'architecture peut aussi s'auto-adapter pour exécuter séquentiellement des tâches à priori parallèles sur une ressource spécifique plutôt que de les exécuter en parallèle sur différentes ressources standardisées. Ces optimisations à l'exécution nécessitent un mécanisme auto-adaptatif de gestion de ressources capable d'appréhender ces variations en-ligne. Nous proposons un modèle d'ordonnancement auto-adaptatif qui prend en compte ces optimisations à l'exécution et fournit les garanties de respect d'échéances pour les tâches temps réel critiques. Ceci est possible à l'aide d'une analyse d'ordonnançabilité pour une application répartie sur plusieurs SANE (Self Adaptive Networked Entity). De plus, un algorithme a été développé estimant le temps restant libre pour exécuter des tâches sporadiques temps réel souple et donc permettre d'augmenter la charge du SANE ou du processeur. Cet algorithme et le calcul d'ordonnançabilité tiennent compte des caractéristiques de l'architecture (communications, processeurs, ressources) et de l'application (parallélisme, contraintes de temps). Ces travaux ont été menés dans le cadre du projet Européen ÆTHER.

2.3 Ordonnancement EEDF et ERM

2.3.1 Introduction

La théorie de l'ordonnancement a été largement étudiée au cours des vingt dernières années. Depuis la levée d'un important jalon dans le domaine de l'ordonnancement temps-réel dur utilisé par Liu et al. [78], deux classes d'algorithmes (c'est-à-dire à priorité dynamique et à priorité fixe) ont été étudiées séparément et beaucoup de résultats tels que l'optimalité, la condition de faisabilité, le temps de réponse, le contrôle d'admission ont été établis.

L'algorithme d'ordonnancement EDF [78] est un algorithme de priorité dynamique qui exécute toujours une tâche dont l'échéance est la plus imminente. Cet algorithme a été prouvé pour être optimal dans le cas de système monoprocesseur [78]. À l'instant de relâchement d'une tâche de priorité supérieure, l'algorithme d'ordonnancement EDF préempte la tâche en exécution et commence à exécuter la tâche de priorité supérieure nouvellement libérée. Cet algorithme ne prend pas en compte la possibilité de retarder l'exécution de la tâche de priorité plus élevée et de laisser la tâche de faible priorité achever son exécution. Ainsi, l'algorithme EDF peut, inutilement, augmenter le nombre de préemptions des tâches.

Réduire le nombre de préemptions peut également être bénéfique d'un point de vue énergétique dans les systèmes demandant une faible consommation d'énergie [160]. Lorsqu'une tâche est préemptée, il y a une forte probabilité pour que son contenu dans le cache soit partiellement ou totalement perdu. Lorsque l'exécution de la tâche est reprise, ceci provoque de nombreux accès à la mémoire qui consomment de l'énergie. La réduction du nombre de préemptions réduit ces accès mémoire supplémentaires et coûteux tout en réduisant la pollution

du cache.

Radu Dorbin [88] a proposé un algorithme qui minimise les préemptions pour l'ordonnancement à priorités fixes mais il n'aborde pas les algorithmes d'ordonnancement dynamique. Sung-Heun [89] ont proposé un algorithme qui résout le problème des conflits de laxité qui apparaissant entre deux tâches au moment de l'exécution. Ces conflits introduisent de sérieux problèmes qui fait que l'algorithme LLF [9] soit impraticable.

Woonseok Kim [90] propose d'augmenter la fréquence des processeurs afin de minimiser le nombre de préemptions. Il réduit la consommation d'énergie en minimisant le nombre de préemptions mais, d'un autre côté, augmente la consommation d'énergie dû au processeur fonctionnant à des fréquences plus élevées. Sanjoy K. Baruah [91] ont proposé un algorithme où l'idée de base pour réduire les préemptions est similaire, mais où la technique pour calculer le paramètre Non Préemption Zone (NPZ) d'une tâche est différente. Il ne fonctionne pas si les tâches quittent ou rejoignent le système dynamiquement. Par ailleurs, ces travaux ne traitent pas de calculs dynamiques des NPZ. Le NPZ pour une tâche calculée dynamiquement peut avoir une valeur supérieure ou égale au NPZ calculé statiquement ce qui réduit encore les préemptions. Nous avons également observé que les préemptions de tâches dépendent aussi de la charge des processeurs. Si la charge du processeur est faible, le nombre de préemptions est également plus faible.

Ainsi, nous proposons de répondre à ces problèmes à l'aide de deux algorithmes, notés EEDF et ERM, dont nous allons présenter le principe et les résultats obtenus. Afin de comprendre le principe utilisé, nous commençons par présenter la manière de minimiser le nombre de préemptions.

2.3.2 Minimisation du nombre de préemptions

La plupart des algorithmes d'ordonnancement sont à priorité : ils assignent des priorités aux tâches dans le système et ces priorités sont utilisées pour sélectionner l'exécution d'une tâche à chaque fois que les décisions d'ordonnancement sont faites.

Les algorithmes à priorité dynamique permettent plus de souplesse dans les affectations des priorités. La priorité d'une tâche peut varier au cours du temps. Un exemple d'algorithme d'ordonnancement qui utilise les priorités dynamiques est l'algorithme EDF. L'algorithme RM à priorité statique est considéré comme un algorithme d'ordonnancement optimal à priorités fixes. Il a l'avantage d'être très peu gourmand en terme de complexité et de temps d'exécution alors que l'algorithme EDF à priorité dynamique a un ordonnancement optimal sur des systèmes monoprocesseur et est plus complexe à implémenter. C'est pourquoi, nous avons proposé de modifier ces deux algorithmes de telle sorte que le nombre de préemptions soit minimisé. Nous commençons par investiguer sur ces points quand une tâche T_i est préemptée par une tâche plus prioritaire. Nous avons également identifié les tâches qui peuvent potentiellement préempter une tâche T_i, et ainsi proposer d'utiliser la laxité de la tâche T_i pour aider à retarder, si possible, cette préemption, comme la laxité d'une tâche donne la flexibilité pour la retarder.

9. Least Laxity First

Théorème 2.1. *La laxité L_i de la tâche T_i peut être décalée n'importe où durant sa période sans causer à la tâche T_i de manquer son échéance.*

Preuve : Une tâche T_i ne manquera jamais son échéance si elle est assignée à C_i unités de temps du processeur pendant la période P_i. Cela implique que la laxité d'une tâche peut être décalée n'importe où pendant la période de la tâche T_i.

Théorème 2.2. *Une tâche T_i peut être préemptée que par la tâche qui a une durée inférieure à celle de T_i (valable pour l'ordonnancement RM et EDF).*

Preuve : Seules les tâches de plus hautes priorités peuvent préempter les tâches de faibles priorités. Dans le cas de l'algorithme d'ordonnancement RM, les tâches ayant de faibles périodes ont des priorités plus élevées que les tâches ayant des périodes élevées. Dans le cas de l'algorithme d'ordonnancement EDF, les priorités sont définies suivant la proximité vis à vis des échéances absolues des tâches. Si la tâche T_i est préemptée lors de son n^{ieme} instant par la tâche T_j prête au k^{ieme} instant, cela implique que le n^{ieme} instant de la tâche T_i a été libéré avant le k^{ieme} instant de la tâche T_j (c'est-à-dire, $r_i^n < r_j^k$). Comme la tâche T_i est préemptée lors de son n^{ieme} instant par tâche T_j prête au k^{ieme} instant, cela implique que l'échéance de la tâche T_j est plus proche que l'échéance de la tâche T_i.

Si $r_i^n < r_j^k$
et $d_j^k < r_i^n$
alors $P_j < P_i$

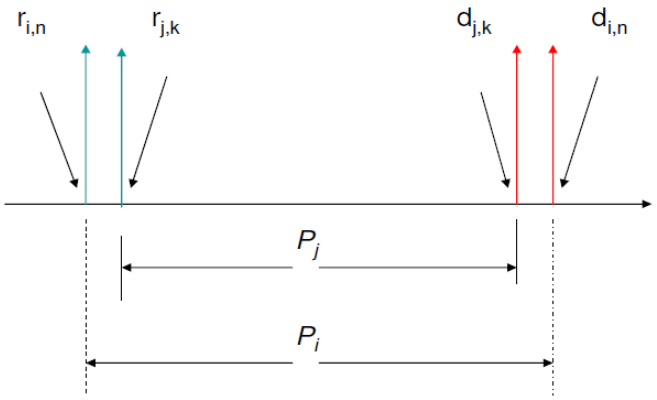


Figure 2.1 – *Tâche ayant une fréquence plus élevée pouvant préempter une tâche de plus faible fréquence.*

Par conséquent, il est prouvé que la tâche T_i peut être préemptée par une tâche qui a une période plus petite que celle de la tâche T_i (Figure 2.1). Ainsi, nous fournissons des techniques pour calculer le NPZ d'une tâche à la fois pour les deux algorithmes d'ordonnancement RM et EDF.

2.3.3 Ordonnancement EEDF

2.3.3.1 Principe

Nous proposons un algorithme visant à réduire le nombre de préemptions. Les préemptions sont directement en rapport avec le changement de contexte. Cet algorithme fonctionne exactement comme l'algorithme d'ordonnancement classique EDF, sauf à ces instants d'ordonnancement où la préemption des tâches en exécution est proposée par l'ordonnanceur modifié. Si une tâche en exécution va être préemptée par la nouvelle tâche arrivée de priorité supérieure, alors ce nouvel algorithme EEDF entre en action et tente de préempter, si c'est vraiment nécessaire, cette tâche en exécution ou de retarder la préemption pendant un certain temps à savoir, le NPZ.

Cette approche est similaire à celle proposée par Baruah [91] où il a examiné l'ensemble des tâches sporadiques. Selon son approche, chaque tâche est assignée à un autre paramètre appelé NPZ. La préemption de la tâche en cours d'exécution est retardée du temps égal à NPZ. Cette approche présentée par Baruah est basée sur des paramètres statiques des tâches et il ne supporte pas la création et la suppression dynamique de tâches. Dans son approche, il a utilisé la fonction limite de demande (DBF, Demand Bound Function) pour calculer la valeur du NPZ pour chaque tâche et afin de prouver la faisabilité de l'ensemble des tâches. La fonction limite qui fournit une condition suffisante et nécessaire pour des tâches périodiques utilisant EDF, est définie comme suit (équation 2.1) :

$$\sum_{i=1}^{n} DBF(T_i, t) = \sum_{i=1}^{n} \left\lfloor \frac{t}{P_i} \right\rfloor \times C_i \tag{2.1}$$

La condition suffisante et nécessaire pour la faisabilité de tâches périodiques est donnée par l'équation 2.2 :

$$\forall t > 0 :: \sum_{i=1}^{n} DBF(T_i, t) \leq t \tag{2.2}$$

Les calculs des NPZ selon l'approche de Baruah sont effectués par l'équation 2.3.

$$NPZ_i = min\left(NPZ_{i-1},\ d_i - \sum_{i=1}^{n} DBF(T_i, t)\right) \tag{2.3}$$

Les tâches sont triées dans un ordre croissant suivant leurs périodes et NPZ_1 est initialisé à L_0 (Laxité de la tâche T_0). Cette approche est itérative ; NPZ_i est calculé avant NPZ_{i+1}. De plus, cette approche est basée sur des paramètres statiques des tâches. C'est pourquoi elle ne prend pas en charge la création et la suppression dynamique des tâches.

Nous proposons de calculer le NPZ de telle façon qu'il supporte la création et la suppression dynamique des tâches. Nous avons deux approches pour calculer le NPZ de chaque tâche, l'une est statique et l'autre dynamique. Dans l'approche statique, nous considérons le pire cas lorsque toutes les tâches plus prioritaires (les tâches qui ont des périodes plus petites que celle de la tâche en exécution) arrivent au même instant (ce qui est plutôt rare dans les systèmes réels). Dans l'approche dynamique, nous prenons en compte les tâches qui ont des échéances absolues plus proches que celle de la tâche en cours d'exécution.

Exemple

La laxité de toute tâche nous donne une mesure de la durée maximale pour laquelle elle peut être retardée. La tâche ayant la plus petite période ne sera jamais préemptée car aucune autre tâche peut avoir une priorité plus élevée au cours de son exécution. L'approche est démontrée en considérant un modèle simple comprenant trois tâches T_1, T_2 et T_3 données dans le Tableau 2.1. Le temps d'exécution pire-cas, que nous appellerons WCET [10], pour les tâches T_1, T_2 et T_3 sont respectivement 6, 4 et 9, tandis que 21, 10 et 31 sont respectivement les périodes des tâches. Les échéances des tâches sont supposées être égales à leurs périodes.

La tâche ayant la plus petite période de temps n'est jamais préemptée. Dans cet exemple, T_2 a la plus petite période. La tâche T_1 peut être seulement préemptée par T_2 puisque T_2 est la seule tâche de période inférieure à celle de T_1. Dans ce cas, T_1 peut causer à T_2 une inversion de priorité pour la durée au plus égale à la laxité de la tâche T_2 ; c'est-à-dire, NPZ_1 = P_2-C_2. La tâche T_3 peut être préemptée par les deux tâches T_1 et T_2 qui ont une période de temps inférieure à celle de la tâche T_3. Dans ce cas, le NPZ pour T_3 est calculé comme suit :

$$
\begin{aligned}
L_{abs} &= P_1 - \sum_{i=1}^{2} \frac{P_1}{P_i} \times C_i \\
L_{abs} &= 21 - 6 - \frac{21}{10} \times 4 \\
L_{abs} &= 6,6 \\
NPZ_3 &= min\left\lfloor NPZ_1, L_{abs} \right\rfloor \\
NPZ_3 &= 6
\end{aligned}
$$

Tâche	C	P	Offset	NPZ
T_1	6	21	0	6
T_2	4	10	3	n'est jamais préemptée
T_3	9	31	0	6

Tableau 2.1 – *Paramètres des tâches.*

Nous observons qu'il y a deux préemptions (Figure 2.2 (b)) selon l'ordonnancement EDF. Les préemptions sont réduites à zéro (Figure 2.2 (a)) si la préemption est retardée suivant les NPZ selon EEDF.

Le NPZ des tâches T_i peut être calculé soit statique ou dynamique, afin de minimiser les préemptions. La méthode de calcul de ces deux approches ainsi que la démonstration de la preuve d'ordonnançabilité de EEDF sont respectivement détaillées dans le manuscrit [93], section 3.2.1.

10. Worst Case Execution Time

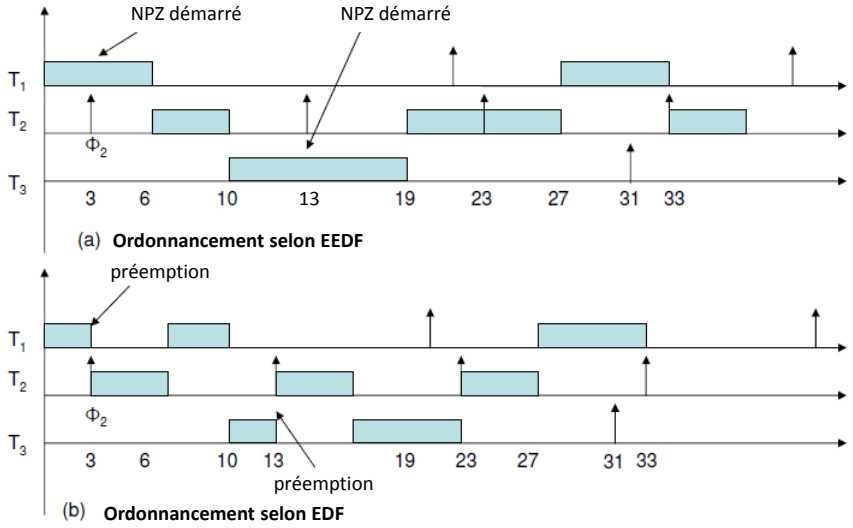


Figure 2.2 – *Exemple d'ordonnancement EDF et EEDF.*

2.3.3.2 Résultats

Nous avons comparé notre algorithme EEDF avec l'algorithme classique EDF et nous avons effectué des simulations à l'aide de l'outil CoFluent [94]. Nous avons mesuré le nombre de préemptions pour un calcul statique des $NPZs$ et également pour un calcul dynamique. Nous avons considéré différents ensembles de tâches afin de comparer nos résultats. Les paramètres de ces tâches sont générés aléatoirement. Le taux d'utilisation de toutes les tâches dans chaque ensemble de tâches est de 98%. Nous pouvons observer sur la Figure 2.3 que l'algorithme EEDF a significativement réduit le nombre de préemptions. Nous avons montré que ces préemptions sont encore réduites si nous faisons nos calculs de façon dynamique et de manière plus réaliste (Figure 2.4). Nous avons également observé que les deux algorithmes, à savoir EDF et EEDF, ont moins de préemptions de tâches si la charge du processeur diminue.

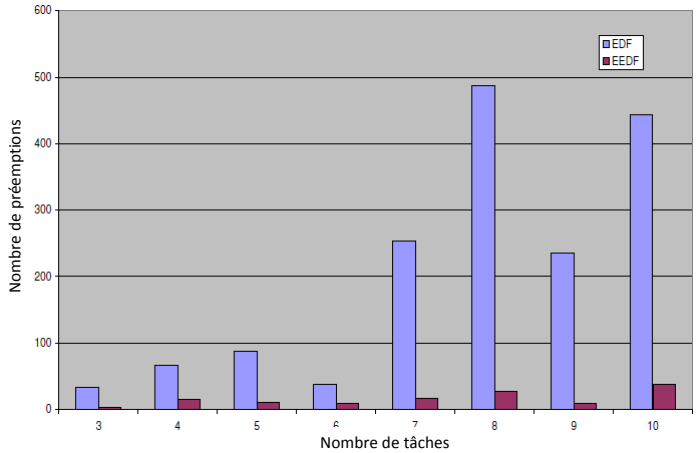


Figure 2.3 – *Comparaison du nombre de préemptions entre EDF et EEDF.*

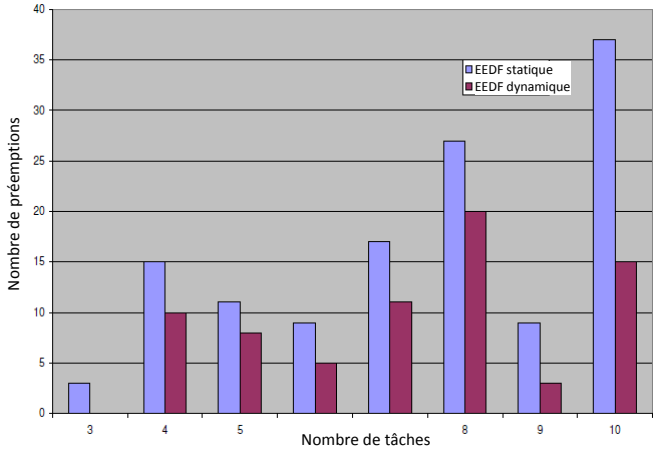


Figure 2.4 – *Calculs statiques versus calculs dynamiques pour EEDF.*

2.3.4 Ordonnancement ERM

2.3.4.1 Principe

L'algorithme d'ordonnancement RM a une règle simple qui attribue des priorités aux tâches en fonction de leur période. Précisément, les tâches avec des périodes plus courtes ont des priorités plus élevées. Puisque les périodes sont constantes, l'algorithme RM assigne statiquement les priorités. En d'autres termes, les priorités sont assignées aux tâches avant l'exécution et ne changent pas au fil du temps. Par ailleurs, l'algorithme RM est intrinsèquement préemptif : la tâche en cours d'exécution est préemptée par une tâche nouvellement arrivée ayant une période plus courte. Nous définissons un paramètre NPZ_i de la tâche T_i comme définie dans la section 2.3.2. la démonstration de la preuve d'ordonnançabilité de ERM est présentée dans le manuscrit [93], section 3.2.2.1.

2.3.4.2 Résultats

Nous avons comparé l'algorithme ERM avec l'algorithme RM et nous avons effectué des simulations avec l'outil STORM [11]. Nous avons mesuré le nombre de préemptions entre l'algorithme RM et ERM. Comme pour l'algorithme EEDF, nous avons considéré plusieurs ensembles de tâches afin de comparer nos résultats. Les paramètres de ces tâches sont également générés aléatoirement. Le taux d'utilisation pour chaque ensemble de tâches varie entre 70% et 100%. Les calculs de NPZ basés sur ERM sont extraits à partir du test d'ordonnançabilité des tâches, et si un ensemble de tâches n'est pas ordonnançable, la simulation n'est pas effectuée. Nous pouvons observer sur la Figure 2.5 que le nombre de préemptions des tâches dans le cas de l'algorithme RM est presque 2,5 fois plus élevé que celui des tâches ordonnancées par l'algorithme ERM.

11. STORM (TOol for Real time Multiprocessor scheduling) est un outil de simulation développé par l'IRCCyN dans le contexte du projet PHERMA, `http://pherma.irccyn.ec-nantes.fr`

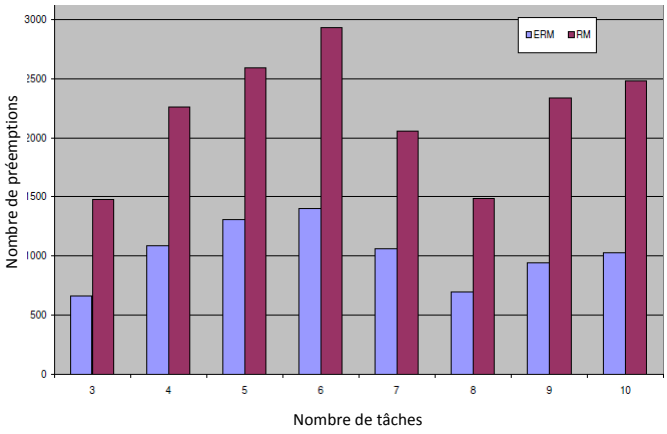


Figure 2.5 – *Comparaison du nombre de préemptions entre RM et ERM.*

2.3.5 Limitation de la préemption en cascade pour EEDF et ERM

La préemptions en cascade est un phénomène qui augmente le nombre de préemptions dans une grande proportion dans le cas des algorithmes d'ordonnancement RM et EDF. Ce phénomène se produit lorsqu'une tâche est préemptée par une tâche plus prioritaire nouvellement relâchée. Alors cette nouvelle tâche en exécution est également préemptée par une autre tâche de priorité supérieure (Figure 2.6). Dans le pire des cas, ces préemptions enchaînées peuvent être égales au nombre de tâches au sein d'un ensemble de tâches. NPZ_i introduit pour chaque tâche peut significativement minimiser ce phénomène (Figure 2.7).

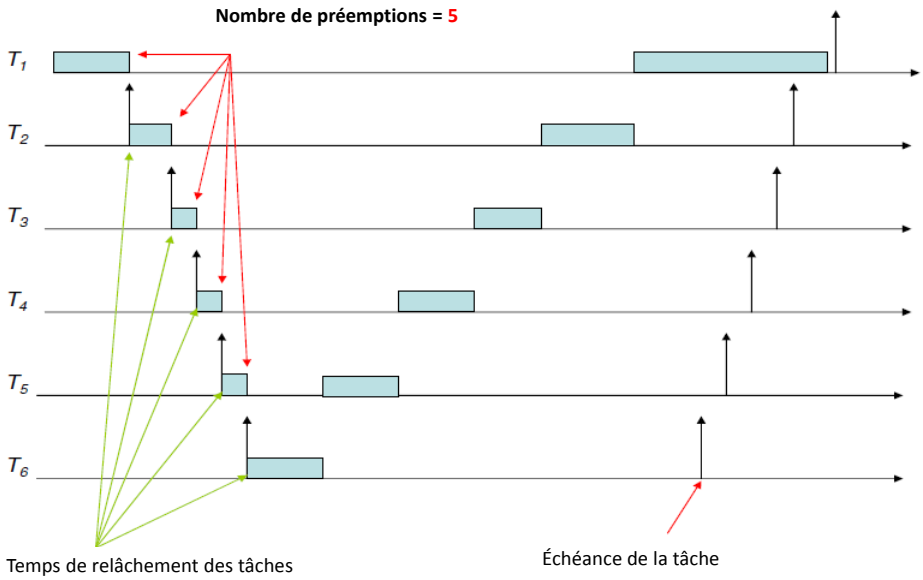


Figure 2.6 – *Exemple de préemptions en cascade avec l'algorithme EDF.*

En effet, les algorithmes EEDF et ERM minimisent efficacement le nombre de préemptions. Nous remarquons que EEDF et EDF introduisent moins de préemptions si la charge du processeur est faible. Cependant, il faut noter que si nous voulons minimiser la consommation d'énergie à l'aide d'algorithmes basés sur le DVFS qui réduisent la fréquence de fonctionnement du système et conduisent à une augmentation de la charge du processeur, le nombre de

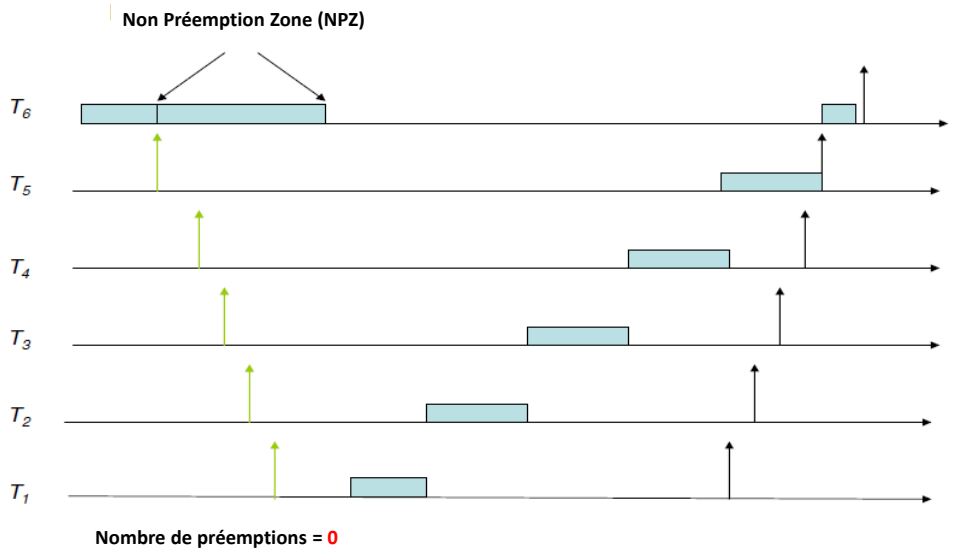


Figure 2.7 – *Exemple de préemptions en cascade avec l'algorithme EEDF.*

préemptions augmentent en conséquence.

2.4 Intergiciel auto-adaptatif pour de futures architectures reconfigurables

2.4.1 Introduction

Le développement de circuits reconfigurables qui pourrait se spécialiser ou s'adapter lui même durant l'exécution, est devenu de plus en plus courant. Les futures architectures reconfigurables auront ces dispositifs de calculs qui seront considérés comme des blocs de base. Ainsi, l'architecture reconfigurable pourrait faire des regroupements de ces blocs à la volée pour exécuter des applications concurrentes. Les blocs logiques génériques gros grains ainsi que leurs connections dans les circuits reconfigurables actuels devraient migrer vers des architectures reconfigurables auto-adaptatives gros grains et ayant des ressources de communications très bien structurées et performantes. Le concepteur devra réfléchir sur la meilleure façon de gérer cette auto-adaptivité afin d'anticiper le bon déroulement de l'exécution des applications sur de telles architectures. Ainsi, il y a un besoin non seulement d'exploitation du parallélisme des applications décrites au niveau micro-thread, mais le système doit également supporter un intergiciel dynamique et auto-adaptatif pour ordonnancer tous ces micro-threads dont sont composés les applications. Nous mettons l'accent sur le problème de l'allocation dynamique et de l'ordonnancement des ressources sur une telle architecture.

Afin de fournir une puissance de calcul suffisante pour répondre aux besoins croissants des applications, les concepteurs s'efforçent d'améliorer les capacités d'une seule machine ou de concevoir un système distribué composé d'un ensemble évolutif de machines. Comparé à autrefois où l'amélioration était principalement liée à l'évolution technologique du matériel, la conception de systèmes distribués partageant des ressources est considérée comme plus

complexe. Certains systèmes distribués bien connus composés de ressources hétérogènes sont Condor [95], NetSolve [96], Nimrod [97], Globus et l'environnement de calcul Grid [98]. Sabin et al. [99] proposent un méta-ordonnanceur centralisé pour ordonnancer des tâches parallèles dans plusieurs machines hétérogènes. De même, Arora et al. [100] présentent un ordonnancement complètement décentralisé, dynamique, et adapté pour l'équilibrage de charge dans un environnement Grid. Toutes ces approches ne traitent pas des modèles d'applications où les threads concurrents sont créés et gérés à l'exécution. Ces méthodes ne visent pas les futures architectures où chaque ressource d'un processeur a une capacité à s'optimiser ou à s'adapter par lui-même et à s'interconnecter avec d'autres ressources pour former un ensemble de blocs qui permettra d'exécuter l'application concurrente. Enfin, ÆTHER est un projet européen IST-FET (Information Society Technology-Technologies futures et émergentes) ayant comme objectif principal d'étudier de nouvelles technologies de cellules auto-adaptatives pour les futures applications embarquées et pervasives dont nous sommes partenaires.

Le système ÆTHER est hiérarchique, tant au niveau fonctionnel que architectural. Au niveau fonctionnel, l'application est écrite pour qu'elle s'auto-adapte afin de bien correspondre à l'objectif de l'application et de faire face aux changements impromptus de l'environnement. Les applications sont très dynamiques par nature où les threads concurrents, qui composent l'application, sont instanciés dynamiquement en fonction du nombre de ressources du système. L'architecture du système n'est pas traditionnelle puisqu'elle n'est pas la seule unité de calcul. Il s'agit d'un réseau d'un nombre donné de SANEs. Un SANE est une entité auto-adaptative en réseau qui peut s'auto-optimiser lui-même.

Nous proposons un algorithme pour l'ordonnancement de tâches temps réel dur et souple dans une architecture qui est composée de plusieurs unités de traitement (SANE). Ces unités de calcul sont auto-adaptatives et ont la capacité à s'optimiser selon les besoins de l'application. L'application aide à la création et à l'instanciation de threads concurrents en cours d'exécution. L'objectif principal de l'algorithme proposé est de maximiser l'ordonnançabilité des tâches souples, sans compromettre l'ordonnançabilité des tâches dures. L'algorithme a la caractéristique inhérente de dégrader/améliorer la Qualité de Service (QoS), par la gestion dynamique et concurrente des tâches en allouant davantage de ressources aux tâches les plus appropriées, c'est-à-dire que les tâches temps réel dur ne sont pas toujours favorisées par rapport aux tâches temps réel souple. Cet algorithme permet de maximiser l'exécution de l'exécution concurrente des tâches en distribuant les ressources disponibles à des milliers de threads concurrents formant les tâches et où l'objectif est d'assurer des garanties d'échéances pour les tâches.

Nous allons tout d'abord définir le problème (section 2.4.2) pour ensuite proposer un modèle architectural (section 2.4.3) et applicatif (section 2.4.4). Pour finir, nous discuterons de l'ordonnancement et de l'allocation de ressources sur cette architecture reconfigurable (section 2.4.6).

2.4.2 Définition du problème

Nous avons n tâches $\tau = T_1, T_2, \ldots, T_n$ et les ressources parallèles R_x qui correspondent à des SANEs. Cet ensemble de ressources s'auto-adapte lui-même selon la tâche qui s'exécute

dessus. Ces ressources forment des ensembles qui se configurent et s'auto-adaptent [101] avec des connexions entre les SANEs pour exécuter des tâches tandis que les tâches ont la capacité de gérer dynamiquement (créer des threads à l'exécution) la concurrence en fonction de la disponibilité des ressources. Les ressources système sont limitées alors que les demandes de toutes les tâches ne peuvent être satisfaites simultanément. Le temps d'exécution de la tâche décroît suivant le nombre de ressources allouées et ce jusqu'au maximum de parallélisme dans la tâche. Pour certaines tâches où les parties concurrentes des tâches sont dépendantes, le temps d'exécution d'une tâche est décroissante jusqu'à un point (appelé seuil), puis il commence à augmenter après ce point pour chaque ressource allouée. La tâche T est représentée comme une séquence d'exécution concurrente de μthreads (μTs [12]). Le niveau de concurrence pour chaque séquence de tâches peut être différent. Nous nous sommes intéressés à l'ordonnancement de ces tâches sur les ressources parallèles R_x telles qu'un maximum de tâches puissent respecter leurs échéances et que l'utilisation des ressources puisse être maximisée. Ceci correspond à l'architecture ÆTHER. Nous allons détailler notre approche afin que les tâches aient des garanties temps réel en calculant la demande de ressources minimale de chaque tâche et la manière par laquelle les ressources sont allouées dynamiquement pour augmenter la qualité de service (QoS).

2.4.3 Le modèle architectural

Un SANE [101] est présenté comme une entité de base de calcul dont l'objectif est d'être mis en réseau avec d'autres entités de même type pour former des systèmes complets. Chacune de ces entités est destinée à être auto-adaptatives, ce qui implique qu'elles peuvent modifier leur propre comportement à réagir suivant les changements de leur environnement ou bien de respecter certaines contraintes. La figure 2.8 représente un SANE. Le contrôleur permet de modifier l'état de l'implémentation matérielle du SANE en changeant certains paramètres de la tâche actuellement chargée, ainsi que de changer d'implémentation de la même tâche ou encore de changer complètement de tâche. Les tâches de calcul sont chargées dans l'unité de calcul reconfigurable. Cette unité peut être programmée par un bitstream si elle est assimilée à un FPGA ou comme un fichier binaire exécutable pour les processeurs. L'existence d'un processus de surveillance associé à un contrôleur d'adaptation offre au SANE la capacité d'auto-adaptation. La dernière partie du SANE est l'interface de communication. Elle est dédiée à la collaboration entre les éléments matériels SANEs qui forment l'architecture. Le processus de collaboration se fait à travers un mécanisme de publication/découverte qui permet à un SANE de publier ses propres capacités et paramètres, et ainsi de découvrir l'environnement de calcul formé par les autres SANEs situés dans son voisinage immédiat. Ce mécanisme permet aux SANEs d'échanger leurs tâches ou tout simplement de cloner leurs états vers d'autres SANEs.

Le processeur SANE (Figure 2.9) est une architecture d'exécution reconfigurable composée de milliers d'éléments SANEs. Ces éléments sont interconnectés de façon dynamique et auto-adaptative pour exécuter des applications concurrentes. Le processeur SANE est un processeur multi-cœur avec des éléments SANEs assimilés à des cœurs de processeur. Le

12. micro Threads

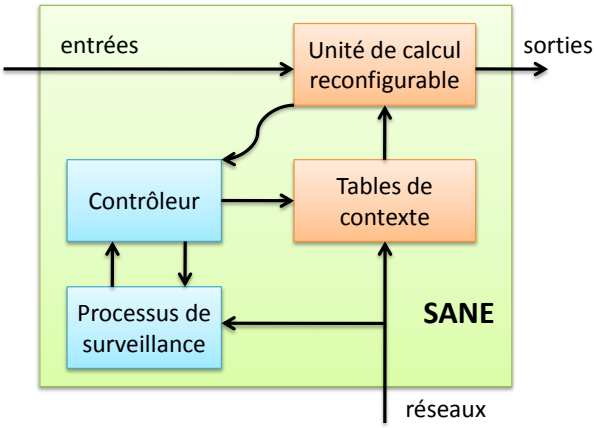


Figure 2.8 – *Représentation d'un SANE.*

processeur SANE est bien plus flexible qu'un simple processeur multi-cœur en raison de sa capacité non seulement de reconfiguration de ses cœurs (SANEs) mais aussi de restructuration de l'interconnexion entre eux. Les éléments SANE (égal aux ressources allouées dynamiquement à la tâche T_i) sont combinés pour former un ensemble de SANEs afin d'exécuter la tâche T_i. Comme le nombre de tâches concurrentes en cours d'exécution varie dynamiquement, les ensembles de SANEs formés au moment de l'exécution, à divers instants, sont de tailles différentes. Par ailleurs, les ressources affectées à chaque tâche varie. Le nombre d'ensembles de SANEs dans un processeur SANE change également à l'exécution (Figure 2.10 (a),(b)).

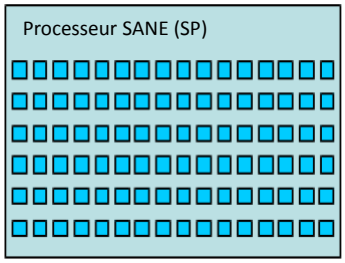


Figure 2.9 – *Exemple d'un processeur SANE composé d'éléments SANEs.*

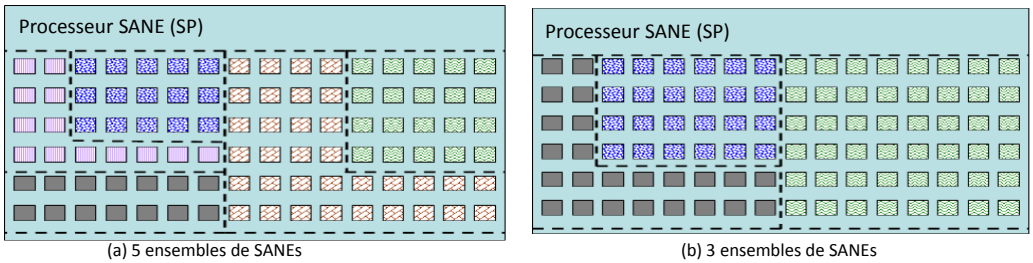


Figure 2.10 – *Ensembles de SANEs.*

2.4.4 Le modèle applicatif

Il existe deux types de parallélisme à exploiter : le parallélisme des tâches et le parallélisme de données. Dans notre approche, le parallélisme est exploité à travers un langage de coordination S-Net [9]. Ce langage orchestre les composants asynchrones qui communiquent les uns avec les autres et avec leur environnement d'exécution uniquement par des flux signés. Les unités de programme d'application sont décrites dans un langage de programmation appropriés à part entière tels que C, Java, etc... tandis que les aspects de communication, de concurrence et de synchronisation (indiqué par le terme coordination) sont capturés par une procédure distincte. Le programme S-Net est compilé en μthreadedC (μTC) [18], qui est utilisé comme langage de programmation utilisateur. μTC est capable d'exprimer la concurrence hétérogène statique et dynamique ainsi que la concurrence homogène. Seul un petit nombre de constructions est ajouté au langage C avec des sémantiques pour la mémoire de synchronisation.

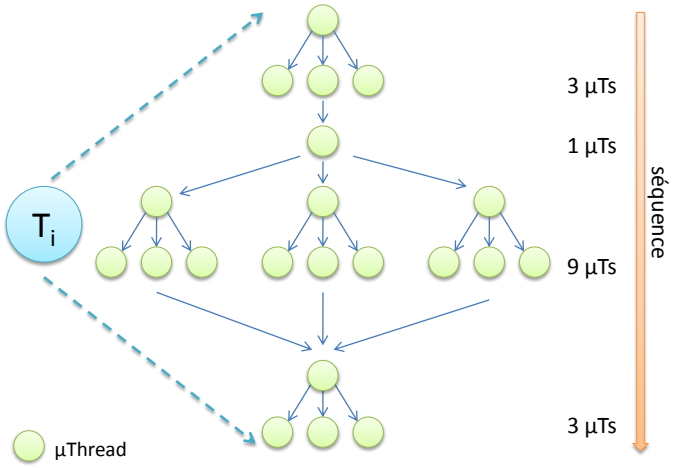


Figure 2.11 – *Représentation d'une tâche.*

Dans l'exemple de la figure 2.11, nous pouvons observer que dans une structure de tâches, nous avons un nombre différent de μTs à des niveaux distincts. Ces μTs représentent la limite maximale correspondant au niveau maximum de parallélisme. Cependant, les μthreads créés lors de l'exécution peuvent avoir des valeurs inférieures à la valeur maximale selon la disponibilité des ressources.

2.4.5 Structuration des tâches auto-adaptives

Les éléments SANE, ayant des capacités d'auto-optimisation, nécessitent un intergiciel dynamique et auto-adaptatif pour faire face à ces optimisations des SANEs. L'objectif de cet intergiciel (que nous appelons également OE [13]) serait de distribuer les ressources d'une manière efficace. L'optimisation réalisée par les SANEs pour n'importe quelle tâche peut être de différents types :

13. Operating Environment

- **SANE dédié pour une tâche :** pourrait s'optimiser pour fournir des fonctionnalités dédiées, implémentées en matériel pour une tâche. Si un élément SANE a du matériel dédié pour une tâche ou qu'il peut s'auto-optimiser lors de l'exécution pour réaliser toutes les séquences de la tâche, alors il n'y aura pas de modification de la structure de cette tâche spécifique.
- **SANE optimisé pour une famille (ou ensemble) de μTs pour une tâche :** pourrait s'optimiser pour exécuter une famille complète ou partielle de μTs concurrents. Un SANE peut être spécialisé seulement pour exécuter une partie de la tâche au lieu de la tâche complète. En effet, il peut fournir de meilleurs résultats pour une famille de μTs. Dans un tel scénario, la structure de tâche sera changée (Figure 2.12) et il y aura un besoin de recalculer les temps d'exécution des tâches avec les nouveaux paramètres, c'est-à-dire le nombre de séquences, le nombre de threads à chaque niveau (Figure 2.12) et le pire-cas d'exécution du thread modifié. Les ressources sont allouées à différentes tâches en fonction de leur structure et de leur échéance.

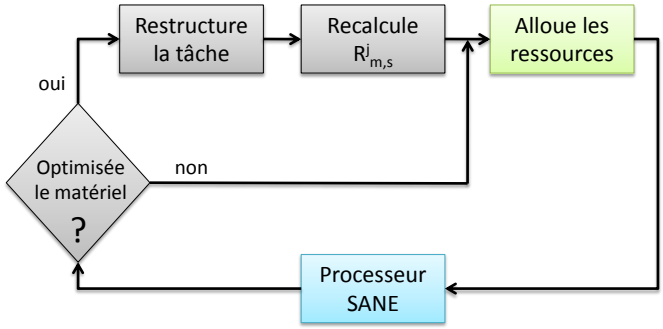


Figure 2.12 – *Restructuration d'une tâche auto-adaptive.*

2.4.6 Ordonnancement et allocation de ressources

Un algorithme d'ordonnancement idéal est celui où chaque tâche est assignée à des ressources proportionnellement à son poids pendant toute sa durée d'exécution. Un algorithme d'ordonnancement idéal est impraticable :

- Dans le cas des processeurs à usage général, lorsque les capacités de calcul d'un processeur ne peuvent pas être affectées à des tâches différentes en proportion de leur poids.
- Dans les cas réels où l'application ne dispose pas de μTs égaux au nombre qui correspond à son poids. Le nombre de μTs qu'une application peut exécuter en parallèle varie au cours de son exécution.

Dans le cas de processeurs SANE, les ressources (SANEs) peuvent être attribuées proportionnellement et dynamiquement pour exécuter une tâche en parallèle. Si le nombre minimum de μTs dans n'importe quelle séquence d'une tâche est un entier multiple du nombre de ressources correspondant à son poids, alors cette tâche peut être ordonnancée idéalement. Mais si une tâche T_i a un nombre de μTs inférieur aux ressources minimales (notées $R^i_{m,s}$) à un certain niveau et un nombre supérieur de ressources minimales à d'autres niveaux (qui est le cas

la plupart du temps), alors cette tâche ne peut être ordonnancée idéalement. Pour fournir des garanties à cette tâche, la réservation de ressources correspondant au minimum à son poids est nécessaire. En contre-partie, certaines ressources resteront inutilisées et provoqueront du gaspillage.

Nous avons développé un algorithme [93], section A.6 qui permet d'allouer dynamiquement des ressources pour des tâches temps réel dure et souple. De plus, l'ordonnançabilité a été démontrée dans certaines conditions [93], section A.7.

2.4.7 Le simulateur SystemC

Afin de valider cette approche, un simulateur SystemC présenté figure 2.13 a été conçu. Nous étions en charge de concevoir l'architecture de l'OE, l'oganisation des fichiers d'entrées, l'ordonnançabilité et l'ordonnancement des applications. Notre partenaire, le Lab-STICC [14], pôle CACS [15], se focalisait sur l'architecture inter-SANEs, et sur la répartition des applications sur ce réseau de SANEs à partir d'algorithmes basés principalement sur le principe des enchères.

L'application est décrite sous la forme de graphes au format XML. L'architecture des SANEs comme le type (fonctions dédiées), la capacité (nombre de SANEs par processeur SANEs), la topologie et capacité (débit) du réseaux d'interconnexion inter-SANEs sont également au format XML. Le modèle de coût, représenté par le tableau dans la figure 2.13, est décrit en XML. Celui-ci représente un exemple de description de la fonction F_1 utilisée dans le graphe de l'application. Ce tableau représente les différentes implémentations possibles de F_1 et fournit des informations tels que le temps de calcul, le taux d'occupation du CPU, les ressources matérielles (Hw), la mémoire utilisée et le taux d'occupation des communications ainsi que le nombre d'instances utiles de SANEs. Pour finir, des scénarios de simulations sont définis afin de tester notre gestion de l'application sur un réseau de SANEs.

C'est à partir de toutes ces entrées que nous décrivons une politique de l'OE. La politique de l'OE permet de spécifier la stratégie de répartition de l'application, la manière d'ordonnancer les graphes de tâches. Contrairement aux autres entrées, celle-ci est décrite sous forme de script, permettant d'obtenir un comportement plus dynamique.

L'OE est composé de deux parties principales : un module de vérification de l'analyse d'ordonnançabilité qui permet de savoir si un processeur SANE est capable d'exécuter l'application ou une partie de l'application, et un module qui ordonnance les tâches des applications en tenant compte des contraintes temps réel. Ces modules implémentent les algorithmes dont le principe a été expliqué ci-avant. Les résultats obtenus sont des traces qui permettent d'évaluer les performances des applications sur les processeurs SANEs comme par exemple le taux d'occupation des ressources sur les processeurs SANEs, l'évolution et l'état des tâches des applications.

14. Laboratoire en sciences et techniques de l'information, de la communication et de la connaissance à Lorient

15. Communications, Architecture et CircuitS

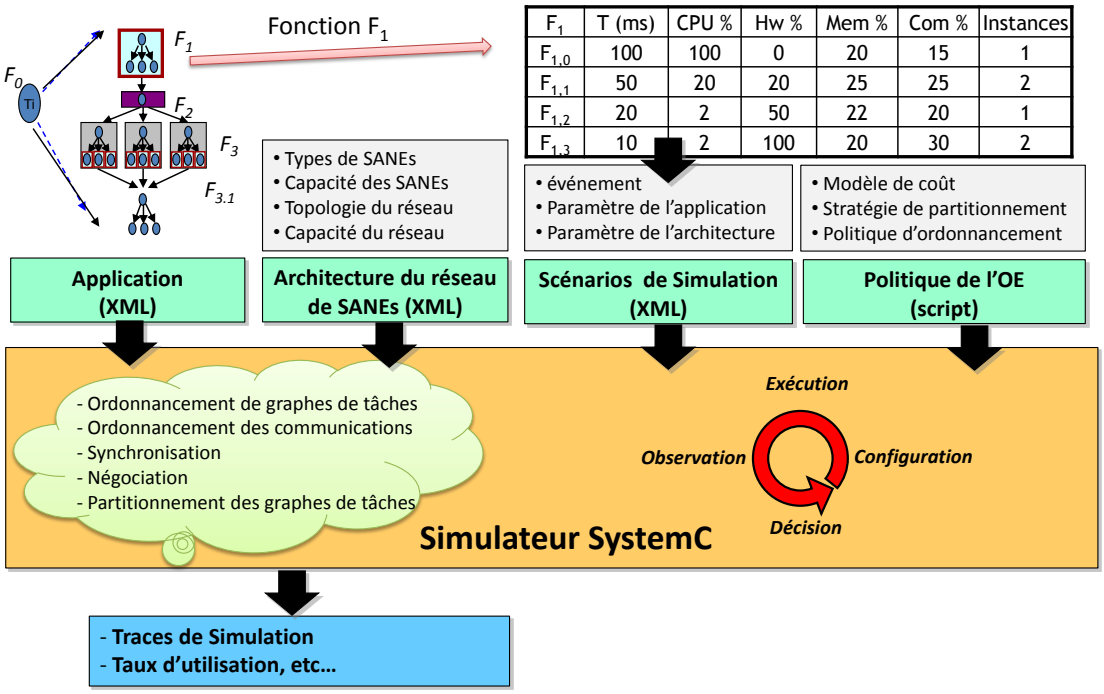


Figure 2.13 – *Environnement du simulateur SystemC.*

2.5 Conclusion

Les algorithmes d'ordonnancement temps réel de type EDF, LLF ou MUF sont optimaux dans le cas monoprocesseur et les algorithmes Pfair ou LLREF [16] le sont dans le cas multiprocesseur. Nous nous sommes attachés à montrer qu'il est possible de proposer des algorithmes d'ordonnancement optimaux (sous certaines hypothèses), c'est-à-dire ayant une borne d'ordonnançabilité égale à la capacité de traitement de l'architecture cible, et ce avec une efficacité accrue du fait d'une réduction du coût de gestion induit par l'ordonnanceur. Ce gain en efficacité est obtenu en utilisant au mieux des paramètres implicites des tâches, évalués à l'exécution. Il en découle en particulier une augmentation de la Qualité de Service (QoS) globale de l'application.

Nous avons proposé différentes techniques telles que les algorithmes RUF, EEDF, ERM, ASEDZL, un algorithme d'ordonnancement hiérarchique et un modèle tenant compte des communications inter-processeurs. Nous avons détaillé les algorithmes EEDF/ERM qui ont fait l'objet d'un brevet.

Enfin, nous avons proposé dans le cadre du projet Européen ÆTHER une technique d'ordonnancement auto-adaptative d'applications pour de futures architectures. On peut penser que les architectures du futur seront intelligentes pour pouvoir s'adapter/s'auto-configurer en fonction de la demande variable de traitement par les applications. Pour obtenir des garanties sur les échéances de ces applications auto-adaptatives, nous avons proposé de calculer de manière statique la demande minimale en ressources pour chaque tâche, puis, à l'exécution d'allouer à chaque tâche plus de ressources que cette demande minimale en fonction du degré de parallélisme intrinsèque de la tâche observée et du nombre de ressources disponibles. Un

16. Least Local Remaining Execution First

simulateur, développé en SystemC, a été conçu pour valider cette approche. L'architecture considérée dans cette étude est quelque peu futuriste mais semble avoir son utilité dans les prochaines architectures reconfigurables où les équivalents des SANEs seront des blocs de granularité gros grain intégrés dans les circuits reconfigurables de nouvelles générations pouvant être comparés à des cœurs de GPUs.

Cette dernière étude nous a permis de mettre en place les deux axes que nous allons détailler dans les prochains chapitres.

Afin de mener à bien ces travaux de recherche, un doctorant [93] a participé à cet axe, un ingénieur d'étude pour le développement du simulateur en SystemC et de stagiaires de MASTER pour mettre en place les concepts de l'intergiciel. L'ensemble des travaux mené au sein de cet axe de recherche a conduit à 1 brevet [161], 10 publications [173, 176, 177, 178, 179, 186, 180, 181, 182, 183] dans des conférences internationales et 2 publications [196, 198] dans des conférences nationales.

CHAPITRE 3

Placement et ordonnancement de tâches sur des architectures reconfigurables

3.1 Contexte

Au cours de ces dernières années, l'utilisation des systèmes embarqués s'est accentuée pour inclure une grande variété de produits, comme les appareils photo numériques, les réseaux de capteurs, les systèmes d'imagerie médicale, les moyens de communications sans fil, etc. La complexité des algorithmes de ces systèmes dépasse fréquemment la capacité de traitement des processeurs. Dans certaines applications, une solution consiste à introduire des co-processeurs afin de déporter une partie du traitement des algorithmes et ainsi de soulager le processeur. De plus, dans certaines applications, ces systèmes imposent aussi aux concepteurs de fortes contraintes temps-réel. Afin de résoudre cette montée en puissance des traitements, le calcul reconfigurable est considéré comme une solution qui consiste essentiellement à modifier les fonctions de calcul en cours d'exécution, et ainsi diminuer la consommation d'énergie et l'espace des ressources de calculs tout en gardant une haute performance de calculs.

De plus, la constante évolution des systèmes embarqués implique une réduction du temps de mise sur le marché de nouveaux produits. Les concepteurs ont tendance à élever le niveau d'abstraction afin de masquer les détails d'exécution qui ne peuvent être pertinents durant les premiers stades du développement. Par ailleurs, la capacité des puces est croissante et les systèmes sur puce (SoC) deviennent de plus en plus complexes, incluant un ou plusieurs microprocesseurs, des mémoires, des interfaces périphériques, etc. Lorsque le système est implémenté sur un circuit programmable, par exemple un FPGA, on parle de système sur puce programmable (SoPC [1]). Les SoPCs ont tendance à être un compromis entre une solution logicielle complète mais lente, et une solution ASIC (Application Specific Integrated Circuit), plus puissante, mais aussi beaucoup plus chère [102].

Les derniers SoPCs utilisant les FPGAs peuvent bénéficier d'une reconfiguration partielle

1. System on Programmable Chip

PR [2]. Cette fonctionnalité permet de reconfigurer dynamiquement une zone du FPGA tandis que le reste du FPGA continue à fonctionner. Ceci est par exemple possible dans les circuits programmables de la société Xilinx, et également bientôt supportés par les circuits de la société Altera. Cette technique présente un grand intérêt quand il s'agit de la mise en œuvre d'applications très gourmandes en calcul comme le transcodage vidéo. À l'aide de cette technique, les ressources peuvent être mutualisées afin de sélectionner un FPGA plus petit et ainsi réduire le prix du produit et la consommation d'énergie [103].

Cependant, en dépit des propriétés plutôt intéressantes, la reconfiguration partielle n'est pas encore bien établie dans l'industrie [104] pour deux raisons principales : La première est le placement et l'ordonnancement de ces zones reconfigurables qui sont considérés comme des problèmes NP-complets [80]. La seconde raison est que la reconfiguration partielle ne parvient pas à trouver une solution satisfaisante pour modéliser son comportement et le contrôle associé à un haut niveau d'abstraction. Il est donc impossible de valider une approche durant les premiers stades du processus de développement. Cette approche, basée sur un modèle de haut-niveau, permettrait de choisir rapidement l'architecture et de la valider, en évitant des retours intempestifs dans les phases avancées de développement du projet. Par ailleurs, il n'est actuellement pas possible de tester et/ou valider un algorithme d'ordonnancement avec les travaux existants et dans des outils de modélisation, et conduisant à apporter une réponse à la reconfiguration partielle.

3.1.1 Le placement et l'ordonnancement des tâches matérielles

L'ordonnancement et le placement de tâches sont prouvés comme des problèmes d'optimisation combinatoire. Généralement, ce type de problèmes est très complexe. Par conséquent, le concept d'optimisation combinatoire doit être bien étudié et les méthodes de résolution possibles pour ce type de problèmes doivent être bien traitées. Les systèmes temps-réel tendent vers des méthodes d'ordonnancement qui gèrent efficacement les exécutions de tâches et respectent leurs contraintes temps-réel dures. Le non respect de ces contraintes peut mener à des conséquences catastrophiques pour le système. En outre, dans le cas de tâches dépendantes, en plus de la périodicité et des contraintes d'échéances, l'ordonnancement doit considérer les contraintes de précédence et doit assurer le transfert de données entre les tâches inter-dépendantes.

Contrairement aux tâches purement logicielles exécutées sur les processeurs, les SoPCs ajoutent un nouveau problème, le problème du placement des tâches matérielles sur des zones reconfigurables qui sont formées de ressources hétérogènes. En plus de l'ordonnancement respectant les contraintes temps-réel, les tâches matérielles requièrent un nombre suffisant de ressources hétérogènes sur les SoPCs pour pouvoir s'exécuter. Autrement, la tâche matérielle, qui ne peut être placée par manque de ressources, sera rejetée même si l'ordonnancement est faisable. Ainsi, le placement consiste à gérer l'espace libre de ressources et à insérer les tâches élues dans leurs zones reconfigurables les plus convenables. Le placement et l'ordonnancement sont deux problèmes hautement corrélés. Nous pouvons considérer que l'ordonnancement de tâches sur les SoPCs peut être vu comme un ordonnancement multiprocesseur avec une

2. Partial Reconfiguration

contrainte additionnelle globale de ressources.

3.1.2 Modélisation de la reconfiguration dynamique et partielle

Les travaux réalisés pour fournir un support dans les outils de modélisation statique de la reconfiguration partielle sont principalement axés sur deux outils principaux, SystemC [105] et UML [3] [106] (Unified Modeling Language).

UML est un langage de modélisation graphique et orienté-objet initialement dédié à la modélisation de logiciels. Néanmoins, il est largement utilisé pour les systèmes matériels et mixtes, principalement à cause des nombreux supports d'outils UML et en raison de son mécanisme d'extension utilisé pour personnaliser des modèles UML. Pour les systèmes embarqués, il y a le profil MARTE [4] [107] (Modeling and Analysis of Real-Time Embedded Systems). MARTE étend la capacité de modélisation d'UML pour les systèmes temps réel embarqués. Pourtant, MARTE ne supporte pas nativement la modélisation PR. Les travaux réalisés dans [108, 109] présentent une approche basée sur MARTE qui remédie à ce manque en utilisant le flot GASPARD2. Une autre approche basée sur MARTE est décrite dans [110]. Elle s'appuie sur le projet MoPCoM qui a défini une méthodologie de co-design basée sur MDA [5].

Quant à SystemC, il fournit des outils pour la conception et la vérification des systèmes matériels, logiciels ou mixtes. Il est devenu un standard de facto (IEEE 1666-2005) dans l'industrie car il est basé sur un langage bien connu dans la communauté des développeurs [111] . Il permet de décrire un système à différents niveaux d'abstraction, du niveau transfert de registres (RTL) jusqu'à des modèles fonctionnels qui peuvent être avec des informations de temps ou non. La modélisation au niveau TLM [112] est un autre paradigme où les communications sont simplifiées et représentées comme des transactions, réduisant ainsi les temps de simulation.

Toutes ces observations nous conduisent à penser que ces problématiques de placement, d'ordonnancement et de modélisation des architectures reconfigurables sont encore jeunes malgré les nombreux travaux déjà réalisés. Dans nos contributions, nous allons nous focaliser sur différentes solutions afin d'appréhender au mieux ces trois problématiques. Nous allons tout d'abord présenté les contributions (section 3.2) de cet axe de recherche. Nous détaillerons ensuite les trois principales contributions où les résultats sont les plus intéressants : le placement et ordonnancement de tâches indépendantes par deux méthodes différentes (section 3.3), le placement et ordonnancement de tâches dépendantes (section 3.4) et le placement et l'ordonnancement par l'utilisation de modèles de haut-niveau et simulation (section 3.5).

3. Unified Modeling Language
4. Modeling and Analysis of Real-Time Embedded Systems
5. Model-Driven Architecture

3.2 Contributions

Dans nos contributions résumées ci-dessous, nous introduisons des méthodes complètes et approximatives ciblant le problème du placement sur des ressources hétérogènes. Nous nous sommes également concentrés sur les problèmes liés à l'ordonnancement des tâches matérielles dépendantes et indépendantes. Des méthodes analytiques, heuristiques et par l'utilisation de modèles en vue de simulation sont suggérées pour faire de l'exploration d'architectures reconfigurables tout en garantissant des contraintes de temps réel et en minimisant le nombre de ressources hétérogènes utilisées.

3.2.1 Placement/Ordonnancement hors-ligne de tâches matérielles indépendantes par des méthodes dynamiques et méta-heuristiques

Cette première contribution s'attaque au placement et ordonnancement de tâches matérielles indépendantes et préemptives à certains endroits de la tâche. Tout d'abord, nous introduisons nos modèles d'abstraction liés aux tâches et à l'architecture (SoPC). Ces modèles sont essentiellement basés sur la technologie choisie (par exemple, FPGA Xilinx Virtex5) et les caractéristiques des tâches (ressources et temps). Nous proposons une approche générique permettant de résoudre cette problématique de placement/ordonnancement. Deux méthodes de résolution appliquées à ce flot sont proposées. Une méthode complète par programmation mathématique en nombres entiers mixtes. Une méthode approximative (méta-heuristique) nommée algorithme *Abeilles* ou *BEE* qui essaye de proposer une solution proche de l'optimale. Les résultats obtenus sont comparés. Les conclusions sur l'efficacité et l'évolution des deux méthodes sont également discutées. Les expériences démontrent que la méthode complète offre une meilleure efficacité en terme de ressources de 33% par rapport à l'algorithme *BEE* et atteint 15% de gain concernant le temps de configuration des zones reconfigurables. Cette contribution est détaillée dans la section 3.3.

3.2.2 Placement/Ordonnancement hors-ligne de tâches matérielles dépendantes par méthode dynamique

Dans cette contribution, nous ajoutons les contraintes de dépendance entre les tâches tout en considérant les contraintes de périodicité et de temps réel dur. Ces tâches sont modélisées par un graphe acyclique orienté. La résolution du placement et de l'ordonnancement hors-ligne de tâches matérielles débute par des analyses de contraintes temporelles et spatiales, puis se poursuit par la vérification de la validité des graphes et la détermination du nombre nécessaire de zones reconfigurables afin de placer et d'ordonnancer les tâches dépendantes constituant le graphe. Au cours de nos analyses, nous résolvons ce problème en utilisant une méthode par programmation linéaire et non-linéaire en nombres entiers mixtes en considérant un seul graphe acyclique. Un exemple de graphe illustre notre démarche. Cette résolution fortement contrainte est optimale et donne la meilleure qualité de placement et d'ordonnancement parmi toutes les solutions candidates. Cependant, elle est prouvée très coûteuse en termes d'espace mémoire et de temps de résolution. Ainsi, elle est uniquement adaptée à des graphes de taille

réduite. Cette contribution est également détaillée dans la section 3.4.

3.2.3 Placement/Ordonnancement en-ligne de multiples graphes acycliques orientés

Comme l'ordonnancement d'un seul graphe acyclique a été considérablement étudié, cette contribution se focalise sur le problème d'ordonnancement de plusieurs graphes acycliques sur des SoPCs hétérogènes. Un schéma dynamique pour l'ordonnancement et le placement de tâches matérielles temps-réel modélisées par des graphes est proposé. De plus, nous prendrons des graphes acycliques à comportements inconnus selon un intervalle d'inter-arrivée fixe. Les approches proposées sont appliquées sur différents ensembles constitués par de multiple graphes temps-réel. En se basant sur des techniques de prédiction et de réutilisation, les mesures démontrent des résultats pertinents dans la réduction du makespan total, sur le taux de rejet de graphes qui ne respectent pas leurs contraintes temporelles, et concernant la minimisation du ralentissement des graphes. D'autres objectifs sont aussi atteints : la réduction des latences de configuration et l'amélioration de l'efficacité en terme de ressources. Quatre approches sont proposées pour exécuter le placement et l'ordonnancement en-ligne de graphes multiples tout en respectant les contraintes temps-réel et de dépendance, et en améliorant la qualité du placement et de l'ordonnancement. Les approches proposées font face aux limitations de notre approche hors-ligne proposée précédemment qui est uniquement applicable pour des petits graphes. En effet, ces approches en-ligne sont capables de placer et d'ordonnancer plusieurs graphes, chaque graphe pouvant avoir un nombre important de tâches. Les résultats montrent que notre approche permet d'obtenir un taux de prédiction du placement des tâches supérieur à 88% qui permet ainsi de réduire le temps de configuration. De plus, la fragmentation interne est proche de 1 ce qui signifie qu'une tâche est placée trés souvent sur sa zone reconfigurable optimale en terme de ressources. Dans cette approche, les contraintes sont de type temps réel souple, c'est-à-dire que le taux de garantie de l'échéance n'atteint pas 100% pour chaque graphe et, par expérimentation, fluctue entre 40% et 100%.

3.2.4 Modélisation haut-niveau pour le placement/ordonnancement de tâches matérielles reconfigurables

Nous constatons un manque d'outils permettant de modéliser à un niveau élevé d'abstraction les FPGAs reconfigurables partiellement, et ceci dans le but de donner suffisamment de liberté au concepteur pour l'évaluation d'algorithmes d'ordonnancement pour des tâches matérielles. Nous proposons une méthodologie prenant en compte la reconfiguration partielle dans notre modélisation de haut niveau. Notre approche s'appuie sur les threads dynamiques permettant de modifier la fonctionnalité des modules en cours d'exécution et sur la modélisation niveau-transactionnel pour toutes les communications. Nous introduisons un gestionnaire de reconfiguration dont l'objectif est de développer et de valider des algorithmes d'ordonnancement pour les tâches matérielles. Par ailleurs, notre simulateur effectue de l'exploration d'architecture afin de trouver une implémentation satisfaisante (en termes de zones reconfigurables) pour une application donnée. Un exemple d'application d'une chaîne vidéo

reconfigurable de transcodage valide notre méthodologie. Cette contribution est détaillée dans la section 3.5 et est en cours au travers du projet ANR ARDMAHN.

3.3 Placement/Ordonnancement hors-ligne de tâches matérielles indépendantes par des méthodes dynamiques et méta-heuristiques

Les développements actuels dans les circuits reconfigurables hétérogènes utilisés dans les SoPCs montrent la nécessité d'intégrer des méthodes efficaces pour la gestion des ressources des tâches matérielles. En outre, la complexité des applications complique la résolution des problèmes du placement et provoque du gaspillage de ressources et un temps de configuration des zones reconfigurables du circuit avant l'exécution de la tâche matérielle. Les problèmes du placement et de l'ordonnancement sont des problèmes multi-objectifs d'optimisation combinatoire qui nécessitent des algorithmes efficaces en terme de temps de résolution pour une solution acceptable.

Nous sommes donc confrontés à deux problèmes difficiles qui sont le placement et l'ordonnancement des tâches matérielles sur les architectures reconfigurables. Nous détaillons notre approche hors-ligne afin de résoudre le placement de tâches matérielles sur SoPCs, indépendantes, temps réel dur et semi-préemptives, c'est-à-dire que des points de préemption sont prédéfinis. Le principe est détaillé dans la section du niveau applicatif. Afin de comprendre notre approche, nous introduisons notre terminologie qui est liée aux caractéristiques des tâches matérielles et à celles des SoPCs. C'est ensuite que nous exposons l'approche proposée dont le problème du placement/ordonnancement est de complexité NP-complet. Nous présentons deux méthodes : une méthode complète par programmation mathématique en nombres entiers mixtes et une méthode itérative approximative, l'algorithme *Abeilles* (algorithme BEE). Pour ces deux méthodes, la qualité de placement est mesurée par le taux d'efficacité des ressources et par le temps de configuration des zones reconfigurables.

3.3.1 Terminologie

Pour notre modèle, nous proposons de prendre en compte les critères suivants : les ressources matérielles de la tâche et l'hétérogénéité du circuit reconfigurable. Nous supposerons que les tâches matérielles sont exécutées dans des zones reconfigurables 2D. La modélisation est classée selon deux niveaux ; le niveau applicatif et le niveau physique.

3.3.1.1 Niveau applicatif

Ce modèle contient les caractéristiques des tâches matérielles. Prenons l'exemple de la Figure 3.1 qui définit le temps d'exécution pire cas (C_A), la période (P_A), l'échéance relative (D_A). Nous supposons que la période d'une tâche donnée est égale à son échéance relative. Chaque tâche matérielle est également caractérisée par un ensemble de points de préemption prédéfinis ($Preemp_{A,l}$). Le nombre de points de préemptions de T_A est noté $NbrPreemp_A$.

Les points de préemption sont des instants où la tâche peut être exclusivement préemptée. Ce choix de préempter la tâche à certains instants prédéfinis est un compromis, proposé dans le projet ANR FOSFOR, et qui permet de sauvegarder un contexte limité de la tâche matérielle en cas de préemption. Si nous avions choisi la préemption totale, la sauvegarde de toute la zone reconfigurable serait alors nécessaire et provoquerait un overhead de sauvegarde/restauration de contexte trop important et ôterait les bienfaits du reconfigurable. De plus, il n'est pas possible de lire certains blocs comme les BRAMs [6] dans les circuits actuels tels que les FPGA Virtex5 et Virtex6.

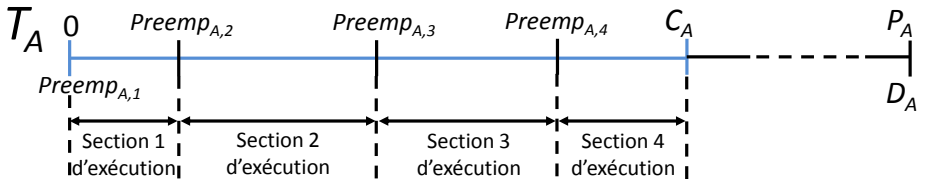


Figure 3.1 – *Caractéristiques d'une tâche T_A.*

3.3.1.2 Niveau physique

Classiquement, un circuit reconfigurable est composé d'un ensemble de cellules logiques reconfigurables telles que des CLBs [7], BRAMs, DSP48s, etc (nous noterons k, le nombre de type de cellules). Cependant, la granularité de reconfiguration n'est pas la cellule logique mais un ensemble de cellules de même type que nous appelons RB_k [8]. Sur l'exemple de la figure 3.2, le circuit Virtex5 a des ensembles de RB_k de 20 CLBMs (RB_1), de 20 CLBLs (RB_2), de 4 BRAMs (RB_3) et de 8 DSP48s (RB_4). En d'autres termes, dans cette technologie, il ne sera pas possible de reconfigurer moins que 20 CLBLs. De plus, les tâches sont également modélisées en RB_k auxquelles un coût ($RBCost_k$) est associé à chacun des types de RB_k et qui est fonction de la rareté du RB_k par rapport aux RBs [9] contenus dans le circuit reconfigurable.

De plus, notre architecture reconfigurable cible est partitionnée en plusieurs régions (figure 3.3) :

- **Une région statique** (SR [10] : elle comprend la zone statique placée, routée et non modifiable durant l'utilisation du circuit. Une région statique est un ensemble de ressources hétérogènes RBs.
- **Des régions reconfigurables** (RR [11]) : ces zones sont dédiées à la partie dynamique où seront situées les différentes zones reconfigurables, appelés $RPBs$ [12] et pouvant accueillir les tâches. Suivant le nombre de ressources disponibles dans le circuit reconfi-

6. Block Random Access Memory
7. Configurable Logic Blocks
8. Reconfiguration Bloc de type k
9. Reconfigurable Blocs
10. Static Region
11. Reconfigurable Region
12. Reconfigurable Physical Blocs

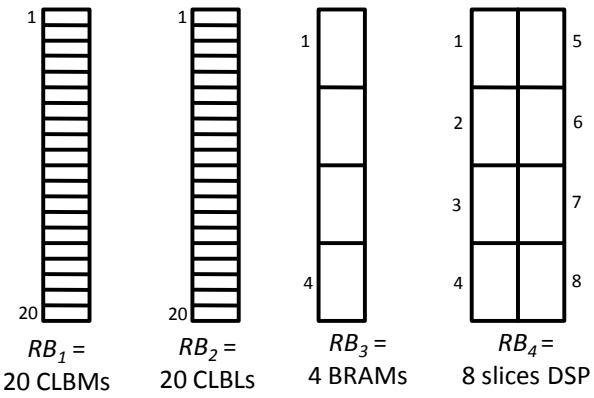


Figure 3.2 – *Exemples de RBs en technologie Virtex5.*

gurable, il est possible de définir plusieurs RRs au sein du circuit. Chaque RR est un ensemble de ressources hétérogènes (RBs).

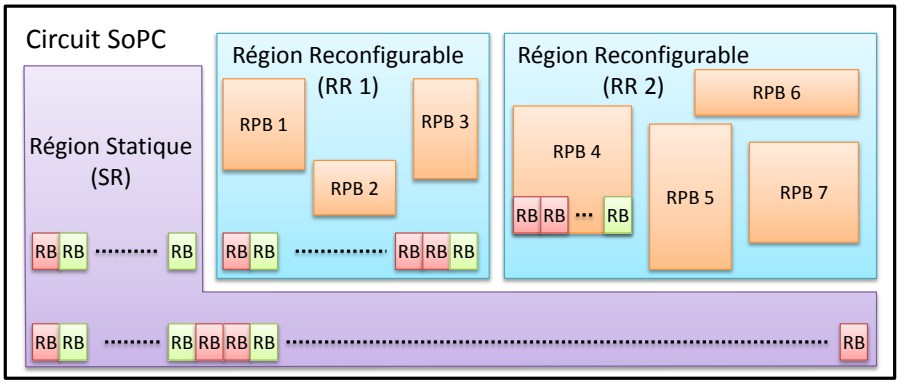


Figure 3.3 – *Les éléments principaux d'un circuit reconfigurable.*

Dans notre approche, nous avons également défini une unité virtuelle, appelée RZ [13], composée de ressources hétérogènes (RB). Chaque type de RZs (RZ_j) est spécialisé pour une classe de tâches matérielles. Ces types de RZs seront déterminés en s'appuyant sur les ressources physiques des tâches matérielles (RB) de l'application, extraites des rapports après-synthèse.

Le modèle de RBs des RZs est intégré dans les blocs physiques reconfigurables (RPB) d'une RR. Ces $RPBs$ rectangulaires 2D représentent les emplacements physiques possibles des RZs. En conséquence, comme illustré par la figure 3.4, les emplacements exacts des RBs dans les RZs (modélisées en RBs) sont déterminés après l'affectation des RZs sur les $RPBs$. Il est alors possible que des $RPBs$ pour une RZ donnée contiennent quelques RBs non utilisés par la RZ. Cette différence ou distance de ressources est appelée la fragmentation interne et est principalement causée par l'hétérogénéité des ressources du circuit reconfigurable. Par exemple, dans la figure 3.4, RPB_2 contient RB_4 qui n'est pas utilisé par la RZ. Cette fragmentation interne est un métrique important pour évaluer la qualité du placement.

13. Reconfigurable Zone

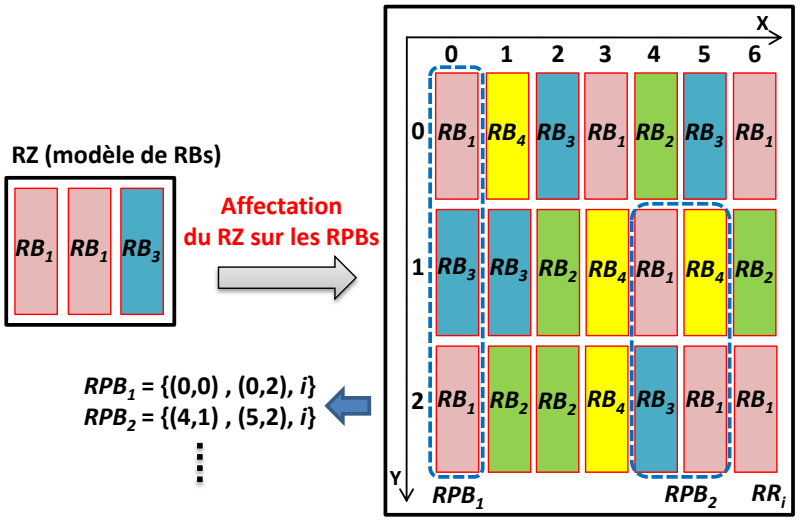


Figure 3.4 – *Exemple d'un RZ allouée à des RPBs.*

3.3.2 Approche proposée pour le placement et l'ordonnancement des tâches

Dans cette section, nous détaillons l'approche pour le placement et l'ordonnancement des tâches matérielles sur un circuit reconfigurable dont le modèle a été présenté ci-avant. Nous adoptons un partitionnement 2D du circuit reconfigurable. À partir de l'ensemble des tâches matérielles et des caractéristiques du circuit reconfigurable cible, nous proposons une stratégie hors ligne composée de trois principaux niveaux.

3.3.2.1 Niveau 1 : Classification des tâches matérielles

Ce niveau prend un ensemble de tâches et fournit les types et les instances de RZs. Les RZs sont des abstractions de classes de tâches et sont définies en fonction des types de ressources nécessaires par les tâches. Ce niveau est composé de trois étapes :

- **Étape 1, La recherche des types de RZ :** cette étape regroupe les tâches partageant les mêmes types de RBs et le même type de RZ. Par exemple (Figure 3.5), les tâches T_A et T_C utilisent uniquement les mêmes types de ressources et ainsi forment un premier type RZ_1. Pour chacune des ressources de RZ_1, nous prenons le maximum de ressource par type de ressource.
- **Étape 2, Classification des tâches :** La seconde étape calcule un coût D pour chaque tâche affectée à une RZ. De plus, le taux d'occupation ($Load\_RZ_j$) est calculé pour chaque RZ_j en fonction de C_A, P_A, $NbrPreemp_A$ et du temps de configuration $Config_j$ qui dépend des ressources de RZ_j.
- **Étape 3, Décision d'ajouter une RZ :** Dans certaines RZ_j, le taux d'occupation $Load\_RZ_j$ peut être supérieur à 100%. Ainsi, cette étape a pour objectif d'ajouter des RZs permettant de diminuer ce taux à une valeur inférieure à 100% si une migration d'une tâche sur une autre RZ compatible n'est pas possible.

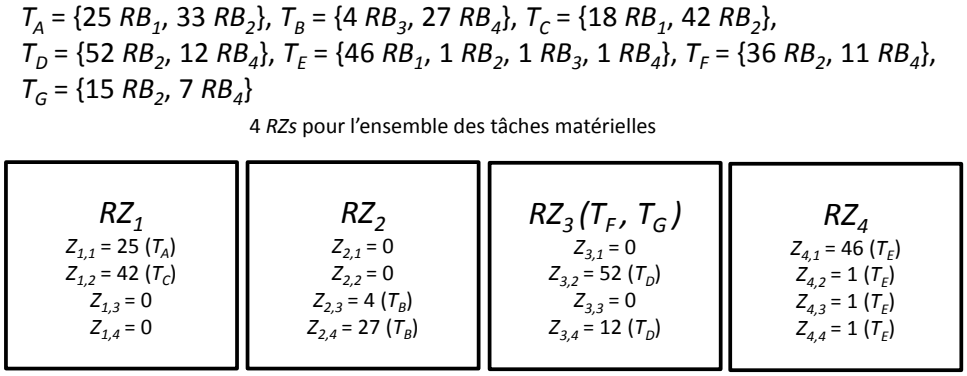


Figure 3.5 – *Exemple de recherche de RZs.*

3.3.2.2 Niveau 2 : Partitionnement des $RPBs$ sur le circuit cible

Le niveau 2 prend les types de RZs fournies par le premier niveau et recherche toutes les possibilités de placement de ces RZs sur le circuit cible. Ces possibilités deviennent alors des blocs physiques reconfigurables (RPB). Un RPB_n doit être compatible en terme de ressources avec une RZ_j c'est-à-dire que le nombre de RBs dans un RPB_n doit être supérieur ou égal au nombre de RBs dans une RZ_j. Néanmoins, les $RPBs$ doivent être les plus proches possibles du modèle de RBs des RZs, afin d'assurer l'efficacité des ressources, c'est-à-dire la fragmentation interne.

3.3.2.3 Niveau 3 : Allocation des tâches sur les RZs et les RZs sur les $RPBs$

Le niveau 3 se compose de deux sous-niveaux indépendants. Le premier sous-niveau assure l'allocation des RZs sur le plus appropriée des $RPBs$ (sans chevauchement) en terme d'efficacité de ressources. Le second sous-niveau effectue l'allocation des tâches sur les RZs en fonction de leurs points de préemption en respectant la charge de travail de chaque RZ et afin de garantir les contraintes temps réel de chaque tâche. Cette allocation incite essentiellement à diminuer le temps de configuration et le coût D. Ainsi, la solution finale permet l'ordonnancement d'un ensemble de tâches matérielles sur chaque RPB en utilisant l'algorithme EDF, dans sa version non-préemptive décrite et démontrée dans [74, 76].

3.3.3 Principe de résolution à partir de la méthode complète

L'objectif est de résoudre le niveau 2 et 3 à l'aide d'une méthode complète *Branch and Bound*. La méthode utilise la formulation mathématique du problème du placement présenté dans [164]. Nous séparons le problème en deux sous-problèmes :

1. Le partitionnement des $RPBs$ sur le circuit cible (niveau 2) et l'allocation des RZs sur le plus approprié des $RPBs$ (premier sous-niveau du niveau 3).
2. l'allocation des tâches aux RZs (second sous-niveau du niveau 3).

Nous avons utilisé l'environnement AIMMS [85] s'appuyant sur des solveurs puissants afin de résoudre nos deux sous-problèmes à l'aide de la méthode *Branch and Bound*. Pour le

premier sous-problème, nous avons utilisé une méthode de programmation linéaire en nombres entiers fixes alors que le second sous-problème s'est résolu en non-linéaire.

3.3.4 Principe de résolution à partir de la méthode méta-heuristique

La seconde méthode de résolution est basée sur la méthode itérative appelée l'algorithme BEE [81] ou Abeille. L'algorithme BEE imite le comportement d'approvisionnement en nourriture des essaims d'abeilles et peut être considéré comme un outil d'optimisation intéressant. La recherche du voisinage combinée à la recherche aléatoire effectuée par l'algorithme BEE ne permet pas de garantir la solution optimale globale au problème du placement, mais on peut généralement trouver une solution acceptable. D'ailleurs, cette méta-heuristique trouve rapidement une solution convenable contrairement à la méthode complète.

À l'inverse de la résolution complète, les deux sous-problèmes sont résolus simultanément. L'algorithme 1 décrit l'algorithme BEE appliqué au problème du placement. Au début, nous créons plusieurs combinaisons de tous les RZs dont l'allocation des tâches est attribuée de façon aléatoire sur les RZs. Cette allocation aléatoire produit une matrice d'occupation des RZs (NZ colonnes) en fonction des tâches (NT tâches). Cette matrice permet de créer des abeilles dit « scouts » au sein des populations pour explorer par la suite des solutions (les RZs allouées sur les $RPBs$) afin d'obtenir les meilleures allocations possibles. Tant que l'objectif n'est pas atteint (ligne 3), un ensemble de sites de $RPBs$ est sélectionné (ligne 4), puis un ensemble de RZs compatible en terme de ressources selon les $RPBs$ sélectionnés (ligne 5), et qui permet d'obtenir un coût (fitness). Chaque site sélectionne le meilleur coût (ligne 6), c'est-à-dire celui qui aura la meilleure solution. Le reste des abeilles sera utilisé pour la prochaine itération si l'objectif n'est pas atteint (ligne 7 vers 3). L'algorithme BEE est répété *niter* fois pour chaque matrice d'occupation créée. C'est ensuite que nous sélectionnerons la solution ayant la meilleure allocation.

Algorithm 1 Algorithme pour le placement des tâches sur les $RPBs$.

1: **for** $i = 0$ to *niter* **do**
2: // ===== Début de l'algorithme BEE =====
3: Initialise la population de manière aléatoire (Tâches allouées sur des RZs)
4: Évaluation de l'aptitude physique de la population (calcul de la fonction objectif)
5: **while** le critère n'est pas atteint **do**
6: Sélection de sites pour la recherche du voisinage (Sélection des meilleurs $RPBs$ sur le circuit)
7: Recrutement des abeilles pour des sites sélectionnés (Allocation des RZs sur les $RPBs$)
8: Sélection des meilleures abeilles (aptitude physique) pour chaque site (Sélection des meilleurs coûts RZs sur les $RPBs$)
9: Assigne les abeilles restantes pour une recherche suivant leur aptitude physique
10: **end while**
11: // ===== Fin de l'algorithme BEE =====
12: **end for**
13: Retourne la meilleure solution en fonction du coût (RZs,$RPBs$)

Tableau 3.1 – *Caractéristiques des tâches matérielles de l'application.*

Tâches	**Instances**	**modèle de** RBs	**WCET** (μs)	**Période** (μs)	**Temps de configuration** (μs)	**Points de préemption** (μs)	RZs
MDCT	$\{T_1, T_2\}$	{2 RB_1, 12 RB_2, 3 RB_3, 0 RB_4}	40552	416666	1856	10000,20000, 30000	RZ_1
AES	$\{T_3\}$	{4 RB_1, 7 RB_2, 1 RB_3, 1 RB_4}	44540	200000	2185	30000,40000	RZ_2
DDS	$\{T_4, T_5\}$	{0 RB_1, 1 RB_2, 1 RB_3, 1 RB_4}	5000	12000	432	1000,2000, 4000	RZ_3
T48	$\{T_6\}$	{5 RB_1, 4 RB_2, 0 RB_3, 0 RB_4}	20000	50000	605	5000, 10000, 15000	RZ_4
JPEG	$\{T_7\}$	{ 8 RB_1, 12 RB_2, 0 RB_3, 2 RB_4}	350000	416666	2421	200000 300000	RZ_5

3.3.5 Comparaison de la méthode complète et méta-heuristique BEE

3.3.5.1 Description de l'application

L'application proposée dans le tableau 3.1 est composée d'un ensemble de tâches matérielles (fournies par OpenCores [14]) de ressources suffisamment hétérogènes et de caractéristiques temps réel variées afin de valider et comparer nos deux approches. L'application comprend un microcontrôleur (T48) qui contrôle l'ensemble des autres tâches de l'application et assure la synchronisation des flux de données. La tâche MDCT calcule la transformée en cosinus discrète modifiée. La tâche JPEG effectue la compression matérielle à 24 images par seconde. L'AES crypte les données résultant de la tâche JPEG par blocs de 128 bits et avec une clé de 256 bits. Le DDS (Synthétiseur Numérique Direct) crée des ondes sinusoïdales programmables. Dans ce tableau, les caractéristiques temporelles sont réalistes à l'exception des points de préemption qui ont été déterminés arbitrairement en fonction de leurs temps d'exécution.

Pour expérimenter notre approche avec l'algorithme BEE, nous avons itéré 20 fois ($niter$). L'algorithme BEE a été programmé en langage C++. Concernant l'approche *Branch and Bound*, nous nous sommes appuyés sur les solveurs fournis par l'environnement AIMMS.

3.3.5.2 Résultats du niveau 1

Cette étape est commune aux deux approches. La première étape consiste à rechercher les modèles de RBs. Pour obtenir ces modèles pour la technologie Virtex5, nous avons synthétisé les différentes tâches par le biais de l'outil ISE 11.3 de la société Xilinx afin d'extraire les ressources utilisées (RB_1=CLBL, RB_2=CLBM, RB_3=BRAM et RB_4=DSP48) comme l'illustre le tableau 3.1. La seconde étape (tableau 3.2) décrit les types de RZs obtenus ainsi que leur charge de travail et les coûts D entre les tâches et les RZs, ∞ signifiant que le placement de la tâche est incompatible en terme de ressources.

Pour finir, l'étape 3 résout les problèmes de surcharge détectés comme RZ_3. Cette surcharge sur RZ_3 est résolue en migrant les deux premières sections d'exécution de la tâche T_4 (DDS) ou T_5 (DDS) sur RZ_2. La migration de tâches a été utilisée à l'instar d'un ajout d'une RZ plus coûteux.

14. `http://opencores.org`

Tableau 3.2 – *Résultats des étapes 1 et 2 du niveau 1.*

	MDCT $\{T_1, T_2\}$	**AES** $\{T_3\}$	**DDS** $\{T_4, T_5\}$	**T48** $\{T_6\}$	**JPEG** $\{T_7\}$
RZ_1 **(26 %)**	**0**	∞	∞	∞	∞
RZ_2 **(25 %)**	∞	**0**	560	∞	∞
RZ_3 **(116 %)**	∞	∞	**0**	∞	∞
RZ_4 **(46 %)**	∞	∞	∞	**0**	∞
RZ_5 **(86 %)**	∞	∞	∞	1380	**0**

Tableau 3.3 – *Efficacité de ressources pour les deux approches.*

Approche *Branch and Bound*		
	modèle de *RBs*	**Δ**
RPB_1	6 RB_1, 12 RB_2, 6 RB_3, 0 RB_4	4 RB_1, 3 RB_3
RPB_2	**12** RB_1, **8** RB_2, **2** RB_3, **2** RB_4	**8** RB_1, **1** RB_2, **1** RB_3, **1** RB_4
RPB_3	**2** RB_1, **1** RB_2, **1** RB_3, **1** RB_4	**2** RB_1
RPB_4	**8** RB_1, **4** RB_2, **0** RB_3, **0** RB_4	**3** RB_1
RPB_5	**18** RB_1, **12** RB_2, **3** RB_3, **3** RB_4	**10** RB_1, **3** RB_3, **1** RB_4
Approche *Bees*		
	modèle de *RBs*	**Δ**
RPB_1	6 RB_1, 12 RB_2, 6 RB_3, 0 RB_4	4 RB_1, 3 RB_3
RPB_2	**14** RB_1, **8** RB_2, **2** RB_3, **2** RB_4	**10** RB_1, **1** RB_2, **1** RB_3, **1** RB_4
RPB_3	**8** RB_1, **4** RB_2, **4** RB_3, **4** RB_4	**8** RB_1, **3** RB_2, **3** RB_3, **3** RB_4
RPB_4	**9** RB_1, **4** RB_2, **1** RB_3, **2** RB_4	**4** RB_1, **1** RB_3, **2** RB_4
RPB_5	**27** RB_1, **12** RB_2, **3** RB_3, **6** RB_4	**19** RB_1, **3** RB_3, **4** RB_4

3.3.5.3 Résultats du niveau 2 et 3

Le tableau 3.3 montre les RPB_j sélectionnés pour chaque RZ_j et les coûts Δ exprimés par les différences des RB_k des RZs et des RB_k des $RPBs$ obtenues après résolution avec les deux méthodes. Les caractères gras représentent les différences entre les deux approches. Nous observons que la méthode complète est meilleure que l'algorithme BEE en se basant sur l'efficacité des ressources (paramètre Δ). Plus précisément, la méthode complète utilise 33% de RBs disponibles sur le Virtex5-SX50 tandis que l'algorithme BEE en utilise 49%. La figure 3.6 montre le floorplanning finale des $RPBs$ sur le Virtex 5-SX50 selon les coordonnées des $RPBs$ obtenues respectivement avec l'algorithme BEE et l'approche *Branch and Bound*.

Par ailleurs, les approches fournissent l'allocations des tâches sur les $RPBs$ (tableau 3.4). En termes de réduction de surcharge du temps de configuration, seulement 20% de l'exécution de T_4 et T_5 est sur RZ_2 (devenu RPB_2). L'approche *Branch and Bound* ne produit que 50380μs de temps de reconfiguration, contre 59091μs produites par l'approche BEE, qui alloue 60% de l'exécution de T_4 sur RZ_2. Le gain en temps de configuration est donc 15% plus favorable avec une approche complète.

3.4 Placement/Ordonnancement hors-ligne de tâches matérielles dépendantes par méthode dynamique

3.4.1 Introduction

Une tendance importante dans les applications temps réel mis en œuvre dans les systèmes de calcul reconfigurables consiste à utiliser des circuits matériels reconfigurables pour

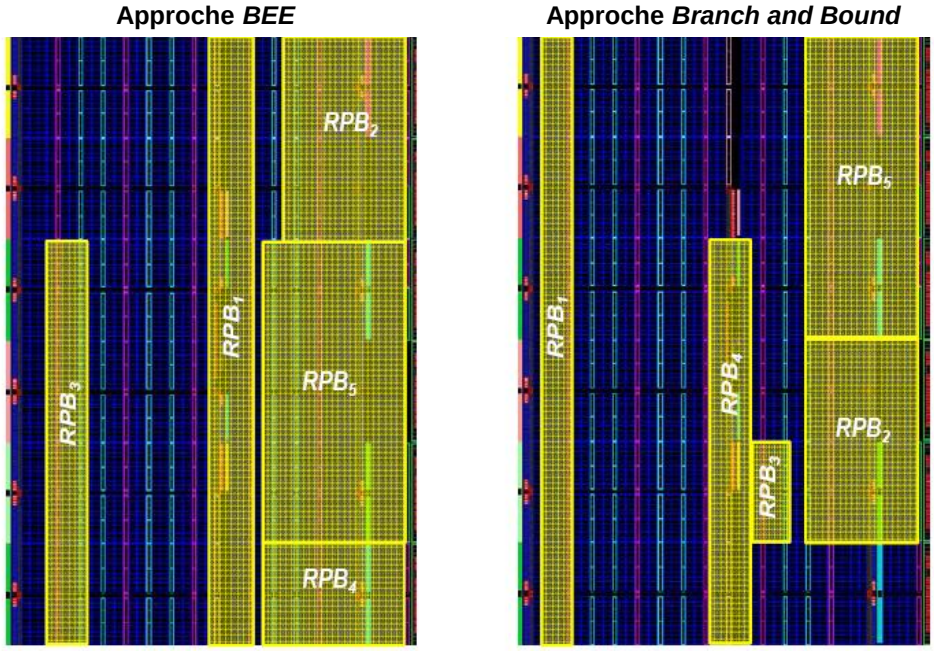


Figure 3.6 – *Floorplanning finale des RPBs pour les deux approches.*

Tableau 3.4 – *Allocations des tâches sur les RPBs.*

	Approche *BEE*	**Approche** ***Branch and Bound***
$\boldsymbol{RZ_1}$ $\boldsymbol{(RPB_1)}$	T_1 (100%) T_2 (100%)	T_1 (100%) T_2 (100%)
$\boldsymbol{RZ_2}$ $\boldsymbol{(RPB_2)}$	T_3 (100%) $T_{4,1}$,$T_{4,2}$,$T_{4,4}$ (60% of T_4)	T_3 (100%) $T_{4,2}$ (20% of T_4) $T_{5,2}$ (20% of T_5)
$\boldsymbol{RZ_3}$ $\boldsymbol{(RPB_3)}$	$T_{4,3}$ (40% of T_4) T_5 (100%)	$T_{4,1}$,$T_{4,3}$,$T_{4,4}$ (80% of T_4) $T_{5,1}$,$T_{5,3}$,$T_{5,4}$ (80% of T_5)
$\boldsymbol{RZ_4}$ $\boldsymbol{(RPB_4)}$	T_6 (100%)	T_6 (100%)
$\boldsymbol{RZ_5}$ $\boldsymbol{(RPB_5)}$	T_7 (100%)	T_7 (100%)

accroître les performances et afin de garantir les contraintes temporelles. En outre, les activités partiellement ordonnées et périodiques représentent une part importante de la demande dans les systèmes temps réel tels que le contrôle en temps réel et le traitement numérique du signal. Cette catégorie de calcul répétitif est généralement décrit par des graphes acycliques orientés DAG [15]. Par conséquent, tous ces facteurs (calcul parallèle, temps de reconfiguration, densité élevée des ressources) encouragent l'emploi de ces circuits reconfigurables.

Des techniques d'ordonnancement statiques multiprocesseurs utilisant des graphes de tâches ont évolué au cours de ces dernières années et de nombreuses stratégies d'ordonnancement ont émergé. Comme ce problème est connu pour être NP-difficile [72], les principaux efforts de recherche dans ce domaine portent sur des méthodes heuristiques et peu d'entre elles proposent des résolutions analytiques. Nous avons inventorié et classifié des ordonnancements statiques et dynamiques multiprocesseur utilisant des DAGs dans un contexte logiciel

15. Directed Acyclic Graph

et matériel. Le tableau 3.5 présente un résumé des paramètres d'optimisation et les techniques employées décrites dans les ouvrages cités d'ordonnancement multiprocesseurs de DAGs.

Tableau 3.5 – *Classification des travaux en fonction des paramètres d'optimisation et des techniques d'ordonnancement de DAGs.*

Références	**Makespan/ Speedup/ Efficacité parallèle**	**Efficacité de ressources**	**Overhead de configuration**	**Techniques**
[71]	X		X	Prefetch/replace/reuse of reconfigurations
[83]	X			Graph unfolding/*genetic algorithm*/ *earliest-start-time* heuristic
[70]	X			Timed automaton model/shortest path
[82]	X		X	1) ILP/reuse 2) $NAPOLEAN/ALAP$/prefetch/ reuse/anti-fragmentation
[87]	X			ILP/loop level task partitioning/task transformation (loop fusion+loop splitting)/task mapping and scheduling (task merging on batches+batch replication)/data mapping
[84]	X			1) Graph decyclification (DFS+shortest critical path)/*CP-MISF* 2) Graph Unfolding/ *genetic algorithm*
[72]	X			*Chaining/task selection first* heuristic/critical path/edge width/communication latencies
[75]	X	X	X	Dynamic programming algorithm/FPGA total configuration
[73]	X	X		Orthogonal packing problem/packing classes
[79]	X		X	Prefetch/local optimization/reuse
[77]	X			Critical path heuristic/ *Branch and Bound*

Comme indiqué dans le tableau 3.5, les travaux se concentrent sur l'optimisation du makespan du graphe et néglige l'overhead de reconfiguration et l'efficacité des ressources, ou alors n'optimisent pas ces trois paramètres simultanément. Les travaux décrits dans [75] et [73] nous conduisent à de l'ordonnancement de DAGs sur FPGA, basé sur de la configuration totale successive du circuit programmable. Leur efficacité de ressources consiste uniquement à grouper le maximum de tâches dans le DAG sur le FPGA afin d'exploiter efficacement les ressources reconfigurables ainsi que d'effectuer le minimum de configuration totale. Ces travaux ne prennent pas en considération la fragmentation interne provoquée par le placement des tâches sur le FPGA qui représente un élément primaire pour l'efficacité des ressources dans nos travaux.

Dans le contexte de l'ordonnancement des tâches matérielles, le placement des tâches ordonnancées n'est pas considéré dans les travaux proposés, ou seulement étudié dans le cas de circuits reconfigurables homogènes. Contrairement à ces travaux, notre stratégie est de cibler les circuits à ressources hétérogènes et ainsi de pouvoir traiter également les circuits à ressources homogènes. Le problème du placement est considéré comme une étape importante qui est très étroitement liée à l'ordonnancement de graphes de tâches sur le dispositif

reconfigurable. Avec notre stratégie, la tâche peut être exécutée sur plusieurs unités reconfigurables en fonction de ses ressources et selon les analyses effectuées pendant la phase de regroupement. Par ailleurs, certains de ces travaux décrits n'exploitent pas le concept de reconfiguration partielle offert par les circuits FPGAs actuels.

Basé sur ces observations, nous proposons une méthodologie qui utilise les avantages de la reconfiguration partielle sur des circuits hétérogènes. Cette méthodologie rend possible le développement de systèmes matériels multi-tâches en partageant la surface reconfigurable en zones reconfigurables ($RPBs$).

3.4.2 Principes de la méthodologie proposée

L'ordonnancement de ces graphes de tâches sur des architectures reconfigurables peut être défini comme la recherche d'un scénario pour un ensemble de tâches périodiques avec des contraintes de précédences, de dépendances et d'échéances, sans oublier leurs placements sur des ressources matérielles hétérogènes. L'objectif est d'obtenir une solution qui minimise le temps d'exécution (makespan), le temps de réponse, la fragmentation interne et le temps de configuration.

Nous rappelons que le terme « précédence » désigne un ordre partiel entre les tâches sans aucun transfert de données tandis que le terme « indépendant » signifie qu'il n'y a pas d'ordre d'exécutions des tâches. Par ailleurs, deux tâches sont considérées comme « dépendantes » si une tâche exige le résultat d'exécution d'une autre tâche pour permettre son exécution, ce qui impose évidemment une relation de précédence entre ces deux tâches. La tâche productrice de données est appelée « prédécesseur » et la tâche consommatrice de données est nommée « successeur ».

Ainsi, la mise en œuvre de ces DAGs sur ces circuits reconfigurables consiste à ordonnancer des tâches dans un nombre limité de zones reconfigurables hétérogènes et de taille différentes tout en respectant les quatre contraintes décrites comme suit :

- **La contrainte de périodicité** : Chaque tâche est répétée périodiquement en fonction de ses dates de réveil dans le graphe. Ainsi, si une tâche A a une période P_A alors : $\forall\ i \in I\!N, (r_{A_{i+1}} - r_{A_i}) = P_A$, où A_i et A_{i+1} sont la i^{ieme} et la $(i+1)^{ieme}$ répétitions de la tâche A, et r_{A_i} et $r_{A_{i+1}}$ sont leurs temps de départ.
- **La contrainte de précédence** : Afin de maintenir l'exactitude de la précédence des tâches, à chaque itération, une tâche peut être exécutée que si tous ses prédécesseurs dans le graphe ont fini leurs exécutions. Par conséquent, chaque tâche A peut lancer l'exécution après l'achèvement des exécutions de ses prédécesseurs définit par le sous-ensemble π_A. Par conséquent, $\forall\ i \in I\!N, s_{A_i} \geq s_{B_i} + C_B, \forall\ B \in \pi_A$, où s_{A_i} et s_{B_i} sont respectivement (au cours de leur i^{ieme} itération) les temps de départ de la tâche A et B, et C_B est le temps d'exécution de la tâche B.
- **La contrainte de dépendance** : L'exécution de chaque tâche dans le DAG est lancée lorsque toutes les données résultantes de tous ses prédécesseurs sont disponibles.
- **La contrainte d'échéance** : Chaque tâche dans le DAG doit terminer son exécution avant son échéance. Ainsi, dans l'itération i, si la tâche A a un temps d'exécution de C_A et une échéance absolue d_{A_i}, alors $s_{A_i} + C_A \leq d_{A_i}$.

Pour respecter ces contraintes, nous devons vérifier la validité des périodes des tâches et leurs temps d'exécution dans le DAG comme détaillé dans [163].

Exemple

La figure 3.7 illustre un exemple de graphe de tâches. Les tâches sont répétées en fonction de leurs périodes. Chaque tâche ayant une contrainte de précédence lance son exécution que lorsque ses prédécesseurs atteignent leurs exécutions, et seulement lorsque cela est nécessaire. Par exemple, la troisième itération de T_2 et T_3 de période 8 n'ont pas besoin d'une troisième exécution de leurs prédécesseurs T_1 car elle est moins répétitive que T_2 et T_3 (la période de T_1 est égale à 12). A chaque répétition, pour permettre l'exécution de tâches, les lignes pointillées désignent le transfert des données entre les tâches inter-dépendantes. Ainsi, lors de ses deux itérations, T_1 doit fournir toutes les données nécessaires pour le traitement des données par ses successeurs T_2 et T_3 au cours de leurs trois itérations. Le problème de la dépendance de données est également détaillé dans [163]. Enfin, à chaque itération, les tâches doivent respecter leurs échéances absolues.

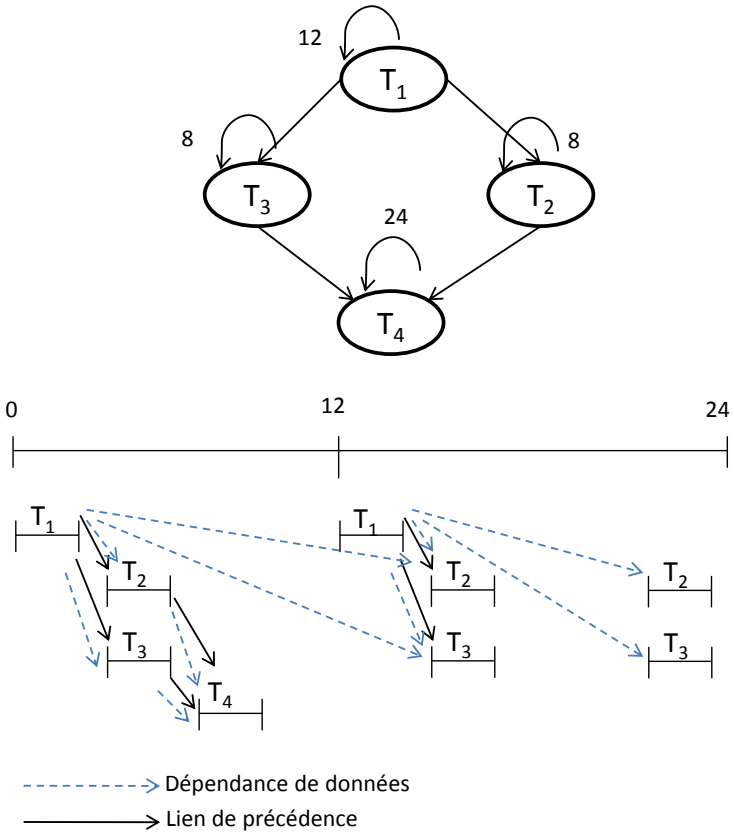


Figure 3.7 – *Exemple d'un graphe acyclique de tâches.*

Comme le montre la figure 3.8, nous proposons une nouvelle méthodologie comprenant trois étapes principales afin de réaliser le placement et l'ordonnancement de ces DAGs avec les contraintes prédéfinies ci-avant sur l'architecture reconfigurable cible :

• **Étape 1, Regroupement des tâches :** Cette étape est tributaire de la technologie. Elle

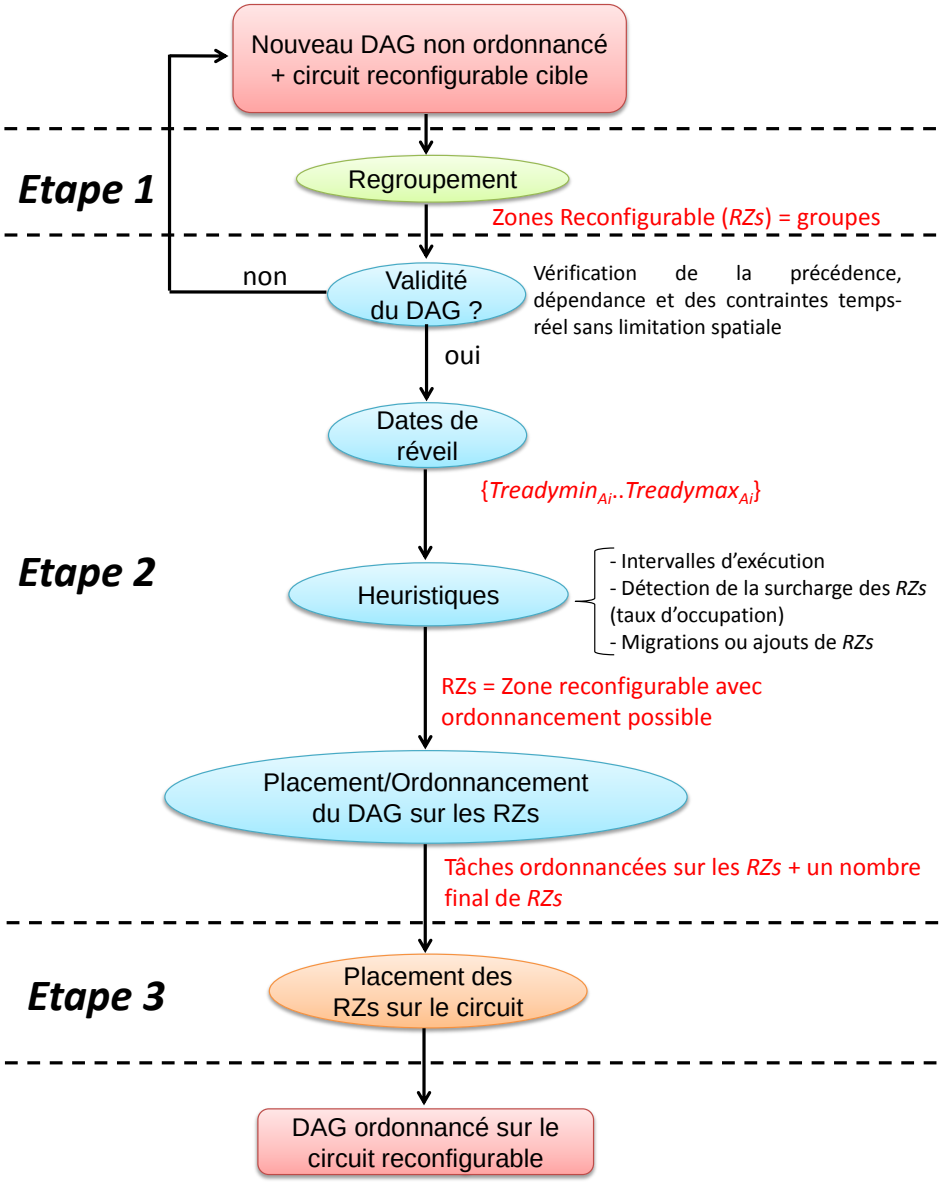


Figure 3.8 – *La méthodologie proposée.*

définit le partitionnement des tâches nécessitant les mêmes types de ressources dans le même groupe. Cette étape est équivalente aux deux premières étapes du niveau 1 dans « l'approche du Placement/Ordonnancement hors-ligne de tâches matérielles indépendantes par des méthodes dynamiques et méta-heuristiques », page 65.

- **Étape 2, Allocation/Ordonnancement des tâches sur les regroupements :** Cette étape commence par réaliser une analyse spatiale et temporelle mentionnées dans la figure 3.8 par la validité de DAG, puis recherche les dates de réveils. L'exécution d'un ensemble d'heuristiques est réalisée afin de calculer des intervalles d'exécution possibles des tâches. De plus, il est indispensable de vérifier la surcharge des RZs et ainsi de faire migrer ou ajouter des RZ si nécessaire comme dans l'approche du « Placement/Ordonnancement hors-ligne de tâches matérielles indépendantes par des méthodes dynamiques et méta-heuristiques », page 65. Nous obtenons un nombre de RZs nécessaires pour

concevoir un ordonnancement réalisable en respectant les contraintes prédéfinies. Par la suite, cette étape, basée sur un modèle de préemptions prédéfinies, traite simultanément l'allocation des tâches sur les groupes de RZs et l'ordonnancement global des tâches sur ces groupes en respectant la périodicité, la priorité, la dépendance et les contraintes de délais. Cette étape optimise la qualité d'ordonnancement.

- **Étape 3, Placement des groupes de RZs sur le circuit reconfigurable :** Cette troisième étape est également dépendante de la technologie. Elle consiste à rechercher l'endroit (RPB) le plus approprié sur le circuit reconfigurable pour chaque groupe obtenu à la seconde étape. Cette étape optimise la qualité de placement.

Cette méthodologie en trois étapes a pour résultat un ordonnancement statique des tâches des DAGs dans un nombre restreint de $RPBs$ sur le circuit reconfigurable, en respectant la périodicité, la précédence, la dépendance et les échéances des tâches. C'est un problème fondamental pour le calcul parallèle, et qui équivaut à la détermination d'ordonnancement statique multiprocesseur pour des DAGs dans un contexte logiciel. Comme c'est bien connu, l'ordonnancement statique multiprocesseur de graphes de tâches est un problème d'optimisation combinatoire. Nous l'avons formulé par de la programmation entière mixte et résolu par des solveurs dédiés à ce type de programmation.

3.4.3 Résultats

Pour illustrer la méthodologie proposée, nous avons conçu un DAG entièrement connecté (figure 3.9). Les tâches, comme dans l'approche précédente, ont été sélectionnées à partir du site OpenCores.

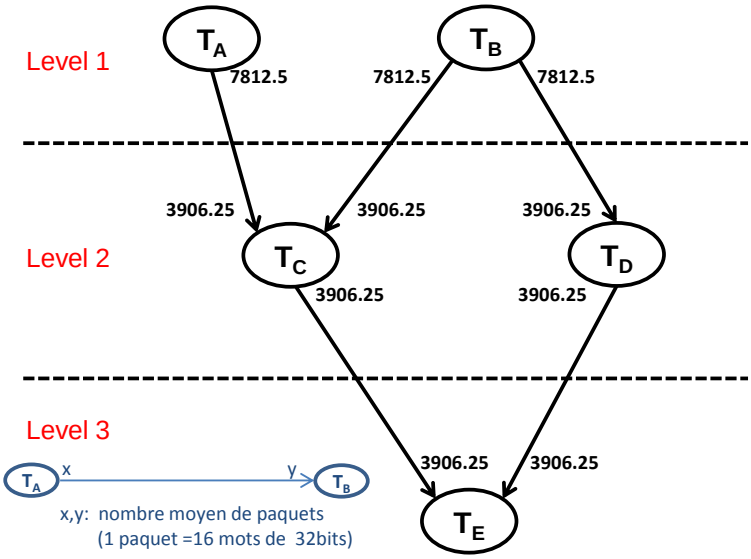


Figure 3.9 – *Application proposée.*

Comme présenté dans le tableau 3.6, chaque tâche est caractérisée par son temps d'exécution pire-cas (WCET) qui intègre également la latence de communication pour l'envoi de données à ses successeurs, sa période (correspondant à son échéance) et l'ensemble des points de préemption prédéfinis. Afin d'évaluer la taille des tâches en terme de ressources, nous avons

Tableau 3.6 – *Caractéristiques des tâches matérielles.*

Tâche	**Nom**	**Slices (LUTs/Regs)**	**WCET (μs)**	**Période (ms)**	**Temps de configuration(μs)**	**Points de préemption (μs)**	**modèle de** RBs
T_A	Reed-Solomon	2234/1224	26576	500	1116	14770,20200	{8 RB_1, 7 RB_2, 1 RB_3, 1 RB_4}
T_B	AES	1150/507	42733	500	675	5000, 11000, 23000,30000	{5 RB_1,5 RB_2, 1 RB_3,1 RB_4}
T_C	IIR	678/565	11805	250	524	3334, 10000	{4 RB_1,1 RB_2, 0 RB_3,1 RB_4}
T_D	FFT	2333/2010	11806	250	1199	5000, 11667	{9 RB_1,7 RB_2, 0 RB_3,1 RB_4}
T_E	Basic DES	387/192	22280	250	188	7500, 15000, 19000	{2 RB_1,2 RB_2, 0 RB_3,0 RB_4}

Tableau 3.7 – *RZ Types and D costs.*

	T_A	T_B	T_C	T_D	T_E
RZ₁ {8 RB_1, 7 RB_2, 1 RB_3, 1 RB_4}	**0**	**68**	292	∞	448
RZ₂ {9 RB_1,7 RB_2, 0 RB_3, 1 RB_4}	∞	∞	**140**	**0**	356
RZ₃ {2 RB_1, 2 RB_2, 0 RB_3, 0 RB_4}	∞	∞	∞	∞	**0**

précisé le nombre de slices. Un slice contient un ensemble de LUTs et de registres dépendant de la technologie cible.

3.4.3.1 Regroupement des tâches

Cette étape est identique à l'approche précédente (page 65). Les résultats sont présentés dans le tableau 3.7. Nous rappelons que l'objectif est de rechercher les types de RZs et de calculer le coût D correspondant à la fragmentation interne entre une RZ et une tâche.

3.4.3.2 Allocation/Ordonnancement des tâches sur les regroupements

Le comportements du DAG est étudié sur l'hyper-période, égale à 500 ms dans notre application. La figure 3.10 illustre l'ordonnancement pour les cinq tâches dans le DAG allouées sur trois RZs, calculé par l'intermédiaire de solveurs de l'environnement AIMMS. Comme le nombre de RZs est le paramètre principal dans la fonction objectif, la résolution a seulement utilisé deux RZs. L'élimination de RZ_3 améliore l'efficacité des ressources globales et permettrait au DAG d'être étendu pour des besoins futurs.

3.4.3.3 Placement des groupes sur le circuit reconfigurable

Les RZ_1 et RZ_2 sont assignées sur leurs $RPBs$ les plus appropriés et définis sur le FPGA Virtex5 FX70T. Le tableau 3.8 illustre la comparaison de ressources (Δ) entre les $RPBs$ et les RZs. Les Δ non nuls sont dus à la forme rectangulaire des $RPBs$ et à l'hétérogénéité du circuit.

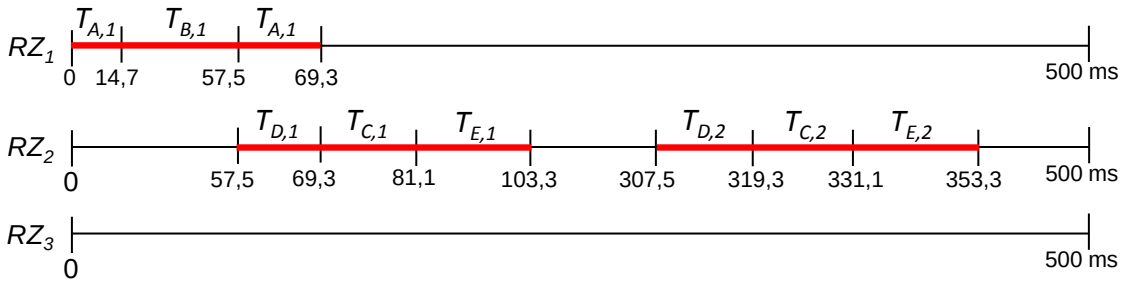


Figure 3.10 – *Résolution du placement/ordonnancement.*

Tableau 3.8 – *Efficacité de ressources.*

	RB_1	RB_2	RB_3	RB_4	Δ
RPB_1	8	7	2	1	1 RB_3
RPB_2	9	7	2	1	2 RB_3

La figure 3.11 représente le placement/routage des tâches T_A (Reed-Solomon) et T_D (FFT) qui s'exécutent respectivement sur RZ_1 et RZ_2 pendant l'intervalle de temps [57,5ms,69,3ms]. Les résultats obtenus montrent une utilisation moyenne de 12,46% des ressources disponibles sur le circuit reconfigurable. Cette moyenne est calculée selon le nombre et le coût de chaque type de RBs. Grâce à la reconfiguration dynamique partielle, l'efficacité des ressources est améliorée de 17,3% par rapport à la solution statique du DAG. La solution statique correspond au placement/routage de toutes les tâches sur le circuit reconfigurable.

3.5 Modélisation haut-niveau pour le placement/ordonnancement de tâches matérielles reconfigurables

3.5.1 Introduction

L'objectif de cette contribution est de développer une méthodologie pour la modélisation de haut niveau des architectures reconfigurables dynamiquement. Nous présentons notre approche, basée sur la bibliothèque SystemC [105]. L'objectif principal est de fournir un moyen aisé de développer des stratégies d'ordonnancement pour la gestion des tâches matérielles dans un système reconfigurable dynamiquement. Nous voulons aussi introduire de l'exploration d'architectures dans notre méthodologie en vue de fournir au futur développeur un flot de conception qui pourrait être intégré, par exemple, dans le flot actuel de reconfiguration partielle de la société Xilinx. Par ailleurs, la méthodologie doit être indépendante de toutes autres bibliothèques ou d'extensions, c'est-à-dire de ne pas modifier le cœur de la bibliothèque SystemC.

Dans l'introduction générale de cet axe de recherche, nous avons discuté du profil MARTE provenant du langage UML mais qui ne supportait pas nativement la modélisation PR. Cependant, à l'aide du flot GASPARD2, les auteurs ont modifié certains concepts de MARTE afin d'ajouter les aspects manquants de PR. Par exemple, ils ont introduit un attribut qui décrit la nature de la région soit statique ou soit reconfigurable dynamiquement. Autant que nous le savons, il est actuellement difficile de développer et valider des stratégies d'ordonnan-

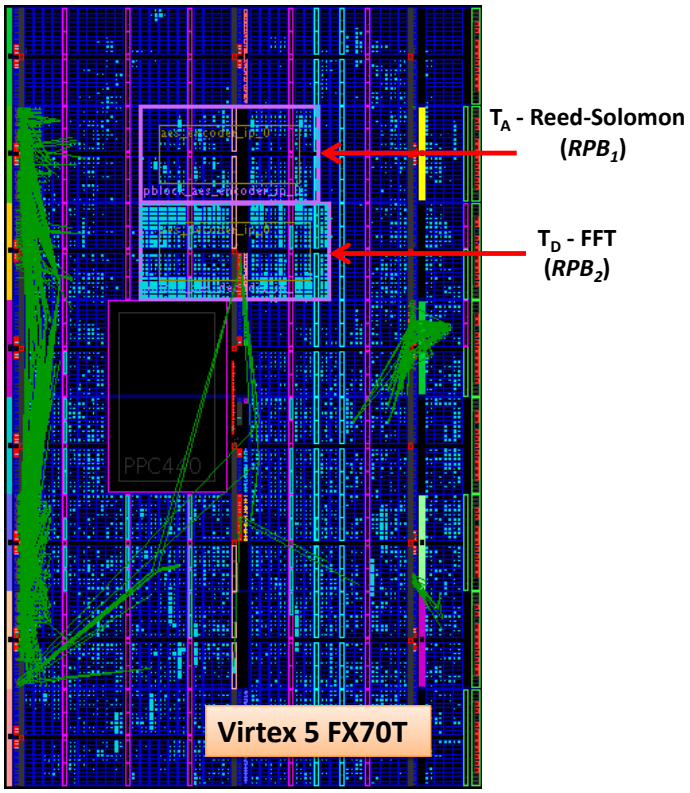


Figure 3.11 – *Floorplanning des RPBs et P&R des tâches T_A et T_D durant l'intervalle d'ordonnancement [57,5ms,69,3ms].*

cement en utilisant ces travaux.

Une autre approche [110] également présentée en introduction est basée sur le projet MoPCoM. La modélisation est effectuée sur trois niveaux, à partir d'un modèle de calcul (MoC [16]) indépendant vers un modèle de plate-forme spécifique. Chaque niveau suit une conception en Y, qui sépare le modèle applicatif du modèle de plate-forme, pour former le modèle architectural ou d'allocation. Le support de la reconfiguration partielle est introduite à ce dernier niveau où plusieurs composants de l'application sont alloués à une seule zone physique reconfigurable. Cette méthodologie nécessite encore aux concepteurs de définir manuellement les composants qui peuvent partager une zone reconfigurable, et ainsi ne permettant pas une exploration du placement (voire même d'ordonnancement) de ces composants.

Comme nous l'avons évoqué dans l'introduction, le langage SystemC est un bon candidat pour modéliser à haut-niveau et réaliser de l'exploration d'architecture. Les auteurs dans [113] présentent leur approche de modèle de PR comme une bibliothèque SystemC, appelé ReChannel. Elle consiste principalement à instancier les différents modules possibles pour chaque zone reconfigurable, mais en activant un seul module à la fois (*Dynamic Switching Circuit*). La connexion entre l'ensemble des modules reconfigurables et le reste du système se fait par une interface qui peut gérer les canaux de base définis dans SystemC. ReChannel définit également une classe de contrôle pour gérer la reconfiguration en utilisant des interfaces dédiées. Cependant, les tâches sont assignées une fois au début de la simulation, ce qui

16. Model of Computation

signifie qu'il n'est pas possible de faire de la migration de tâche d'une zone reconfigurable à l'autre.

Les travaux effectués, basé sur SystemC, dans ADRIATIC [17] a conduit à un circuit reconfigurable dynamique [114]. Le circuit contient plusieurs contextes et est capable de basculer dynamiquement de l'un à l'autre, rappelant l'approche ReChannel. Les auteurs dans [115] ont amélioré les travaux précédents en ajoutant un ordonnanceur de configuration qui tient compte du temps associé à une reconfiguration.

Les auteurs dans [116] utilisent les threads dynamiques introduit dans SystemC 2.1. Par opposition aux threads statiques, les threads dynamiques peuvent être engendrés lors de l'exécution de la simulation et pas seulement durant la phase d'élaboration. Un module reconfigurable est composé de deux threads dynamiques : l'un pour le processus de l'utilisateur et l'autre pour son contrôle (création et destruction). À notre avis, cette méthode est la plus flexible et par conséquent, nous basons notre approche sur les threads dynamiques. Par ailleurs, ils définissent le concept de port dynamique, rendant la construction d'un port possible après la phase d'élaboration. Ceci ouvre une porte pour l'interconnexion de modules reconfigurables avec des interfaces différentes.

3.5.2 Notre approche

3.5.2.1 Vue d'ensemble du flot de conception

Nous basons notre approche sur un modèle de conception en Y qui fusionne notre modèle applicatif et notre modèle de plateforme en un modèle architectural qui fournit au concepteur une implémentation en conformité avec une stratégie d'ordonnancement. Cette approche repose sur un simulateur décrit en SystemC. Notre simulateur peut être utilisé à deux stades différents de conception.

Tout d'abord, il peut être utilisé indépendamment durant les premiers stades de développement, afin d'étudier la faisabilité et les avantages potentiels tirant partis d'une conception intégrant le reconfigurable dynamique et partiel. Dans ce cas, les entrées du simulateur doivent être estimées afin qu'il donne une bonne évaluation des performances du système reconfigurable finale.

Ensuite, il peut également être utilisé pendant la phase d'implémentation, complètement intégré dans un flot de conception existants, tel que décrit dans la figure 3.12. Généralement, une fois les netlists générées pendant la phase de synthèse, le développeur doit définir manuellement les zones reconfigurables disponibles sur le FPGA. Nous proposons de remplacer cette phase par notre simulateur en inférant de l'information pertinente à partir des fichiers générés lors de la synthèse. Par exemple, la netlist générée pour chaque tâche dans l'application peut être utilisée pour récupérer des informations sur les besoins en ressources qui alimentent le modèle applicatif. Ce modèle utilise également un graphe flot de données représentant les liens entre les tâches et les informations de synchronisation. Cette information représente, d'une part l'échéance des tâches déterminée par les spécifications de l'application, et d'autre part les temps de reconfiguration. Ces temps peuvent être estimés en utilisant les travaux

17. Advanced Methodology for Designing Reconfigurable SoC and Application Targeted IP-entities in wireless Communications

que nous menons sur un gestionnaire de reconfiguration efficace couplé avec un modèle précis de coût [162, 168] que nous présentons dans le chapitre 4, page 94. Le modèle d'architecture n'a besoin que du circuit cible pour inférer potentiellement des zones reconfigurables. Les résultats du simulateur sont l'algorithme d'ordonnancement et un ensemble de $RPBs$. Ces $RPBs$ répondent aux exigences de l'application et seront utilisées pour alimenter les dernières étapes du flot, c'est-à-dire l'implémentation du matériel et la génération des bitstreams.

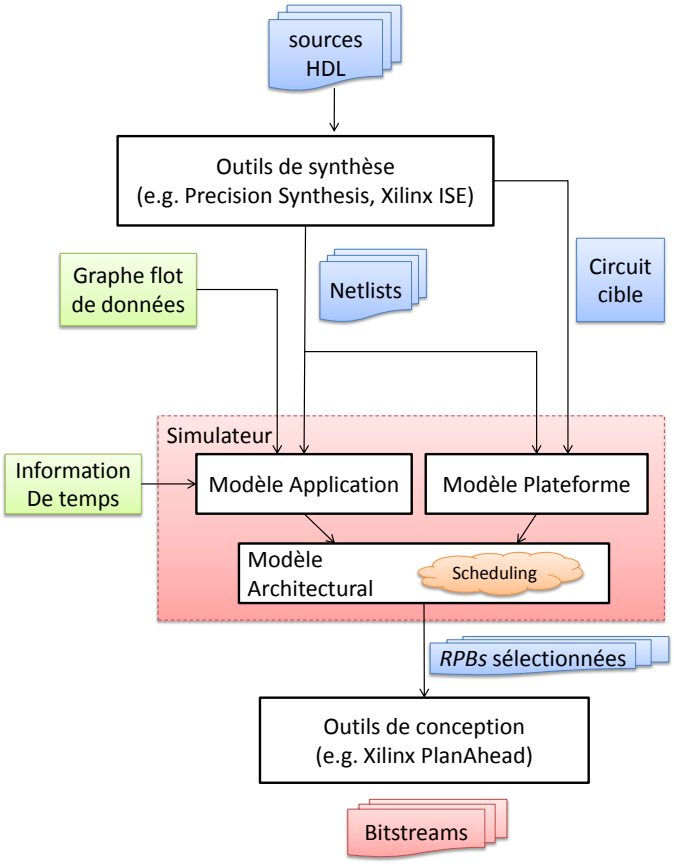


Figure 3.12 – *Flot amélioré de reconfiguration partielle.*

La reconfiguration partielle montre deux aspects majeurs que nous voulons prendre en compte dans notre méthodologie de modélisation. Le premier est le changement de fonctionnalité où la reconfiguration partielle est utilisée pour permuter entre les différentes implémentations d'une seule tâche, et le second est le partage des ressources où la reconfiguration partielle est utilisée pour mutualiser un ensemble de ressources entre plusieurs tâches. Faire face à ces aspects de manière efficace peut exiger une certaine forme de contrôle au niveau des FPGAs. Ainsi, nous introduisons un gestionnaire de reconfiguration, divisé en deux niveaux : un contrôleur de bas niveau dans le FPGA et une couche de niveau supérieur qui est plus intelligente et qui contrôle les tâches matérielles de l'intérieur ou de l'extérieur du FPGA. La figure 3.13 décrit notre approche basée en Y, en séparant l'application, la plateforme cible et le modèle architectural.

L'application est d'abord décrite de façon statique (modèle de l'application). Dans cet exemple, l'application est composée de quatre tâches activées séquentiellement avec deux implémentations possibles pour la deuxième tâche (B_1 et B_2). Le modèle applicatif a besoin

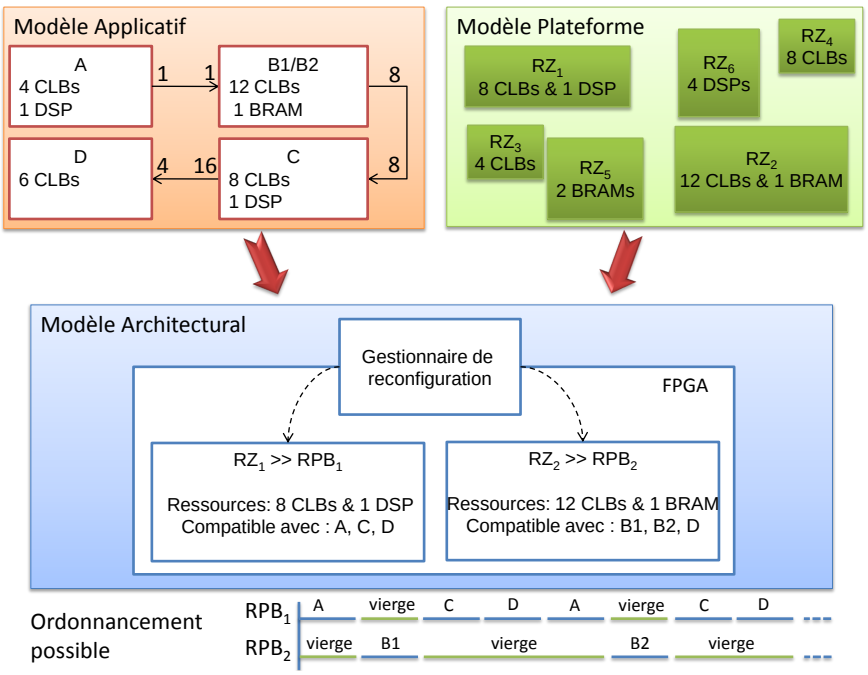


Figure 3.13 – *Approche proposée.*

de connaître la quantité de données transférées entre les tâches. Dans l'exemple, la tâche C reçoit des blocs de données de huit octets alors que la sortie génère des blocs de seize octets. Les tâches sont également modélisées par leurs besoins en ressources (colonnes de CLBs, BRAM et DSP48). Ensuite, la plate-forme est décrite (modèle de la plateforme). Puisque notre méthodologie se focalise uniquement sur la partie reconfigurable du système qui devrait être mis en œuvre dans un FPGA, le concepteur définit des zones reconfigurables disponibles (RZs) en termes de ressources. Puis, lors de la simulation, le gestionnaire alloue les tâches sur la plate-forme décrite (modèle architectural). Les RZs deviennent alors des $RPBs$. Dans cet exemple, le premier RPB peut accueillir les tâches A, C et D tandis que la seconde peut accueillir B1, B2 et D. Un schéma d'ordonnancement possible est également présenté dans la figure 3.13. Au départ, l'ordonnancement introduit des configurations vierges (blank bitstreams) pour chaque RPB, permettant de réduire la consommation d'énergie. Ici, le second RPB est conservé inactif autant que possible car c'est la zone les plus importante et, par conséquent, la plus consommatrice d'énergie.

Les détails de niveau inférieur du gestionnaire de reconfiguration sont directement inclus dans la tâche tels que le temps de reconfiguration ou le temps de changement de contexte. Le gestionnaire, présent dans le modèle, est en fait la couche de niveau supérieur en charge de l'ordonnancement.

Nous basons notre modèle sur trois principaux concepts. Tout d'abord, une tâche fournit des interfaces pour communiquer avec d'autres tâches et une interface dédiée à la communication avec le gestionnaire de reconfiguration. Ce gestionnaire est en charge de la reconfiguration de chaque RPB. Enfin, toutes les communications dans ce modèle sont décrites en utilisant le standard TLM-2.0 permettant d'élever le niveau d'abstraction et d'accélérer le temps de simulation.

3.5.2.2 Le module reconfigurable

Notre module typique est basé sur les threads dynamiques de SystemC, utilisé pour commuter entre les tâches et ainsi modifier les fonctionnalités d'un module en cours d'exécution. Cette méthode est de loin la plus souple et la plus complète. La figure 3.14 montre l'architecture du module complet.

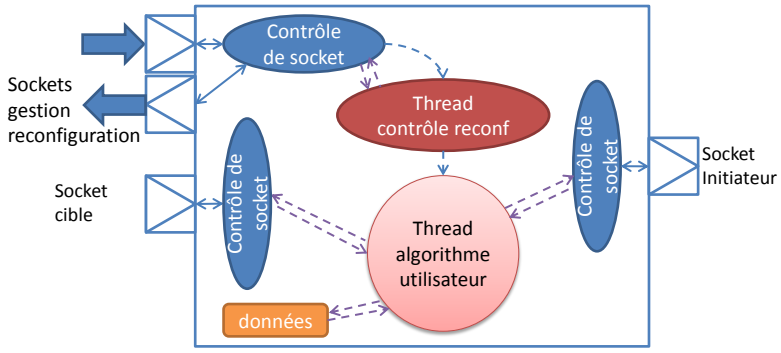


Figure 3.14 – *Architecture du module SystemC accueillant les tâches.*

D'abord, ce module contient une interface d'entrée et de sortie utilisée pour communiquer entre les autres modules. Dans la norme TLM, elle est dénommée respectivement comme des sockets cibles et initiateurs. Le nombre d'E/S des sockets peut être adapté aux besoins de l'application. Chaque interface est associée à un ensemble de services, *contrôle de socket*, utilisée pour connecter les sockets au module. Il y a également une interface liée au gestionnaire qui contrôle les aspects de reconfiguration partielle. Elle est composée de deux sockets pour une communication bidirectionnelle.

Afin de modéliser le comportement dynamique des tâches, le module est également constitué de deux threads dynamiques. Le premier est l'*algorithme utilisateur* et représente la fonctionnalité d'une tâche dynamiquement reconfigurable. Cette tâche est créée lors de l'exécution de la simulation et peut être détruite si elle n'est plus nécessaire. Le second est *le contrôle reconf* : il communique avec l'interface de configuration et est responsable de la création et de la destruction de l'*algorithme utilisateur*.

3.5.2.3 Le gestionnaire de reconfiguration

Le gestionnaire est informé de l'architecture du système, décrite en termes de zones reconfigurables. En utilisant cette information, le gestionnaire est en mesure de placer dynamiquement les tâches reconfigurables et d'effectuer l'ordonnancement.

Le placement d'une tâche est décidé lorsque la tâche est prête, c'est-à-dire lorsque la tâche précédente est terminée où lorsque les premiers paquets sont envoyés à la première tâche. La figure 3.15 montre les interactions entre les tâches et le gestionnaire de reconfiguration conduisant à leur placement et à leur ordonnancement sur le FPGA.

Le module précédent doit attendre que la prochaine tâche soit configurée pour envoyer les données et ainsi mettre fin à la transaction. Ainsi, ce module avise le gestionnaire de la fin de son exécution. À ce stade, le gestionnaire, informé des modules actuellement configurés sur le FPGA, ordonnance la tâche suivante. À cet instant, plusieurs cas peuvent se produire :

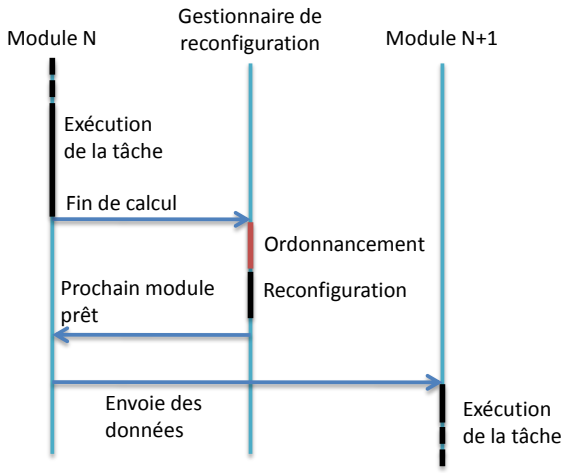


Figure 3.15 – *Interaction entre des tâches et le gestionnaire de reconfiguration.*

- Tout d'abord, si la tâche n'est pas présente sur le FPGA, le gestionnaire recherche une RPB compatible, vierge ou inutilisé (c'est-à-dire que la tâche est configurée sur le RPB mais ne s'exécute pas). Si un tel RPB est trouvé, le gestionnaire de reconfiguration configure le RPB. Une fois le processus terminé, le gestionnaire informe le module précédent qu'il peut envoyer les données vers le module suivant.
- Si la tâche est présente sur le FPGA, l'état de la tâche peut être soit en exécution, soit au repos. Au repos, le gestionnaire avise simplement le module précédent. Si la tâche est en exécution, la tâche suivante est mise dans une file d'attente jusqu'à la fin du traitement de la tâche sur la donnée précédente.

Un autre point important concerne la préemption. Dans notre cas, le développeur doit explicitement définir quelques points de préemption dans l'*algorithme utilisateur* lorsque la tâche peut être interrompue, ce qui correspond aux points où la sauvegarde du contexte peut se faire facilement. Un point de préemption est défini implicitement à la fin de l'exécution des tâches. La figure 3.16 montre un exemple de communication entre une tâche et le gestionnaire. Lorsque nous atteignons un point de préemption, le module utilise le socket afin d'aviser le gestionnaire qu'il est possible de préempter la tâche en cours d'exécution sur ce RPB. Ensuite, le gestionnaire décide si la préemption de la tâche est nécessaire et répond par socket au module avec la commande appropriée (arrêt ou reprise). Si la tâche doit être arrêtée, le gestionnaire envoie une commande de démarrage au module suivant.

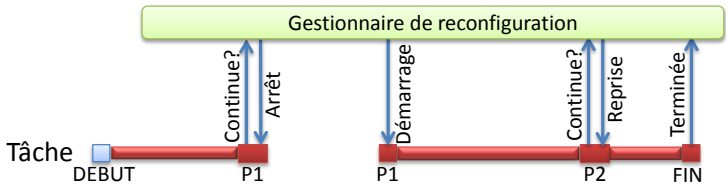


Figure 3.16 – *Exemple de préemption pour modéliser le partage de ressource.*

3.5.2.4 L'exploration d'architecture

Comme mentionné précédemment, notre objectif est de fournir au concepteur une solution viable qui réponde aux contraintes temporelles de l'application une fois son implémentation sur le FPGA. Pour ce faire, le simulateur effectue l'exploration architecturale, c'est-à-dire le choix des RZs qui deviendront des $RPBs$. Il est capable de simuler différents scénarios (un nombre différents de $RPBs$) jusqu'à ce que la simulation atteigne sans erreur l'ordonnancement d'une hyper-période. Pour ce faire, le simulateur doit être alimenté par un ensemble de RZs prédéfinies.

Cet ensemble de RZs est généralement hétérogène afin de réussir le placement de chaque tâche de l'application. Cet ensemble est décrit dans un fichier de configuration qui est analysé au début de la simulation. Un sous-ensemble de RZs est choisi pour lancer la simulation. Cependant, il y a certaines conditions pour constituer cette sélection qui doivent être prises en compte. La première et la plus évidente est que l'ensemble peut accueillir toute l'application. Si une seule tâche ne correspond pas à toutes les RZs, alors l'application ne peut pas être mise en œuvre. En outre, on peut vouloir choisir des RZs qui peuvent accueillir plusieurs modules afin d'améliorer les performances du système et de faciliter l'allocation. Dans notre algorithme, la simulation doit trier les RZs suivant la compatibilité descendante avec l'application. Ici, la compatibilité correspond au nombre de tâches qui peut être exécutées sur la RZ donnée. Même si l'ensemble des RZs est trié, il n'est pas toujours possible de sélectionner les premiers $RPBs$ pour effectuer la simulation. En fait, certaines tâches ne peuvent pas être implémentées sur n'importe quelle RZ, et par conséquent, ne remplit pas la première contrainte. Afin de satisfaire cette contrainte, nous devons sélectionner les RZs dont la tâche est uniquement compatible avec une seule RZ. Après cette étape, nous pouvons choisir les RZs restantes de l'ensemble.

Le bien-fondé de la simulation réside dans la vérification des échéances des tâches. Chaque fois qu'une tâche termine son exécution, le gestionnaire de reconfiguration vérifie si sa contrainte temporelle est satisfaite. Si non, le gestionnaire arrête la simulation en cours et permet au simulateur d'incrémenter le nombre de $RPBs$ maximum à utiliser.

3.5.3 Évaluation de notre approche

3.5.3.1 L'application proposée

Dans le cadre du projet ANR ARDMAHN [117], nous développons une passerelle domestique adaptative qui devrait fournir des flux vidéo vers différents terminaux, par exemple un lecteur vidéo portable, un ordinateur ou une télévision. Bien sûr, ces appareils n'ont pas les mêmes encodages vidéo et/ou d'exigences de taille d'image, de sorte qu'il devient nécessaire d'adapter le flux vidéo avant de les transférer vers ces terminaux. Une fois de plus, l'hétérogénéité du circuit cible implique de nombreux cas d'utilisation de la chaîne de transcodage et il est impossible de mettre en œuvre tous ces cas dans une seule puce. Par conséquent, la fonctionnalité de la reconfiguration partielle est utilisée pour changer la fonctionnalité du transcodeur (d'un cas d'utilisation à l'autre) et de mutualiser les ressources à l'intérieur du transcodeur (pour l'énergie et/ou l'économie des ressources). Par conséquent, notre applica-

tion consiste en une chaîne de transcodage vidéo de manipulation de plusieurs codecs vidéo. L'application porte sur des cas d'utilisation suivants : H.264 vers H.264, H.264 vers MPEG-2, MPEG-2 vers H.264 et VC-1 vers H.264. Même si les codecs vidéo sont différents, les chaînes partagent une architecture similaire, comme le montre la figure 3.17.

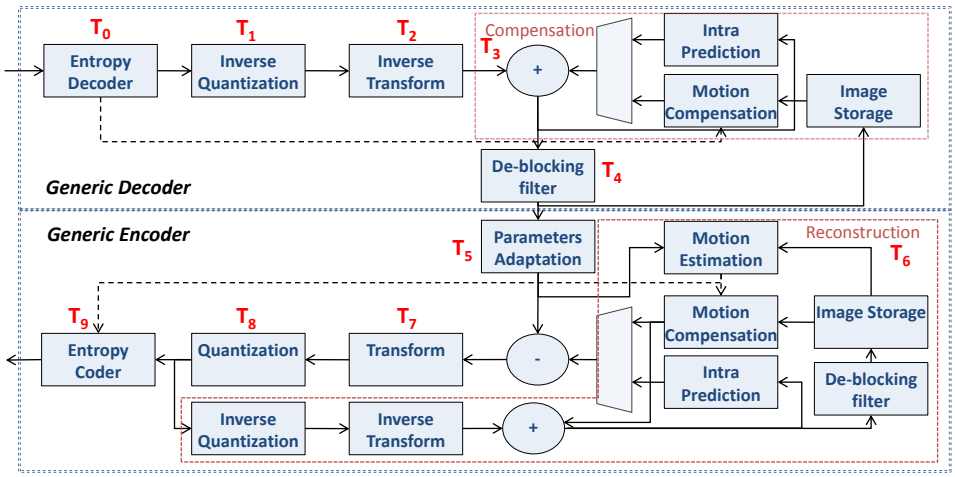


Figure 3.17 – *Chaîne de transcodage générique.*

3.5.3.2 Résultats

Nous avons créé manuellement un ensemble de 15 zones reconfigurables (RZs) pour alimenter le simulateur. Pour le moment, nous n'avons pas encore intégré l'algorithme permettant de déduire un ensemble de zones reconfigurables directement à partir d'un type FPGA. En effet, nous allons nous baser sur l'algorithme de la première phase d'initialisation avant le démarrage de l'algorithme BEE en intégrant des zones rectangulaires et également plus complexes. Ces travaux sont actuellement en cours. Nous avons pris soin de garder un ensemble hétérogène pour s'adapter à la nature hétérogène de l'architecture FPGA. Le simulateur s'assure également que toutes les tâches dans l'application trouvent une zone compatible.

La simulation nous donne comme résultat une implémentation sur cinq zones reconfigurables et nous permet également d'évaluer les performances globales du système. Par exemple, il est possible d'obtenir des informations sur les différents modules, l'état du gestionnaire de reconfiguration, les taux d'occupation des $RPBs$ et ainsi de suite afin d'aider au développement d'algorithme d'ordonnancement. Il est également possible d'évaluer le rapport entre le temps de reconfiguration et le temps d'exécution des tâches. S'il est trop élevé, ce rapport peut dégrader la consommation d'énergie ou même créer un goulot d'étranglement au niveau de la mémoire où les fichiers de configuration (bitstreams) sont stockés. Le tableau 3.9 montre le taux d'occupation du temps dans chaque état possible des $RPBs$ (l'état correspond aux états des tâches) pour chacune des 5 zones reconfigurables.

Ces résultats ont été obtenus avec une stratégie d'ordonnancement utilisant tous les $RPBs$ disponibles et un algorithme d'ordonnancement classique *RoundRobin* qui évoluera très certainement à mesure que nous allons développer de nouvelles stratégies. Nous pouvons voir que grâce à cette stratégie d'ordonnancement, les $RPBs$ sont en état exécution plus de 90%

Tableau 3.9 – *Taux d'occupation (en % du temps total de simulation)*

	RPB_0	RPB_1	RPB_2	RPB_3	RPB_4
Vierge	0.1	0.3	0.5	0.8	1
Reconfiguration	4.1	4	4.1	4	4.1
Inactif	2.2	2.9	2.9	4	4.1
En exécution	93.6	92.8	92.5	91.2	90.8

du temps, ce qui représente une trés bonne utilisation de ces zones. Par ailleurs, la simulation se termine en quelques secondes sur un processeur Intel Core2Quad fonctionnant sous Windows7 x64.

Le simulateur génère également un fichier de trace en fin de la simulation, contenant des informations de débogage concernant chaque module de la chaîne et le gestionnaire de reconfiguration. Un exemple de trace est illustré sur la figure 3.18, montrant les différents états pris par les $RPBs$ et par les tâches pendant la simulation, en visualisant l'ordonnancement effectué par le gestionnaire de reconfiguration. La figure montre également les reconfigurations qui se produisent physiquement sur le FPGA (configuré via le port d'accès de configuration interne (ICAP), une cellule présente sur le FPGA Xilinx). Dans notre cas, l'ICAP est gérée par notre contrôleur, FaRM [162, 168].

Par exemple, considérons la tâche T_2 (*transformée inverse*). À l'instant t_1, le gestionnaire vérifie si la tâche est présente sur le FPGA. Comme elle n'est pas présente, il demande une reconfiguration au gestionnaire de configuration FaRM. Par conséquent, l'ICAP passe à l'état *CONF* T_2. La tâche T_2 bascule en état *Exécution de* RPB_2, ce qui signifie que RPB_2 a été choisi pour accueillir la tâche (vérifiable avec l'état de RPB_2). Entre t_1 et t_2, le gestionnaire reçoit une demande d'implémentation de la tâche T_1. Cependant, cette tâche n'est pas présente sur le FPGA ce qui entraîne une reconfiguration. Toutefois, il n'est pas possible d'utiliser le module FaRM puisqu'il reconfigure déjà RPB_2. Par conséquent, le gestionnaire de reconfiguration interdit les reconfigurations multiples en même temps. Une fois la première reconfiguration terminée, à l'instant t_2, le module FaRM commence à reconfigurer la tâche T_1 sur RPB_4 et RPB_2 passe à l'état *Exécution de* T_2.

3.6 Conclusion

Dans cet axe de recherche, nous avons contribué au placement et à l'ordonnancement de tâches matérielles sur des architectures reconfigurables dynamiquement. Nous avons proposé différentes approches, dynamiques, méta-heuristique et par simulation au niveau système pour résoudre ces problèmes.

Les méthodes dynamiques ont l'avantage de rechercher une solution optimale mais ne peuvent résoudre des problèmes de placement/ordonnancement avec de nombreuses tâches. Les méthodes approximatives, comme l'algorithme BEE, permettent, au contraire, d'examiner un ensemble de solutions sur un nombre de tâches plus important et de se rapprocher

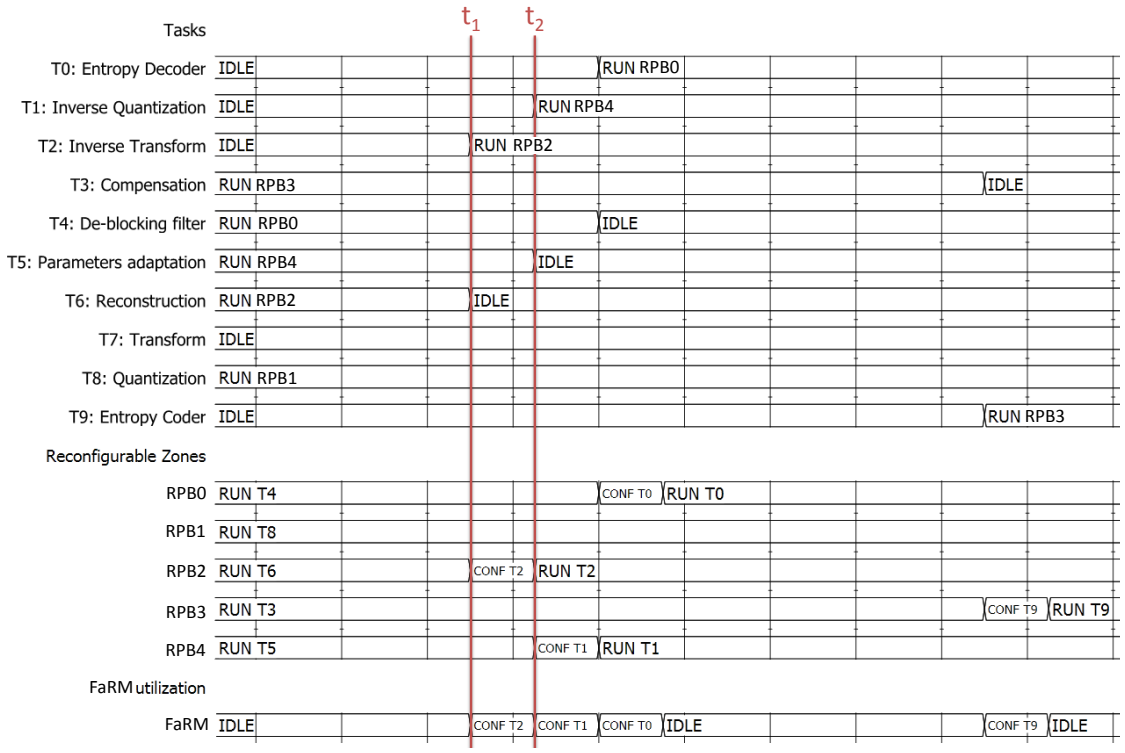


Figure 3.18 – *Exemple de simulation de la chaîne de transcodage.*

de la solution optimale. La dernière approche par modélisation et simulation est également très intéressante et pourrait être couplée à la méthode *BEE*. En effet, cette approche par modélisation/simulation permet d'obtenir un ordonnancement et un placement pour un ensemble de tâches plus conséquent. Nous estimons qu'un nombre de 20 à 40 tâches serait tout à fait réalisable. Pour le moment, le concepteur doit définir manuellement un ensemble de zones reconfigurables à donner au simulateur mais des travaux sont en cours sur cette problématique. Nous aimerions inclure entièrement notre simulateur dans le flot de conception PR de Xilinx comme démonstrateur. Par exemple, le simulateur déduirait les ensembles de *RPBs* à partir du circuit reconfigurable utilisé dans les premières phases du flot, après la génération des netlists par exemple. Un autre point très important à garder à l'esprit est qu'actuellement nous n'avons pas effectuer toutes les vérifications sur les *RPBs* sélectionnés par l'approche de simulation : nous supposons ici que les *RPBs* ne se chevauchent pas (une partie de l'algorithme *BEE* pourrait être judicieusement utilisée). Si l'ensemble de *RPBs* n'est pas correctement défini, alors cela entraînera très certainement une défaillance lors de la recherche d'une implémentation par simulation.

Pour la contribution à ces travaux, deux doctorants ont participé à cette axe de recherche ainsi que des stagiaires de Master. L'ensemble des travaux menés au sein de cet axe de recherche a conduit à 1 chapitre de livre [165], 2 revues [164, 163], 8 publications dans des conférences internationales [175, 174, 172, 169, 167, 170, 166, 184] et 5 publications dans des conférences nationales [197, 193, 195, 191, 189].

CHAPITRE 4

Systèmes d'exploitation logiciel/matériel au sein des MPSoC reconfigurables

4.1 Contexte

De manière cyclique depuis ses débuts, le domaine de la conception des circuits intégrés a été obligé de s'adapter aux pressions cumulées de la technologie, des contraintes commerciales et financières par des innovations en rupture complète avec les solutions de conception précédentes. Ces facteurs d'innovation sont intervenus de manière régulière à des niveaux de plus en plus hauts dans les flots de conception aussi bien d'un point de vue logiciel que matériel. Depuis quelques années, notamment avec l'apparition des systèmes sur puce et des architectures reconfigurables dynamiquement, l'évolution de la densité d'intégration a proposé de répondre aux contraintes de conception par la parallélisation au niveau tâches ou données, augmentant ainsi le nombre des unités de calcul, notamment dans les systèmes embarqués. Cette tendance semble se confirmer aujourd'hui et risque de devenir un véritable « mur » de complexité, lorsque à moyen terme, les systèmes intégreront plusieurs dizaines de blocs (processeurs et/ou coprocesseurs).

Cette complexité ajoutée à l'hétérogénéité croissante de ces architectures devenues dynamiques ou adaptables nous conduit aujourd'hui à imaginer de nouveaux mécanismes de gestion et d'exploitation de ces plateformes qui seront le support d'exécution des applications de demain. Les solutions classiques de centralisation du contrôle de toute la gestion de l'architecture semblent aujourd'hui ne plus pouvoir répondre à la fois à l'augmentation du nombre d'unités d'exécution et à leur hétérogénéité. Seule une approche distribuée, pensée de manière plus flexible et plus évolutive, pourra relever ce défi d'exploitation des architectures du futur.

Les caractéristiques habituellement utilisées pour dimensionner les plateformes embarquées et pour y déployer les applications temps réel deviennent de plus en plus variables ou difficiles à quantifier de manière significative. Jusqu'à présent, les systèmes destinés à inté-

grer des applications relativement dynamiques étaient dimensionnés au pire-cas. On parle maintenant de tâches à temps réel dur et de tâches à temps réel souple. Par exemple, les caractéristiques actuellement essentielles que sont le nombre et le type des tâches exécutées par le système ne sont plus forcément connues à l'avance. Les changements de formats de décompression pour les applications vidéo (comme présenté dans le chapitre 3, page 85), la prolifération des services pour la téléphonie mobile, l'omniprésence de l'intelligence de bord dans l'automobile, les changements de mission en robotique autonome sont des exemples émergents à ce besoin d'adaptation des plateformes reconfigurables de calcul. Dans ce contexte, l'architecture devra être capable de supporter des applications multi-tâches, dynamiques et temps réel qui devront s'adapter au contexte environnemental dans lequel elles évoluent.

4.1.1 L'évolution du système d'exploitation

Le système d'exploitation (OS) a clairement un rôle important à jouer dans ces nouvelles architectures reconfigurables dynamiquement. Un OS est classiquement organisé autour de services fournis aux tâches logicielles ou matérielles. Les principaux services qui constituent le cœur du noyau sont donc liés à la création/destruction des tâches, au placement et à l'ordonnancement des tâches comme nous l'avons expliqué dans les deux premiers chapitres, à l'allocation de mémoire, à la communication entre tâches et aux synchronisations entre tâches. L'OS est donc amené à gérer des objets, ou structures de données, permettant de connaître précisément l'état du système. Comme tous les éléments entrant en jeu dans la conception des systèmes, les OS ont subit des évolutions importantes au cours des 40 dernières années. Ces évolutions montrent que celles-ci ont notamment été poussées par l'évolution des applications, ces dernières se complexifiant de manière importante avec l'explosion des systèmes embarqués/enfouis qui inondent notre quotidien.

Ces systèmes d'exploitation, qui doivent exploiter au mieux les architectures reconfigurables, doivent avoir les moyens de gérer de manière efficace le reconfigurable. Par exemple, les FPGAs récents comme ceux de la société Xilinx fournissent cette fonctionnalité via le port de configuration interne (ICAP [124]). Toutefois, le contrôleur de Xilinx, *xps_hwicap* [125], montre de faibles performances. Dans un tel cas, la reconfiguration partielle ajoute une surcharge de temps non négligeable qui est considérée comme l'un des grands défis de cette technologie. Par exemple, si le système effectue l'ordonnancement des tâches matérielles, le temps de reconfiguration à l'exécution des tâches pourrait devenir inacceptable, supprimant ainsi aux développeurs l'utilisation de la reconfiguration dynamique [126, 127].

4.1.2 L'évolution des architectures et des processeurs

Depuis quelques années, on assiste à un changement dans la façon d'augmenter les performances d'un processeur. En effet, après avoir principalement utilisé la montée en fréquence depuis les débuts de l'informatique, les fondeurs se tournent désormais vers l'augmentation du nombre de cœurs. La raison principale de ce changement est que la montée en fréquence est généralement accompagnée d'une augmentation de la consommation énergétique et de la production de chaleur même si la technologie 28nm prévoit une diminution de 40% de la consommation d'énergie [157] par rapport au 40nm.

Les prémisses du parallélisme au sein d'une puce « on-chip » de type CPU remontent à la technologie Hyper-Threading d'Intel, qui permet d'exécuter virtuellement plusieurs threads simultanément. Les premiers processeurs réellement parallèles sont apparus en 2005, et continuent actuellement à se développer, le nombre de cœurs augmentant régulièrement. Néanmoins, après plusieurs années de mise en œuvre du parallélisme, la plupart des programmes actuels peinent à utiliser le parallélisme qui leur est offert. En effet, leur conception reste, à de rares exceptions près, séquentielle. Heureusement, les systèmes d'exploitation modernes permettent l'exécution multi-tâches, ce qui autorise à utiliser différentes unités d'exécution (PEs [1]) pour différents programmes tournant simultanément, et donc tirent partie de ce parallélisme. Les puces que nous avons mentionnées entrent dans la catégorie multicœur (« multicore »). Mais une autre catégorie de parallélisme on-chip commence à se dessiner : le « manycore » [121, 120]. En effet, au delà d'une dizaine de PEs [159], le système actuel d'échange de données ne peut-être conservé sans dégrader fortement les performances. Dans ce cas, on utilise alors un paradigme équivalent à celui des réseaux, considérant chaque PE comme un nœud d'un réseau. Les PEs sont alors reliées entre elles par un NOC [2] ou réseau sur puce [119].

En constante évolution, les HPCs ont commencé récemment à introduire des FPGAs en tant que PEs dans leurs architectures. Cet ajout permet de programmer matériellement un IP [3] afin d'accélérer une section particulière du programme, en la réalisant à la place du CPU. De telles architectures sont alors appelées HPRC. Cependant, dans les HPRCs actuels, les FPGAs sont contrôlés par les CPUs et reconfigurés en un seul bloc. Les premières expérimentations intégrant la reconfiguration partielle sur des HPRCs commerciaux, les Cray XD1, ont été menées par El-Araby et al. [122].

Notons que les GPUs sont également une solution actuellement exploitée. Tout de même, quelques facteurs de base ont tendance à distinguer les architecture HPRC à base de FPGA et ceux à base de GPU.

Concernant le chemin de données, les GPUs utilisent des ASICs dédiés pour optimiser les performances des types de données et des opérations. Alors que les FPGAs ont également de l'ASIC dédiée ou plutôt des blocs matériels, ces blocs sont à un niveau de granularité inférieur. En ce qui concerne la mémoire, les GPUs atteignent une bande passante importante en utilisant des structures mémoires et caches pour les opérations de rendering par exemple. Les FPGAs offrent de la flexibilité dans le nombre et la largeur (en bits) de la mémoire externe, et la possibilité de regrouper des centaines de blocs mémoires adressables indépendamment, situés à l'intérieur du FPGA. De plus, les GPUs offrent un modèle de parallélisme fixe tandis que le FPGA permet des combinaisons de parallélisme pipeliné et peut mettre en œuvre des communications complexes et optimisées parmi les éléments de calcul. En bref, le GPU et le FPGA sont complémentaires. Si une application peut être décrite par un modèle de calcul de GPUs et que notre architecture cible permet de créer assez de parallélisme (une concurrence de 100 à 1000 par exemple), le GPU est une solution tout à fait pertinente qui, en plus, a un modèle de programmation commun. Cependant, les FPGAs sont intéressants pour du

1. Processing Elements
2. Network On Chip
3. Intellectual Property

contrôle, de la manipulation de données et des modèles de concurrence qui demandent des comportements plus particuliers, plus spécifiques.

Le projet européen ÆTHER[86], qui nous impliquait et détaillé dans le chapitre 2, a également présenté une architecture distribuée basée sur des éléments reconfigurables nommés SANEs [123] montés en réseau.

4.2 Contributions

Nos contributions proposent différentes approches pour répondre aux problèmes soulevés dans la section précédente. Concernant les systèmes d'exploitation, les premiers travaux ont consisté à concevoir un RTOS multiprocesseur décrit matériellement (HwRTOS [4]) présenté en section 4.2.2. À partir de ces travaux, l'idée a été d'étendre ces derniers, décrite dans la section 4.2.3, à des tâches logicielles et matérielles sur la base d'un RTOS multiprocesseur logiciel/matériel. Dans un même temps, dans la section 4.2.4, nous nous intéressons également à utiliser les architectures reconfigurables dans les systèmes répartis, en particulier pour le domaine du HPRC.

Ces travaux sont possibles si la technique de gestion de la reconfiguration est efficace. Ceci n'est pas le cas actuellement comme nous l'avons expliqué dans le contexte. Ainsi, nous proposons un module FaRM, résumé dans la section 4.3, qui permet de générer de manière très efficace la reconfiguration dynamique et partielle des FPGAs de la société Xilinx et de fournir des modèles de temps pour alimenter notre simulateur SystemC présenté dans le chapitre 3. La plupart de ces travaux sont en collaboration avec des partenaires universitaires et industriels à travers des projets de recherche.

4.2.1 Amélioration des performances de la reconfiguration dynamique sur FPGA

L'un des problèmes majeurs de la reconfiguration partielle est sa faible performance. Par conséquent, son utilisation est limitée puisque le temps de reconfiguration est trop élevé par rapport au temps d'exécution des tâches. Pour surmonter ce problème, nous avons conçu un contrôleur d'ICAP efficace, appelé FaRM, offrant une haute vitesse de configuration et une relecture de zone reconfigurable (readback) efficiente, réduisant ainsi le temps de configuration autant que possible. Afin d'améliorer les performances, FaRM utilise des techniques telles que les accès maîtres sur le bus, l'overclocking de l'ICAP, le préchargement de bitstream à l'intérieur du contrôleur et notre technique de compression de bitstream, Offset-RLE, qui est une amélioration de l'algorithme de codage RLE [5]. La combinaison de ces approches nous permet d'atteindre un débit théorique à l'entrée de l'ICAP de 800Mo/s à 200MHz. Afin de compléter notre approche, nous proposons des modèles de coût correspondant aux temps de reconfiguration pour le niveau système. En effet, ces modèles peuvent être utilisés durant les premiers stades de développement comme c'est le cas de notre simulateur SystemC. Nous

4. Hardware Real Time Operating System
5. Run Length Encoding

avons validé notre approche sur des modules AES de cryptage/décryptage. Cette contribution est détaillée dans la section 4.3.

4.2.2 HwRTOS : RTOS multiprocesseur matériel centralisé

Ce HwRTOS est capable d'ordonnancer des applications allouées sur un ensemble de processeurs homogènes ou hétérogènes. Le HwRTOS est matériel, c'est-à-dire complètement décrit en langage HDL (Hardware Description Language). C'est pourquoi, un ensemble de paramètres génériques a été défini avant l'étape de la synthèse (nombre de processeurs, nombre de tâches ...). Ces paramètres sont à ajuster en fonction d'une ou des applications qui s'exécuteront sur la plateforme. Le HwRTOS supporte la plupart des services logiciels comme le sémaphore, les files de messages mais également de l'ordonnancement local, global et mixte ainsi qu'un débogueur en mode multiprocesseur. Ce dernier service permet d'envoyer des traces de debug par Ethernet (UDP) afin qu'un outil (Trace Analyzer) puisse extraire les performances et le comportement de l'application en simulation ou sur carte de prototypage. Par exemple, l'outil évalue l'overhead, le temps d'exécution des tâches, le temps de réponse, le nombre de changements de contexte, etc.

Afin de valider une application et l'ensemble des services du HwRTOS, nous avons développé une plateforme virtuelle mixte SystemC/VHDL. L'avantage d'une telle plateforme est d'avoir la possibilité d'exécuter le code final de l'application sur des processeurs abstraits, pilotée par le HwRTOS au cycle près. En effet, seul le HwRTOS est décrit en VHDL. Le reste de l'architecture est de niveau système (niveau TLM). La plateforme et le HwRTOS offrent également la possibilité de migration de tâches nécessaire pour certains algorithmes d'ordonnancement (chapitre 2, page 37). La validation du HwRTOS en condition réelle a été réalisée sur des cartes de prototypage (ML310, VirtexII Pro, ML405, Virtex4, et ML506, Virtex5). Les résultats obtenus montrent une solution simple (code quasi-compatible VxWork) et efficace (overhead réduit) pour ordonnancer des tâches sur une architecture MPSoPC.

Cependant, l'inconvénient de ce HwRTOS est principalement la centralisation. Ce HwRTOS peut piloter N processeurs qui, tout de même, ne sont pas obligatoirement connectés sur le même bus car le HwRTOS est multi-port. Ce nombre N dépend principalement de l'architecture et de la contention sur les bus. De plus, le HwRTOS étant matériel, une perspective serait de gérer des tâches matérielles, non réalisée dans ce HwRTOS. Cependant, ces travaux ont servi de base pour proposer et déposer le projet FOSFOR. Ce projet, présenté ci-après, doit justement répondre, entre autre, à ces deux inconvénients précédemment identifiés.

4.2.3 L'OS matériel FOSFOR pour le reconfigurable

Le caractère multiprocesseurs, ou plutôt multi-ressources d'exécution de la plateforme lié à la distribution de l'OS requiert des mécanismes particuliers pour parvenir à assurer une cohérence globale du système. En effet, il est nécessaire que chaque tâche puisse accéder à l'ensemble des services du système d'exploitation et ceci sans distinction de type de tâches et/ou de localisation. De plus, compte tenu de l'hétérogénéité de la plateforme d'exécution et de son caractère multiprocesseurs et reconfigurable, les services précédents doivent exhiber des propriétés particulières que nous avons identifiées : Accès aux services quelque soit leur

localisation, ordonnancement, gestion des ressources, gestion mémoire, gestion des communications, gestion des synchronisations.

Nous avons identifié les fonctionnalités à implémenter dans l'OS FOSFOR, afin de prendre en charge la flexibilité et la distribution de celui-ci au sein de la plateforme. La solution retenue est une implémentation supportant ce principe de distribution de l'OS : RTEMS [6], et la couche supportant l'exécution multiprocesseur de façon transparente, le MPCI [7]. Cette contribution est détaillée dans la section 4.4 sur les systèmes d'exploitation matériel. Afin de tester notre approche, nous avons ciblé une application de type flot de données de suivi de cibles, fournie par notre partenaire TRT.

4.2.4 Plateformes HPRC et auto-adaptatives

Contrairement aux deux contributions précédentes, cette contribution traite les aspects HPRC et non les aspects temps réel. Dans ces architectures distribuées et fortement parallèles, nos travaux ont pour objectif d'offrir une méthodologie et des propositions de plateformes de prototypage associées pouvant exécuter des applications HPRC ou flot de données où il est possible de mixer des applications contenant des descriptions logicielles et matérielles des fonctionnalités à exécuter. Toute la gestion de ces plateformes est basée sur de l'OS Linux embarqué.

Les architectures de calcul parallèle commencent progressivement à intégrer des unités matériellement reconfigurables. Ces unités peuvent accueillir différents circuits logiques, et donc implémenter des accélérateurs matériels en lien direct avec l'application exécutée. Plus précisément, nous développons une nouvelle approche permettant une plus grande mixité entre les éléments de calcul logiciels et matériels afin de tirer parti de toute la puissance que peut apporter un accélérateur matériel pour l'exécution d'une tâche spécifique. À cette fin, nous proposons un flot destiné à la conception d'applications parallèles tout en gardant une compatibilité avec les applications déjà existantes. Nous présentons également des plateformes d'exécution destinées à ces applications et utilisant la reconfiguration dynamique partielle, qui permet de réduire la granularité de la reconfiguration, trop souvent limitée au FPGA entier. Cette contribution est détaillée dans la section 4.5.

4.3 Amélioration des performances de la reconfiguration dynamique sur FPGA

4.3.1 Introduction

La reconfiguration partielle a été introduite à la fin des années 90 pour répondre à l'accroissement des ressources associées au FPGA. En effet, il permet des changements de comportement au sein d'une partition reconfigurable de l'architecture FPGA tandis que la logique restante est toujours en fonctionnement. La reconfiguration partielle peut donc être utilisée

6. Real-Time Operating System for Multiprocessor Systems
7. MultiProcessor Communications Interface

pour changer la fonctionnalité d'un système, économiser des ressources FPGA et de l'énergie [103, 128]. Les premiers FPGAs à supporter cette caractéristique ont été la série XC6200, programmés en utilisant JBITS et introduisant certains concepts qui sont encore utilisés aujourd'hui [129].

Cette reconfiguration partielle pose deux problèmes majeurs, le placement/ordonnancement de ces zones, et le temps de reconfiguration des zones. Ainsi, notre approche se concentre, dans ce chapitre, sur l'optimisation des temps de configuration en travaillant sur la configuration effective du FPGA, c'est-à-dire lors de la transmission du bitstreams à l'ICAP. Par conséquent, nous ne considérons pas le placement des tâches dans cette contribution, qui peuvent également participer à l'amélioration de le temps de configuration (c.f. chapitre 2, pages 61 et 69) en choisissant une zone reconfigurable optimale pour un ensemble de tâches [169].

Il est possible de répondre à ce problème de performance de plusieurs façons. Tout d'abord, en travaillant sur les données de configuration, appelé bitstreams. Ces fichiers binaires sont transférés à partir d'une mémoire vers l'ICAP. Réduire la taille du bitstream en utilisant des techniques de compression diminue donc le temps de configuration. De plus, cette compression de bitstream minimise également les besoins en mémoire dans le système. Cependant, le système doit aussi décompresser les données, donc utilise des ressources supplémentaires pour la décompression qui est une métrique significative.

Une autre façon d'améliorer la performance est de travailler sur l'architecture du contrôleur d'ICAP. Une solution consiste, par exemple, à intégrer un contrôleur d'accès mémoire (DMA [8]), afin d'augmenter directement le débit d'écriture/lecture de l'interface de configuration [130]. L'ordonnancement peut également prédire les futures tâches à exécuter sur une zone reconfigurable (RPB) [165, 71, 131]. Ainsi, le préchargement de bitstream dans une mémoire peut considérablement réduire le temps de configuration.

Nous proposons de combiner ces approches dans notre contrôleur FaRM. Nous présentons une architecture offrant des performances allant au delà de la limite théorique de l'ICAP. Afin de réduire les temps de transfert, notre contrôleur intègre une interface intelligente maître/esclave, et est capable de gérer la décompression et le préchargement de bitstream. En outre, nous voulons que FaRM puisse récupérer facilement les données d'une zone reconfigurable du FPGA. Par ailleurs, afin d'estimer ces temps de reconfiguration en utilisant FaRM au cours des premières phases de développement, nous proposons des modèles précis de coûts de temps de reconfiguration qui peuvent être utilisés au niveau système.

4.3.2 L'architecture FaRM

L'IP FaRM est basé sur l'architecture représentée sur la figure 4.1. Il est construit sur la base d'une interface intelligente maître et d'une interface esclave qui peuvent être connectées à un wrapper (dans notre cas, nous utilisons un bus PLB [9] et AXI4.0 [10]), tandis que l'IP Xilinx a seulement une interface esclave. L'interface esclave configure les accès aux registres de FaRM alors que l'interface maître est responsable des accès à la mémoire, où les bitstreams

8. Direct Memory Access
9. Processor Local Bus
10. Advanced eXtensible Interface version 4.0, société ARM

et les contenus relus des zones reconfigurables doivent être stockés. Il gère les différents modes de fonctionnement que nous décrirons ci-après. L'interface maître permet à l'IP d'être indépendante du microprocesseur et d'agir comme un DMA. En effet, le microprocesseur n'accède qu'aux registres de FaRM afin de contrôler l'ordonnancement des tâches matérielles et n'effectue pas les transferts de données comme le fait l'IP de Xilinx. Les deux interfaces peuvent être connectées sur deux bus différents, séparant les accès rapides à la mémoire en lecture/écriture des accès plus lents à des fins de configuration. Par ailleurs, il convient de noter que FaRM est complètement indépendant de l'application ciblée : il n'a besoin que d'un microprocesseur très peu performant (par exemple Picoblaze ou MicroBlaze), ou d'un IP matériel pour contrôler FaRM ainsi que d'un accès direct au bitstream situé généralement dans une mémoire (dans notre cas, nous utiliserons la DDR2).

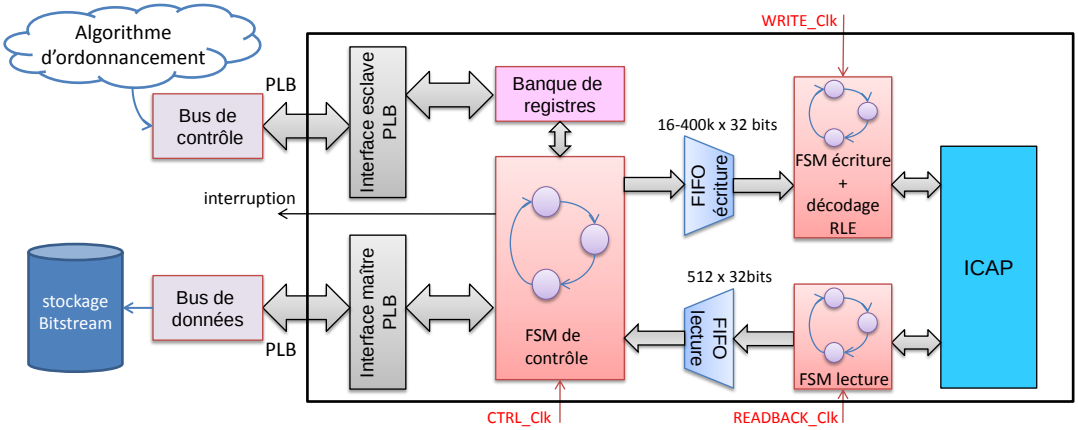


Figure 4.1 – *L'architecture FaRM.*

Comme l'IP de Xilinx, l'ICAP et le bus système ont des domaines d'horloge différents afin de faire face à la limitation de l'horloge de l'ICAP à 100MHz et sans limiter la fréquence du bus. Toutefois, nous avons pu monter au-delà de 100MHz pour l'ICAP. Nous avons également choisi de différencier les horloges pour la lecture et l'écrire sur l'ICAP car nous avons atteint des fréquences beaucoup plus élevées en configuration qu'en relecture de zones reconfigurables. Nous allons présenter succinctement les différentes techniques que nous avons utilisées pour FaRM.

4.3.2.1 La compression du bitstream

Les bitstreams sont des fichiers binaires utilisés pour configurer le FPGA. Dans le cas de la reconfiguration partielle, à partir de l'intérieur du FGPA, les bitstreams doivent être transférés à partir d'un endroit de stockage (par exemple, la mémoire DDR) vers l'ICAP. Ces bitstreams sont composés d'une séquence de contrôle qui initialise l'ICAP et d'une séquence de données utilisée pour la configuration physique du FPGA [132] (par exemple la configuration de LUTs et d'interconnexions). Après l'analyse de certaines séquences, nous avons remarqué qu'il y avait une certaine redondance dans la partie des données du bitstream qui pourrait facilement être compressée, en particulier lors de la configuration des contenus de BRAMs qui sont souvent initialisées à zéro. Par conséquent, nous avons choisi d'utiliser une technique de compression basée sur RLE qui semble être un bon compromis entre efficacité et complexité.

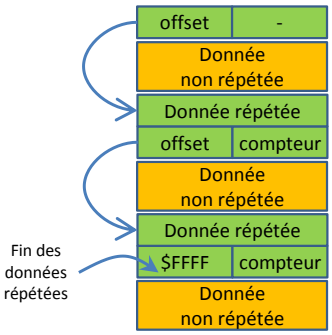


Figure 4.2 – *Principe de notre technique O-RLE*

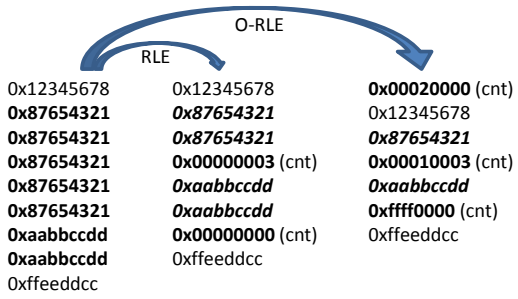


Figure 4.3 – *RLE versus O-RLE*

Comme amélioration de la technique RLE canonique, nous avons essayé d'enlever le mot supplémentaire dans le bitstream compressé. Notre approche O-RLE [11] est illustrée à la figure 4.2. Afin d'indiquer la répétition des données, nous utilisons un mécanisme de pointeur. Au début du bitstream compressé, on insère un mot contenant le décalage de la donnée répétée suivante. Il n'est plus utile de répéter les données deux fois comme la technique RLE canonique le fait, puisque le décalage nous donne déjà cette information. Juste après les données répétées, nous insérons un mot contenant deux éléments d'information : le nombre de fois que les données sont répétées et le décalage vers le mot suivant répété. Si les données sont les dernières données répétées du bitstream, nous utilisons le mot réservé 0xffff comme offset. Notons que le compteur et le décalage sont codés sur deux octets chacun. Ceci est possible parce que nous travaillons sur des mots de 32 bits. Dans notre approche, c'est un bon compromis puisque la valeur maximale de 65535 n'est pas susceptible d'être atteinte pour les bitstreams. De toute manière, si nous avions besoin de plus de 65535, il suffit de lancer une autre séquence. La figure 4.3 montre un exemple de codage de notre technique O-RLE.

4.3.2.2 Mode de préchargement du bitstream

Ce mode permet à l'utilisateur de pré-charger un bitstream ou du moins le début selon la taille de la FIFO d'écriture. En utilisant cette mémoire rapide au plus près de l'ICAP, cela nous permet d'approcher le débit théorique de l'ICAP. La taille de la FIFO n'a pas besoin d'être supérieure à la taille de bitstream. Nous avons prouvé que cela n'est pas utile la plupart du temps. En fait, l'utilisation des modes de compression et de préchargement réduisent considérablement les besoins en mémoire de stockage du système pour les mêmes performances [162].

4.3.2.3 Mode de relecture automatique du bitstream

Une autre caractéristique majeure de FaRM est la relecture automatique de zones reconfigurables. FaRM fournit un moyen facile mais efficace pour lire les données de configuration de n'importe quelle zone du FPGA. Cette fonctionnalité peut être utilisée à des fins de sauve-

11. Offset Run Length Encoding

garde du contexte de deux manières différentes. Tout d'abord, en relisant la partition entière sur laquelle la tâche s'est exécutée. Cette solution est très coûteuse en terme de temps mais elle peut être utilisée pour migrer la tâche sur une autre partition, avec le respect de certaines règles comme mentionné dans [133]. L'alternative est de lire directement les registres ou latches des colonnes CLBs qui contiennent l'état des registres. Cette solution est beaucoup plus efficace si aucune migration n'est nécessaire.

Avec cette fonctionnalité de relecture, le processus de relecture est entièrement automatisé. Il est réalisé en trois étapes. D'abord, la préparation de la relecture. Les séquences de commande sont écrites dans la mémoire accessible par l'ICAP au travers de l'interface maître. Ensuite, l'IP FaRM doit être configuré. Trois registres contiennent les adresses pour chaque partie de la relecture (l'entête de la séquence d'initialisation, l'adresse pour le stockage du bitstream relu, et l'entête de fin). Le processus de relecture automatique est alors démarré. Pour finir, le microprocesseur attend la fin du processus, soit en mode d'attente active (polling) ou en mode passif (interruption). Dans ce second mode, le processeur peut ainsi réaliser d'autres opérations dans le FPGA sauf dans la zone en cours de reconfiguration.

4.3.3 Les modèles de coût pour le temps de reconfiguration

Comme nous l'avons évoqué précédemment, nous proposons des modèles de coût de temps de reconfiguration en tenant compte des différents modes de fonctionnement de FaRM. Le temps de reconfiguration a été estimé pour une opération d'écriture canonique. En utilisant ChipScope [12], il est possible de capturer certains signaux de notre contrôleur FaRM, y compris l'interface du bus PLB. Dans nos expérimentations, l'ICAP ne limite pas le transfert : une fois que l'accès est terminé sur le bus, l'ICAP permet de traiter tous les mots avant le prochain accès sur le bus en mode burst. Ainsi, ce temps ne reflète que les transferts du bitstream sur le bus. Le temps de reconfiguration est estimé par les séquences d'accès de type burst et le dernier transfert qui utilise un simple accès. Soit $N_{bitstream}$ la longueur du bitstream en mots de 32 bits, T_{bus} la période du bus en ns, $latency$ la latence initiale en cycles d'horloge, t_{burst} le temps moyen pour un accès de type burst en cycles et N_{burst} le nombre de mots dans un burst. L'équation 4.1 donne une estimation du temps de reconfiguration pour un bitstream non compressé.

$$t_{\text{write}}\,(\text{ns}) = (latency + t_{\text{burst}} * \lfloor \frac{N_{\text{bitstream}}}{N_{\text{burst}}} \rfloor + (t_{\text{burst}} - N_{\text{bitstream}} \bmod N_{\text{burst}})) * T_{\text{bus}} \tag{4.1}$$

Dans le cas d'un bitstream compressé, le modèle de temps de reconfiguration ne peut pas être estimé correctement car il est très dépendant de la régularité de compression du fichier. Si la compression est uniforme et également répartie sur le bitstream, alors l'équation 4.1 peut être utilisée pour calculer le temps de reconfiguration. Dans le cas de compression irrégulière qui est le cas réel, cette équation fournie la borne inférieure. Par contre, la borne supérieure se calcule à partir de notre algorithme présenté dans [162]. De plus, nous sommes

12. Outil logiciel et IP matériel fourni par Xilinx pour capturer des signaux à l'intérieur du FPGA

également capable de fournir un modèle de coût en mode préchargement et donc de choisir la profondeur de la FIFO en écriture en fonction des performances voulues avec et sans compression. L'estimation de tous ces coûts de temps de reconfiguration est plus compliquée à calculer et est présentée dans [162]. Ces modèles pourront être utilisés dans notre simulateur SystemC.

4.3.4 Résultats obtenus

FaRM a été testé sur une carte ML507 (Virtex5) à la fois avec du cryptage/décryptage AES et une IP FFT64, comme illustrée à la figure 4.4. Cette architecture contient un *RPB* compatible au sens des ressources (soit l'encodeur ou le décodeur AES), et un autre *RPB* pour l'IP FFT64. Les *RPBs* sont connectés au bus PLB via un wrapper. Le système est piloté par un processeur MicroBlaze. Les bitstreams sont stockés dans la carte Compact Flash par défaut, puis au départ du programme de test, copiés par le MicroBlaze dans la mémoire principale DDR2.

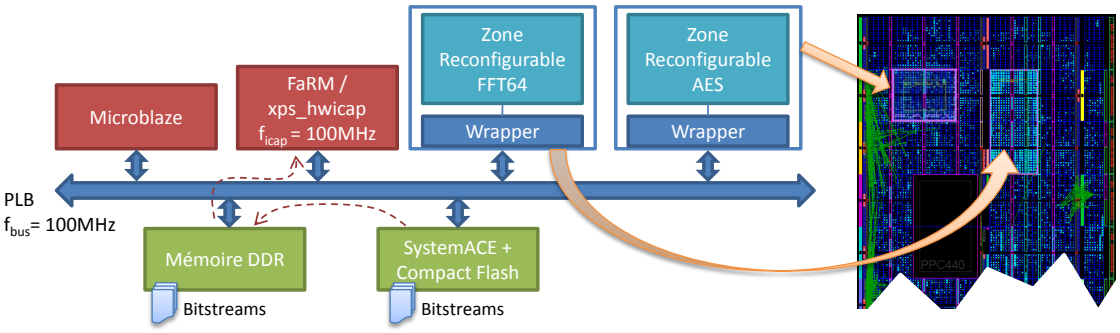


Figure 4.4 – *Architecture et application de test.*

Le *RPB* compatible AES dispose des ressources suivantes : 12 colonnes de CLBs et une colonne de BRAM, représentant un total de 590 trames. Il a été conçu pour être aussi proche que possible des ressources nécessaires aux tâches de l'application, minimisant la fragmentation interne. Chaque trame se compose de 41 mots de 32 bits. Cependant, la relecture correcte des colonnes BRAM est actuellement impossible donc pour les opérations de relecture, nous avons travaillé sur un sous-ensemble de *RPBs* comprenant 462 trames. Notez que le décodeur AES a besoin d'un peu plus de ressources que l'encodeur AES qui gaspille des ressources lorsque celui-ci est chargé dans le *RPB*. Toutefois, FaRM réalise une meilleure compression avec ce bitstream qui, par conséquent, compense ce gaspillage. Le *RPB* compatible FFT64 est composé de 2 colonnes de BRAM, 2 colonnes de DPS48 et de 16 colonnes de CLBs (952 trames).

4.3.4.1 Performances

Nous avons utilisé différents contrôleurs d'ICAP pour les tests, en commençant par le contrôleur de Xilinx et ses pilotes associés. Dans ce cas, le microprocesseur exécute toutes les opérations par lui-même. Nous avons un peu amélioré le contrôleur Xilinx en le couplant à un

Tableau 4.1 – *Temps de reconfiguration et de relecture d'un bitstream encodeur AES*

f_{bus} = 100MHz f_{icap} = 100MHz	xps_hwicap			xps_hwicap + DMA		
	Temps (μs)	BP COM (Mo/s)	BP ICAP (Mo/s)	Temps (μs)	BP COM (Mo/s)	BP ICAP (Mo/s)
Reconfiguration	72800	1,29	1,29	1550	60,5	**60,5**
Relecture trame/trame	159000	0,92	0,92	49200	2,97	2,97
Relecture globale	-	-	-	-	-	-
	FaRM			FaRM + O-RLE		
Reconfiguration	731	128	**128**	538	127	**174**
Relecture trame/trame	1536	47	95	1600	45	91
Relecture globale	560	129	129	560	129	129

DMA et en prenant soin des transferts de données. Enfin, nous avons utilisé notre contrôleur FaRM, avec et sans compression du bitstream.

Nos résultats sont présentés dans le tableau 4.1. Nous avons mesuré les temps nécessaires pour les opérations de lecture ou d'écriture et de calcul de la bande passante (BP) pour une reconfiguration, une relecture trame par trame et une relecture globale. La relecture trame par trame, beaucoup moins efficace que la relecture globale, a été testée avec FaRM afin de la comparer à IP Xilinx qui ne peut pas lire plus d'une trame à la fois. Les résultats du tableau 4.1 correspondent à l'encodeur AES présenté dont les caractéristiques du bitstream sont illustrées dans le tableau 4.2.

Nous avons testé l'overclocking de l'ICAP avec le mode de préchargement. Le bus PLB est configuré à 125 MHz tandis que l'ICAP a fonctionné à 200MHz sans aucune erreur. Les bitstreams utilisés étaient count1 et count2 du tableau 4.2. La reconfiguration a pris 8304 ns tandis que l'ICAP a besoin d'au moins 8290 ns (1658 cycles d'ICAP nécessaires pour écrire 6632 octets), ce qui signifie que nous avons atteint le débit maximum de 800 Mo/s. Pour le moment, nous n'avons pas pu dépasser la limite de 100MHz en mode relecture.

4.3.4.2 Compression de bitstreams

Le tableau 4.2 compare différents algorithmes de compression pour un ensemble de bitstreams. Nous avons testé chaque algorithme sur des IPs hétérogènes en terme de taille, de complexité et de ressources (CLB, BRAM, DSP48).

Avec un taux de compression moyen de 26,7% (en utilisant la moyenne géométrique pour lisser le résultat), l'algorithme de compression FaRM est légèrement plus efficace que les versions 16 bits et 32 bits de l'algorithme RLE canonique. Bien que nous ne pouvons pas rivaliser avec les algorithmes tels que ZIP et 7z (plus lourds en terme de latence et de ressources pour leurs implémentations sur FPGA), notre solution ne nécessite pas beaucoup de ressources pour la mise en œuvre du matériel de décompression et, de plus, a un débit d'un mot de 32 bits par cycle.

Tableau 4.2 – *Taux de compression en % pour différents algorithmes et bitstreams.*

IP	Taille (octet)	FaRM	RLE 16	RLE 32	ZIP	7z
décodeur AES	97676	23,4	21,6	20,9	57	60,8
encodeur AES	97676	31	31,6	26,6	67,6	69,7
RPB AES vierge	97676	90,8	93	88,1	97,3	97,5
Basic DES	24348	7	4,4	4,1	48,3	50,9
Basic RSA	68156	36,9	35,4	35,8	61,8	65,4
DCT	66844	41,1	40,5	38,8	65,3	67,9
FFT64	161308	32,9	31,3	32,2	57,3	60,8
FIR_7_16_8	17132	11,7	7,6	7,3	64,3	65
FIR_7_16_16	23036	12,8	10,3	7,5	65,3	67,1
Hibert	24348	19,9	18,7	14,8	65,4	66,7
Count 1	6632	51,2	48,9	44,8	78,3	78,5
Count 2	6632	51	48,8	44,6	78,2	78,3
Moyenne géométrique		**26,7**	**23,9**	**21,5**	**66,1**	**68,1**

4.4 Le système d'exploitation matériel FOSFOR

4.4.1 Introduction

Les nouvelles tendances dans la conception des systèmes embarqués temps réel dans divers domaines (télécommunications, automobile, multimédia, missions spatiales, etc) indiquent que les futures applications seront probablement implémentées sur des architectures reconfigurables hétérogènes incorporant des unités de traitement logiciels (processeurs) et matériels (zones reconfigurables). L'émergence du besoin de flexibilité dans ces systèmes avec la demande croissante de performance et les exigences de la réduction de la consommation d'énergie motivent la recherche de moyens pour gérer efficacement l'infrastructure en ligne. Ces nouveaux paramètres doivent être pris en compte dans un système d'exploitation pour faciliter le développement des nouvelles applications tout en minimisant la complexité de l'architecture. Etant donné que les architectures visées sont hétérogènes, les modules du système d'exploitation peuvent être partitionnés sur des zones reconfigurables et sur des processeurs comme nous l'avons déjà présenté dans ce manuscrit.

Une autre tendance est d'abstraire les différents systèmes d'exploitation en ajoutant une couche logicielle supplémentaire (intergiciel) facilitant le développement des applications et masquant les détails et les mécanismes du système d'exploitation. Dans le but d'abstraire l'architecture et d'assurer la communication entre les services logiciels/matériels du système d'exploitation, l'orientation est d'établir une couche mixte matérielle/logicielle permettant la virtualisation des unités d'exécution. Cette couche produit une interface uniforme vis-à-vis des services logiciels du système d'exploitation et gère l'accès aux ressources matérielles de façon transparente. Effectivement, elle emploie divers mécanismes qui facilitent l'exécution

des services du système d'exploitation sans connaître les détails des supports d'exécution matériels.

L'application proposée est représentative des applications de suivi de cibles développées dans notre partenaire TRT. Il s'agit de détecter, pister et reconnaître les cibles dans une vidéo en infrarouge ayant une résolution spatiale de 640 × 512 pixels dans chaque trame et une résolution temporelle de 25Hz (fps). Cette application a été sélectionnée pour ses caractéristiques de dynamicité qui permettront de valider la capacité de l'OS proposé à gérer la reconfiguration dynamique partielle dans le respect des contraintes temps réel.

Notre contribution dans ce projet concerne la conception d'une partie de l'OS FOSFOR, à savoir, les services de l'OS sauf l'intergiciel. Nous sommes également en charge de réfléchir à des méthodes d'ordonnancement/placement de tâches matérielles sur l'architecture reconfigurable, traitées dans le chapitre 3.

4.4.2 L'architecture FOSFOR

L'objectif du projet FOSFOR est de présenter une approche pour concevoir un système d'exploitation reconfigurable distribué (matériel/logiciel), un intergiciel et une couche d'abstraction/communication pour faciliter l'implémentation de ces systèmes embarqués et pour améliorer la gestion de leurs ressources. La figure 4.5 présente les couches de l'architecture multi-niveaux :

- **Processeur (nœud logiciel) :** Son rôle est d'exécuter une partie des tâches logicielles de l'application et devra supporter certains services de l'OS et la couche d'intergiciel.
- **Zone reconfigurable (nœud matériel) :** Son rôle est d'exécuter des tâches matérielles. Ces zones reconfigurables supportent aussi certains services de l'OS.
- **Couche d'abstraction/communication :** Cette couche mixte matérielle/logicielle assure la communication entre les services matériels et logiciels de l'OS (HAL [13]) par le transfert de commandes et de données. Elle garantit aussi la virtualisation des supports d'exécution en offrant une interface uniforme vis-à-vis des services logiciels de l'OS ; elle emploie des mécanismes efficaces permettant de gérer les ressources matérielles.
- **Système d'exploitation (OS) :** L'OS a une structure modulaire répartie entre les unités d'exécution matérielles et les unités d'exécution logicielles.
- **Intergiciel :** Cette couche d'abstraction logicielle, fournit une interface de haut niveau aux applications, masque l'hétérogénéité des modules des OS sous-jacents.
- **Application :** L'application est constituée d'un ensemble de tâches et d'un ensemble de contraintes de temps réel et/ou qualité de services.

Chaque processeur possède sa propre mémoire locale. Cette mémoire stocke les données locales et, le cas échéant, le code logiciel. Une mémoire partagée, reliée sur le bus de données, rend possible le partage des données entre les threads (instances de tâches) partitionnés sur différents processeurs. Chaque unité d'exécution peut atteindre cette mémoire partagée, contenant les ressources d'exécution logiciel des programmes et des données. Chaque ressource

13. Hardware Abstraction Layer

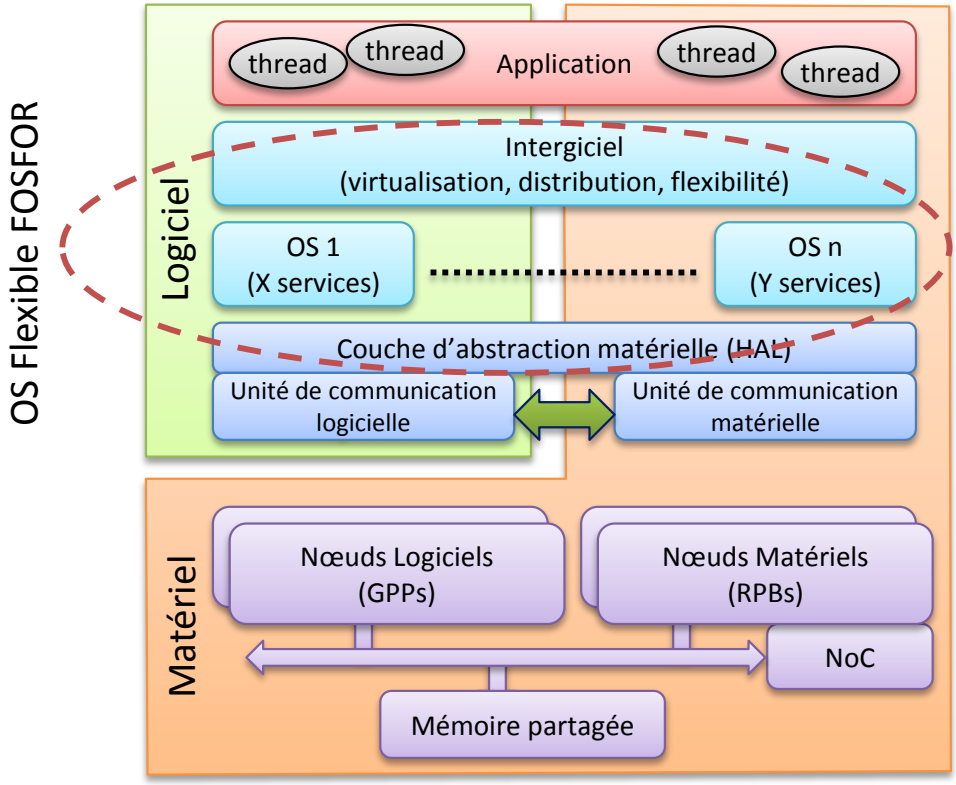


Figure 4.5 – *Structure de l'architecture générique FOSFOR.*

peut également atteindre une mémoire de configuration, afin d'enregistrer et de restaurer son contexte d'exécution. Cette structure générique donne toute la liberté possible pour mettre en œuvre n'importe quels threads et services sur n'importe quelle ressource d'exécution. La RR est divisée en deux parties principales : l'une statique (SR), l'autre dynamique (DR).

L'objectif de la région statique concerne les besoins de communication entre les threads matériels. Cette zone peut également contenir des services de l'OS. De même, la région dynamique doit supporter la dynamicité de la manière la plus flexible. Ces zones dynamiques doivent en outre être librement connectées à la région statique pour assurer la communication des threads avec l'extérieur de la région reconfigurable. C'est la raison pour laquelle nous proposons d'utiliser un nouvel intergiciel FOSFOR qui abstrait les communications implicites.

Une RR est organisée selon le modèle défini dans la figure 4.6. Deux régions statiques entourent la région dynamique. L'une implémente le système d'exploitation FOSFOR matériel et les services de l'intergiciel, l'autre le NoC dédié aux flux de données des communications entre les threads matériels. La plateforme FOSFOR contient des composants FPGA reconfigurables dynamiquement qui peuvent déjà supporter ce modèle d'architecture. Nous avons choisi la technologie Virtex5 pour leur capacité à reconfigurer des régions rectangulaires ($RPBs$). Des algorithmes d'ordonnancement/placement qui permettent une utilisation efficace des éléments du FPGA (LUT [14], registres, mémoire distribuée, IO, etc.) à l'intérieur de chaque RR en fonction des besoins de l'application ont été étudiés dans le chapitre 3.

14. Look-Up Table

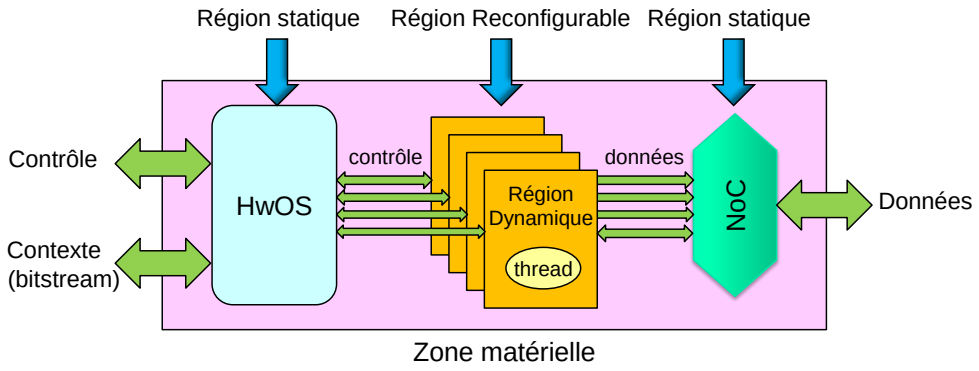


Figure 4.6 – *Structure de l'architecture générique FOSFOR.*

4.4.3 La conception de l'OS FOSFOR matériel

Comme nous l'avons expliqué dans l'introduction, l'une de nos contributions concerne, en partie, la conception de l'OS matériel FOSFOR. Nous rappelons que l'objectif est de gérer les tâches logicielles et matérielles de la même manière. Ainsi, nous nous sommes basés sur RTEMS [134], qui a l'avantage de supporter les architectures multiprocesseurs et d'être un système open-source, obligatoire pour concevoir l'intergiciel coté logiciel et de comprendre l'architecture de RTEMS afin de concevoir l'OS matériel compatible RTEMS.

De plus, ce choix de l'OS RTEMS a notamment été guidé par son mode de communication entre les diverses unités d'exécutions ou nœuds. Assurée par un ensemble de services regroupés sous le nom de MPCI, cette communication inter-nœuds a l'avantage d'être transparente pour les tâches exécutées uniquement localement. En effet, chaque nœud peut définir des objets (sémaphores, tâches, boîtes aux lettres, ...) comme étant globaux, ce qui signifie que ceux-ci peuvent être manipulés par un autre nœud. L'objectif est de concevoir en matériel la partie communication de RTEMS, de manière à ce que l'OS matériel soit capable de s'interfacer avec une ou plusieurs instances logicielles de RTEMS (nœuds logiciels). On choisit comme nœud logiciel le processeur LEON3, instancié autour d'un bus AHB, disponible également en open source au sein de la librairie GRLIB [135].

Tout d'abord, et comme le montre la figure 4.7, il est nécessaire de définir un mode de communication entre le MPCI matériel (HwMPCI [15]) et les autres services du HwOS. Chaque service du HwOS dispose d'une mémoire tampon segmentée en blocs de la taille d'un message, dans laquelle seront stockés les différents messages reçus ou à émettre. Lors de la réception d'un message, et après l'avoir stocké dans la mémoire tampon, le HwMPCI avertit le service destinataire de la présence d'un message reçu à traiter en postant une donnée dans une file fonctionnant selon le comportement d'une FIFO. Cette donnée contiendra différents paramètres tels que le numéro du bloc dans lequel le message a été stocké. De la même manière, les services préviendront le HwMPCI de la présence d'un message à émettre en postant une donnée dans la file des tâches du HwMPCI. Il est à noter que les files de réception des services du HwOS peuvent également être utilisées par les tâches locales. Si les services de l'OS FOSFOR peuvent être implémentés de manière concurrente afin de tirer partie de la structure matérielle, les services du HwMPCI sont dépendants de l'accès au bus. Pour cette

15. Hardware MPCI

raison, un seul service HwMPCI pourra être exécuté à la fois.

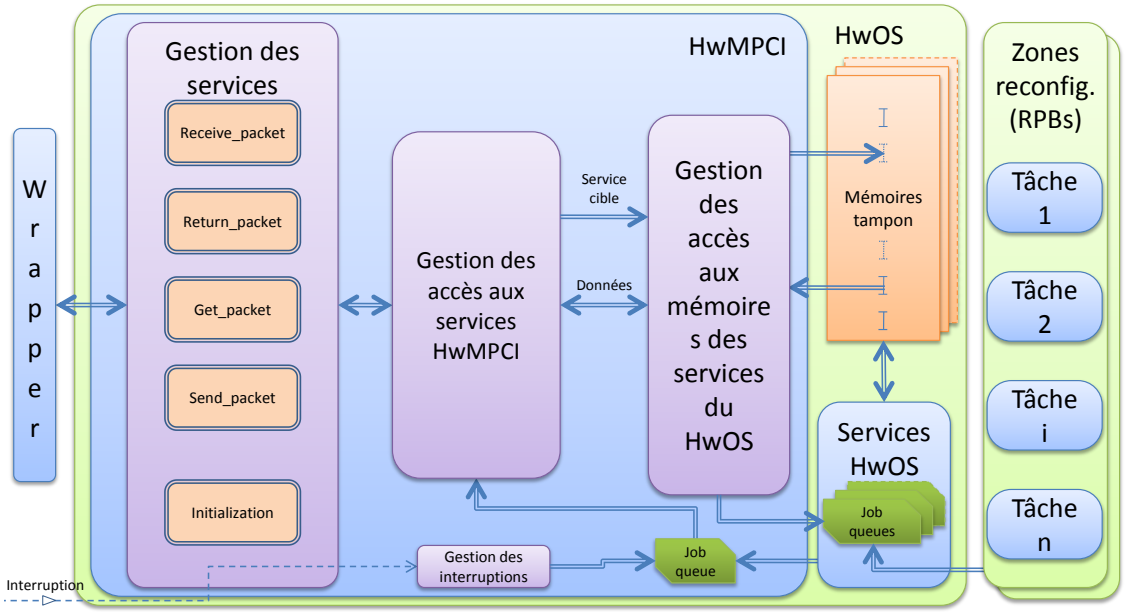


Figure 4.7 – *Architecture simplifiée de l'OS matériel FOSFOR.*

Le cœur du système d'exploitation réside dans les différents services qu'il offre aux tâches. Nous avons implémenté le service sémaphore (booléen, compteur et exclusion mutuelle), le service message queue avec quelques restrictions sur la taille des messages et l'ordonnanceur simplifié. L'OS matériel est connecté aux tâches matérielles par l'intermédiaire d'un bus que nous avons défini avec notre partenaire TRT, responsable de l'organisation et de la conception des tâches matérielles. Ces tâches matérielles peuvent communiquer en local ou bien communiquer avec des tâches logicielles ou matérielles situées sur un autre nœud logiciel ou matériel par l'intermédiaire du HwMPCI.

4.4.4 Résultats

Les principales raisons de la conception d'un tel système d'exploitation matériel sont les performances et la flexibilité. L'OS pourrait être entièrement logiciel ou entièrement matériel. Comme chaque appel à une primitive OS provoque un overhead et signifie, par conséquent, un temps d'attente pour le thread en cours d'exécution, plus l'OS est rapide, plus le temps perdu par les threads diminue. Afin d'évaluer cet overhead, il faut comparer les temps de l'OS matériel avec l'OS RTEMS logiciel original.

Les services matériels locaux ont seulement besoin de quelques dizaines de cycles, tandis que les services matériels globaux exigent quelques centaines de cycles en raison de l'accès à la mémoire partagée. Nous avons évalué une accélération d'environ 60 sur la création locale et les services de suppression, et d'environ 50 pour les autres services, par rapport au système d'exploitation RTEMS.

L'utilisation des ressources de l'HwOS (tableau 4.3) varie considérablement, selon le nombre et les capacités des services activés. Par exemple, nous sélectionnons le nombre d'objets (les sémaphores, les threads, ...) pour chaque service. Nous utilisons un FGPA Xilinx

Virtex5 FX100T pour implémenter le système. Le tableau répertorie les ressources utilisées par le HwOS. L'espace restant permet la mise en œuvre des autres composants du système (partie statique) et les threads matériels eux-mêmes (zones reconfigurables ou $RPBs$). Ces évaluations ont été faites avec notre partenaire TRT.

Tableau 4.3 – *Utilisation des ressources de l'OS matériel FOSFOR sur Virtex5-FX100.*

Nb. d'objets par service	8	16	32
Slices CLBs	2408 (15%)	3151 (20%)	4327 (27%)
DFF	5498 (8,5%)	6650 (10,4%)	8918 (13,9%)
BRAMs	8 (3,5%)	16 (7%)	32 (14%)

4.5 Plateformes HPRC et auto-adaptatives

4.5.1 Introduction

L'architecture actuelle des structures d'exécution parallèles, notamment les HPRCs, avec le matériel reconfigurable vu comme une unité d'appoint pour aider le logiciel souffre, à nos yeux, d'un héritage trop lourd. En effet, jusqu'à récemment, une plateforme ne pouvait exécuter des applications différentes qu'en ayant recours au logiciel. Et si parfois certains IPs permettent d'accélérer une partie de l'application, comme par exemple un décodeur MPEG2, il est impossible de disposer de tous les IPs possibles et futurs de manière statique. Logiquement, les architectures HPRC vont donc dans la continuité des architectures passées, en ajoutant un soupçon de matériel, mais sans remettre en cause le modèle actuel dans lequel le logiciel est moteur.

Pourtant, et maintenant que le matériel n'est plus figé comme auparavant, nous commençons à disposer de moyens importants pour changer le rapport de force entre le logiciel et le matériel. En effet, à nos yeux, le matériel ne doit plus être vu comme une aide ponctuelle au logiciel, mais que la totalité de l'application doit être conçue dans une logique de codesign logiciel/matériel en ligne. Cette notion de codesign en ligne permet de faire le choix d'une implémentation logicielle ou matérielle pour chaque élément d'une application, à la fois lors de sa conception, mais également dynamiquement en fonction des ressources disponibles, tant sur les processeurs que sur les zones reconfigurables. Ainsi, notre approche est considérée comme flexible et non dédiée.

4.5.2 L'architecture proposée

Pour cette raison, nous proposons une architecture reconfigurable dans laquelle la distinction entre PEs logiciels et matériels n'a que peu d'importance, ces PEs correspondant à des cellules. Celles-ci pourraient être comparées aux SANEs du projet ÆTHER présenté au chapitre 2. La figure 4.8 présente une vue simplifiée du système.

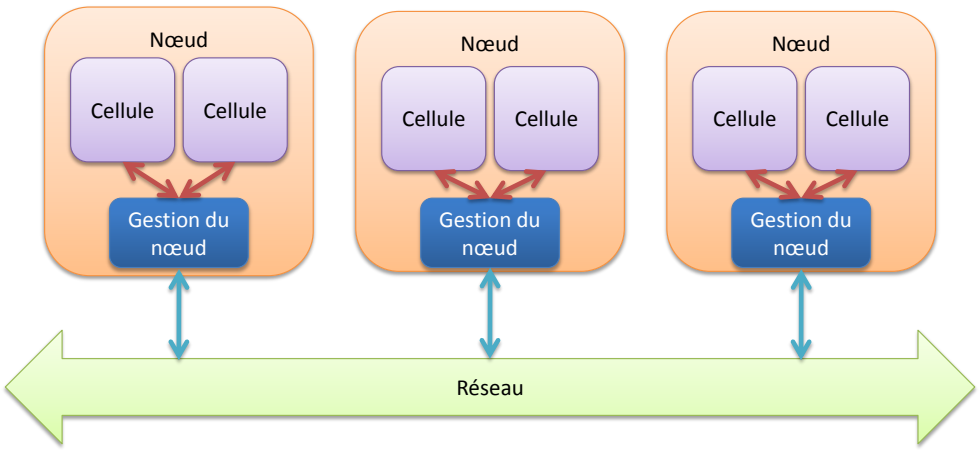


Figure 4.8 – *Vue du système.*

Dans cette architecture, une application est découpée en diverses tâches séquentielles et/ou concurrentes. Chaque tâche peut être conçue logiciellement, matériellement, ou disposer des deux types d'implémentations. De la même manière, une tâche peut disposer de plusieurs implémentations matérielles utilisant des ressources différentes. Ainsi, une même tâche peut avoir une implémentation utilisant des DSP48s, tandis qu'une autre pourra utiliser une zone du FPGA ne comportant pas de telles ressources, en remplaçant les DSP48s par leurs circuits logiques équivalents au détriment des performances.

Notre objectif est de proposer une méthodologie de développement pouvant s'appliquer à la plupart des programmes parallèles, ainsi qu'une plateforme pouvant accueillir ces programmes, tout en conservant une compatibilité maximale avec des programmes déjà existants.

Notre architecture est distribuée, comme sur les HPCs actuels, sur différents nœuds, chacun contenant plusieurs cellules reconfigurables. Les cellules pouvant contenir indifféremment un PE logiciel ou matériel. Notre architecture s'apparente au type NNUS [16].

Cette plateforme permet de « virtualiser », ou tout au moins de faire abstraction du matériel sous-jacent, en proposant un modèle unifié sur lequel les applications bâties en utilisant notre flot pourront s'exécuter. Les catégories d'applications pouvant tirer parties d'une telle plateforme sont celles pouvant nécessiter de l'auto-adaptation et/ou du parallélisme. Concernant le parallélisme, la plupart des applications scientifiques fonctionnant sur HPC gagneraient à déporter une partie de leurs calculs intensifs sur du matériel, comme le montre déjà la tendance HPRC. Notre approche permet d'aller plus loin en concevant l'application dès l'origine pour une architecture où le matériel est présent et flexible. Concernant l'auto-adaptativité, de nombreuses applications de traitement du signal en temps réel ont des besoins changeants en fonction du temps et doivent par exemple s'adapter aux conditions extérieures. Dans ce cas, une telle plateforme leur permettrait de réguler leurs besoins en puissance de calcul en fonction des PEs disponibles, et même de modifier le type de PE en fonction des besoins.

16. Nonuniform Node, Uniform System

4.5.3 Flot de conception et modèle

Notre flot de conception, présenté sur la figure 4.9, débute par une description de l'application de type graphe (étape A.1). À cette fin, l'application est segmentée en noyaux applicatifs (étape A.2), un noyau étant un élément de l'application réalisant une tâche de calcul précise, par exemple une transformée de Fourier discrète. Pour une nouvelle application, le choix de la segmentation des noyaux peut être fait selon des critères de disponibilité des IPs. Par exemple, pour une application de transcodage vidéo, il est possible d'utiliser un IP de décodage avec un certain format d'entrée suivie d'un IP de codage générant un autre format en sortie. Dans le cas d'une application logicielle existante, on peut l'analyser en vue de la découper selon les différentes routines déjà existantes, pour ensuite en déplacer certaines vers des IPs matériels correspondants au même traitement.

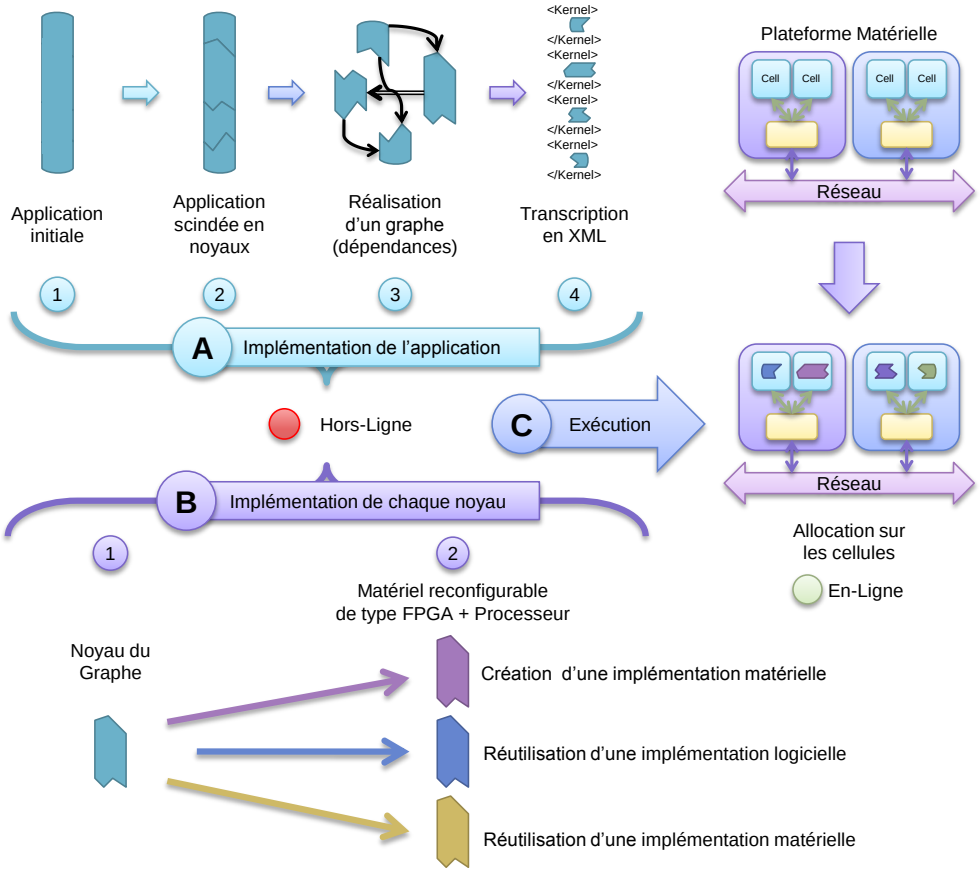


Figure 4.9 – *Flot de conception.*

Une fois les noyaux isolés, on détermine les dépendances de données entre ces noyaux (étape A.3). On peut alors bâtir un graphe, indépendant du contenu des noyaux, permettant de visualiser les dépendances internes de l'application. Ceci permet d'identifier quels noyaux pourront être exécutés simultanément. Ce graphe est également utile pour identifier les échanges intensifs de données entre les noyaux, et ainsi détecter lesquels doivent bénéficier d'un placement minimisant le temps de communication entre eux. Le graphe étant une représentation abstraite de l'application, on se basera sur une description XML pour transformer celui-ci en un ensemble de données compréhensibles par un ordonnanceur (étape A.4).

L'étape B présente la mise en œuvre des noyaux de l'application (étape B.1) qui peuvent être implémentés ou réutilisés (étape B.2). Pour finir, dans l'étape C, l'application est exécutée sur la plateforme et les noyaux sont dynamiquement instanciés sur les ressources matérielles et logicielles.

La définition du modèle utilisant les graphes est fonctionnelle, dans le sens où elle permet de modéliser plusieurs types d'applications et plusieurs modèles de parallélisme. Par exemple, elle permet de modéliser aussi bien des applications utilisant le standard MPI [17] qu'OpenMP [18], bien que leur comportement diffère. En effet, MPI utilise un modèle dans lequel les threads existent du début à la fin d'une application, tandis qu'OpenMP utilise un modèle fork/join dans lequel les threads sont créés en fonction des besoins puis détruits lorsqu'ils deviennent inutiles. Notre modèle utilise pour l'instant une représentation non standard, mais notre objectif est de le faire évoluer de manière à converger vers des graphes déjà existants, tels les automates finis parallèles [136]. Il devrait ainsi être possible de définir notre modèle en tant qu'extension d'un modèle déjà existant.

4.5.4 Nos plateformes et son évolution

Nos différentes plateformes prennent pour base l'architecture générale des HPCs, c'est-à-dire la division en nœuds contenant chacun plusieurs PEs, que nous appellerons ici cellules. La division en nœuds résulte en une architecture mémoire globalement distribuée – localement partagée. En effet, chaque nœud possède sa propre mémoire, celle-ci étant partagée entre les différentes cellules.

Afin de disposer d'une architecture reconfigurable, on implémente celle-ci sur une cible de type FPGA. Notre plateforme finale devra respecter les différents éléments déjà détaillés dans la section 4.5.2. À cette fin, nous détaillons la plateforme actuelle et les évolutions qui y seront apportées.

Une première plateforme, *Software HPC Platform*, a permis de valider le modèle général et son adéquation à la structure HPC, tout en pointant du doigt certains problèmes inhérents à la méthode d'utilisation de la mémoire. La seconde plateforme, *Hardware Stream Dynamic Platform* permet de prouver la viabilité de l'utilisation de la PR dans une architecture orientée flot de données. Enfin, une future évolution, *Hybrid HPRC&Stream Platform*, devrait apporter une réponse complète en rassemblant les éléments des deux autres. Un résumé des caractéristiques des plateformes est présenté dans le tableau 4.4.

4.5.5 Architecture et performance de la *Software HPC Platform*

Dans cette plateforme, les cellules de calcul sont toutes composées d'un PE logiciel statique basé sur un processeur de type MicroBlaze. En effet, cette plateforme préliminaire, présentée en [171], a servi à valider l'architecture globale de la plateforme. Une vue du système avec le détail d'un nœud est présentée sur la figure 4.10.

Afin de garantir une compatibilité avec les applications actuelles, cette plateforme utilise le standard industriel MPI via la librairie open-source MPICH2. Les communications inter-

17. Message Passing Interface
18. Open Multi-Processing

Tableau 4.4 – *Caractéristiques des différentes plateformes.*

Nom de la plateforme	Dynamique	Communication via MPI	Type des IPs	Communications intra-nœud
Software HPC Platform	Non	Oui	Logiciels	Mémoire partagée
Hardware Stream Dynamic Platform	Oui	Non	Matériels	Mémoire partagée + bus
Hybrid HPRC& Stream Platform	Oui	Oui	Logiciels et Matériels	NOC

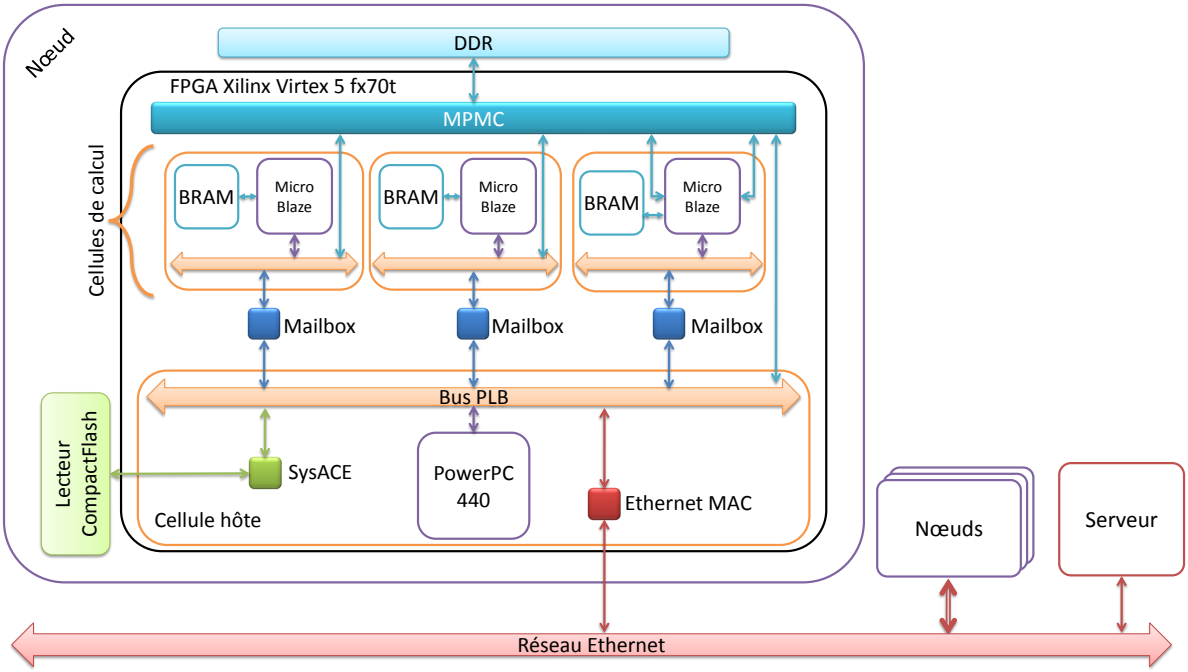


Figure 4.10 – *Software HPC platform avec détail sur un nœud.*

nœuds sont ainsi gérées par la cellule hôte, qui héberge un OS linux embarqué s'exécutant sur un PowerPC440. Les communications intra-nœud sont assurées par le biais de la mémoire partagée. Ainsi, les deux types de communication sont gérées à l'aide de routines MPI.

Le système complet étant composé de plusieurs nœuds, il est nécessaire de disposer d'un point d'entrée pour lancer les calculs. Pour cela, on ajoute au système un nœud spécifique, que l'on appelle serveur, et dont le rôle se résume à lancer les calculs sur les nœuds. Ce rôle peut être rempli par n'importe quelle machine disposant de MPICH2. Dans notre cas, il s'agit d'un poste de travail de type x86_64.

Nous avons testé la plateforme à l'aide de l'application Integer Sort (IS) du benchmark NPB [19] [137]. La plateforme est testée selon les critères suivants :

- la charge totale de calcul, représentée par la taille du bench, ou classe dans la terminologie NPB,

19. NAS Parallel Benchmark

- le degré de parallélisation, représenté par le nombre total de cellules de calcul participant au bench,
- la charge locale de calcul, représentée par le nombre de cellules de calcul utilisées sur chaque nœud,
- la présence ou non de cache d'instruction dans les processeurs des cellules de calcul.

La figure 4.11 présente le temps d'exécution du bench selon diverses configurations, tandis que la figure 4.12 se focalise sur le temps passé par les différentes cellules à communiquer. Les résultats présentés sur les figures concernent des benchmarks de classe A.

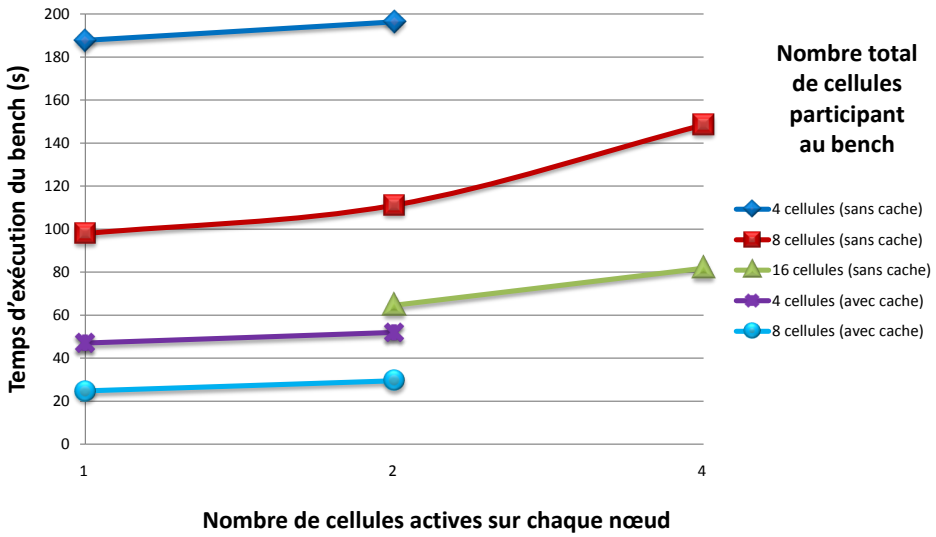


Figure 4.11 – *Résultats obtenus pour le temps total d'exécution du bench NPB IS avec différentes configurations de la plateforme.*

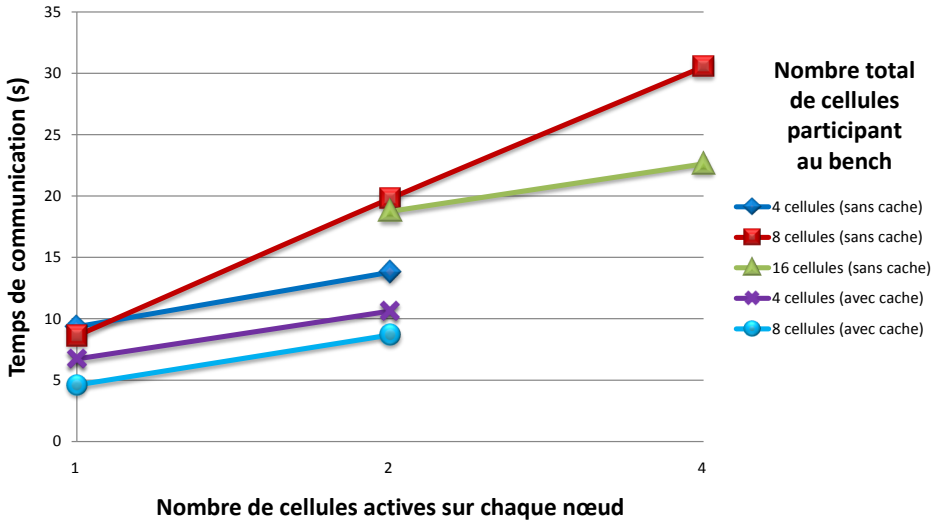


Figure 4.12 – *Résultats obtenus pour le temps de communication du bench NPB IS avec différentes configurations de la plateforme.*

On constate sur les graphiques, et principalement sur celui présentant le temps de communication, à nombre constant de cellules, le temps présenté en ordonnée croit lorsque l'on rassemble les cellules participant au bench sur les mêmes nœuds. En effet, la mémoire partagée est utilisée par toutes les cellules simultanément, ainsi que pour les communications intra-nœud, et se comporte comme un goulot d'étrangement, limitant sévèrement les performances. Ainsi, notre approche globale est confirmée par l'obtention d'une plateforme fonctionnelle malgré la mise en lumière de certains problèmes inhérents au mécanisme de communication intra-nœud.

4.5.6 Architecture et performance de la *Hardware Stream Dynamic Platform*

Sur cette plateforme, nous nous concentrons sur l'aspect dynamique des IPs. À cet effet, cette architecture repose sur des cellules purement matérielles, basée sur des IPs existants. La figure 4.13 présente le système avec une configuration comportant deux nœuds, chacun pourvu de trois cellules de calcul.

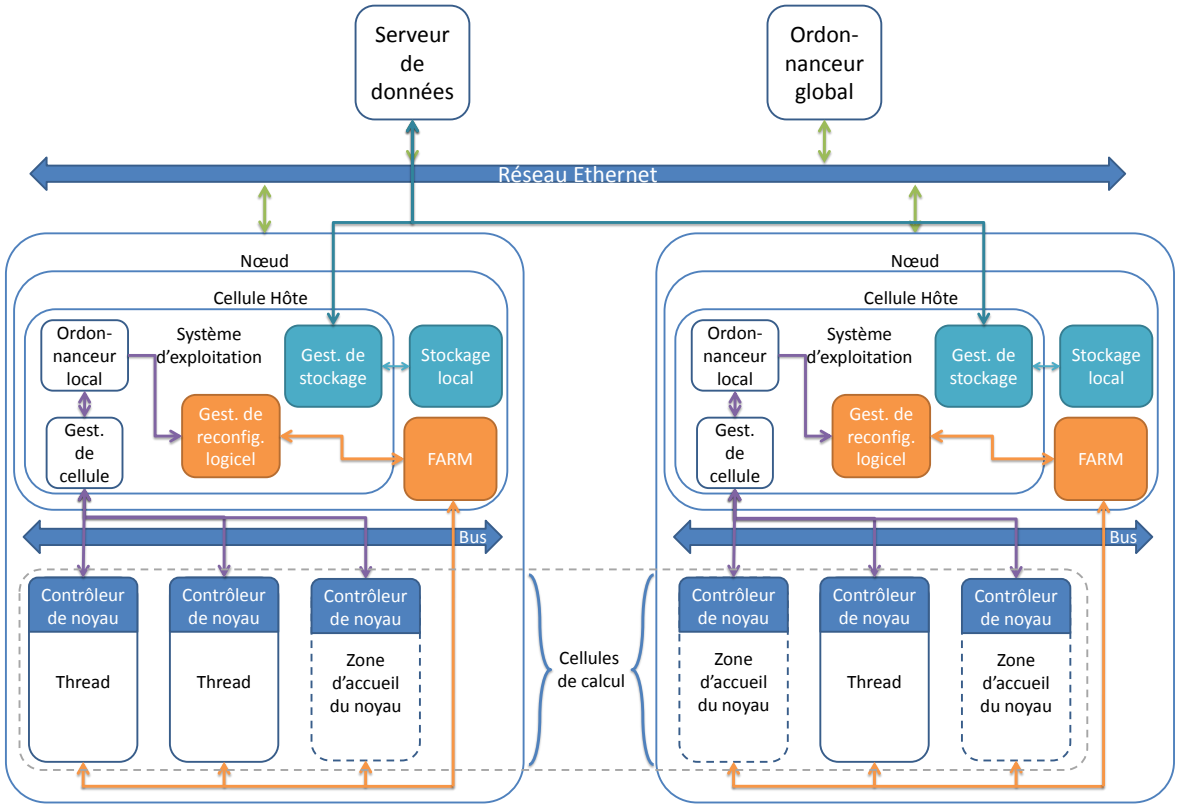


Figure 4.13 – *Vue de la Hardware Stream Dynamic Platform avec deux noeuds.*

Elle ne dispose pas du standard MPI, qui n'est pas réellement nécessaire dans cette version dans la mesure où les IPs utilisés disposent de leur propre manière de récupérer et fournir des données, différente de MPI. Cette plateforme est donc moins orientée HPC, et plus flot de données. Elle permet l'exécution d'applications auto-adaptatives par la présence d'un

ordonnanceur local répartissant dynamiquement les instances des noyaux, ou threads, sur les cellules.

Afin d'éviter de reproduire les problèmes de la précédente plateforme, les transferts de données peuvent cette fois emprunter le bus inter-cellules et ne reposent donc plus uniquement sur la mémoire partagée. Ici, le simple serveur de la plateforme précédente est remplacé par un ordonnanceur global, décidant de la répartition des tâches sur les nœuds. La nouveauté est que cet ordonnanceur global est secondé par un ordonnanceur local, qui décide de la répartition des tâches au sein du nœud.

De plus, un serveur de données est ajouté, qui permet de distribuer les différents éléments de l'application, permettant ainsi un fonctionnement indépendant des nœuds. En effet, le serveur de données évite un stockage préalable des éléments de l'application sur chaque nœud, ces éléments étant récupérés dynamiquement sur le serveur de données.

Contrairement à la *Software HPC Platform*, dont toutes les cellules étaient homogènes, la *Hardware Stream Dynamic Platform* dispose de cellules dont les ressources peuvent différer. Ainsi, à la répartition figée de la plateforme statique succède une répartition dynamique prenant en compte les spécificités des cellules pour décider de la localisation d'un thread.

Concernant les cellules matérielles, nous devons gérer une dimension supplémentaire, qui se trouve être la reconfiguration partielle des cellules de calcul. Celle-ci s'appuie sur le gestionnaire de reconfiguration, qui épaule l'ordonnanceur local en s'occupant de réaliser les ordres de reconfigurations. Ce gestionnaire de reconfiguration est composé de deux parties. La partie logicielle s'occupe de récupérer le bitstream partiel sur le serveur de fichiers, et de le stocker localement. Ainsi, à la différence de Crenne [118], nous stockons localement les bitstreams de manière à ce qu'ils soient déjà disponibles au moment où la reconfiguration doit s'effectuer. Afin de gérer l'encombrement local, les bitstreams sont remplacés à l'aide d'une politique comparable au LRU [20] utilisée par les contrôleurs de mémoires caches.

L'autre partie du gestionnaire de reconfiguration est matérielle et utilise l'IP FaRM présenté précédemment. Un driver Linux permet de commander l'IP FaRM à partir du gestionnaire de reconfiguration logiciel, transformant les commandes de reconfiguration en chargement de bitstreams partiels.

Cette plateforme a été testée à l'aide d'une chaîne de codage/décodage AES. Nous écrivons un descripteur de travail (job descriptor) contenant l'application, à savoir un encodeur AES suivi d'un décodeur AES. Le comportement et les caractéristiques des IPs sont contrôlés par des descripteurs décrits en XML.

La mise en œuvre de l'encodeur AES consomme 2430 LUTs alors que le décodeur AES utilise 2975 LUTs. Cette plateforme, dont une implémentation est présentée sur la figure 4.14, est fonctionnelle du point de vue local, c'est à dire que chaque nœud peut fonctionner indépendamment si nous lui communiquons la description de l'application à exécuter.

L'ordonnanceur global aura pour tâche de découper une application en plusieurs éléments qui seront distribués sur les nœuds, avec pour objectif de minimiser les échanges de données inter-nœud, afin d'éviter les communications lentes. Ces travaux d'ordonnancement global sont actuellement en cours.

Pour finir, l'application a été testée en utilisant notre technique de compression de bit-

20. Least Recently Used

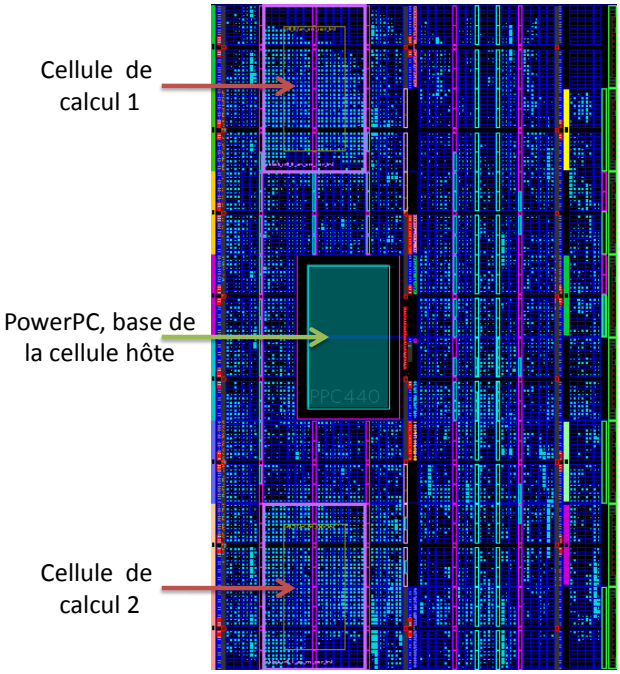


Figure 4.14 – *Implémentation d'un nœud de la Hardware Stream Dynamic Platform contenant deux cellules sur Virtex 5 FX70t.*

stream disponible dans FaRM. Les temps de configuration ont été mesurés et sont présentés dans le tableau 4.5.

Tableau 4.5 – *Temps de reconfiguration exprimés en cycles d'horloge.*

Compression	Encodeur	Décodeur
oui	79 633	91 232
non	103 295	103 287

4.6 Conclusion

Ce chapitre a abordé la problématique du reconfigurable au sein des systèmes d'exploitation : le HwOS centralisé, l'OS FOSFOR et l'approche HPRC et auto-adaptative en utilisant l'OS Linux. Ces contributions sur les architectures reconfigurables dynamiques et partielles sont possibles grâce à la technologie actuelle qui est capable de se reconfigurer partiellement.

La première contribution de ce chapitre a proposé de réduire les temps de reconfiguration de ces zones reconfigurables à l'aide d'un IP FaRM qui combine plusieurs techniques comme la compression O-RLE, un contrôleur interne sophistiqué. Ceci a permis d'obtenir une vitesse de reconfiguration très importante par rapport à l'IP de Xilinx, allant jusqu'à 800Mo/s pour FaRM. De plus, nous proposons également un modèle de reconfiguration précis pour le calcul du temps de reconfiguration qui peut être utilisé pour alimenter notre simulateur SystemC et développer des stratégies de placement et d'ordonnancement.

Les deux contributions suivantes ont eu comme objectif de concevoir des systèmes d'exploitation en matériel. Dans le premier cas, l'OS multiprocesseur matériel est centralisé et ne gère pas les tâches matérielles. Quant à l'OS FOSFOR, son objectif est justement d'avoir une approche d'OS multiprocesseur distribué et de pouvoir gérer des tâches matérielles comme des tâches logicielles avec un OS logiciel (RTEMS), un OS matériel (Compatible RTEMS) et une couche d'intergiciel qui fournit des services de haut-niveau aux tâches.

La dernière contribution propose une méthodologie pour les systèmes HPRC et auto-adaptatif. Nous avons détaillé les différents éléments de notre méthodologie destinée à l'implémentation d'applications parallèles. Nous avons étudié le flot de conception de l'application, permettant nativement un codesign logiciel–matériel en ligne des applications. De plus, nous avons vu que ce flot pouvait facilement s'appliquer aux applications existantes désirant profiter de notre méthodologie. Nous avons également abordé les différentes plateformes que nous avons développées pour permettre d'accueillir ces mêmes applications.

Cet axe de recherche a été possible grâce aux projets ANR FOSFOR et ARDMAHN, et à l'aide de deux doctorants qui sont actuellement en 3^{e} année de thèse, et également de stagiaires. Ces travaux ont conduit à 3 revues internationales [5, 6, 8], 2 revues internationales en première soumission [11, 10], 1 revue nationale en première soumission [188], 6 conférences internationales [168, 166, 171, 173, 185, 187], et 5 conférences nationales [189, 190, 192, 194, 199].

CHAPITRE 5

Conclusions et Perspectives

5.1 Bilan

Depuis une trentaine d'années, les circuits intégrés, la technologie et les applications ont considérablement évolué. En 1996, lors de mon DEA, je réalisais mes premiers travaux sur microcontrôleur 68332 et circuit programmable xc4010 de la société Xilinx. Quinze ans plus tard, je travaille sur des architectures comportant un ou plusieurs SoPC intégrant de nombreuses portes logiques, de la mémoire, des processeurs, des blocs plus complexes comme des modules Ethernet, PCI-Express, etc. De plus, ces architectures SoPC sont capables de supporter des systèmes d'exploitation embarqués bien plus lourd tels que Linux. C'est à partir de ce constat que j'ai défini un sous-thème traitant de ces problématiques selon 3 axes présentés dans le manuscrit.

Les applications exécutées sur des architectures reconfigurables demandent toujours plus de QoS qui deviennent encore moins évidentes à maîtriser et à garantir, due à la dimension supplémentaire du reconfigurable. Finalement, nous avons une architecture plus flexible, plus dynamique, avec plus de possibilités et de solutions architecturales qui engendre des décisions plus complexes à prendre en ligne. D'une manière générale, la QoS est tributaire de ces décisions en ligne qui peuvent être des modifications : de l'ordonnancement, de la répartition de l'application, du réseaux d'interconnexion et même de l'architecture en elle-même. Si nous ajoutons à ceci que certaines applications demandent des contraintes temps réel dur, cela complique encore plus la prise de décision. Par conséquent, cela remet en cause, dans une certaine mesure, le flot de conception classique, c'est-à-dire les méthodes et les outils hors-lignes de conception et de vérification du couple application/architecture. Il faut donc rechercher de nouveaux moyens pour répondre à cette problématique.

L'architecture reconfigurable évolue et les systèmes d'exploitation doivent également prendre en compte ce changement de cap important. Dans nos contributions, nous essayons également d'y répondre en ajoutant la dimension reconfigurable dans les architectures actuelles distribuées et réparties. Nous avons évalué des solutions pour des systèmes d'exploitation

temps réel qui doivent gérer aussi bien les tâches logicielles que matérielles de manière la plus égalitaire possible. De plus, cette évolution du reconfigurable a donné naissance aux architectures HRPC qui ont l'avantage de prendre en considération cette dimension de flexibilité de l'architecture, mais ainsi pose le problème du modèle de programmation associé (FPGA, GPU et CPU).

D'une façon plus générale, ses contributions répondent à ces problématiques d'application temps réel, de qualité de service, de placement et d'ordonnancement de tâches matérielles et, bien sûr, des systèmes d'exploitation sur des architectures reconfigurables, distribuées et réparties. Ceci nous amène à développer des perspectives de recherche pour envisager de nouveaux travaux dans ce domaine très large des systèmes reconfigurables.

5.2 Perspectives de recherche

L'évolution des technologies, des besoins applicatifs créés par l'homme conduisent à concevoir des systèmes de plus en plus complexes et qui doivent interagir, s'adapter, évoluer rapidement tout en respectant le cahier des charges. Ce dernier est devenu plus compliqué, avec des contraintes fortes comme par exemple des performances de calcul à atteindre pour une consommation d'énergie à ne pas dépasser. Afin de répondre à ces contraintes fonctionnelles et non fonctionnelles, je propose des perspectives non exhaustives conditionnées, en partie, à mes projets à venir.

5.2.1 Processus de conception et modélisation

Depuis de nombreuses années, l'effort s'est principalement focalisé sur des flots en Y afin de concevoir des systèmes embarqués, de complexité croissante, à partir de descriptions de plus haut niveau. Cependant, l'évolution de la technologie, des architectures reconfigurables et des architectures réparties ou distribuées utilisant, par exemple, des réseaux de capteurs et/ou des réseaux ad hoc n'a été prise en compte que partiellement. D'une manière générale, il existe toujours un décalage entre l'avancée technologique et l'utilisation de ces technologies de manière viable. En effet, nos principales préoccupations sont la décomposition de l'application, l'architecture cible la mieux adaptée, les contraintes de temps réel et de consommation d'énergie.

Le défi actuel est de prendre en compte d'autres types de spécifications que des spécifications structurelles et fonctionnelles. Les systèmes embarqués sont à la fois soumis à des contraintes de capacités de mémoire, de bande passante, à des exigences non fonctionnelles comme l'exécution sous contraintes de temps réel, mais également à ces nouvelles dimensions que sont les architectures reconfigurables au sens large, les réseaux de capteurs, les réseaux ad hoc etc. Des outils de modélisation comme MARTE, AADL [1], CAL [146], MCSE (Cofluent Design) ou d'autres, ont déjà permis d'apporter des solutions pour concevoir et vérifier ces systèmes. Cependant, ces outils méritent des évolutions pour intégrer cette nouvelle opportunité de flexibilité de l'architecture dans un milieu pervasif.

1. Architecture Analysis and Design Language

Il est nécessaire d'adapter le processus de conception de manière à répondre à ces évolutions architecturales et applicatives devenues très dynamiques. Dans un premier temps, ceci nécessite de vérifier localement ces nœuds de calcul par des méthodes telle que de la simulation ou des modèles formels. C'est ensuite qu'il faut déterminer des modèles permettant de prendre en considération cet aspect de flexibilité qui pourrait être vu comme des « services » à part entière de l'architecture. Ceci nous amène également à faire évoluer le système d'exploitation comme nous essayons de le faire dans le projet FOSFOR.

5.2.2 Optimisation d'architectures multi-cœurs

Les architectures multi-cœurs sont devenues incontournables comme le montre la figure 5.1. Cependant, ces architectures CPU/GPU sont plutôt orientées logicielles alors que l'accélération matérielle permettrait de monter encore plus en puissance [147]. Cependant, les problèmes majeurs restent le moyen de communiquer de manière standard entre les cœurs de calcul, les systèmes d'exploitation et d'une manière générale, les applications réparties sur ces architectures reconfigurables, appelées HPRC. De plus, cette hétérogénéité de ressources au niveau des zones reconfigurables en vue de l'accélération matérielle en ligne est également un problème pour les cœurs de calcul.

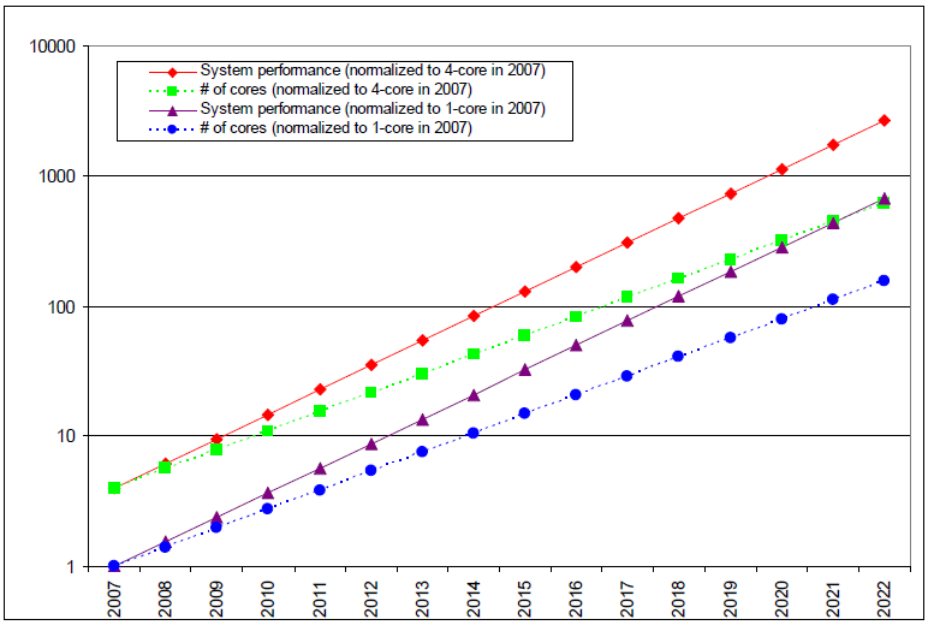


Figure 5.1 – *Deux scénarios possibles pour l'évolution de SoC multicœurs en partant du 65 nm vers le 32 nm et inférieur (source ITRS 2010) [153].*

La communication inter-thread et le placement des threads logiciels ou matériels sur l'architecture répartie sont intimement liés. Cela signifie que les algorithmes de placement/ordonnancement vont influer directement sur les performances globales du système. Nous l'avons d'ailleurs évoquer dans le chapitre 4, section 4.5, page 106. La problématique serait alors de proposer des algorithmes en ligne de placement et d'ordonnancement hiérarchique pour

optimiser les performances. De plus, cette approche devrait prendre en compte l'axe de la consommation d'énergie qui est un critère qui concerne de plus en plus les systèmes HPC/H-PRC.

L'autre problématique concerne l'interfaçage logiciel/matériel qui est toujours présent depuis de nombreuses années. Une solution serait, par exemple, d'utiliser une solution comme OCP [2] [144] comme standard entre les zones matérielles et le NoC. Cependant,d'autres standards de protocoles (interfaces de communication) existent dans la littérature comme CoreConnect [142] d'IBM, VCI [3] [141], AMBA AXI [143] de la société ARM en sont des exemples bien connus. Au niveau communication inter-threads logiciels ou matériels, nous avons par exemple MPI qui est également bien connu.

Le dernier point concerne la programmation de ces architectures multi-cœurs où je souhaite également y associer du reconfigurable. Ainsi, il est nécessaire de rechercher des solutions de langages communs pour exécuter le code applicatif sur n'importe quel noyau (GPU, CPU et FPGA). Nous avons comme exemple FCUDA [148] ou plus récemment OpenCL [149].

5.2.3 La technologie 3D pour le reconfigurable

Dans ce manuscrit, nous avons uniquement traité les architectures dites 2D. Cependant, de nombreuses entreprises comme Tezzaron [4], Xilinx et d'autres envisagent la mise en œuvre de leurs futurs circuits en utilisant les technologies 3D au lieu du circuit monolithique. le 3D résout plusieurs problèmes importants dans la fabrication de FPGA de grande capacité. Le premier problème est le nombre d'E/S que nous obtenons lorsque nous partitionnons un FPGA, qui représente environ 90% d'interconnexion et 10% pour la logique d'après Xilinx. L'objectif majeur de la technologie 3D est la performance et une consommation d'énergie réduite (dûe à la réduction de l'interconnexion) par rapport à la technologie 2D.

Bien que l'intégration 3D promet des avantages considérables, plusieurs défis doivent être satisfaits. Un défi important est l'exploration de l'espace de conception, qui est essentiel pour construire des circuits efficaces en termes de haute performance, de faible consommation d'énergie tout en exploitant tous les avantages offerts par l'intégration 3D. Il existe également des idées d'architecture basées sur de la technologie hybride CMOS-nano, appelé 3D-nFPGA [150]. En outre, les outils de CAO qui facilitent la conception de circuits en 3D sont nécessaires. Les premiers circuits reconfigurables devraient, au mieux, sortir en 2013.

L'idée serait de garder la reconfiguration partielle au sein de ces 3D-FPGA et de permettre le placement et l'ordonnancement de tâches sur les zones reconfigurables. Ainsi, mon objectif serait de proposer des méthodes et des algorithmes en avance de phase permettant de répondre à cette problématique. Il serait alors nécessaire de trouver d'autres pistes que celles décrites dans ce manuscrit pour répondre à ce problème encore plus complexe.

2. Open Core Protocol
3. Virtual Component Interface
4. Tezzaron Semiconductor, `www.tezzaron.com`

5.2.4 Tolérances aux fautes

Les architectures reconfigurables doivent également supporter des applications critiques. Dans ce manuscrit, les applications critiques sont vues comme des applications à temps réel dur, c'est-à-dire que le résultat doit arriver avant l'échéance établie sinon le résultat est considéré comme erroné. Cependant, dans des systèmes embarqués soumis à des radiations, des bruits électromagnétiques ambiants, le comportement de l'application peut être perturbé. Pourtant, la fiabilité des circuits s'est considérablement améliorée. Néanmoins, compte tenu de l'informatisation croissante nécessaire aux applications actuelles et futures, il faut tenir compte des risques potentiels majeurs pouvant résulter de la défaillance des calculateurs, comme de nombreux exemples l'attestent : pertes de vies humaines, atteinte à la santé, dommages à l'environnement, pertes économiques. Aussi, les techniques de tolérance aux fautes constituent un moyen essentiel pour permettre de sécuriser les calculateurs basés sur les architectures reconfigurables dans le cadre d'applications critiques. De plus, nous pouvons imaginer que la tolérance aux fautes peut s'appliquer aux systèmes HPRC qui est un système distribué et réparti où nous pourrions considérer que les communications inter-nœud, comme au sein des calculateurs, peuvent engendrer des fautes.

Inévitablement, l'évolution des architectures reconfigurables partiellement demande de nouvelles techniques permettant de détecter, de corriger ces fautes. Dans le projet Européen ÆTHER, nous avons traité des architectures et des applications « self-adaptive ». Dans cette problématique de la tolérance aux fautes, nous aurions des architectures « self-healing », c'est-à-dire détectant et corrigeant de manière autonome ses dysfonctionnements.

5.2.5 Maîtrise de la consommation d'énergie

D'une manière générale, toutes les perspectives abordées ont un point commun, la réduction de la consommation d'énergie. Au sein de l'équipe MCSOC, cette problématique est déjà en cours d'étude sur de l'ordonnancement sur architecture à multiprocesseur, sur les systèmes d'exploitation et les réseaux de capteurs, mais n'est cependant pas encore bien initiée dans le domaine du reconfigurable. Par exemple, nous essayons actuellement à partir de nos travaux sur le reconfigurable et notre module FaRM, d'estimer le coût énergétique pour la phase de reconfiguration dynamique d'une zone d'un FPGA. Dans le domaine des systèmes HPRC, l'un des intérêts du reconfigurable est notamment une réduction de la consommation d'énergie. Il est prouvé qu'un circuit matériel cadencé à faible fréquence peut offrir les mêmes performances que le logiciel à vitesse d'horloge plus élevée tout en consommant moins d'énergie [145].

Cependant, je pense que la consommation d'énergie doit être gérée sur différentes couches (physique du composant, antenne RF, matériel couche basse, système d'exploitation, et même application) et de communiquer entre-elles afin d'optimiser encore plus la consommation d'énergie. Dans les systèmes embarqués sans fils, c'est l'émission et la réception qui consomment largement plus que la partie numérique. Cependant, la partie numérique a un rôle à jouer. Par exemple, il a été étudié que si un rendez-vous entre un émetteur et un récepteur est établie, la consommation diminue. Les réveils à répétition n'est également pas souhaitable pour l'optimisation de l'énergie. C'est en regroupant toutes ces astuces que l'ef-

fort de réduction de la consommation d'énergie sera réel. L'exemple de ce type d'approche se retrouve dans les réseaux de capteurs et ad hoc qui seront de plus en plus omniprésents dans notre société. Il sera donc nécessaire de faire évoluer les outils de conception afin que la prise en compte de la consommation d'énergie soit prise en compte au niveau d'abstraction le plus élevé comme le montre la figure 5.2.

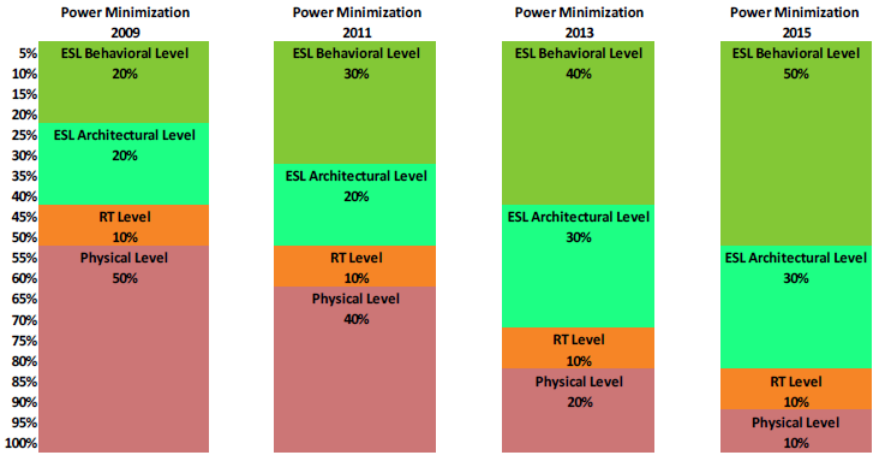


Figure 5.2 – *Évolution du rôle des étapes de conception pour minimiser la consommation d'énergie globale (source ITRS 2009) [158].*

Bibliographie Générale

Ce chapitre correspond à la bibliographie générale du manuscrit sans la bibliographie personnelle qui est située dans le chapitre 1, paragraphe 1.5.

[70] Y. Abdeddaim, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. In *17th International Symposium on Parallel and Distributed Processing (IPDPS)*, page 237, Nice, France, April 2003. IEEE.

[71] J.A. Clemente, C. Gonzalez, J. Resano, and D. Mozos. A hardware task-graph scheduler for reconfigurable multitasking systems. In *International Conference on reconfigurable computing and FPGAs*, pages 79–84, Cancun, Mexico, December 2008.

[72] G.L. Djordjevic and M.B. Tosic. A heuristic for scheduling task graphs with communication delays onto multiprocessors. *Elsevier Parallel Computing*, 22(9) :1197–1214, 1996.

[73] S.P. Fekete, E. Kohler, and J. Teich. Optimal fpga module placement with temporal precedence constraints. In *Design Automation and Test in Europe*, pages 658–665, Munich, Germany, 2001.

[74] L. George, N. Rivierre, and M. Spuri. *Preemptive and non-preemptive real-time uniprocessor scheduling*. PhD thesis, INRIA, 1996.

[75] M. Huang, H. Simmler, P. Saha, and T. El-Ghazawi. Hardware task scheduling optimizations for reconfigurable computing. In *2nd International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA'08)*, pages 1–10, Austin, Texas, USA, November 2008.

[76] O. KERMIA. *Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence*. PhD thesis, University Paris XI, France, 2009.

[77] W.H. Kohler. A preliminary evaluation of the critical path method for scheduling tasks on multiprocessor systems. *IEEE transactions on computers*, 24(12) :1235–1238, December 1975.

[78] C. L. Liu and J. W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[79] Z. Pan and B.E. Wells. Hardware supported tasks scheduling on dynamically reconfigurable soc architectures. *IEEE transactions on very large scale integration (VLSI) systems*, 16(11) :1465–1473, November 2008.

[80] C.H. Papadimitriou and K. Steiglitz. In *Combinatorial Optimization Algorithms and Complexity*, 1998.

[81] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, and M. Rahim Sand Zaidi. The bees algorithm : A novel tool for complex optimisation problems. In *Innovative Production Machines and Systems*, 2006.

[82] F. Redaelli, M.D. Santambrogio, and Memik. An ilp formulation for the task graph scheduling problem tailored to bi-dimensional reconfigurable architectures. *International Journal for Reconfigurable Computing*, pages 97–102, December 2008.

[83] F.E. Sandnes and G.M. Megson. Improved static multiprocessor scheduling using cyclic task graphs : a genetic approach. *Parallel Computing : Fundamentals, Applications and New Directions*, 12 :703–710, 1998.

[84] F.E. Sandnes and O. Sinnen. A new strategy for multiprocessor scheduling of cyclic task graphs. *International Journal High Performance Computing and Networking*, 3(1) :62–71, 2005.

[85] aimms. Aimms web site. `http://www.aimms.com`, 2011.

[86] Aether web site. `http://www.aether-ist.org`, 2011.

[87] Y. Yi, W. Han, X. Zhao, A.T. Erdogan, and T. Arslan. An ilp formulation for task mapping and scheduling on multicore architectures. In *Design, Automation and Test in Europe (DATE)*, pages 33–38, Nice, France, March 2009.

[88] Radu Dobrin and Gerhard Fohler. Reducing the number of preemptions in standard fixed priority scheduling, 2004.

[89] S.-H. Oh and S.-M. Yang. A modified least-laxity-first scheduling algorithm for real-time tasks. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, RTCSA '98, pages 31–, Washington, DC, USA, 1998. IEEE Computer Society.

[90] Woonseok Kim, Jihong Kim, and Sang Lyul Min. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, pages 393–398, New York, NY, USA, 2004. ACM.

[91] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 137–144, Washington, DC, USA, 2005. IEEE Computer Society.

[92] Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, RTSS '99, pages 294–, Washington, DC, USA, 1999. IEEE Computer Society.

[93] L. George, N. Rivierre, and M. Spuri. *Ordonnancement de Tâches efficace et à Complexité Maîtrisée pour des Systèmes Temps Réel*. PhD thesis, Université de Nice-Sophia Antipolis, 2009.

[94] Cofluent Design. `http://www.cofluentdesign.com`, 2011.

[95] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[96] H. Casanova and J. Dongarra. Netsolve : A network server for solving computational science problems. Technical report, Knoxville, TN, USA, 1995.

[97] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod : A tool for performing parametised simulations using distributed workstations. In *4th IEEE Symposium on High Performance Distributed Computing*, 1995.

[98] Douglas Thain and Miron Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[99] Gerald Sabin, Rajkumar Kettimuthu, and Arun Rajan. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science*, pages 87–104, 2003.

[100] A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *Proceedings of the 2002 International Conference on Parallel Processing Workshops*, pages 499–, Washington, DC, USA, 2002. IEEE Computer Society.

[101] Katarina Paulsson, Michael Hübner, Jürgen Becker, Jean-Marc Philippe, and Christian Gamrat. On-line routing of reconfigurable functions for future self-adaptive systems - investigations within the æther project. In *FPL*, pages 415–422, 2007.

[102] Katherine Compton and Scott Hauck. Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34 :171–210, June 2002.

[103] Cindy Kao. Benefits of Partial Reconfiguration. *Xcell Journal*, 55 :65–67, 2005.

[104] Philippe Manet, Daniel Maufroid, Leonardo Tosi, Gregory Gailliard, Olivier Mulertt, Marco Di Ciano, Jean-Didier Legat, Denis Aulagnier, Christian Gamrat, Raffaele Liberati, Vincenzo La Barba, Pol Cuvelier, Bertrand Rousseau, and Paul Gelineau. An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications. *EURASIP J. Embedded Syst.*, 2008 :1 :1–1 :11, January 2008.

[105] Open SystemC Initiative (OSCI). SystemC home. `http://www.systemc.org`.

[106] Object Management Group (OMG). Unified Modeling Language (UML). `http://www.uml.org/`.

[107] Object Management Group (OMG). MARTE profile. `http://www.omgmarte.org/`.

[108] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser. MARTE based modeling approach for Partial Dynamic Reconfigurable FPGAs. In *Sixth IEEE Workshop on*

Embedded Systems for Real-time Multimedia (ESTIMedia 2008), Atlanta États-Unis, 10 2008.

[109] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser. From MARTE to dynamically reconfigurable FPGAs : Introduction of a control extension in a model based design flow. Research Report RR-6862, INRIA, 2009.

[110] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Jean-Philippe Diguet, and Philippe Soulard. UML design for dynamically reconfigurable multiprocessor embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1195–1200, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[111] Jerry Gipper. SystemC : the SoC system-level modeling language. *Embedded Computing Design*, may 2007.

[112] Lukai Cai and Daniel Gajski. Transaction level modeling : an overview. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '03, pages 19–24, New York, NY, USA, 2003. ACM.

[113] Andreas Raabe. *Describing and Simulating Dynamic Reconfiguration in SystemC Exemplified by a Dedicated 3D Collision Detection Hardware*. PhD thesis, Bonn University, 2008.

[114] Antti Pelkonen, Kostas Masselos, and Miroslav Cupḱ. System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC. *Parallel and Distributed Processing Symposium, International*, 0 :174b, 2003.

[115] Yang Qu, Kari Tiensyrjä, and Kostas Masselos. System-Level Modeling of Dynamically Reconfigurable Co-processors. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application*, volume 3203 of *Lecture Notes in Computer Science*, pages 881–885. Springer Berlin / Heidelberg, 2004.

[116] Kenji Asano, Junji Kitamichi, and Kuroda Kenichi. Dynamic Module Library for System Level Modeling and Simulation of Dynamically Reconfigurable Systems. *Journal of Computers*, 3 :55–62, feb 2008.

[117] ARDMAHN consortium. ARDMAHN project. `http://ARDMAHN.org/`.

[118] Jérémie Crenne, Pierre Bomel, Guy Gogniat, and Jean-Philippe Diguet. End-to-end bitstreams repository hierarchy for FPGA partially reconfigurable systems. In Guy Gogniat, Dragomir Milojevic, Adam Morawiec, and Ahmet Erdogan, editors, *Algorithm-Architecture Matching for Signal and Image Processing*, volume 73 of *Lecture Notes in Electrical Engineering*, pages 171–194. Springer, 2011.

[119] Luca Benini and Giovanni De Micheli. Networks on Chips : A new SoC paradigm. *Computer*, 35 :70–78, 2002.

[120] Xian-He Sun and Yong Chen. Reevaluating amdahl's law in the multicore era. *J. Parallel Distrib. Comput.*, 70 :183–188, February 2010.

[121] Dong Hyuk Woo and Hsien-Hsin S. Lee. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41 :24–31, 2008.

[122] Esam El-Araby, Ivan Gonzalez, and Tarek El-Ghazawi. Exploiting partial runtime reconfiguration for High-Performance Reconfigurable Computing. *ACM Trans. Reconfigurable Technol. Syst.*, 1 :21 :1–21 :23, January 2009.

[123] Jose Antonio Casas, Juan Manuel Moreno, Jordi Madrenas, and Joan Cabestany. A novel hardware architecture for self-adaptive systems. In *AHS '07 : Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems*, pages 592–599, Washington, DC, USA, 2007. IEEE Computer Society.

[124] Xilinx, Inc. *Virtex-5 FPGA Configuration User Guide*, August 2010.

[125] Xilinx Inc. *LogiCORE IP XPS HWICAP datasheet*, 2010.

[126] Katherine Compton and Scott Hauck. Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34 :171–210, June 2002.

[127] Russell Tessier and Wayne Burleson. Reconfigurable computing for digital signal processing : A survey. *J. VLSI Signal Process. Syst.*, 28 :7–27, May 2001.

[128] Katarina Paulsson, Michael Hübner, Salih Bayar, and Jürgen Becker. Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 699–700, September 2008.

[129] Steve Guccione, Delon Levi, and Prasanna Sundararajan. JBits : Java based interface for reconfigurable computing. 1999.

[130] Ming Liu, Wolfgang Kuehn, Zhonghai Lu, and Axel Jantsch. Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, August 2009.

[131] Francesco Redaelli, Marco D. Santambrogio, and Donatella Sciuto. Task scheduling with configuration prefetching and anti-fragmentation techniques on dynamically reconfigurable systems. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 519–522, New York, NY, USA, 2008. ACM.

[132] Xilinx Inc. *Virtex-5 Configuration User Guide, UG191*, 2010.

[133] Adam Flynn, Ann Gordon-Ross, and Alan D. George. Bitstream relocation with local clock domains for partially reconfigurable fpgas. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 300–303, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[134] Xilinx. Real-Time Executive for Multiprocessor Systems (RTEMS) web site. `http://www.rtems.org`, 2011.

[135] Gaisler. Disponible également sur le site : RCC (RTEMS LEON/ERC32 GNU cross-compiler system) permettant la compilation RTEMS pour LEON3. `http://www.gaisler.com`, 2011.

[136] P. David Stotts and William Pugh. Parallel finite automata for modeling concurrent software systems. *J. Syst. Softw.*, 27 :27–43, October 1994.

[137] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. *The NAS Parallel Benchmarks.* http ://www.nas.nasa.gov/News/Techreports/1994/PDF/RNR-94-007.pdf, 1994.

[138] J. P. Calvez and D. Isidoro. A codesign experience with the mcse methodology. In *Proceedings of the 3rd international workshop on Hardware/software co-design*, CODES '94, pages 140–147, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[139] Heller D. Bakowski P. Calvez, J.P. Functional-level synthesis with vhdl. In *EUROVHDL'93*, Hamburg, Germany, Sept. 20-24 1993.

[140] J.P. Calvez. Asics specification and design. In *Chapman&Hall*, pages 1–570, 1995.

[141] Annette Bunker and Ganesh Gopalakrishnan. Formal specification of the virtual component interface standard in the unified modeling language. Technical report, 2001.

[142] Coreconnect protocol. Technical report, 2011.

[143] Coreconnect protocol. Technical report, 2011.

[144] Open core protocol. Technical report, 2011.

[145] P. Kwan and C. Clarke. Fpgas for improved energy efficiency in processor based systems. In Thambipillai Srikanthan, Jingling Xue, and Chip-Hong Chang, editors, *Advances in Computer Systems Architecture*, volume 3740 of *Lecture Notes in Computer Science*, pages 440–449. Springer Berlin / Heidelberg, 2005. 10.1007/11572961 35.

[146] J. Eker and J.W. Janneck. Cal language report, language version 1.0 - document edition 1. Technical report, December 2003.

[147] Christopher W. Fletcher, Ilia A. Lebedev, Narges B. Asadi, Daniel R. Burke, and John Wawrzynek. Bridging the gpgpu-fpga efficiency gap. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 119–122, New York, NY, USA, 2011. ACM.

[148] Alexandros Papakonstantinou, Karthik Gururaj, John A. Stratton, Deming Chen, Jason Cong, and W. Hwu Wen-Mei. Fcuda : Enabling efficient compilation of cuda kernels onto fpgas. In *Proceedings of the 7th Symposium on Application Specific Processors, SASP'2009*, pages 35–42, July 2009.

[149] Singh Deshanand. Higher level programming abstractions for fpgas using opencl. In *FPGA 2011 Pre-Conference Workshop : The Role of FPGAs in a Converged Future with Heterogeneous Programmable Processors, Session Future-Looking CAD/Compiler Synthesis Flows*, February 2011.

[150] Dong Chen, Chen Deming, Tanachutiwat Sansiri, and Wang Wei. Performance and power evaluation of a 3d cmos/nanomaterial reconfigurable architecture. In *International Conference on Computer Aided Design*, pages 758–764, 2007.

[151] Marco Caccamo, Giorgio Buttazzo, and Lui Sha. Capacity sharing for overrun control. In *IEEE Real-Time Systems Symposium*, pages 295–304, 2000.

[152] David Stewart and Pradeep K. Khosla. Real-time scheduling of sensor-based control systems. In *in Real-Time Programming (W. Halang)*, pages 144–150. Pergamon Press, 1991.

[153] Fawzi Behmann. Nextgen multicore/accelerators soc platform defined by itrs, February 2010.

[154] Kevin Jeffay and Steve Goddard. Rate-based resource allocation models for embedded systems. In *in Proc. First International Workshop on Embedded Software*, 2001.

[155] Luca Abeni, Giuseppe Lipari, Giorgio Buttazzo, Scuola Superiore, and S. Anna. Constant bandwidth vs proportional share resource allocation. In *In Proceedings of the IEEE International Conference on Mutimedia Computing and Systems*, pages 107–111, 1999.

[156] Ion Stoica, Hussein Abdel-wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems, 1996.

[157] Taiwan Semiconductor Manufacturing Company (TSMC). 28 nanometer process technology. `http://www.tsmc.com/download/brochures/2011_28%20Nanometer%20Process%20Technology.pdf`, 2011.

[158] International Technology Roadmap for Semiconductors. Design, edition 2009, 2009.

[159] Krste Asanovic, Ras Bodik, Bryan C. Catanzaro, Joseph J. Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William L. Plishker, John Shalf, Samuel W. Williams, and Katherine A. Yelick. The landscape of parallel computing research : a view from Berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, december 2006.

[160] Rakesh Kumar, Tusar Patra, and Anupam Basu. Software energy optimization of real time preemptive tasks by minimizing cache-related preemption costs. In Hans Zima, Kazuki Joe, Mitsuhisa Sato, Yoshiki Seo, and Masaaki Shimasaki, editors, *High Performance Computing*, volume 2327 of *Lecture Notes in Computer Science*, pages 435–438. Springer Berlin / Heidelberg, 2006. 10.1007 :3-540-47847-7 :28.

[161] Farroq Muhammad, Michel Auguin, and Fabrice Muller. Procédé de gestion des préemptions dans un système d'exploitation temps réel, n° d'application wo/2009/087317, international application no. : Pct/fr2008/001481, european patent office n° 088699004-2211, 22.10.2008.

[162] François Duhem, Fabrice Muller, and Philippe Lorenzini. Reconfiguration time overhead on fpga : Reduction and cost model (accepted). *IET Computers and Digital Techniques*, Accepted in August 2011.

[163] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Static scheduling of periodic hardware tasks with precedence and deadline constraints on reconfigurable hardware devices. *Special Issue in EURASIP International Journal of Reconfigurable Computing (IJRC)*, February 2011.

[164] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. New three-level resource management enhancing quality of off-line hardware task placement on fpga. *EURASIP International Journal of Reconfigurable Computing (IJRC)*, April 2010.

[165] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. *Algorithm-Architecture Matching for Signal and Image Processing : "A New Three-Level Strategy for Off-line Placement*

of Hardware Tasks on Partially and Dynamically Reconfigurable Hardware", volume 73. Springer, 2011.

[166] François Duhem, Fabrice Muller, and Philippe Lorenzini. Methodology for designing partially reconfigurable systems using transaction-level modeling. In *IEEE Design and Architectures for Signal and Image Processing (DASIP) (Accepted)*, Tampere, Finland, November 2-4 2011. IEEE.

[167] I. Belaid, F. Muller, and M. Benjemaa. Schedulers-driven approach for dynamic placement/scheduling of multiple dags onto sopcs. In *IEEE International Symposium on Rapid System Prototyping (RSP)*. IEEE, May 2011 2011.

[168] François Duhem, Fabrice Muller, and Philippe Lorenzini. Farm : fast reconfiguration manager for reducing reconfiguration time overhead on fpga. In *Proceedings of the 7th international conference on Reconfigurable computing : architectures, tools and applications*, ARC'11, pages 253–260, Berlin, Heidelberg, 2011. Springer-Verlag.

[169] B. Ouni, I. Belaid, F. Muller, and M. Benjemaa. Placement of hardware tasks on fpga using the bees algorithm. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011)*, Vilamoura, Algarve, Portugal, March 2011.

[170] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Optimal static scheduling of real-time dependent tasks on reconfigurable hardware devices. In *IEEE International Conference on Communications, Computing and Control Applications (CCCA 2011)*, Hammamet, Tunisia, March 2011. IEEE.

[171] Clément Foucher, Fabrice Muller, and Alain Giulieri. Exploring FPGAs capability to host a HPC design. In *IEEE CAS 28th Norchip Conference (Norchip 2010)*, pages 1–4, Tampere Finlande, November 2010.

[172] I. Belaid, F. Muller, and M. Benjemaa. New three-level resource management for off-line placement of hardware tasks on reconfigurable devices. In *Reconfigurable Communication-centric Systems on Chip (ReCoSoC 2010)*, Karlsruhe, Germany, May 2010.

[173] Farooq Muhammad Fabrice Muller. An embedded, generic and multiprocessor hardware operating system. In *Design and Architectures for Signal and Image Processing (DASIP)*, Sophia-Antipolis, France, 22-24 September 2009.

[174] I. Belaid, F. Muller, and M. Benjemaa. Off-line placement of reconfigurable zones and off-line mapping of hardware tasks on fpga. In *Design and Architectures for Signal and Image Processing (DASIP)*, Sophia-Antipolis, France, 22-24 September 2009.

[175] I. Belaid, F. Muller, and M. Benjemaa. Off-line placement of hardware tasks on fpga. In *19th IEEE International Conference on Field Programmable Logic and Application (FPL'09)*, pages 591–595, Prague, Czech Republic, September 2009. IEEE.

[176] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Slack-conserving based scheduling of periodic real-time tasks. In *Proceedings of the 2008 Fifth IEEE International Symposium on Embedded Computing*, pages 37–42, Beijing, China, October 6-8 2008. IEEE Computer Society.

[177] Farooq Muhammad, Khurram Bhatti, Fabrice Muller, and Michel Auguin. Precognitive dvfs : Minimizing switching points to further reduce the energy consumption. In *14th IEEE Real-Time and Embedded Technology and Applications Symposium, WIP session*, St. Louis, MO, USA, 22-24 april 2008. IEEE.

[178] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Weight bound limits in supertasking approach for guaranteed timeline constraints. In *Proceedings of the 2008 International Conference on Parallel Processing - Workshops (ICPPW'08)*, pages 9–16, Krakow, Poland, 1-5 july 2008. IEEE Computer Society.

[179] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Hierarchical scheduling approach for real time tasks. In *In Proc. International Conference on Parallel Processing, Workshop on Scheduling Ressource Management for Parallel and Distributed Systems*, Portland (Oregon), USA, 8-12 september 2008.

[180] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Aether : dynamic and self-adaptive middleware. In *in Proc. IEEE International Multitopic Conference*, Lahore, Pakistan, 28-30 december 2007. IEEE.

[181] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Dynamic and self adaptive resource management : Aether operating environment. In *in Proc. 3rd IEEE Int. Conference on Emerging Technologies*, Islamabad, Pakistan, 12-13 november 2007. IEEE.

[182] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Self balancing computational load on multiprocessor architecture. In *IEEE International Conference on Self-Organization and Autonomous Systems in Computing and Communications - SOAS'2006*, Erfurt, Germany, sept. 2006.

[183] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Contentions-conscious dynamic but deterministic scheduling of computational and communication tasks. In *Annual ACM Symposium on Applied Computing*, Dijon, France, Apr. 2006. ACM Press.

[184] François Duhem, Fabrice Muller, and Philippe Lorenzini. Transaction-level modeling of dynamically reconfigurable systems using systemc. In *Sophia Antipolis MicroElectronics, SAME'2011, Poster session)*, Sophia Antipolis, France, October 12-13 2011.

[185] Fabrice Muller and Farooq Muhammad. Virtual platform for hw rtos - multiprocessor hardware rtos. In *IEEE in Proc. Design Automation and Test in Europe, (DATE 2009), University Booth*, Nice, France, 21-23 april 2009. IEEE.

[186] J-Ph Diguet, M.El Khodary, G. Gogniat, F. Muller, and M. Auguin. On simulating operating environment decisions in a sane network. In *AMWAS'08 (2nd AETHER - MORPHEUS Workshop- Autumn School From Reconfigurable to Self-Adaptive Computing)*, Lugano, Switzerland, October 7-9 2008.

[187] Fabrice Muller, Farooq Muhammad, and Michel Auguin. Design of a hardware multiprocessor real-time operating system. In *IEEE in Proc. Design Automation and Test in Europe (DATE 2007), University Booth*, Nice, France, 17-19 april 2007. IEEE.

[188] Clément Foucher, Fabrice Muller, and Alain Giulieri. Méthodologie dédiée aux applications parallèles sur plateforme reconfigurable dynamiquement. *Technique et Science Informatiques*, Soumise en Septembre 2011.

[189] François Duhem, Fabrice Muller, and Philippe Lorenzini. Dynamic and partial reconfiguration transaction-level modeling in systemc. In *Colloque GDR SoC/SiP*, Lyon, France, 15-17 juin 2011.

[190] Clément Foucher, Fabrice Muller, and Alain Giulieri. Flot de conception d'applications parallèles sur plateforme reconfigurable dynamiquement. In *Symposium en Architectures nouvelles de machines (Sympa'14)*, St Malo, France, 10-13 mai 2011.

[191] Ikbel Belaid, Fabrice Muller, and Maher Benjemaa. Off-line placement/scheduling of hardware tasks on reconfigurable devices. In *Colloque GDR SoC/SiP*, Cergy-Pontoise, France, 9-11 juin 2010.

[192] François Duhem, Fabrice Muller, and Philippe Lorenzini. Services pour des systèmes reconfigurables dynamiquement. In *Colloque GDR SoC/SiP*, Cergy-Pontoise, France, 9-11 juin 2010.

[193] Ikbel Belaid, Fabrice Muller, Maher Benjemaa, and Alain Giulieri. Off-line placement of hardware tasks on fpga. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[194] Clément Foucher, Fabrice Muller, and Alain Giulieri. Implémentation d'un système d'exploitation matériel compatible rtems. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[195] Bassem Ouni, Fabrice Muller, and Maher Benjemaa. Placement et ordonnancement des tâches matérielles sur des zones reconfigurables en utilisant le bees algorithm. In *Colloque GDR SoC/SiP*, Paris-Orsay, France, 10-11-12 juin 2009.

[196] Farooq Muhammad, Khurram Bhatti Muhammad, Fabrice Muller, and Michel Auguin. Efficient and optimal multiprocessor scheduling for real time tasks. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

[197] Ikbel Belaid, Fabrice Muller, and Alain Giulieri. Virtualisation de l'ordonnancement matériel/logiciel sur plateforme reconfigurable dynamiquement. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

[198] Farooq Muhammad, Fabrice Muller, and Michel Auguin. Proportionate scheduling of hard and soft real time tasks. In *Colloque GDR SoC/SiP*, Paris-Jussieu, France, 13-14-15 juin 2007.

[199] Fabrice Muller. Os & reconfigurable. In *Colloque GDR SoC/SiP*, Paris-ENST, France, 4-5-6 juin 2008.

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

HABILITATION A DIRIGER LES RECHERCHES

Université de Nice-Sophia Antipolis
Mention : Electronique

présentée par

Fabrice Muller

Architectures multiprocesseurs, distribuées et reconfigurables dynamiquement pour les applications temps réel

ANNEXE A

Sélection des publications significatives

Septembre 2011

ANNEXE A

Sélection des publications significatives

Cette annexe illustre certaines contributions menées en présentant plusieurs articles scientifiques.

1. Revue IJRC, 2010 : New Three-Level ResourceManagement Enhancing Quality of Offline Hardware Task Placement on FPGA
2. Revue IJRC, 2011 : *Static Scheduling of Periodic Hardware Tasks with Precedence and Deadline Constraints on Reconfigurable Hardware Devices*
3. Revue IET Computers and Digital Techniques, (Acceptée le 28 juillet 2011) : *Reconfiguration Time Overhead on FPGA : Reduction and Cost Model*

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 980762, 20 pages
doi:10.1155/2010/980762

Research Article

New Three-Level Resource Management Enhancing Quality of Offline Hardware Task Placement on FPGA

Ikbel Belaid,[1,2] Fabrice Muller,[1,2] and Maher Benjemaa[1,2]

[1] *University of Nice Sophia-Antipolis/LEAT-CNRS, 250 rue Albert Einstein, bât 4. 06560, Sophia Antipolis - Cedex, France*
[2] *Research Unit ReDCAD, National Engineering School of Sfax, B.P. 1173-3038 Sfax, Tunisia*

Correspondence should be addressed to Ikbel Belaid, ikbel.belaid@unice.fr

Received 14 November 2009; Revised 6 March 2010; Accepted 24 April 2010

Academic Editor: Christophe Bobda

Currently, reconfigurable hardware devices feature a high density of heterogeneous resources to enable multitasking and offer flexibility in application needs. These concepts raise the need for efficient management of hardware tasks and hardware resources. The scheduling of hardware tasks is highly dependent on placement. Placement focuses on allocation of hardware resources required by the scheduled hardware tasks. In this paper, we propose novel three-level resource management that investigates enhancement of placement quality by reducing task rejection, configuration overheads, and by optimizing resource utilization. Improving placement quality will produce significant enhancement of performance for scheduling and overall execution time of the application in FPGA. Hence, the placement problem is formulated into a constrained optimization problem and resolved with powerful solvers using the Branch and Bound method. The obtained results of an application of heterogeneous hardware tasks show an average resource utilization of 36% of the available resources on the reconfigurable region and an overall overhead of 11% of total application running time, and we have eliminated the issue of task rejection. Compared to static implementation, the gain in resource utilization within the reconfigurable region achieves up to 43%.

1. Introduction

Scheduling and placement are strongly linked. The scheduler decides which of the ready tasks should be executed next and calls the placer to find a feasible location. The scheduler decision should be taken in accordance with the ability of placer to allocate free resources required by the elected task. Field-Programmable Gate Array (FPGA) is the most widely used reconfigurable hardware device. Today's FPGA devices provide several million reconfigurable heterogeneous resources. The development of dynamic partial reconfiguration in the FPGAs allows reconfiguring only the necessary part of the FPGA when required without interfering with any other parts running on the same FPGA. While this technique can increase device utilization and performance of scheduling and application, it also leads to high configuration overhead, fragmentation, and complex allocation situations of hardware tasks [1]. Frequently, the existing methods of placement face these issues. Consequently, the quality of placement and performance of the scheduling degrades while the overall response time increases. So, there exists a serious need to define an efficient method that helps manage the area of resources.

In general, the placement of hardware tasks consists of two main functions: (i) *partitioning,* which handles the free space in the device and identifies any potential sites enabling execution of hardware tasks, and (ii) *fitting,* which selects the feasible placement solution. In this paper, under FOSFOR project [2], we address the aspect of placement and introduce new three-level offline resource management that challenges all the above mentioned issues. The main concern of our method is enhancing placement quality by targeting the optimized use of FPGA's resources and taking into account the physical and functional features of hardware tasks (FOSFOR (Flexible Operating System For Reconfigurable platform) is French national program (ANR) targeting the most evolved technologies. Its main objective is to design a real time operating system distributed on hardware and software execution units which offers required flexibility to application tasks through the mechanisms of dynamic

reconfiguration and homogeneous Hw/Sw OS services.). During the conception of three-level resource management, we rely on the physical architecture of target technology and on the advantages of dynamic partial reconfiguration. The contribution of this paper is the development of

(i) offline flow for hardware task classification which is an enhancement of our previously proposed work in [3] and which creates task classes,

(ii) a formulation of the task placement as a constrained optimization problem and its resolution by powerful solvers using Branch and Bound method by considering two independent sublevels: the first sublevel ensures the fitting of obtained task classes on physical blocs partitioned on target technology which improves resource efficiency up to 36% and the second sublevel performs the mapping of tasks to obtained classes by optimizing the overall overhead up to 11% of total running time and by minimizing the number of task relocations.

The remainder of this paper is organized as follows. Section 2 reviews related work of placement. Section 3 details our three-level resource management. Section 4 focuses on the formulation of hardware task placement as a constrained optimization problem and its resolution by the Branch and Bound method. The obtained results are depicted in Section 5. In Section 6, we summarize our work, make some concluding remarks, and present future works.

2. Related Work

Current strategies dealing with task placement are divided into two categories: offline placement and online placement.

2.1. Online Methods for Hardware Task Placement. The main reference is [4], Bazargan et al. suggest online scenario for hardware task placement. In fast on-line placement, Bazargan et al. introduce two partitioning techniques; the first technique denoted *Keeping All Maximal Empty Rectangles (KAMER)* searches all the Maximal Empty Rectangles (MER) after each task insertion/deletion operation. The MERs are defined as the empty rectangles which are not contained in another empty rectangle and are not necessarily disjoined. The second technique of partitioning called *Keeping Nonoverlapping Empty Rectangles* keeps all the nonoverlapping holes and is evoked after each split/merge operation. For both previous techniques of partitioning, the fitting in the on-line placement is conducted by the *First Fit*, *Best Fit (BF)* and *Bottom Left* bin-packing algorithms [5].

Fast two-dimensional on-line placement is presented in [6] which is an extension of Bazargan's partitioning; the *Keeping Nonoverlapping Empty Rectangles*. In [6], Walder et al. propose placement methods that rely on efficient algorithms of partitioning enhancing the quality of Bazargan's partitioning by 70% and on a hash matrix data structure that finds a feasible placement in constant time. Based on a nonpreemptive system without precedence constraint, the Walder's partitioner delays split decision instead of using the Bazargan's heuristics for decision of split. Reference [6] details four enhancements of Bazargan's partitioner. The main partitioner is On-The-Fly (OTF) partitioner which consists of resizing empty rectangles only if the newly arrived task overlaps them. After each task insertion or deletion, the Walder's partitioner updates the data on the hash matrix. If a new arrived task fits into more than one empty rectangle determined by the hash matrix, a fitting strategy is used to choose a rectangle [6]. Reference [6] implements four types of fitting: *BF*, *Worst Fit*, *Best Fit with Exact Fit,* and *Worst Fit with Exact Fit*.

Ahmadinia et al. present in [7] a new method of on-line placement by managing the occupied space instead of free space because of the difficulty of managing empty space and the huge increase of empty rectangles. In [7], at the arrival of a new task, the space manager starts by delimiting the *Impossible Placement Region (IPR)* relative to placed modules and to device. Thereafter, the *Nearest Possible Position* fitter selects the optimal point that gives the optimal communication cost, which is not included in the *IPR*.

In [8], Marconi et al. extend Bazargan's placement by means of an *Intelligent Merging* (*IM*) algorithm. IM dynamically combines three techniques of managing free resources: *Merging Only if Needed*, *Partially Merging,* and *Direct Combine*. IM accelerates Bazargan's partitioner by 3 and improves placement quality by increasing the rate of accepted tasks.

Handa et al. introduce the staircase method in [9]. The staircase method handles free space during the first subfunction of the on-line placement. This method is considered efficient as it tries to cover the faults of the *KAMER* method, especially for task rejection.

Some approximate metaheuristics are adopted to resolve the hardware task placement such as [10] that employs an on-line task rearrangement by using genetic algorithm approach. When a newly arrived task could not be placed immediately, the proposed approach tries to rearrange a subset of tasks executing on the FPGA to allow the processing of the pending task sooner. The approach is based on the First-Fit strategy and genetic algorithm. By allowing the rotation of tasks and by using input buffer to save the data of suspended tasks, the approach combines two genetic algorithms to resolve two subproblems. The first subproblem identifies a feasible rearrangement and the second one consists on scheduling the moves of executing tasks to attain the feasible rearrangement.

Reference [11] copes with task placement problem and adopts interconnection-based FPGA as support for run-time reallocation of hardware tasks. It applies matador task concurrency management methodology for scheduling hardware tasks on identical tiles by minimizing run-time reconfiguration. This goal is reached by two new techniques implied in the scheduler named configuration reuse and configuration prefetch. Reduction in configuration overhead decreases significantly the execution time and energy consumption.

2.2. Offline Methods for Hardware Task Placement. In the offline scenario for hardware task placement, [4] defines 3D templates depicting the tasks in time and space dimensions

and uses slow heuristics: simulated annealing and greedy research, and KAMER-BF to perform a high-quality placement in terms of resource utilization and task rejection.

Reference [12] models the problem of resource allocation as a 0-1 integer linear programming problem which aims necessarily at minimizing the resources area which is reconfigured at runtime. Reference [12] considers an application with known sequential execution trace. Hence, the huge configuration latency is tackled by reducing the overlapped areas between tasks.

By considering the placement of hardware tasks as rectangular items on hardware device as rectangular unit, several approaches for resolving the two-dimensional packing problem are proposed. For example, in [13], the offline approximate heuristics: Next-Fit Decreasing Height, First-Fit Decreasing Height, and Best-Fit Decreasing height are presented as strip-packing approaches based on packing items by levels.

In addition, Lodi et al. in [14] propose different offline approaches to resolve hardware task placement as 2D bin-packing problem. For instance, the Floor-Ceiling algorithm that considers alternate directions for packing tasks, either from left to right when their bottom edges touch the level floor or from right to left; when their top edges are on the top of the level floor. The Knapsack packing algorithm is also proposed in [15] which initializes each level by the tallest unpacked item and completes it by packing tasks as the associated Knapsack problem that maximizes the total area within level. For both latter algorithms, the second phase of 2D bin packing is achieved by Best-Fit Decreasing algorithm.

Baker et al. define in [16] the Bottom-left (BL) offline algorithm. BL packs each hardware task in the bottom left position.

Martello and Vigo propose in [17] an enumerative offline approach to exact solution for 2D bin-packing. Their algorithm is based on a two-level branching scheme: the outer branch-decision tree that assigns tasks to the bins without specifying their position and the inner branch-decision tree that enumerates all possible patterns.

By optimizing the total execution time and the resource utilization, the method of placement in [18] consists of two phases: the first phase is the recursive bi-partitioning by means of slicing tree that defines the relative position of each hardware task towards the other hardware task placement and finds the appropriate room in the reconfigurable device for each hardware task according to task's resources and intertask communication. The second phase uses the obtained room topology to achieve the sizing that computes the possible sizes for each room.

Reference [19] minimizes task rejection and presents offline algorithms for 3D floorplanning of hardware tasks on reconfigurable functional unit such as: KAMER-BF Decreasing, Simulated Annealing, Low-temperature Annealing, and Zero-temperature Annealing. The 3D placement models tasks as 3D boxes having a base corresponding to the spatial dimensions of tasks and a height corresponding to their time-span.

In [20], as bin-packing problem, an offline approach is proposed by Fekete et al. through a graph-theoretical characterization of the packing of a set of items into a single bin. Tasks are presented as three-dimensional boxes and the feasible packing is decided by the orthogonal packing problem within a given container. Their approach considers packing classes, precedence constraints, and the edge orientation to solve the packing problem. Similarly, in [21], Teich et al. definesthe task placement as more-dimensional packing problem. Tasks are modeled as 3D polytopes with two spatial dimensions and the time of computation. Based on packing classes as well as on a fixed scheduling, they search a feasible placement on a fixed-size chip to accommodate the set of tasks. The resolution is performed by Branch and Bound technique to optimality of dynamic hardware reconfiguration.

The major shortcoming of all the above-proposed methods of placement is that they are applicable only in homogeneous devices. In fact, these methods assume that the relocation of tasks is allowed and enable the allocation of resources whenever sufficient free space is available. Furthermore, the placement disregards the routing constraints as it does not address the issue of intertask communication and I/O routing. Moreover, tasks are nonpreemptive and almost-identical. Unfortunately, the algorithms for 2D packing focus only on the objective of minimizing resource waste and do not satisfy all other goals. All the existing strategies of placement provide a nonguarantee system as they suffer from task rejection and fragmentation. We believe the issue of task rejection is caused by the constructive way in which the placement is performed throughout all existent strategies of placement. The issue of fragmentation may lead to undesirable situations where a new task cannot be placed although there would be sufficient free space.

As we have full knowledge about the set of hardware tasks and the features of the reconfigurable device, in this paper, we present a realistic three-level resource management solution as a new strategy to perform offline placement of hardware tasks in FPGA. This new strategy aims at enhancing placement quality by trimming the previously mentioned issues. Our proposed method is technology-dependent and in accordance with generic placement as the second level partitions the available resources in FPGA according to the task classes provided by the first level. Nevertheless, the third level ensures the subfunction of fitting. The task model is preemptive and preemption points are predefined. Our resource management allows the relocation of tasks and results in strict positions for each hardware task by respecting its preemption points and types of resources.

3. Three-Level Resource Management

We use Xilinx's Virtex FPGA as a reference for the hardware reconfigurable device to lead our hardware resource management study. We offer a definition of a few terms which are used throughout the paper: *NT* is the number of tasks, *NR* the number of Reconfigurable Physical Blocs, *NZ* the number of Reconfigurable Zones, and *NP* the number of resource types in the chosen technology. In the beginning, we should start by introducing the hardware task models. We have defined three models.

(i) *The functional model*: this contains the functional features of hardware tasks T_i as the worst case execution time (C_i), the period (P_i), and preemption points l ($Preemp_{i,l}$). The number of preemption points of T_i is denoted by $NbrPreemp_i$. This number also includes the first point of execution of T_i. Preemption points are specified by the designer.

(ii) *The behavioral model*: This includes the finite state machine controlling each task and which handles a set of *NbrReg* registers of 32 bits to conduct the context switch. The behavioral model defines the functional overhead ($Context_i$) that is needed to preempt or resume the execution of tasks. This functional overhead is fixed for all the hardware tasks as they have similar register banks with *NbrReg* registers. The functional overhead is computed as two times (save and load) the access to a bus having 32 bits of width and functioning at 80 MHz. In addition, we have considered the worst case, when tasks need the *NbrReg* 32bit-registers to perform context switch. Hence, this functional overhead represents sequential access of *NbrReg* registers associated for a given task to save and load its context through a 32 bit-bus ($Context_i$ = 2x *NbrReg*/80 MHz).
In our preemptive modeling, we do not use the classical method of readback and load bitstream since it takes a significant latency, complicates the preemption, and requires a large space memory as a new readback bitstream must be saved at each preemption. Thus, we resort to save the state of finite state machine with an acceptable amount of data by keeping always the same bitstream for each task. Preemption points of hardware tasks are fixed in a way to reduce the data dependency that could exist between two states. In fact, we must avoid keeping a preemption point between two states processing the same data because we need to save these data into an external memory which might increase the overhead at run-time. Otherwise, it is recommended to put a preemption point when the task is in a blocked state waiting for receiving external resource to allow the ready tasks to be executed in the RZ. As tasks are periodic, a preemption point could be inserted after the last state before restarting the task to avoid any data dependency. In the finite state machine, the longest execution time between two states must be considered in order to deduct the worst case execution time.

(iii) *The RB-model*: tasks are presented as a set of reconfigurable resources called Reconfigurable Blocs (RB). The RBs are closely shaped to the reconfiguration granularity in the chosen technology. The determination of the RB-model of hardware tasks is well-detailed in our work in [3]. Each type of RB is characterized by specified cost $RBCost_k$ which is defined according to three parameters: the number of the RB type in the device, its power consumption and the importance of its functionality. The more consistent these parameters are, the higher the cost of RB type. The RB-model of each hardware task is described by (1).

$$T_{i\_}RB = \{\alpha_{i,k}RB_k\}, \quad \alpha_{i,k} \text{ is natural}, \quad 1 \le i \le NT,\ 1 \le k \le NP. \tag{1}$$

The functional model and the RB-model are the basic models for resource management. However, the behavioral model is employed during scheduling. The FPGA provides reconfigurable resources organized according to column-based technology [22]. The management of hardware resources on FPGA consists of three levels.

3.1. Level 1: Offline Flow of Hardware Task Classification. Level 1 takes a set of tasks as input and provides the types and instances of Reconfigurable Zones (RZ). The RZs are abstractions of task classes and are defined according to the types of resources needed by the tasks. As the RZs model the classes of hardware tasks, they are described by their RB-model given by

$$RZ_{i\_}RB = \{\beta_{i,k}\, RB_k\}, \quad \beta_{i,k} \text{ is natural}, \quad 1 \le i \le NZ,\ 1 \le k \le NP. \tag{2}$$

Level 1 consists of three steps:

Search of RZ types: Step 1 gathers the tasks sharing the same types of RBs under the same type of RZ. Step 1 is essentially based on RB-model of hardware tasks and is achieved by Algorithm 1 of complexity in the worst case O ($NT * NP * NZ$).

Step 1 scans the RB-model of each hardware task and checks whether there exists in the list of RZ types *List-RZ* an already inserted type of RZ that closely matches the required types of RBs in the task (line 6). In this case, step 1 updates the number of RBs within this type of RZ by the maximum between the number of RBs in the task and that in the RZ (line 9). If the required types of RBs in the task do not match any type of RZ included in the *List-RZ*, the algorithm of the search of RZ types decides the creation of a new type of RZ as required by the task (line 13) and inserts it in *List-RZ* (line 14). At the end of step 1, we obtain the possible types of RZs. The number of RZ types is limited by the number of tasks. As shown in Figure 1, step 1 groups T_1 and T_3 in the same type of RZ (RZ_1) as both need RB_1 and RB_2 and adjusts the number of each RB type within RZ_1 by the maximum number of RBs between $T_1$and T_3. Similarly, RZ_2 is created by T_2 and T_4, and T_5 defines the third type of RZ (RZ_3).

Classification of hardware tasks: Step 2 starts by computing cost D between hardware tasks and RZ types resulting from step 1. Based on RB-models of hardware tasks (T_i) and RZs (RZ_j), cost D is computed as follows according to two cases.

We define by

$$d_{i,j,k} = \alpha_{i,k} - \beta_{j,k}, \quad 1 \le i \le NT, \quad 1 \le j \le NZ,\ 1 \le k \le NP. \tag{3}$$

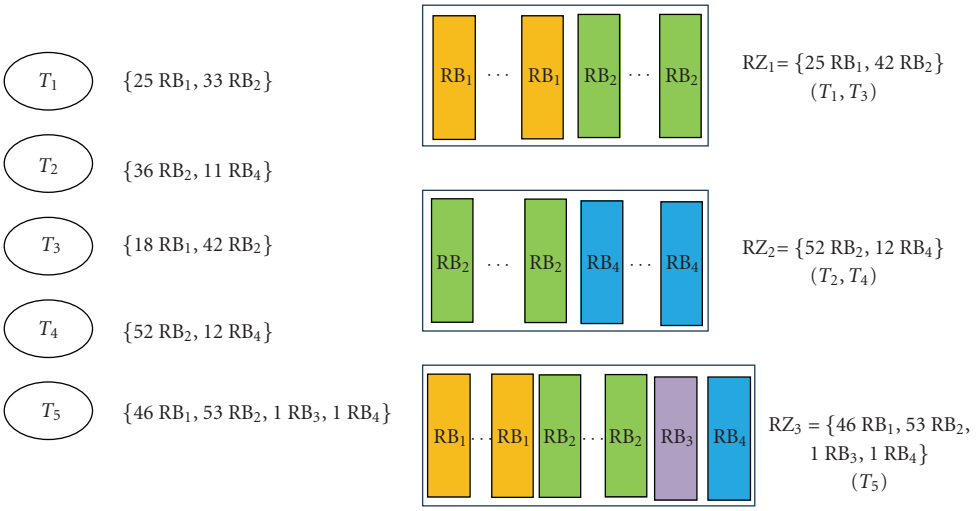
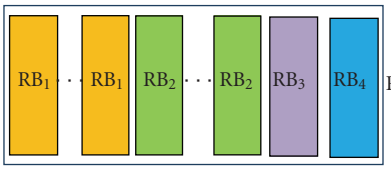


Figure 1: Example of RZ types search.

RZ-type = 0 // RZ types
List-RZ // list of RZ types
n // natural
for all tasks T_i **do**
  // T_i_$RB = X_{i,k}\, RE_k$
  if ((*RZ-type* $\neq 0$) **and** ($\exists\ n, 1 \leq n \leq$ *RZ-type*)/for all $k((X_{i,k} \neq 0$ **and** $Z_{n,k} \neq 0)$ **or** $(X_{i,k} = 0$ **and** $Z_{n,k} = 0)))$ **then**
    // this test checks whether the task matches with an RZ type that already exists in *list-RZ*
    for all *k* **do**
      $Z_{n,k} = \max(X_{i,k}, Z_{n,k})$ // update RB number of RZ_n
    end for
  else
    Increment *RZ-type*
    $RZ_{RZ\text{-}type}$ = *Create new* $RZ(X_{i,k})$ // new type of RZ, $RZ_{RZ\text{-}type} = \{X_{i,k} RE_k\}$
    Insert(*list-RZ*, $RZ_{RZ\text{-}type}$)
  end if
End for

Algorithm 1: Search of RZ types.



RZ$_3$= {46 RB$_1$, 53 RB$_2$, 1 RB$_3$, 1 RB$_4$}
(T_5)

RBCost$_1$= 20, RBCost$_2$= 80, RBCost$_3$= 192, RBCost$_4$= 340

D (T_1, RZ$_3$) = 20 × |25 − 46| + 80 × |33 − 53| + 192 × |0 − 1| + 340 × |0 − 1|
D ($T2$, RZ$_3$) = ∞ (lack of 10 RB$_4$ in RZ$_3$ for T_2)
D (T_3, RZ$_3$) = 20 × |18 − 46| + 80 × |42 − 53| + 192 × |0 − 1| + 340 × |0 − 1|
D (T_4, RZ$_3$) = ∞ (Lack of 11 RB$_4$ in RZ$_3$ for T_4)
D (T_5, RZ$_3$) = 0

Figure 2: Example of computing cost *D* with RZ_3.

Case 1. *for all* $k, d_{i,j,k} \leq 0$, RZ_j contains a sufficient number of each type of RB (RB_k) required by T_i. In this case, cost D is equal to the sum of differences in the number of each RB type between T_i and RZ_j weighted by $RBCost_k$ (see (4))

$$D\left(T_i, RZ_j\right) = \sum_{1 \leq k \leq NP} \text{RBCost}_k \times \left| d_{i,j,k} \right|. \tag{4}$$

Case 2. $\exists\ k, d_{i,j,k} > 0$, the number of RBs required by T_i exceeds the number of RBs in the RZ_j or T_i needs RB_k which is not included in RZ_j. In this case, the cost D between T_i and RZ_j is infinite (see (5))

$$D\left(T_i, RZ_j\right) = \infty \tag{5}$$

Figure 2 illustrates the computing of costs D between the five tasks and RZ_3 described in Figure 1.

As shown in Table 1, step 2 assigns each task to the RZ giving the lowest cost D as described by the third column. For example, T_1 is assigned to RZ_1 since D (T_1, RZ_2) and D (T_1, RZ_3) are superior to D (T_1, RZ_1). Then, by using (6), step 2 computes the workload of each RZ according to this assignment and by using the functional models of hardware tasks

$$\text{Load of } RZ_j = \sum_{i \text{ in } RZ_j} \left(C_i/P_i + Nbr\text{Preemp}_i \times \text{Overhead}_{j,i}/P_i \right),$$
$$\text{Overhead}_{j,i} = \text{Config}_j + \text{Context}_i. \tag{6}$$

The overhead $Overhead_{j,i}$ is the sum of $Config_j$ corresponding to each RZ_j and $Context_i$ (save and load) common for all tasks T_i. $Config_j$ corresponds to the configuration overhead to place RZ_j on the target technology. We compute this configuration overhead by making the floorplan of each RZ_j on the chosen device and by conducting the whole partial reconfiguration flow up to the creation of partial bitstream. According to the configuration frequency and the configuration port, the $Config_j$ is determined by

$$\text{Config}_j = \frac{\text{size of bit stream}}{(\text{Configuration frequency} \times \text{configuration port width})}. \tag{7}$$

For example, the workload of RZ_1 resulting from T_1 and T_3 is 66%. The last column in Table 1 gives costs D of the other tasks not assigned to the RZ.

We notice an overload in RZ_2 caused by the workloads of execution of T_2 and T_4 as well as by their overheads. This overload is resolved during Step 3.

Decision of increasing the number of RZs: Step 3 takes place only when an overload within some RZs is detected in Step 2 and is achieved by Algorithm 2. Step 3 aims to lighten the overload in RZs by conducting the migration of task execution sections to nonoverloaded RZs before resorting to the solution of increasing the number of overloaded RZs. Hence, for each task, we search all the possible combinations of task execution sections. For each overloaded RZ, Algorithm 2 searches all the nonoverloaded RZs that could accept at least one of its assigned tasks; that is, $D \neq \infty$. Then, Algorithm 2 checks task by task the possibility of migration of an execution section or a combination of execution sections of the current task in order to reduce the overload of its RZ by respecting the workload of the nonoverloaded receiving RZ. In the worst case, the complexity of Algorithm 2 is O ($M * N * NTM * TS$), where M denotes the number of overloaded RZs, N is the number of nonoverloaded RZs, NTM is the maximum number of tasks assigned to an overloaded RZ and TS the maximum number of execution section combinations for a task assigned to an overloaded RZ.

Step 3 groups the workloads of overloaded RZs in *L1* (line 7) and the workloads of nonoverloaded RZs in *L2* (line 7). Step 3 goes throughout the RZs in *L1* to resolve their overloads independently (line 11). Step 3 uses nonoverloaded RZs in *L2* to lighten the workloads of RZs in *L1* (line 15). This step searches the nonoverloaded RZ in *L2* that gives finite cost D with at least one task assigned to the overloaded RZ during Step 2 (line 19). Once Step 3 finds the set of tasks that could be executed in the nonoverloaded RZ, it balances the workloads between both RZs by respecting the tasks' preemption points (line 22–line 37). If the overload persists in the RZ of *L1*, the algorithm decides adding other instances of this RZ up to workload of RZ?(line 42). When the processed nonoverloaded RZ_n do not affect the added number of overloaded RZ_m, Step 3 reinitializes their workloads to their values before dealing with the overload of RZ_m(line 43).

Without any loss of generality, our proposed strategy of resource management includes the main functions of generic placement: partitioning and fitting, which are fulfilled by the two following levels.

3.2. Level 2: Partitioning of Reconfigurable Physical Blocs on the Target Technology. Level 2 takes the types of RZs provided by level 1 as inputs and searches all the possible locations for them on the target device. These locations, called Reconfigurable Physical Blocs (RPB), are partitioned on the specified Reconfigurable Regions (RR) delimited in the target device. The RPBs are depicted by their RB-model as presented in

$$RPB_{i\_}RB = \{\gamma_{i,k}\ RB_k\}, \quad \gamma_{i,k} \text{ is natural}, \quad 1 \leq i \leq NR, 1 \leq k \leq NP. \tag{8}$$

The RPBs must contain all the types of RBs required by the RZ type. The number of RBs in RPBs is greater than or equal to the number of RBs in RZs. Figure 3 shows an example of RPBs partitioned in RR_1 which are associated to an RZ requiring two RB_1 and one RB_3. The RPBs are presented by the five dotted rectangles.

3.3. Level 3: Two-level Fitting. Level 3 consists of two independent sublevels. The first sublevel ensures the fitting of

Table 1: Example of final results of Step 2.

RZ types	RZ resources	Assigned tasks T_i (D) and RZ workloads	Costs D $T_i(D)$
RZ_1	{25 RB_1, 42 RB_2}	T_1 (720), T_3 (140) **66**%	$T_2(\infty)$, $T_4(\infty)$, $T_5(\infty)$
RZ_2	{52 RB_2, 12 RB_4}	T_2 (1620), T_4 (0) **137**%	$T_1(\infty)$, $T_3(\infty)$, $T_5(\infty)$
RZ_3	{46 RB_1, 53 RB_2, 1 RB_3, 1 RB_4}	T_5 (0) **35**%	$T_1(2552)$, $T_2(\infty)$ $T_3(1972)$, $T_4(\infty)$

$Load_m$: the load (%) of overloaded RZ_m
$Load_n$: the load (%) of nonoverloaded RZ_n
$Load_{n,i}$: the load (%) of nonoverloaded RZ_n after adding a section of execution of T_i
$Section_i$: the list of possible execution sections of task T_i determined by its preemption points
Exe_i: execution section of T_i
p, q, r, j, i, l: naturals
$L1$ = {loads of overloaded RZ_j}; $L2$ = {loads of nonoverloaded RZ_j}
$L3$: list of tasks
Sort $L1$ in descending order
Sort $L2$ in ascending order, in case of equality. Sort $L2$ in ascending order according to confi-guration overhead
for $p = 1$ **to** sizeof($L1$) **do**
  $RZ_m = L1(p)$
  $Load_m$ = load(RZ_m)
  $q = 1$
  while $q \leq$ sizeof($L2$) **and** $Load_m > 100$ **do**
    $RZ_n = L2(q)$
    $Load_n$ = load (RZ_n)
    //Search Ti from RZ_m to migrate to RZ_n
    if$\exists$ $\{T_i\}$ assigned to $RZ_m/D(T_i, RZ_n) \neq \infty$ **then**
      Sort $\{T_i\}$ in ascending order according to $D(T_i, RZ_n)$ in $L3$
      $r = 1$
      while ($r \leq$ size of ($L3$)) **and** ($Load_m > 100$) **do**
        $\tau_i = L3(r)$
        $l = 1$
        // checking the possibility of relocation of the sections of T_i by respecting the load of RZ_n
        while $l \leq$ size of ($Section_i$) **and** $Load_m > 100$ **do**
          Select the first execution section Exe_i and discard it from $Section_i$
          $Load_{n,i} = Load_m + Exe_i/P_i + Overhead_{n,i}/P_i$
          if $Load_{n,i} \leq 100$ **then**
            // Migration of Exe_i from RZ_m to RZ_n is accepted
            $Load_m = Load_m$-Exe_i/P_i-$Overhead_{m,i}/P_i$ // Removing Exe_i from RZ_m
            $Load_n = Load_{n,i}$// Migration of Exe_i to RZ_n
          end if
          l++
        end while
        r++
      end while
    end if
    q++
  end while
  if $Load_m > 100$ **then**
    New $RZ_m^*(\lceil Load_m/100 \rceil - 1)$// Adding new RZ_m
    Reinitialize the load of $\{RZ_n\}$ when it does not affect the number of added RZ_m
  end if
end for

Algorithm 2: Decision of increasing the number of RZs.

RZs on the most suitable nonoverlapped RPBs in terms of resource efficiency. The second sublevel performs the mapping of tasks to RZs according to their preemption points by respecting the workload of each RZ and guaranteeing the total execution of each task. Such mapping essentially promotes the solution giving the lowest overhead and lowest cost D. The second fitting sublevel provides an execution unit for each task; consequently, there is no longer the issue of task rejection. The mapping of tasks to RZs is strongly based on dynamic partial reconfiguration. This latter concept enables

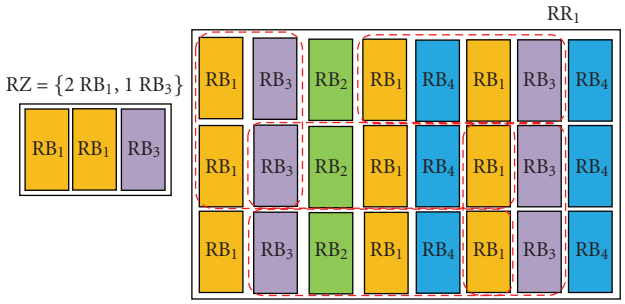


Figure 3: Example of partitioning RR_1 into RPBs.

multitasking as well as execution of several hardware tasks on the same RZs. In fact, dynamic partial reconfiguration allows the reconfiguration of subareas on the FPGA during runtime without affecting other running tasks.

4. Resolution of the Hardware Task Placement Problem

Previously, in our work presented in [23], we proposed a straightforward method to resolve partitioning and fitting. Nevertheless, with the rapid growth of resources in recent technologies and with the increasing complexity of applications, this exhaustive search is not efficient. In fact, the problem of partitioning/fitting is NP-complete, the search space is immense and the temporal complexity of the execution of our proposed algorithm in [23] is exponential. In this paper, we formulate the partitioning/fitting problem as a constrained optimization problem. Our work is based on the smart nonexhaustive complete method called Branch and Bound [24] which employs efficient techniques for scanning search space and extracting the optimal solution.

4.1. Formulation of Hardware Task Placement as a Constrained Optimization Problem. The problem of partitioning/fitting is modeled as a constrained combinatory optimization problem as it uses discrete solution set, chooses the best solution out of all possible combinations and aims the optimization of multicriteria function. Partitioning/fitting problem is mixed integer problem as it uses some natural and binary variables. This problem is described by the quadruplet (Constants, Variables, Constraints, and Objective Function).

4.1.1. Constants. *NT:* number of tasks constituting the application, *NZ*: number of RZs resulting from level 1 (Offline flow of hardware task classification), *NP*: number of RB types existing in the target technology, D (T_i, RZ_j): the cost D between T_i and RZ_j, $RBCost_k$: the cost of each RB type.

Device Features. *Device_Width*: the width of the device, *Device_Height*: the height of the device, *Device_RB*: the RB-model of the device.

Task features. C_i: the Worst Case Execution Time (WCET) of T_i, P_i: the period of T_i, $Preemp_{i,l}$: the preemption point l of T_i, $NbrPreemp_i$: the number of preemption points in T_i, $Context_i$: the functional overhead for preempting and resuming T_i, $Overhead_{j,i}$: the full overhead for execution of task T_i on RZ_j, T_i_RB: the RB-model for T_i.

RZ features. RZ_j_RB: the RB-model for RZ_j, $Config_j$: the configuration overhead for RZ_j in target technology.

4.1.2. Variables

RPB_j Features for Each RZ_j. X_j: the abscissa of the upper left vertex of RPB_j, Y_j: the ordinate of the upper left vertex of RPB_j, $WRPB_j$: the abscissa of the upper right vertex of RPB_j, $HRPB_j$: the ordinate of the bottom left vertex of RPB_j, RPB_j_RB: the RB-model of the RPB_j constructed by the above coordinates.

The Task Preemption Points. Under the task functional model, it is assumed that the preemption points$l$of each task T_i are known. $PreempUnicity_{j,i,l}$: Boolean variable controls whether the mapping of $Preemp_{i,l}$ of T_i is performed on RZ_j. It is equal to 1 when $Preemp_{i,l}$ is mapped to RZ_j. $PreempTask_{j,i,l}$: each preemption point assigned to RZ_j is taken from the predefined preemption points of each task(see(9)).

$$\begin{aligned} &\text{PreempTask}_{j,i,l} = \text{Preemp}_{i,l} \times \text{PreempUnicity}_{j,i,l} \\ &\forall\ 1 \le i \le NT, \quad 1 \le l \le N\text{brPreemp}_i, 1 \le j \le \text{NZ}. \end{aligned} \tag{9}$$

$SumPreemp_{j,i}$: The sum of preemption points of T_i performed within $RZ_j$is expressed by

$$\begin{aligned} \text{SumPreemp}_{j,i} = &\sum_{\substack{1 \le l \le Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} 1 \\ &\forall\ 1 \le i \le NT, \quad 1 \le j \le NZ. \end{aligned} \tag{10}$$

$OccupationRate_{j,i}$: The full occupation rate of T_i in RZ_j resulting from the mapping of preemption points of T_i to RZs. The occupation rate is computed as expressed in

$$\begin{aligned} &\text{OccupationRate}_{j,i} \\ &= \sum_{\substack{1 \le l < Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} \left(\text{Preemp}_{i,l+1} - \text{Preemp}_{i,l}\right) \\ &\quad + \left(C_i - \text{Preemp}_{i,Nbr\text{Preemp}_i}\right), \\ &\text{if PreempUnicity}_{j,i,Nbr\text{Preemp}_i} \ne 0 \\ &\quad \sum_{\substack{1 \le l < Nbr\text{Preemp}_i \\ \text{PreempUnicity}_{j,i,l} \ne 0}} \left(\text{Preemp}_{i,l+1} - \text{Preemp}_{i,l}\right), \\ &\quad \text{else.} \end{aligned} \tag{11}$$

AverageLoad: The average of RZ workloads obtained after task mapping is calculated by

$$\text{AverageLoad} = \left(\sum_{\substack{1 \le i \le NT \\ 1 \le j \le NZ}} \left(\text{OccupationRate}_{j,i}/P_i + \text{SumPreemp}_{j,i} \times \text{Overhead}_{j,i}/P_i \right) \right) /NZ. \tag{12}$$

4.1.3. Constraints

Heterogeneity Constraint. The RZs must be fitted on RPBs containing a sufficient number of their required types of RBs. This constraint must be respected during partitioning and fitting of RZs. The heterogeneity constraint is formulated by

$$\beta_{j,k} \le \sum_{\substack{X_j \le m \le WRPB_j \\ Y_j \le n \le HRPB_j}} \sum_{\text{device\_RB}[m][n]=RB_k} 1, \quad \forall\ 1 \le j \le NZ, 1 \le k \le NP$$
$$RZ_{j\_RB} = \left\{ \beta_{j,k}\ RB_k \right\}, \quad 1 \le j \le NZ,\ 1 \le k \le NP. \tag{13}$$

Nonoverlapping between RPBs. As expressed by (14), this constraint restricts the fitting of RZs on nonoverlapped RPBs

$$X_q > WRPB_j \text{ or } X_j > WRPB_q \text{ or } Y_q > HRPB_j \text{ or } Y_j > HRPB_q \quad \forall\ j \ne q, 1 \le j, q \le NZ. \tag{14}$$

Nonoverload in RZs. As mentioned in (15), the non-overload in RZs must be respected during mapping of tasks

$$\sum_{1 \le i \le NT} \left(\text{OccupationRate}_{j,i}/P_i + \text{SumPreemp}_{j,i} \times \text{Overhead}_{j,i}/P_i \right) \le 100\%, \forall\ RZ_j. \tag{15}$$

Infeasibility of Mapping for Preemption Points. This constraint prohibits the mapping of preemption points of tasks to RZs giving infinite cost D (see (16)).

$$\text{PreempUnicity}_{j,i,l} = 0 \text{ when } D\left(T_i, RZ_j\right) = \infty, \quad \forall\ 1 \le i \le NT, \quad 1 \le l \le Nbr\text{Preemp}_i, 1 \le j \le NZ. \tag{16}$$

Uniqueness of Preemption Points. This constraint claims that each preemption point l of T_i must exist on unique RZ_j and guarantees the achievement of task execution as well as the elimination of task rejection (see (17))

$$\sum_{1 \le j \le NZ} \text{PreempUnicity}_{j,i,l} = 1, \quad \forall\ 1 \le i \le NT, \quad 1 \le l \le Nbr\text{Preemp}_i. \tag{17}$$

Domains of RPB Coordinates. Equation (18) defines the allowed domain of values that can be assigned to RPB coordinates during partitioning

$$1 \le X_j, WRPB_j \le \text{Device\_Width}, \quad 1 \le Y_j, HRPB_j \le \text{Device\_Height}. \tag{18}$$

4.1.4. Minimization Objective Function (F). As with all optimization problems, we have defined a minimization objective function F that helps in selecting the optimal solution. As described by (19), F promotes a solution giving the best values for the two objective subfunctions *MappingFunction* and *PlaceFunction*

$$F = \text{MappingFunction} + \text{PlaceFunction}. \tag{19}$$

PlaceFunction focuses on the subproblem of fitting RZs on the most suitable RPBs after partitioning the target device by respecting the predefined constraints. In (20), *PlaceFunction* evaluates the efficiency of resources after fitting RZs on the selected RPBs. *PlaceFunction* promotes the fitting of RZs on the RPBs that strictly contain the number and type of RBs required by RZs

$$\text{PlaceFunction} = \sum_{\substack{1 \le j \le NZ \\ 1 \le k \le NP}} RB\text{Cost}_k \times \left(\gamma_{j,k} - \beta_{j,k} \right), \quad RPB_j\_RB = \left\{ \gamma_{j,k}\ RB_k \right\}, RZ_j\_RB = \left\{ \beta_{j,k}\ RB_k \right\}, \quad 1 \le j \le NZ, 1 \le k \le NP. \tag{20}$$

MappingFunction focuses on the subproblem of fitting hardware tasks on RZs by respecting the predefined preemption points. *MappingFunction* targets the full exploitation of RZs and their workloads balance. It also aims at mapping tasks to the RZs providing the lowest cost D to optimize resource utilization. Moreover, *MappingFunction* promotes the solution that minimizes the overall overhead and number of task relocations. Thus, *MappingFunction* is expressed by four subfunctions targeting these goals

$$\text{MappingFunction} = Map1 + Map2 + Map3 + Map4. \tag{21}$$

Map1 focuses on the RZ workloads by means of (22). Its first expression evaluates whether the RZs are fully exploited. While minimizing this expression, the RZ workloads approach 100% of their exploitation. Its second expression computes the variance of the RZ workloads

towards the obtained average workload. Minimization of this second expression ensures load balancing between RZs

$$Map1 = \sum_{1\le j\le NZ}\left(100\% - \left(\sum_{1\le i\le NT}\mathfrak{A}\right)\right) + \sum_{1\le j\le NZ}\frac{\left(\left(\sum_{1\le i\le NT}\mathfrak{A}\right) - \text{Average Load}\right)^2}{NZ}. \tag{22}$$

Where

$$\mathfrak{A} = \frac{\text{OccupationRate}_{j,i}}{P_i} + \frac{\text{SumPreemp}_{j,i}\times\text{Overhead}_{j,i}}{P_i} \tag{23}$$

Map2 computes the overhead in (24) resulting from the mapping of task preemption points to the RZs. *Map2* takes into account all the possible preemption points, even when two successive preemption points of one task are mapped to the same RZ, for obtaining the worst case overhead. In fact, the scheduler could preempt a task on these successive preemption points in the same RZ in favor of a higher priority task. Minimizing *Map2* promotes the solutions that map the tasks to the RZs providing the lowest configuration overhead

$$Map2 = \sum_{\substack{1\le i\le NT\\ 1\le j\le NZ}}\text{SumPreemp}_{j,i}\times\text{Overhead}_{j,i}. \tag{24}$$

By minimizing *Map3* in (25), we promote a solution that maps tasks by a high occupation rate to the RZs giving the lowest cost *D*. The benefit of this minimization is optimizing the use of available resources in the technology. Indeed, these *D* costs between tasks and RZs reveal the rate of resource waste. Moreover, these *D* costs consider the weight of each resource in terms of three parameters which are: its frequency on the technology, the importance of its functionality, and its power consumption. Our objective is to minimize the utilization of these costly resources whenever possible. Hence, the more we promote mapping of tasks with high occupation rates to the RZs giving the lowest cost *D*, the more we optimize resource utilization

$$Map3 = \sum_{\substack{1\le i\le NT\\ 1\le j\le NZ}} D\left(T_i, RZ_j\right)^2\times\text{OccupationRate}_{j,i}{}^2/4, - D\left(T_i, RZ_j\right)\times\text{OccupationRate}_{j,i}. \tag{25}$$

Map4 computes in (26) the total number of task relocations obtained after such mapping. Although the migration of tasks between RZs solves the conflicts between tasks on the same RZ, minimizing the number of migrations optimizes the number of preemptions and overhead

$$Map4 = \sum_{\substack{1\le i\le NT\\ 1\le j\le NZ}}\ \sum_{\substack{1\le l<Nbr\text{Preemp}_i\\ \text{PreempUnicity}_{j,i,l}\neq 0,\ \text{PreempUnicity}_{j,i,l+1}=0}} 1 \tag{26}$$

4.2. Branch and Bound Method for Solving the Hardware Task Placement. Our work is based on the global optimization method called Branch and Bound during partitioning of resource space and fitting RZs and tasks. The method of branch and Bound consists in enumerating all the possible solutions in an intelligent way by relying on the features of the specified problem. This technique succeeds in eliminating all the partial solutions that do not lead to the optimal solution. The Branch and Bound method relies on a predefined bound function which is our objective function *F* to make boundary on solutions to be excluded or to be kept as potential solutions. The performance of the Branch and Bound method depends on the quality of this function. Branch and Bound is adapted to mixed integer linear and non-linear programming.

Initially only one subset of solutions exists namely, the root subset that contains all the solutions for the problem. The unexplored subsets are represented as nodes in a dynamically generated search tree. Each iteration on Branch and Bound processes one such node. The iteration has three components: the selection of the node to process, branching and the bound function calculation of the ramified nodes. For each node, the bound function calculation considers the best fitting for the remaining RZs or tasks without constraints. Whereas, only nodes respecting predefined constraints are only ramified after node selection. Our strategy for selecting the next node is based on the value of the bound function *F* of the node. We start by the node giving the best bound function *F* at the current level and we use the strategy of *Depth First Search* (*DFS*). This strategy starts by exploring, at each level, the node giving the best *F* until obtaining the complete solution which gives the last best *F*. Then, the process is repeated for the next best node on the last visited level if its *F* does not exceed the last best *F*. In this case, we branch this node and compute the bound functions of its ramified nodes and compare them to the last best *F*. If *F* of these partial solutions is greater than or equal to the last best *F*, they are rejected. If not, they are kept and the best partial solution is selected to be processed by Branch and Bound. Once a new complete solution is founded, the method checks whether its *F* optimizes the last best *F*. When an optimization is detected, the last best *F* is updated. The process is repeated for all the next best nodes in each level as described above. Algorithm 3 describes the Branch and Bound method applied on our problem.

Partitioning resource space and fitting RZs can be resolved independently from task mapping. In the subproblem of partitioning resource space and fitting RZs, *PlaceFunction* is employed as the bound function *F* in Algorithm 3. While in the task mapping subproblem, *F* corresponds to *MappingFunction*. To fit RZs on their suitable RPBs, the two subproblems of partitioning the RPBs on the target device and fitting RZs on the selected RPBs are performed in parallel. In fact, the resolution starts by partitioning the RPBs corresponding to the RZ root (for example RZ_1) with respect to the heterogeneity constraint. This partitioning is done randomly by assigning all potential values to the RPB coordinates and the RZ root is fitted onto the RPB that gives the best *PlaceFunction*. After selection

```
c: narural // the counter on branched nodes
b: natural // the number of branched nodes provided by the current node
Best F = +00,
Live = {(Node in root level, F(node in root level))}// the set of nodes to be processed
while Live ≠ ⊘ do
   Select the node N from Live to be processed which gives the best F by using DFS
   Live = Live \ {(N, F(N))}
   if F(N) = F(X) for a feasible complete solution X and F (X) < Best F then
      Best F = F(X)
      Solution =X // A complete solution is founded
   else if F(N) ≥ Best F then
      Discard N from processing // partial candidate is rejected
   else
      // partial candidate is kept
      Branch on N generating N1, ..., Nb by respecting the problem constraints
      for c = 1 to b do
         Bound Nc: compute F(Nc) // bound calculation for the node Nc
         Live = Live U {(Nc, F(Nc)}
      end for
   end if
end while
Optimal Solution = Solution, Optimal Value = Best F
```

Algorithm 3: Branch and Bound for hardware task placement [24].

of the best RPB for the RZ root that is, the selected node, this node is branched into all the possible RPBs for RZ_2 by respecting the predefined constraints and the *PlaceFunction* of each ramified node is computed. The first selected node for the RZ_2 level must satisfy the heterogeneity constraint as well as the nonoverlapping constraint. The subset of solutions ramified from this first selected node in RZ_2 level is explored by searching the RPBs of RZ_3. The resolution is carried out similarly until the achievement of exploration branched from the first selected node which gives the last best *PlaceFunction*. The process is repeated for the next best nodes and recursively, the remainder RZs are fitted on RPBs as described for the RZ root by respecting the predefined constraints. During exploration, if a partial solution gives *PlaceFunction* worse than the last best *PlaceFunction*, this partial solution is rejected and its branching is stopped. If not, this partial partitioning/fitting of RZs is kept as a potential partial solution. During this repetitive process, all possible combinations of RPBs respecting the constraints are tested, when the process is achieved, the optimal solution is extracted.

Figure 4 illustrates the running of the Branch and Bound method on the partitioning/fitting of four RZs. The nodes with an X mark present the subsets of solutions that do not contain the optimal solution and with a $\surd$ mark present potential solutions. In the *RZroot level* and RZ_2 *level*, the best node is selected, branched on partial solutions and the *PlaceFunction* of its ramified nodes are computed. As can be noticed, there are two processed best fittings in the RZ_3 *level*. The method starts by the first best node that is, *PlaceFunction3p* and processes this selected node. It is branched into two nodes by the RPBs of RZ_4. After computing the *PlaceFunction* for these ramified nodes, the RPB_1-RZ_4 is selected and a complete solution is obtained. The last best *PlaceFunction* is *PlaceFunction41*. RPB_2-RZ_4 is rejected as it does not optimize *PlaceFunction41*. The next best node in the last visited level not yet processed is RPB_1-RZ_3. The node is kept and branched into two RPBs on the RZ_4 *level*. *PlaceFunction43* optimizes the last best *PlaceFunction* and the solution is complete, thus the optimal solution is obtained by *PlaceFunction43*. The next iteration processes the next best node in the last visited level; partial solution RPB_2-RZ_3. This partial solution is rejected as *PlaceFunction32* exceeds the last best *PlaceFunction*. The other nodes are also not processed as their partial bound function exceeds the last best *PlaceFunction*. Finally, the optimal solution of partitioning/fitting of RZs is obtained by fitting *RZroot* (RZ_1) on RPB_i-*RZroot*, RZ_2 on RPB_2-RZ_2, RZ_3 on RPB_1-RZ_3, and RZ_4 on RPB_3-RZ_4.

The resolution of mapping starts by randomly assigning the preemption points of task root (T_1) to RZs giving finite cost D and computing the bound function of each node. The selected node is the node that gives the most optimal mapping of preemption points according to *Mapping Function.* This node is ramified for new processing for the next task. This ramification must respect the total execution of tasks as well as the non-overload on RZs. Recursively, as task root, the remainder tasks are mapped by satisfying the two previous constraints and computing the *Mapping Function* of their ramified nodes. When a complete solution is obtained, it represents the last best *Mapping Function*. Similarly to partitioning/fitting of RZs, the mapping is resolved in a recursive manner by computing the *Mapping Function* of each partial mapping branched from the selected best nodes and keeping only the ones optimizing the last best *Mapping Function*. The mapping resolution is finished when all the

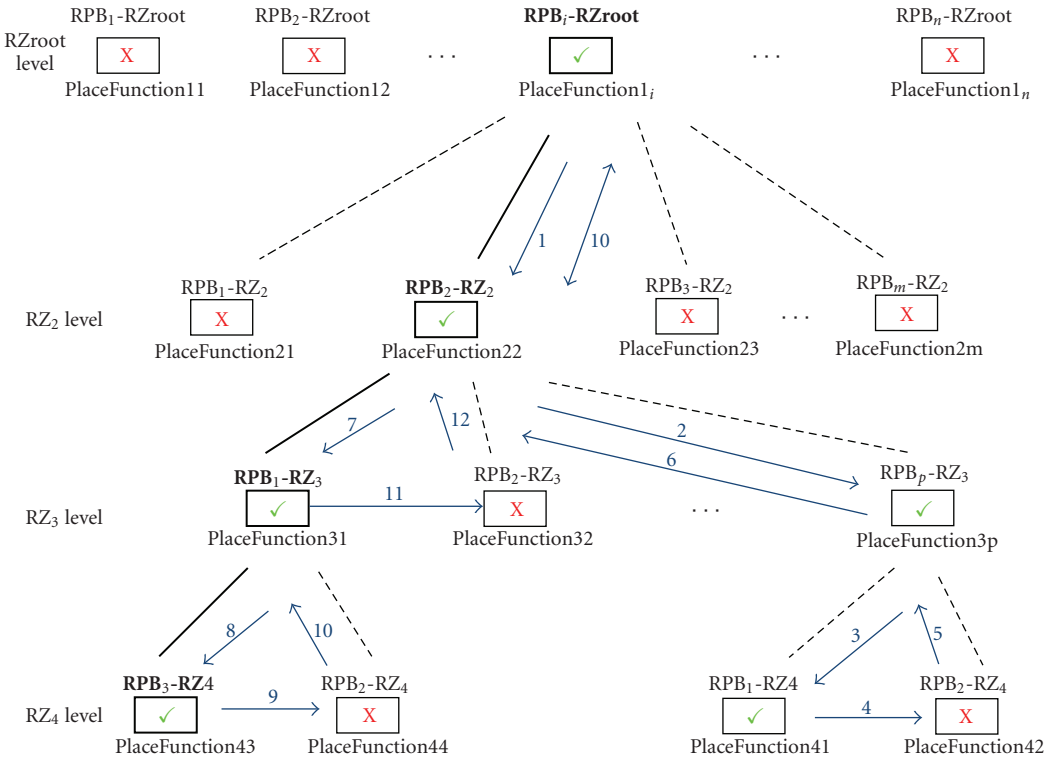


Figure 4: Branch and Bound for partitioning/fitting of RZs.

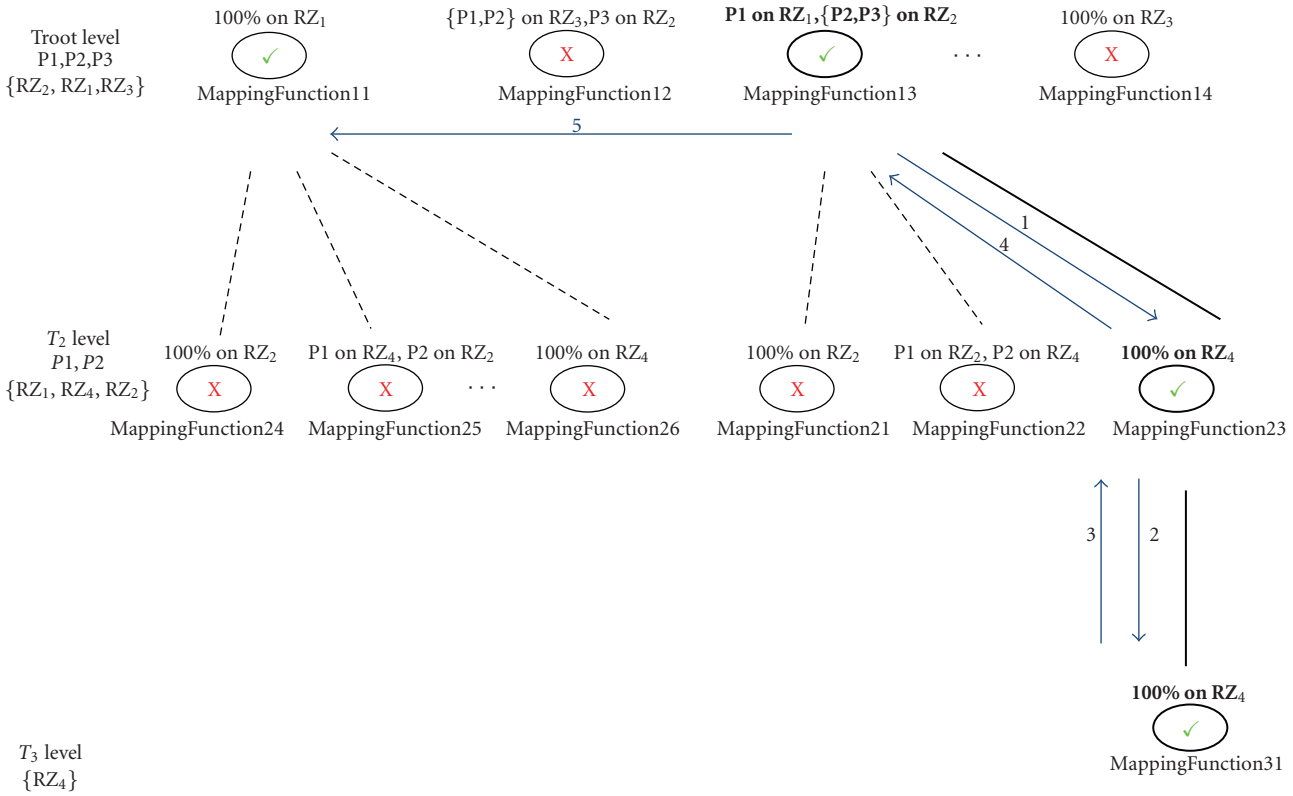


Figure 5: Branch and bound for fitting tasks.

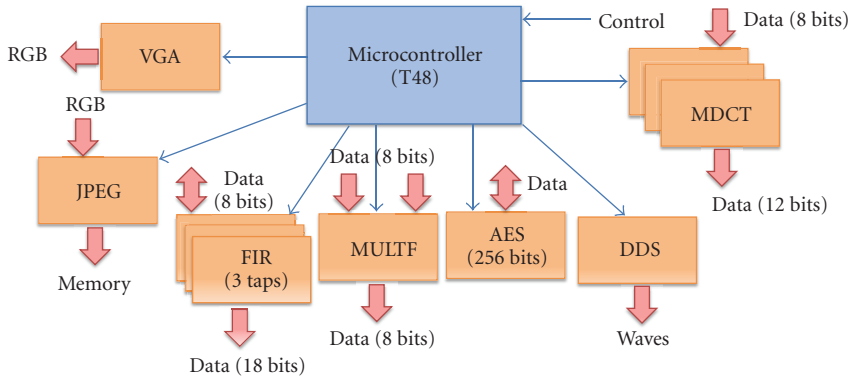


Figure 6: Hardware tasks of the application.

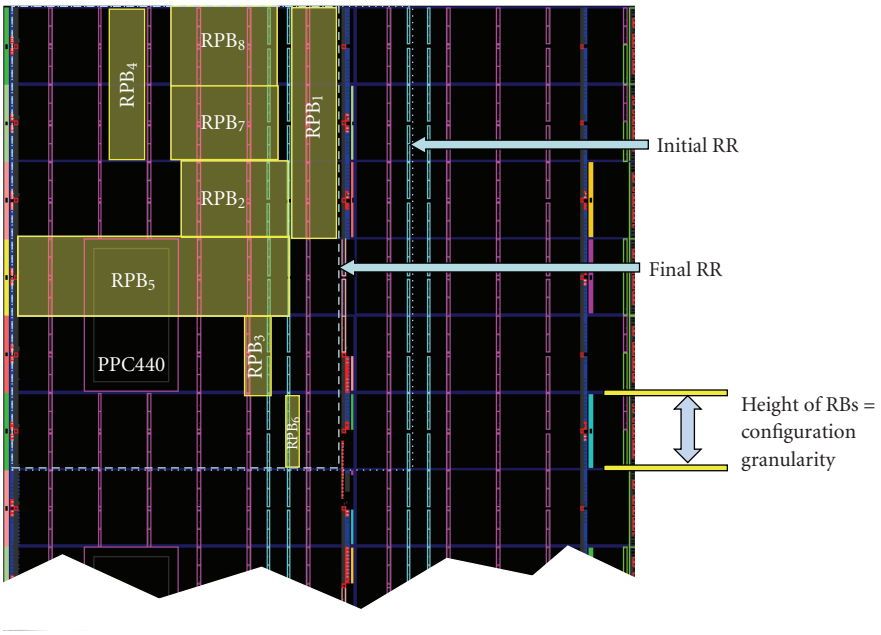


Figure 7: Floorplanning of RPBs on Virtex 5 FX200.

potential mappings of preemption points for all tasks to RZs respecting mapping constraints are checked.

Figure 5 describes the progress of mapping three tasks to four RZs. The set of RZs giving finite cost D is provided for each task. In *Troot level*, the resolution selects the first best node that is, *Mapping Function 13*, after branching it and after computing the *Mapping Function* of the partial solutions, the next iteration is released on T_2 *level*. In T_2 *level*, *MappingFunction23* is the best partial solution, its node is ramified on T_3 *level* and the last best *Mapping Function* is obtained by *MappingFunction31*. The process is repeated for the next best nodes in the last visited level. The nodes *MappingFunction21 and MappingFunction22* are not processed since they exceed the last best *Mapping Function*. The method processes the next best node in the last visited level which is *MappingFunction11*. The branching of this node does not optimize the last best *MappingFunction*. Thus, its ramified nodes are not explored in T_2 *level*. Finally, the optimal solution of mapping hardware tasks to RZs is obtained by fitting the first preemption point P1 of T_1 on RZ_1 and its remaining preemption points P2 and P3 on RZ_2 and by fitting all the preemption points of T_2 and T_3 on RZ_4.

As well-known, in the worst case, the complexity of the classical Branch and Bound method is exponential [25]. In fact, in the worst case, the complexity of the first subproblem: partitioning/fitting RZs is equal to O $((4*B+8*B^2+(NZ\ +1)*B^2)^{NZ-1}+4*B)$ where B denotes a possible combination of RPB coordinates for a given RZ as presented by a node in Figure 4. Thus, B is a possible value assignment for the decision variables X_j, $WRPB_j$, Y_j, $HRPB_j$. The values domain of B is equal to $|\text{domain}_{Xj}|*|\text{domain}_{WRPBj}|*|\text{domain}_{Yj}|*|\text{domain\_}HRPBj|$, which is equal to $(Device\_Width)^{2*}$ $(Device\_Height)^2$. $4*B$ depicts the selection of a node, its removal from the set live, and both following test: the test of final and optimal solution (line 8) and the test of failed solution (line 11). $8*B^2$

Table 2: Hardware task features.

	Instances	RBs	WCET(μs)	Period (μs)	Configuration overhead (μs)	Preemptio points (μs)
MDCT	$\{T_1, T_9, T_{10}, T_{11}, T_{12}\}$	$\{2\ RB_1, 12\ RB_2, 3\ RB_3, 0\ RB_4\}$	40552	416666	1856	10000, 20000, 30000
AES	$\{T_2\}$	$\{4\ RB_1, 7\ RB_2, 1\ RB_3, 1\ RB_4\}$	51540	100000	2185	30000, 40000
DDS	$\{T_3\}$	$\{0\ RB_1, 1\ RB_2, 1\ RB_3, 1\ RB_4\}$	5000	12000	432	none
T48	$\{T_4\}$	$\{5\ RB_1, 4\ RB_2, 0\ RB_3, 0\ RB_4\}$	20000	50000	605	800, 10600, 16300
JPEG	$\{T_5\}$	$\{8\ RB_1, 12\ RB2, 0\ RB_3, 2\ RB_4\}$	350000	416666	2421	200000, 300000
MULTF	$\{T_6\}$	$\{1\ RB_1, 1\ RB_2, 0\ RB_3, 1\ RB_4\}$	5600	10000	491	1400, 2350, 4020, 5170
FIR	$\{T_7, T_{13}, T_{14}\}$	$\{0\ RB_1, 1\ RB_2, 0\ RB_3, 1\ RB_4\}$	300	2000	112	120, 210, 255
VGA	$\{T_8\}$	$\{2\ RB_1, 4\ RB_2, 1\ RB_3, 0\ RB_4\}$	5000	10000	681	1650, 2700

Table 3: The results obtained for Step 1 and Step 2 of Level 1.

	MDCT (T_1,T_9,T_{10},T_{11},T_{12})	AES (T_2)	DDS (T_3)	T48 (T_4)	JPEG (T_5)	MULTF (T_6)	FIR (T_7,T_{13},T_{14})	VGA (T_8)
$RZ_1\{2\ RB_1, 12\ RB_2, 3\ RB_3, 0\ RB_4\}$**57%**	**0**	∞	∞	∞	∞	∞	∞	1024
$RZ_2\{4\ RB_1, 7\ RB_2, 1\ RB_3, 1\ RB_4\}$**338%**	∞	**0**	560	∞	∞	**732**	752	**620**
$RZ_3\{0\ RB_1, 1\ RB_2, 1\ RB_3, 1\ RB_4\}$**45%**	∞	∞	**0**	∞	∞	∞	192	∞
$RZ_4\{5\ RB_1, 4\ RB_2, 0\ RB_3, 0\ RB_4\}$**44%**	∞	∞	∞	**0**	∞	∞	∞	∞
$RZ_5\{8\ RB_1, 12\ RB_2, 0\ RB_3, 2\ RB_4\}$**85%**	∞	∞	∞	1380	**0**	1360	1380	∞
$RZ_6\{0\,RB1, 1\ RB2, 0\ RB3, 1\ RB4\}$**112%**	∞	∞	∞	∞	∞	∞	**0**	∞

is the branching operation by respecting the constraints (line 15) and $(NZ{+}1){*}B^2$ is the bound computation for the branched nodes (line 17) by assigning each already processed RZ to its RPB and the remaining ones to the most suitable RPBs in terms of resource efficiency without constraints and the insertion of this branched nodes into the set live (line 18). Consequently, the complexity of the subproblem in the worst case is $\sim$ O $(((Device\_Width)^{4*}(Device\_Height)^4)^{NZ})$. In the worst case, the complexity of the second subproblem: task fitting is O $((4{*}B{+}B^{2*}\ (2{+}NT{+}6{*}NT\ {*}NZ))^{NT-1}{+}4{*}B)$ where B denotes a possible fitting of preemption points on RZs RZ_j for a given task as associated to a node in Figure 5. Thus, B is a possible value assignment for the decision variables $PreempUnicity_{j,i,l}$. The values domain of B for a given task T_i is $|\text{domainPreempUnicity}\,j, i, l|$ which is equal to: $|\{0,1\}|^{\text{MaxPreemp}^*NZ} = 2^{\text{Max Preemp}^*NZ}$ and we consider the maximum number of preemption points is $MaxPreemp$. After the branching of the selected node, we compute the bound of its branched candidate by considering that the remaining tasks which are not yet processed are mapped with 100% to their optimal RZs in terms of distance and configuration overhead. Consequently, in the worst case, the complexity of the second sub-problem by using the Branch and Bound search is $\sim\ O(2^{\text{MaxPreemp}^*NT^*NZ})$. Thus, in the worst case, the search by Branch and Bound algorithm grows exponentially with NT and NZ. This complexity is expected as the placement of hardware tasks on the heterogeneous reconfigurable devices is NP-complete problem. Currently, many enhancements on Branch and Bound algorithm are conducted as in [26] which could reduce the exponential complexity of some problems to logarithmic complexity. But despite this possible exponential complexity, compared to an exhaustive exact method, the Branch and Bound method immensely lightens the search space. Effectively, the search tree branches are discarded when the current bound function exceeds the last best bound function. The Branch and Bound ensures complete resolution of placement problem with a performance better than or equal to that of complete exhaustive method in terms of resolution speed and storage space. With this smart method, we ensure a full combination of RZ fittings as well as task mapping which solves the problem of task rejection confronted in the previous works of placement. Unlike the approached methods such as metaheuristics [27], this method is complete and affords an optimal solution. In fact, within these metaheuristics, the computation in each generation is rather complicated and time consuming and the number of generations must be increased to ensure a more favorable solution. The quality of the obtained solution in metaheuristics is conditioned by the best choice of the initial generation.

5. Application and Results

To investigate the influence of our proposed method for hardware task placement, we implemented an application composed of several hardware tasks taken from opencores [28]. We have chosen examples of hardware tasks that are frequently used in recent embedded systems performing video and audio applications. Our application features hardware tasks of varied sizes and of heterogeneous resources. The hardware tasks are guided by an application manager; the microcontroller. We do not consider the control overhead between the microcontroller and the other hardware tasks. During the design of this application, we synthesized the resources of each hardware task by the ISE 11.3 Xilinx tool. We defined the configuration overheads of the obtained RZs by performing the partial reconfiguration flow by means of the Planahead 11.3 Xilinx tool.

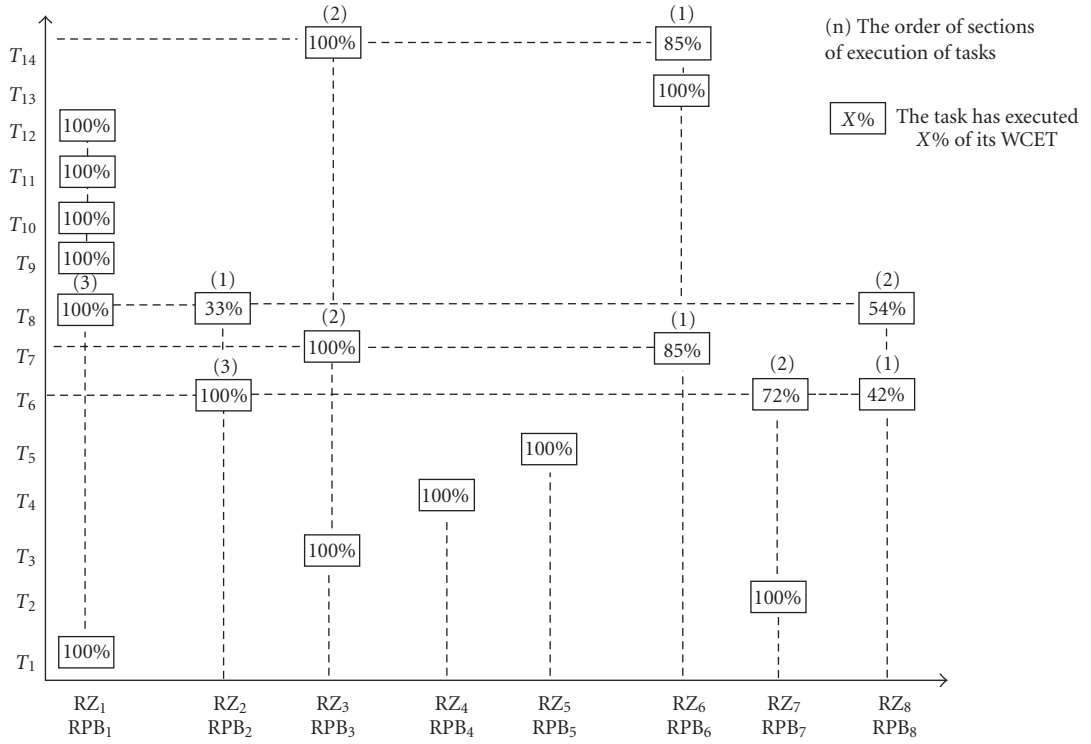


Figure 8: Resolution of fitting tasks on RZs.

Table 4: Coordinates of obtained RPBs.

	X_j	WRP_j	Y_j	$HRPB_j$
RPB_1 for RZ_1	40	45	1	3
RPB_2 for RZ_2	25	38	3	3
RPB_3 for RZ_3	33	36	5	5
RPB_4 for RZ_4	14	18	1	2
RPB_5 for RZ_5	1	39	4	4
RPB_6 for RZ_6	39	40	6	6
RPB_7 for RZ_7	24	37	2	2
RPB_8 for RZ_8	24	37	1	1

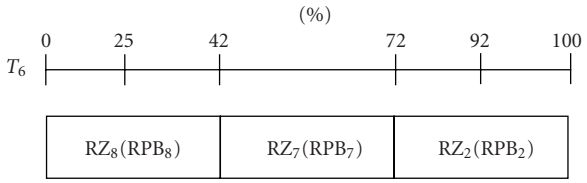


Figure 9: Mapping of preemption points of T_6.

5.1. Application. Figure 6 shows the hardware tasks included in the application. The core of the application is the microcontroller T48. The microcontroller guides the other hardware tasks either for speeding up the computing such as FIR and MULTF or for performing complicated processing such as JPEG compression. Consequently, there are three categories of hardware tasks.

(i) Application manager.

(a) Microcontroller T48: Hardware task configuration and data flow synchronization.

(ii) Fine-grained hardware tasks (for speeding up microcontroller).

(a) FIR: Performs 1000 FIR (Finite Impulse Response) filters. Each FIR features 3 taps, 8bit-input data and 48bit-coefficients.

(b) MULTF: Performs 1000 floating point multiplications between two vectors of 8 bits. The exponent precision and mantissa precision are 3 and 4 respectively. These multiplications are pipelined with the latency of 8 clock cycles.

Table 5: Comparison of number of RBs between RZs and RPBs.

	RB_1	RB_2	RB_3	RB_4	$\triangle$
RPB_1	3	12	3	0	1 RB_1
RPB_2	4	7	2	1	1 RB_3
RPB_3	0	2	1	1	1 RB_2
RPB_4	6	4	0	0	1 RB_1
RPB_5	8	12	3	2	3 RB_3
RPB_6	0	1	0	1	0
RPB_7	4	7	2	1	1 RB_3
RPB_8	4	7	2	1	1 RB_3

Table 6: *Placement results comparison.*

Placement Methods	Task rejection	Resource utilization	Configuration overhead	Complexity
3-level offline placement				
–RR composed of 55 $*$ 6 of heterogeneous RBs –14 heterogeneous hardware tasks –Configuration frequency: 100 MHz	0	36%	11%	$O(((Device\_Width)^4 \times (Device\_Height)^4)^{NZ}) + O(2^{MaxPreemp \times NT \times NZ})$
Metaheuristics				
Simulated annealing [4] –Device cells: 100 × 100 –100 tasks –Task size: 17 × 17	13%	—	—	—
Genetic algorithm [10] –Device cells: 64 × 64 –Sets of 3,000 tasks –Task size: 32 × 32	45%	60%–80%	—	—
Heuristics				
KAMER-BF [4] –Device cells: 100 × 100 –2048-16384 tasks –Task size: 17 × 17	13%–18%	—	—	$O(n \log(n))n$: number of tasks on the device
Blind compaction [1] –Device cells: 64 × 64 –10.000 tasks –Task size: 32 × 32	—	60%	—	$O(n^2)n$: number of tasks on the device
IM [8] –Device cells: 100 × 100 –13 task sets each of 1000 tasks –Task size: 2 × 2 − 20 × 20	10%	—	—	—
Configuration Reuse + Configuration Prefetch [11] –Device cells: 4 tiles –JPEG decoder, MPEG encoder –Configuration frequency: 50 MHz	—	—	5% (JPEG), 18% (MPEG)	$O(NT\ x\ N) + O(N^2)NT$: number of tiles N: number of threads
LFD [32] –Virtex II Pro –2 JPEG + 1 MPEG –Configuration frequency: 100 MHz	—	—	8%	—

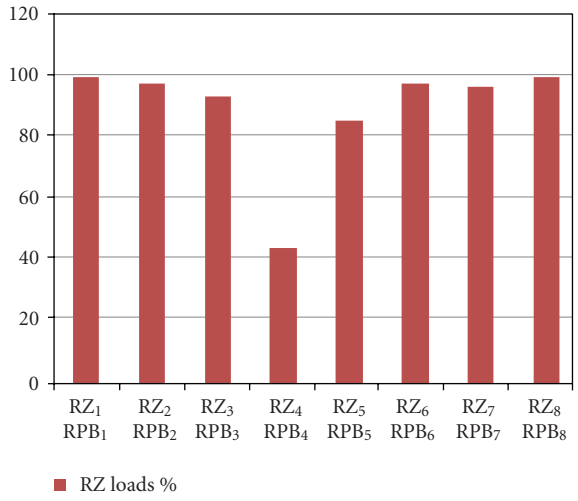


Figure 10: RZ workloads after task mapping.

(iii) Coarse-grained hardware tasks.

(a) MDCT: Computes Modified Discrete Cosine Transform.

(b) AES: (Advanced Encryption Standard): Performs encryption of blocs of 128 bits with 256bit-key.

(c) DDS: (Direct Digital Synthesizer): Creates sinusoidal waves programmable with frequency and phase on-time.

(d) JPEG: Performs hardware compression of 24 frames per second.

(e) VGA: Drives VGA monitors with an 800x600 resolution, it can display one picture on the screen either of chars or color waveforms or color grid.

The features of hardware tasks and their instances are presented in Table 2. These hardware tasks are characterized by considering the resource area in Virtex 5 technology [29]. Virtex 5 contains four main types of resources CLBL, CLBM, BRAM, and DSP. The RBs are vertical stacks composed of the same type of resources and match the reconfiguration granularity. Hence, RB_1 is 20 CLBMs, RB_2 is 20 CLBLs, RB_3 is 4 BRAMs, and RB_4 is 8 DSPs. We have assigned 20, 80, 192, and 340 as *RBCost,* respectively, for RB_1, RB_2, RB_3, and RB_4. Configuration overhead is determined as described in (7) by considering that each task defines an RZ. After synthesizing hardware tasks by the ISE tool, they are modeled under their RB-model reported in [3]. The partial reconfiguration flow dedicated by the Planahead tool enables the floorplanning of hardware tasks on the chosen device to create their bitstreams independently for estimating configuration overheads. The estimation of configuration overheads considers the best case fitting of each task, as we believe the subproblem of partitioning/fitting RZs is solved efficiently and independently from the subproblem of task mapping. We rely on parallel 8bit-width configuration ports and use 100 MHz as the configuration clock frequency. Preemption points are determined arbitrarily according to the granularity of hardware tasks and their WCET. For all tasks, we consider the first preemption point is equal to 0 μs.

5.2. Results. We applied the three levels of our resource management on the application and obtained the following results.

5.2.1. Level 1 Results. Step 1 creates 6 types of RZ depicted in Table 3. RZ_1 is created by MDCT and VGA tasks, RZ_2 by AES, RZ_3 by DDS, RZ_4 by T48, RZ_5 by JPEG, and MULTF and RZ_6 by FIRs. If the RBs of one type of RZ are not constructed from the same task (i.e. there exist some RBs created by other tasks), the configuration overhead corresponding to this RZ must be recomputed as described in (7) before performing Step 2. In our application, the RBs of all RZs are created by one task. Hence, the RZ configuration overhead is provided by the predefined features of hardware tasks in Table 2.

During Step 2, the D costs between tasks and RZs are computed; in Table 3, the column specific for each task and its instances shows the values of obtained D costs. Thereafter, step 2 calculates the workloads of obtained RZs by assigning to each RZ the tasks giving lowest cost D as mentioned by the bold numbers. WorkLoad values are presented in the first column of Table 3. For example, the workload of RZ_2 is computed by assigning the hardware tasks AES, MULTF, and VGA. We detected an overload in RZ_2 and RZ_6. The overloads on these RZs are the result of the assigned hardware task execution time as well as the overheads, especially the configuration overheads of the RZs. These overloads are dealt with in step 3.

To resolve these overloads, step 3 adds two other RZs having the same type of RZ_2 since the other RZs cannot totally resolve its overload. In fact, the only possible migration is performed by T_8 on its second 1650 μs preemption point which loads off RZ_2 up to 299%. Whereas, the overload of RZ_6 could be resolved by performing the migration of two tasks among T_7, T_{13}, and T_{14} on RZ_3 on their second preemption points that is, 120 μs, since RZ_3 is the least loaded. Consequently, the final number of RZs is equal to 8: RZ_1, $3 \times RZ_2$ (RZ_2, RZ_7, RZ_8), RZ_3, RZ_4, RZ_5, and RZ_6.

5.2.2. Level 2 and Level 3 Results. Among all the available solvers [30], our work is based on the AIMMS environment [31] relying on powerful solvers. AIMMS environment has independently resolved the two subproblems: partitioning/fitting of RZs and fitting of tasks on RZs based on the Branch and Bound method. For the subproblem of partitioning/fitting of RZs, we used the Mixed Integer Programming model and for task fitting we employed Mixed Integer Nonlinear Programming model. At the end of resolution on CPU of 2 GHz with 2 GB of RAM, each RZ is fitted on its most suitable RPB and each preemption point of each task is mapped to a unique RZ. The resolution respects the consistency of constraints and extracts the optimal solution.

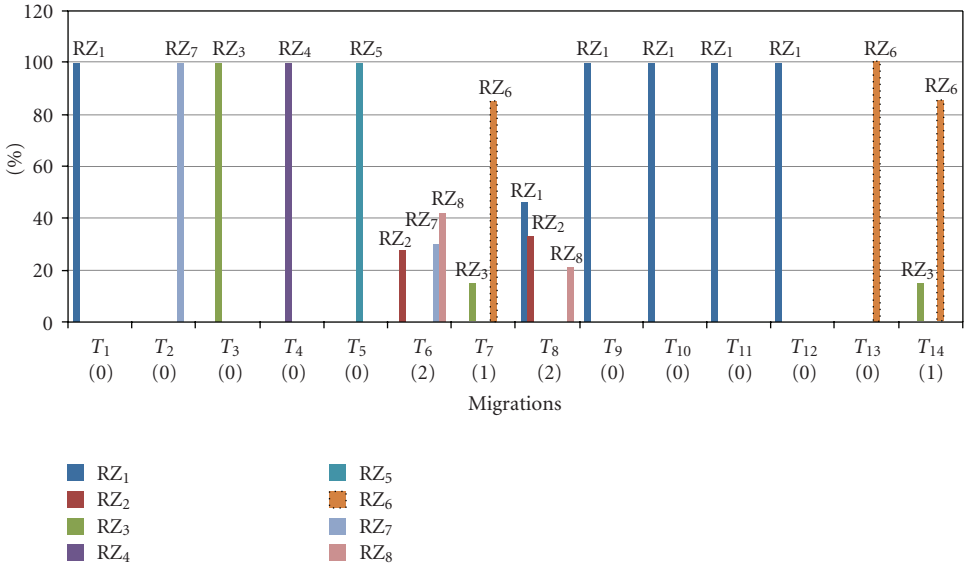


Figure 11: Task Occupation rates on RZs after task mapping.

The subproblem of RZ partitioning/fitting was resolved after 2 hours and 30 minutes. Table 4 shows the four coordinates of the most suitable RPBs for RZ fitting on the initial RR limited on the Virtex 5 FX200 device (82×12 RBs) as depicted in Figure 7. The initial RR is defined by the designer and by taking into account the set of resources dedicated only for the static part.

Table 5 shows comparison between the RBs of the obtained RPBs and their RZs. We observe high resource efficiency as the number of RBs within the RPBs is nearly equal to that of the RZs. Consequently, the internal fragmentation within the RPBs is considerably reduced. The differences of RBs are given by the last column ($\triangle$) of Table 5. For all RZs, except RZ_5 and RZ_6, their corresponding RPBs have one RB in excess. The RPB of RZ_5 contains 3 extra RB_3 and the RPB of RZ_6 strictly contains the required number of each RB type. The nonnull $\triangle$ is explained by the rectangular shape of the RPBs, thus the number of RBs included in the RPB could exceed the required RBs in the RZ. The nonnull $\triangle$ is also due to the heterogeneity on the device; the partitioning could book some RBs which are not used by the RZ.

Figure 7 depicts the floorplanning of RPBs conducted on the Virtex 5 FX200 device. The obtained results show an average of resources utilization of 36% of the available resources on the initial RR. This average is computed according to the number and the cost of each RB type. The optimization in utilization of resources minimizes the area of FPGA which is reconfigured at runtime. We have created static design by floorplanning each instance of each hardware task on its RPB without using the concept of dynamic partial reconfiguration. The obtained utilization of such static design resources is 63% of the available resources on the initial RR. Therefore, the gain of configuration overhead in a static design is paid by the resource waste, which is 43% compared to our obtained results employing dynamic partial reconfiguration. The RPBs are closely packed on the initial RR which avoids the resources waste and the external fragmentation in the device. For this reason, the initial RR could be resized in order to dedicate sufficient space for the remainder static part as depicted in Figure 7 by final RR.

Once the final RPB floorplanning is conducted, the final configuration overheads corresponding to RZs are determined. Thus, these new values are used to resolve the subproblem of task mapping to RZs.

The subproblem of task fitting on RZs was solved within 9 seconds. Figure 8 shows the results of preemption point mapping of hardware tasks in the application. T_7 and T_{14} start on RZ_6, they are preempted after 85% of their WCET and continue their execution on RZ_3. T_8 is mapped first to RZ_2, it is stopped on RZ_2 on its second preemption point that is, after 33% of execution and restarts on RZ_8 up to 54%. At this third preemption point, T_8 migrates to RZ_1 where it completes its execution. T_1, T_9, T_{10}, T_{11}, and T_{12} are totally mapped to RZ_1. T_2, T_3, T_4, T_5, and T_{13} are totally mapped, respectively to RZ_7, RZ_3, RZ_4, RZ_5, and RZ_6.

As shown in Figure 9, task T_6 starts its execution on RZ_8 then is preempted on its third preemption point that is, after 42% of execution. T_6 resumes its execution on RZ_7 till 72% of its execution. Hence, it is preempted again on RZ_7 on the fourth preemption point and restarts on RZ_2 where it is achieved.

This resolution of mapping hardware tasks to RZs discards the problem of task rejection as it guarantees an execution unit for each task to achieve its execution by respecting its predefined preemption points.

The bar chart on Figure 10 presents obtained RZ workloads after task mapping resolution. Except RZ_4 having a workload of 44%, the remainder RZs have balanced workloads which are closest to the average workload equal to 89%.

The bar chart in Figure 11 depicts the task occupation rates on the RZs as a result of mapping. This bar chart shows the number of migrations that must be performed to

avoid the RZ overloads and ensure total execution of tasks. We obtained 6 migrations. T_8 realizes two migrations. In fact, T_8 is mapped to RZ_1, RZ_2, and RZ_8. The mapping of T_8 combines the two objectives expressed by *Map2,* which is the minimization of configuration overhead and *Map4* which optimizes utilization of costly resources. Similarly T_6 is mapped to RZ_2, RZ_7, and RZ_8. T_7 and T_{14} sustain both, migration from RZ_6 to RZ_3.

If we consider independent tasks, within each RZ respecting 100% as workload bound, execution sections mapped to this RZ could be scheduled by Earliest Deadline First (EDF) algorithm as they respect the monoprocessor sufficient schedulability condition ($\sum_{Ti\ mapped\ to\ RZ}(Exe_i/P_i) \leq 1$).

After RZ fitting on their RPBs and according to the obtained task mapping, the final RPB configuration overheads are determined. In the worst case, by counting all the possible preemption points, the total overhead including configuration overheads and functional overheads of tasks is 72959 μs. The overall overhead is 11% of total running time.

Table 6 gives some comparisons with previous work in hardware task placement in terms of task rejection, resource utilization, configuration overhead, and the complexity of the performed technique. To the best of our knowledge, there are several placement algorithms proposed for each goal. These algorithms could be classified as metaheuristics or online heuristics. Nevertheless, these algorithms target only one goal and do not take into account other goal satisfactions. In opposition to [11, 32], our multiobjective placement computes the configuration overhead in the worst case before scheduling (11%) and targets an application of 14 tasks which is not the case of [11, 32] that optimizes the configuration overhead only for two or three tasks during the scheduling (18%, 8%). Comparing to [10] (sets of 3,000 homogeneous tasks) and [1] (10.000 homogeneous tasks) applied in homogeneous devices; we have reduced efficiently the resource utilization (36%) for an application of 14 heterogeneous tasks and by taking into account the heterogeneity of recent reconfigurable devices. The heterogeneous resources in FPGA could fit the RZs on large RPBs giving a significant resource waste. Comparing to the offline simulated annealing in [4] performed for 100 tasks which produces 13% of task rejection, our three-level offline placement discards totally the task rejection for an application of 14 tasks.

6. Conclusion and Future Works

In this paper, we propose novel three-level resource management that investigates enhancing placement quality by reducing task rejection, resource waste, and configuration overheads. Our work is based on the optimization of resource utilization relying on the features of heterogeneous reconfigurable devices and on constrained optimization problems. We reported on resolution showing an improvement of placement quality compared to previous works. In fact, the overall overhead is 11% of total running time, the average resource utilization is 36% of the available resources on the reconfigurable region and we enhanced resource utilization by 43% compared to static design. In addition, the non-overload in some RZs improves the possibility of mapping other tasks by respecting their deadlines without performing another resource allocation. Unlike the works achieved in placement that deal with independent tasks, we eliminated the issue of task rejection as we pack tasks on RZs and employ dynamic partial reconfiguration.

Our results do not agree with other works on hardware task placement since experimental conditions are not the same. In fact, we used the recent FPGA technology that provides the highest configuration frequency and wide data ports that speed up configuration overhead. The improvement of resource utilization is explained by the optimality of our solution. The cancellation of task rejection is due to the employment of dynamic partial reconfiguration to map several tasks to the same RZ instead of dealing with each task placement independently. We have exploited powerful solvers that ensure the achievement of searching. Consequently, the most optimal solution provides the least rates of resources utilization and configuration overhead.

To conclude, it is recommended to take advantage of the obtained results for purposes of establishing the rules of hardware task scheduling for real-time applications. By attempting to follow the obtained partitioning/fitting, we guarantee the highest placement quality equal to or better than that obtained in offline three-level resource management.

Further work includes the defining of efficient on-line scheduling guided by obtained offline results. Moreover, we aim improving our offline task mapping by adding precedence constraints as well as deadline and periodicity constraints to achieve an offline mapping/scheduling of hardware tasks. Consequently, we ensure a full offline placement/scheduling of hardware tasks on hardware reconfigurable devices. The dependency between tasks should be investigated, especially in considering intertask communication with the overall overhead presented in this paper. Intertask communication will be an important criterion in deciding the most optimal RZ fitting. Intertask communication relies on the management of an efficient communication network such as FAT-Tree [33] as well as on the management of a shared memory.

Acknowledgments

This paper was supported by AIMMS technical support and Xilinx tools. It is sponsored by the national agency of research in France and the world-ranking "Secured Communicating Solutions" (SCS) cluster that pool together with industrial, research and higher education players which are involved in the microelectronics, telecommunications, software, and multimedia sectors.

References

[1] A. A. El Farag, H. M. El-Boghdadi, and S. I. Shaheen, "Improving utilization of reconfigurable resources using two dimensional compaction," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '07)*, pp. 135–140, Nice, France, April 2007.

[2] http://www.polytech.unice.fr/~fmuller/fosfor/,FOSFOR.2008.

[3] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of hardware tasks on FPGA," in *Proceedings of the 19th International Conference on Field Programmable Logic and Application (FPL '09)*, pp. 591–595, Prague, Czech republic, September 2009.

[4] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test, Special Issue on Reconfigurable Computing*, vol. 17, no. 1, pp. 68–83, 2000.

[5] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin Packing: A Survey*, Chapter 2, PWS Publishing Company, Boston, Mass, USA, 1997.

[6] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 178, Nice, France, April 2003.

[7] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, vol. 4, p. 134, Santa Fe, NM, USA, April 2004.

[8] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Intelligent merging online task placement algorithm for partial reconfigurable systems," in *Proceedings of the Design Automation Test Europe (DATE '08)*, pp. 1346–1351, Munich, Germany, March 2008.

[9] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 960–965, San Diego, Calif, USA, June 2004.

[10] H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, "Task rearrangement on partially reconfigurable FPGAs with restricted buffer," in *Proceedings of the Field Programmable Logic and Applications*, vol. 1896, pp. 379–388, Vienna, Austria, August 2000.

[11] J. Resano, D. Mozos, D. Verkest, S. Vernalde, and F. Catthoor, "Run-time minimization of reconfiguration overhead in dynamically reconfigurable systems," in *Proceedings of the International Conference on Field Programmable Logic and Application*, vol. 2778 of *Lecture Notes in Computer Science*, pp. 585–594, Lisbon, Portugal, September 2003.

[12] E. M. Panainte, K. Bertels, and S. Vassiliadis, "FPGA-area allocation for partial run-time reconfiguration," in *Proceedings of the Design Automation Test Europe (DATE '05)*, pp. 100–105, Munich, Germany, March 2005.

[13] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: a survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002.

[14] A. Lodi, S. Martello, and D. Vigo, "Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem," in *Proceedings of the Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pp. 125–139, Sophia Antipolis, France, July 1997.

[15] A. Lodi, S. Martello, and D. Vigo, "Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems," *INFORMS Journal on Computing*, vol. 11, no. 4, pp. 345–357, 1999.

[16] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM Journal on Computing*, pp. 846–855, 1980.

[17] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Management Science*, vol. 44, no. 3, pp. 388–399, 1998.

[18] K. Danne and S. Stuehmeier, "Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants," in *From Specification to Embedded Systems Application*, vol. 184 of *International Federation for Information Processing*, Springer, New York, NY, USA, 2005.

[19] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "3-D floorplanning: simulated annealing and greedy placement methods for reconfigurable computing systems," *Design Automation for Embedded Systems*, vol. 5, no. 3, pp. 329–338, 2000.

[20] S. P. Fekete, E. Kohler, and J. Teich, "Optimal FPGA module placement with temporal precedence constraints," in *Proceedings of the Conference Design Automation and Test in Europe*, pp. 658–665, Munich, Germany, 2001.

[21] J. Teich, S. P. Fekete, and J. Schepers, "Optimization of dynamic hardware reconfigurations," *The Journal of Supercomputing*, vol. 19, no. 1, pp. 57–75, 2001.

[22] F. Rivoallon and A. Cosoroaba, *Achieving Higher System Performance with the Virtex 5 Family of FPGAs*, Xilinx White Paper, San Jose, Calif, USA, 2006.

[23] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of reconfigurable zones and off-line mapping of hardware tasks on FPGA," in *Proceedings of the Design and Architectures for Signal and Image Processing (DASIP '09)*, Sophia Antipolis, France, September 2009.

[24] J. Clausen, *Branch and Bound Algorithms-Principles and Examples*, University of Copenhagen, Copenhagen, Denmark, 1999.

[25] G. Pataki, M. Tural, and E. B. Wong, "Basis reduction and the complexity of branch-and-bound," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1254–1261, Austin, Tex, USA, January 2010.

[26] S. M. Azam, M. ur-Rehman, A. K. Bhatti, and N. Daudpota, "Parallel branch and bound model using logarithmic sampling (PBLS) for symmetric traveling salesman problem," in *Proceedings of the World Academy of Science, Engineering and Technology*, vol. 6, pp. 66–69, June 2005.

[27] J.-K. Hao, P. Galinier, and M. Habib, "Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes," *Revue d'Intelligence Artificielle*, vol. 13, no. 2, pp. 283–324, 1999.

[28] http://opencores.org/.

[29] "Virtex-5 FPGA Configuration User Guide," Xilinx white paper, August 2009.

[30] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó, "A comparison of complete global optimization solvers," *Mathematical Programming*, vol. 103, no. 2, pp. 335–356, 2005.

[31] http://www.aimms.com/.

[32] J. A. Clemente, C. González, J. Resano, and D. Mozos, "A hardware task-graph scheduler for reconfigurable multitasking systems," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, pp. 79–84, Cancun, Mexico, December 2008.

[33] L. Devaux, D. Chillet, S. Pillement, and D. demigny, "Flexible communication support for dynamically reconfigurable fpgas," in *Proceedings of the 5th Southern Conference on Programmable Logic (SPL '09)*, pp. 65–70, São Paulo, Brazil, April 2009.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 591983, 28 pages
doi:10.1155/2011/591983

Research Article

Static Scheduling of Periodic Hardware Tasks with Precedence and Deadline Constraints on Reconfigurable Hardware Devices

Ikbel Belaid,[1] Fabrice Muller,[1] and Maher Benjemaa[2]

[1] *LEAT-CNRS, University of Nice Sophia Antipolis, 06560 Valbonne, France*
[2] *Research Unit ReDCAD, National Engineering School of Sfax, University of Sfax, 3038 Sfax, Tunisia*

Correspondence should be addressed to Ikbel Belaid, ikbel.belaid@unice.fr

Received 27 August 2010; Revised 12 January 2011; Accepted 10 February 2011

Academic Editor: Michael Hübner

Task graph scheduling for reconfigurable hardware devices can be defined as finding a schedule for a set of periodic tasks with precedence, dependence, and deadline constraints as well as their optimal allocations on the available heterogeneous hardware resources. This paper proposes a new methodology comprising three main stages. Using these three main stages, dynamic partial reconfiguration and mixed integer programming, pipelined scheduling and efficient placement are achieved and enable parallel computing of the task graph on the reconfigurable devices by optimizing placement/scheduling quality. Experiments on an application of heterogeneous hardware tasks demonstrate an improvement of resource utilization of 12.45% of the available reconfigurable resources corresponding to a resource gain of 17.3% compared to a static design. The configuration overhead is reduced to 2% of the total running time. Due to pipelined scheduling, the task graph spanning is minimized by 4% compared to sequential execution of the graph.

1. Introduction

An important trend in real-time applications implemented in reconfigurable computing systems consists in using reconfigurable hardware devices to increase performances and to guarantee temporal constraints. These reconfigurable devices provide a high density of heterogeneous resources in order to satisfy application requirements and especially to enable parallel computing. Furthermore, the devices employ the pertinent concept of run-time partial reconfiguration which allows reconfiguration of a portion of available resources without interrupting the remainder parts running in the same device. Consequently, the concept increases resource utilization and application performance.

Periodic partially ordered activities represent the major computational demand in real-time systems such as real-time control and digital signal processing. This category of repetitive computation is described by directed acyclic graphs (DAGs). Implementation of these DAGs in reconfigurable hardware devices consists in scheduling tasks to a limited number of nonidentical units shaped on the area of reconfigurable resources, while respecting the four constraints described as follows. (1) The periodicity constraint: each task is repeated periodically according to its ready times in the graph. Thus, if task A has a period P_A, then for all $i \in \mathbb{N}$, $(r_{A_{i+1}} - r_{A_i}) = P_A$, where A_i and A_{i+1} are the ith and the $(i+1)$th repetitions of task A, and r_{A_i} and $r_{A_{i+1}}$ are their start times. (2) The precedence constraint: to maintain the rightness of task precedences, in each iteration, a task can be executed only if all its predecessors in the graph have finished their executions. Therefore, each task A must start execution after the completion of executions of its predecessors defined by the subset Π_A, thus for all $i \in \mathbb{N}$, $s_{A_i} \geq s_{B_i} + C_B$, for all $B \in \Pi_A$, where s_{A_i}, s_{B_i} are the start times of task A and task B, respectively, during their ith iteration, and C_B is the execution time of task B. (3) The dependence constraint: the execution of each task in DAG is launched when all the data resulting from all its predecessors are available. This constraint guides the choice of task periods as detailed in Section 3. (4) The deadline constraint: as this paper focuses on hard real-time systems, each task in the DAG must finish its execution before its hard deadline. Thus, within iteration i, if task A has an execution time C_A and an absolute deadline d_{A_i}, then $s_{A_i} + C_A \leq d_{A_i}$.

Figure 1 illustrates an example of the targeting task graph. As can be seen in Figure 1, the tasks are repeated according to their fixed periods. Each task with precedence link launches its execution only when its predecessors achieve their executions and only when it is required. For example, the third iterations of T_2 and T_3 of periods 8 do not need a third execution of their predecessor T_1 as it is less repetitive than T_2 and T_3 (period of T_1 is equal to 12). At each repetition, to enable the task execution, the dotted lines ensure the data transfer between interdependent tasks. The issue of data dependence is detailed later in the paper. Finally, at each iteration, the real-time tasks must respect their hard deadlines.

As shown in Figure 2, this paper proposes a new methodology comprising three main stages to achieve the scheduling of these DAGs with the predefined constraints on reconfigurable devices.

Task Clustering. This stage is technology dependent. It targets the partitioning of tasks requiring the same types of resources into the same cluster.

Mapping/Scheduling of Tasks in Clusters. This stage starts by performing spatial and temporal analyses mentioned in Figure 2 by DAG validity, Ready Times, and a set of heuristics. Subsequently, based on a predefined preemption model, it deals with simultaneous resolution of mapping tasks to the obtained clusters and global scheduling of tasks in clusters respecting the periodicity, precedence, dependence, and deadline constraints. This stage aims at optimizing scheduling quality.

Cluster Placement on the Reconfigurable Device. This stage is also technology dependent. It involves searching for the most suitable physical location partitioned on the reconfigurable device for each cluster obtained at the second stage. This stage aims at optimizing placement quality.

The resolution of these three stages results in static scheduling of tasks in the DAGs into a limited number of reconfigurable units partitioned on the device, respecting the periodicity, precedence, dependence, and deadline constraints. This is a fundamental problem in parallel computation, equivalent to determining static multiprocessor scheduling for DAGs in a software context. As is well known, static multiprocessor task graph scheduling is a combinatorial optimization problem, and it is formulated in this paper through mixed integer programming and solved by means of powerful solvers.

The paper details the spatial and temporal analyses required to check scheduling task graph feasibility and aims at determining the optimal solution in terms of schedule length, waiting time, parallel efficiency, resource efficiency, and configuration overhead. Schedulability analysis is not the focus of the present paper. However, before dealing with DAG scheduling on reconfigurable device, the rightness of the precedences and dependences between tasks within the graph and the accuracy of real-time functioning are analyzed, and a set of heuristics are performed to provide

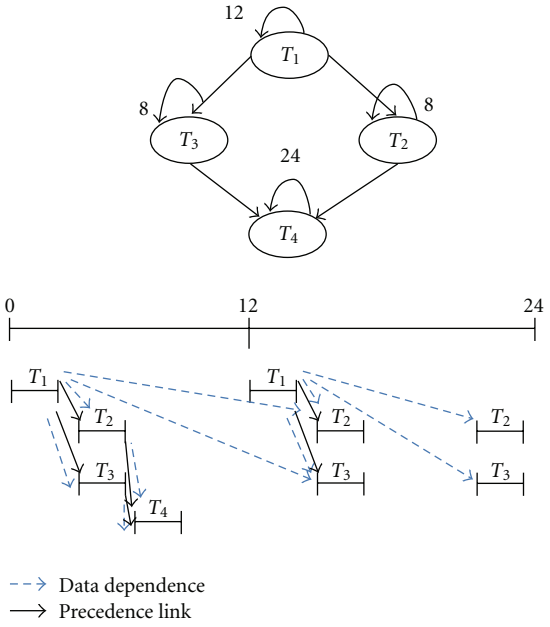


Figure 1: The targeted acyclic task graph.

the number of reconfigurable physical units needed to ensure the existence of valid DAG scheduling. The analyses are expressed by some constraints to ensure the validity of the chosen task graph.

The paper is organized as follows: Section 2 presents related works of the DAG scheduling problem. Section 3 details the methodology we propose to achieve the placement and scheduling of DAGs on reconfigurable devices. Section 4 describes an illustration of our proposed methodology on a given DAG and evaluates the obtained enhancements by metric measuring of placement and scheduling quality. The conclusion and future works are presented in Section 5.

2. Related Works

Static multiprocessor scheduling techniques using task graphs have matured over the last years, and many powerful scheduling strategies have emerged. As this problem is known to be NP-hard [1], the main research efforts in this area focus on heuristic methods and few of them propose analytic resolutions. We have studied static and dynamic multiprocessor scheduling using DAGs in both the software and hardware contexts.

In [2], Clemente et al. implement a static hardware scheduler employing efficient techniques which greatly reduce reconfiguration latencies and schedule length. Taking into account that configuration latency drastically reduces the efficiency of hardware multitasking systems, they introduce a new hardware scheduler communicating directly with the reconfigurable units and using optimization techniques: prefetch, reuse and replace while guaranteeing the precedence constraints. The prefetch technique manages in advance the reconfigurations and replacements required to improve task reuse. Reference [2] presents three algorithms

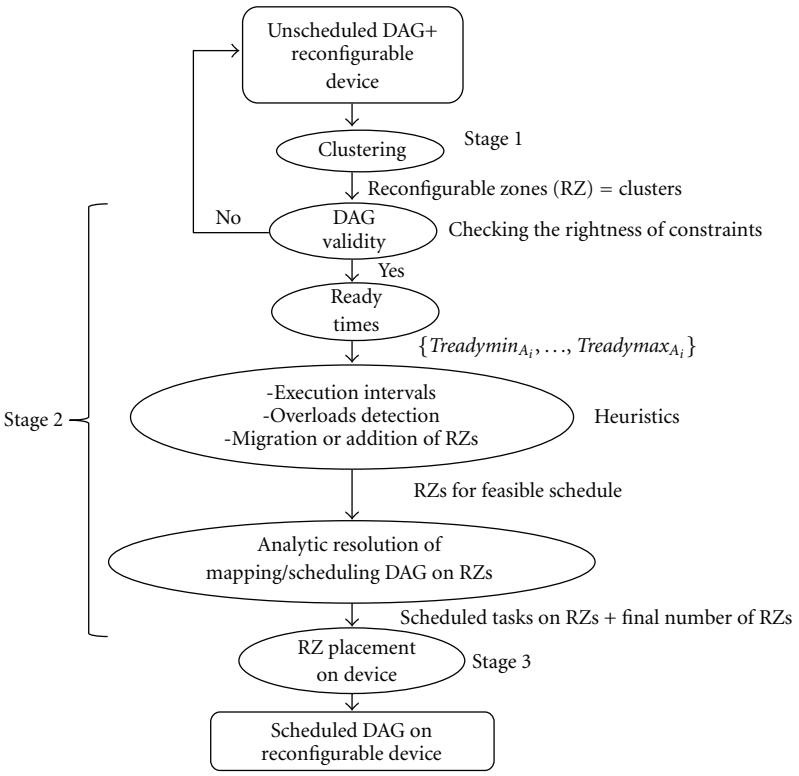


Figure 2: The proposed methodology.

for the replace technique, that is, the *Least Recently Used*, *Longest Forward Distance*, and *Look Forward + Critical* (*LF + C*) algorithms. The paper focuses especially on *LF + C* to schedule graph tasks on several reconfigurable units. In order to maximize the task reuse that reduces reconfiguration latency, *LF + C* classifies tasks from the most critical ones in terms of required reconfiguration to the least critical ones and tries to replace the latter tasks, so that their reconfiguration does not generate any overhead during task graph execution. This advantage is ensured by the prefetch technique which reconfigures a given task during execution of its predecessors.

Static multiprocessor scheduling is a very difficult problem, but genetic algorithms have successfully been applied to the search for acceptable solutions. In [3], the authors investigate scheduling for cyclic task graphs using genetic algorithms by transforming the cyclic graph into several alternate DAGs. To create an efficient schedule, the paper considers both the intracycle dependencies and the dependencies across different cycles. After unfolding the cyclic task graph for two cycles by incorporating the intercycle dependencies, the paper presents an algorithm investigating all the subgraphs extracted from a two-cycle task graph. Based on measurements such as the height and the width of the task graph, connection degree, degree of simultaneousness, and independent parts in the graph, the method evaluates the resulting subgraphs to select the configuration that best suits a chosen application and the available hardware configuration. Suitable allocation to processors is obtained by the *earliest start time* heuristic where tasks are assigned to the processor that offers the earliest start time. The employed genetic algorithm tries to optimize the schedule length which is expressed by the finishing time on all processors.

In [4], Abdeddaim et al. describe a method based on a timed automaton model for solving the problem of scheduling partially ordered tasks on parallel identical machines. The proposed method formulates for each task in the graph a 3-state automaton consisting of the waiting, active, and finish states. Therefore, by searching the tasks related by a partial order in the graph, the possible disjoint chains in the graph are extracted. The automaton of every chain is constructed using the individual task-specific automaton. The global automaton is then composed by the chain-specific automata and takes care that the transitions do not violate the precedence, and resource constraints. Thus, optimal scheduling consists in finding the shortest path in the timed automaton. The proposed methodology is also extended to include two additional features in the task-specific automaton which are the release and the deadline times.

Integer linear programming (ILP) formulation is exploited in some works of static multiprocessor scheduling using task graphs. The authors of paper [5] propose an exact ILP formulation to perform task graph scheduling on dynamically and partially reconfigurable architectures and that minimizes schedule length. In addition, this work proposes a dynamic reconfiguration-aware heuristic scheduler called *NAPOLEON*, which adopts an ALAP (as late as possible)

order of tasks and exploits configuration pre-fetching, module reuse, and antifragmentation techniques. Both methods are extended to the Hw/Sw codesign. The ILP formulation is based on nonpreemptive tasks, allows the execution of tasks on software processors or on the FPGA, and respects the FPGA physical constraints as well as the precedence and temporal constraints in the graph. Both methods provide a solution for the complete scheduling of the DAG and determine for each task its Sw or Hw execution unit, its time of reconfiguration start, its position on FPGA, and its execution starting time.

Another exact ILP formulation for performing task mapping and scheduling on multicore architectures is presented in [6]. The technique of these authors incorporates loop level task partitioning, task transformations by using loop fusion and loop splitting, and it aims at reducing system execution time. The paper focuses on an ILP-based approach for task partitioning, task mapping, and pipelined scheduling while taking data communication between processors into consideration for DSP applications on the multicore platform. The authors in [6] divide the problem into two parts. The first assigns and schedules tasks on processors by including the task merging on the same batch and the replication of batches to several processors. The second step conducts a mapping of data to memory architecture by minimizing memory access latencies.

In [7], Sandnes and Sinnen consider the scheduling of iterative computing that can be represented by cyclic task graphs. In order to avoid costly classic graph unfolding and to shorten the makespan during scheduling, the authors propose a new strategy for transforming cyclic task graphs into acyclic task graphs; an efficient scheduling from the literature named *Critical Path/Most Immediate Successor First* (*CP-MISF*), proposed by Kasahara and Narita in 1985, is then applied to the transformed graph. The strategy is based on a decyclification step involving three parts: (1) a decyclification algorithm for transforming the cyclic graph into an acyclic graph based on a given start node and depth first search (DFS) strategy, (2) by assuming that the critical path in the graph is a good estimator for its schedule length, it searches the start node that yields the shortest critical path in the transformed graphs, and (3) quantifying the acyclic graph quality in terms of makespan. In addition, based on an adjacency matrix representing the graph dependencies and simplifying the unfolding formulation, the paper presents a new intuitive graph unfolding formulation which decomposes the adjacency matrix into two matrices, one for intraiteration dependences and another for interiteration dependences. The unfolded graph is then scheduled using a genetic algorithm approach.

In [1], Djordjević and Tošić propose a new compile-time single-pass scheduling technique applied for task graphs onto fully connected multiprocessor architectures called *chaining* and which takes into account communication delays. The proposed technique consists in a generalized list scheduling with no preconditions concerning the order in which tasks are selected for scheduling. The main idea is to build an heuristic providing a trade-off between maximizing parallelism on processors, minimizing communication overheads, and minimizing overall execution time of the task graph. *Chaining* technique uses nonpreemptive tasks and constructs a scheduled task graph incrementally by scheduling one task at each step. The intermediate partially scheduled task graphs are obtained by selecting a nonscheduled task at each step and by placing it on the most appropriate precedence edge. The policy of selection of tasks to be scheduled is based on a *Task Selection First* heuristic, and the selection of the most suitable valid edge where the task will be placed is guided by the critical path and edge width criteria. The tasks encompassed within the same chain are scheduled on the same processor.

In [8], the authors aim at improving the performance of hardware tasks on the FPGA. Intertask communication and data dependences between tasks are analyzed in order to reduce configuration overhead, to minimize communication latency, and to shorten the overall execution of tasks. The work exploits the proposed works in reconfigurable computing and addressing resource efficiency to present three algorithms. *Reduced Data Movement Scheduling* (*RDMS*) is the most efficient dynamic algorithm for reducing configuration and communication overheads and provides the optimal performance for scheduling tasks in DAGs on the FPGA. *RDMS* uses the total reconfiguration of the FPGA and tries to minimize the number of reconfigurations by grouping communicating tasks in the same configuration. By conducting a width search, *RDMS* schedules tasks while respecting their data dependences. *RDMS* is based on dynamic programming algorithm and ensures that each configuration includes the combination of tasks that exploits the hardware resources to the maximum and that encompasses the highest possible number of task dependences.

In [9], Fekete et al. consider the optimal placement of hardware tasks in space and in time on the FPGA. Tasks are presented as three-dimensional boxes in space and time. The authors integrate intertask communication expressing data dependence and use a graph-theoretical characterization of the feasible packing determined by means of a decision of an orthogonal packing problem with precedence constraints. By searching the transitive orientations and by performing projections, the authors of paper [9] transform the 3D boxes representing tasks into 3×1D arrangements and then verify whether the three obtained arrangements referred to as packing classes satisfy the conditions of the feasible packing and determine the optimal spatial and temporal packing. This work enhances the makespan of the graph and optimizes the used reconfigurable space on the FPGA.

The major contribution in [10] is the development of a multitasking microarchitecture to perform a dynamic task scheduling algorithm on reconfigurable hardware for nondeterministic applications with intertask dependences which are not known until runtime. The task system is modeled as a modified directed acyclic graph which contains directed data edges and directed control edges labeled with scalar values indicating the probability of occurrence of the corresponding sink task in multiple task graph iterations [10]. Based on dynamic priority assignment for nonpreemptive tasks, the dynamic scheduler assigns each task to a software or hardware processing element, schedules

the contexts (bitstreams) and the data, and the fetch and the prefetch reconfigurations, and activates task execution. In order to minimize reconfiguration overhead, the dynamic scheduler uses the configuration prefetching technique to prefetch the task bitstream ahead of time or exploits the previous context configured in the logic cell. In addition, it aims to minimize application execution time by employing a local optimization technique.

In [11], Kohler defines a new heuristic to schedule DAGs on a system of independent identical processors. The author describes a simple critical path priority method which is shown to be optimal or near optimal in the most randomly generated computation graphs compared to the Branch and Bound method. This heuristic aims to minimize the finishing time of the computation graph. Critical path scheduling is based on a list (L) containing permutation of the tasks. Any time a processor is idle, it instantaneously scans the list L from the beginning and begins to execute the first free task which may validly be executed because all its predecessors have been completed [11]. The construction of the list is based on the critical path of tasks which is defined by the longest path from a given task to a terminal node. The paper also presents the exact Branch and Bound method used to obtain optimal scheduling, and the results obtained are compared to the critical path heuristic to prove the high quality of the latter method.

Table 1 provides a summary of the optimization parameters and employed techniques described in the cited works.

The major common drawback of most described techniques is that they do not address real-time constraints. Further-more, as shown in Table 1, most of them seek to optimize the makespan of the graph and neglect reconfiguration overhead and resource inefficiency or do not optimize the three parameters simultaneously. The works described in [8, 9] that conduct scheduling of DAGs on FPGA devices are based on successive total configurations of the device. Their resource efficiency consists only in the packing the maximum of tasks in the DAG on the FPGA in order to efficiently exploit the reconfigurable resources as well as to perform the minimum of total configurations. These works therefore do not consider the internal fragmentation caused by task placement on the FPGA which represents resource efficiency in our work.

In the context of hardware task scheduling, in the proposed works, the placement of scheduled tasks is not considered. Either the placement of the task is allowed in whatever position in the device (in this case, they do not take into account device heterogeneity) or the task is fixed to a unique reconfigurable unit which will reduce application flexibility. Contrary to these works, our strategy may be generalized for all types of devices, that is, both homogeneous and heterogeneous devices; the placement problem is considered an important stage, that is, highly interlinked with the scheduling of task graphs on the reconfigurable device. With our strategy, the task may be executed on several reconfigurable units according to its resources and according to the analyses conducted during the clustering stage.

Moreover, some of the described works do not exploit the relevant concept of run-time partial reconfiguration afforded by recent reconfigurable devices and employ the total configuration of FPGAs.

Based on these observations, our challenge is to utilize the benefits of the run-time partial reconfiguration concept for recent heterogeneous devices. The concept opens up the possibility of developing a hardware multitasking system by dividing the reconfigurable area into smaller reconfigurable units and by customizing them as required by the running application.

Considering multitasking, scheduling of task graphs on reconfigurable hardware devices is similar to heterogeneous multiprocessor scheduling in the software context. With full knowledge of the characteristics of DAG tasks and technology features, our methodology targets constrained applications and endeavors to provide pipelined scheduling in multi-reconfigurable-unit system while optimizing schedule length, waiting time, parallel efficiency, resource efficiency, and configuration overhead.

3. Proposed Methodology for Placement and Scheduling of Dags on Reconfigurable Devices

Our methodology can be viewed as two separate subproblems: (i) the mapping and scheduling of hardware tasks on predefined clusters by satisfying periodicity, precedence, dependence, and deadline constraints and (ii) the placement of obtained clusters on reconfigurable device taking into account its heterogeneity. Our resource and task management is essentially based on features of hardware tasks and reconfigurable hardware devices. The most recent Xilinx's Virtex FPGA was used as a reference for the reconfigurable hardware device to perform the placement and scheduling of the DAGs. Virtex SRAM-based FPGAs are characterized by a column-based architecture, a high density of heterogeneous resources, and several parallel configuration ports functioning at a high speed.

3.1. Terminology and Definitions. Throughout the paper, NT refers to the number of tasks in the graph, NZ the number of clusters, and NP the number of resource types in the chosen technology. The directed acyclic task graph is denoted by the pair (N, E). N is the set of nodes representing tasks in the DAG, and E is the set of edges linking the dependent tasks, $E \subseteq N \times N$.

As shown in Figure 3, on each edge, the outgoing value $(x_{A,B})$ from the source node A to the sink node B depicts the amount of data that A must produce at each repetition for B. The incoming value $(y_{B,A})$ represents the amount of data that must be consumed by the sink node B at each execution iteration after completion of the repetition of its predecessor A.

Each task in the graph has three models as follows.

3.1.1. Functional Model. Each hardware task (A) is represented by a set of parameters fixed at compile time and which are kept static throughout the DAG execution. A is characterized by its worst-case execution time (C_A), its

Table 1: Optimization parameters and techniques for scheduling works.

References	Makespan/Speedup/ Parallel efficiency	Resource efficiency	Configuration overhead	Techniques
[2]	x		x	Prefetch/replace/reuse of reconfigurations
[3]	x			Graph unfolding/genetic algorithm/*earliest start time* heuristic
[4]	x			Timed automaton model/shortest path
[5]	x		x	(1) ILP/reuse (2) *NAPOLEAN*/ALAP/prefetch/reuse/ antifragmentation
[6]	x			ILP/loop level task partitioning/task transformation (loop fusion + loop splitting)/task mapping and scheduling (task merging on batches + batch replication)/data mapping
[7]	x			(1) Graph decyclification (DFS + shortest critical path)/*CP-MISF* (2) Graph unfolding/genetic algorithm
[1]	x			*Chaining*/*task selection first* heuristic/critical path/edge width/communication latencies
[8]	x	x	x	Dynamic programming algorithm/FPGA total configuration
[9]	x	x		Orthogonal packing problem/packing classes
[10]	x		x	Prefetch/local optimization/reuse
[11]	x			Critical path heuristic/Branch and Bound

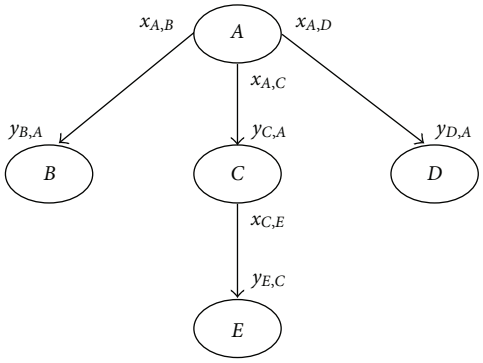


Figure 3: Directed acyclic graph.

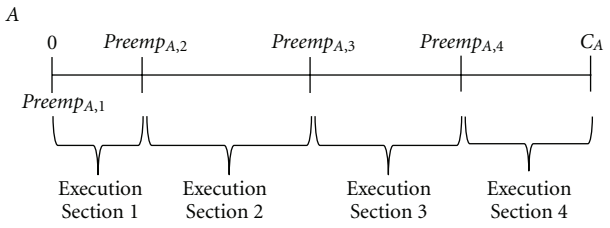


Figure 4: Predefined preemption points in Task *A*.

period (P_A), which is equal to its relative deadline (D_A), and a set of preemption points ($Preemp_{A,l}$). The preemption points are instants of the time taken throughout the worst-case execution time as shown in Figure 4. The number of preemption points of task *A* is denoted by $NbrPreemp_A$. This number also includes the first point of execution of the task. The set of preemption points is determined by the designer according to the known states in the behavioral model and according to possible data dependences between these states. The predefinition of preemption points gives rise to the execution sections within the hardware task. Our methodology is based on preemptive modeling to create a reactive system, to increase flexibility towards application needs, and consequently to enhance the respect of real-time constraints.

3.1.2. Behavioral Model. This includes the finite state machine controlling each task and which handles a set of registers or a small memory bank useful for context switching during preemption. The latency required to preempt and to resume the execution of tasks is disregarded as in the worst case; the time to access the system bus and memory is negligible. With the preemptive model, we do not use the classical method of readback and load modified bitstream since latency with this method is significant; it complicates preemption and requires a large space memory. With the classical method, a new readback bitstream must be saved at each preemption. With our model, the number of preemptions within tasks is limited by specifying the

possible preemption points according to task states outside of which preemption is not allowed. Thus, we resort to saving/loading the current state of the finite state machine with an acceptable amount of data by maintaining the same bitstream for each task within a given reconfigurable unit when the task needs to be preempted or resumed. Preemption points of hardware tasks are set in a way that reduces the data dependences that could exist between two states. In fact, maintaining a preemption point between two states processing the same data must be avoided because these data need to be saved into an external memory which might increase the preemption overhead at runtime. Otherwise, it is recommended to insert a preemption point when the task is in a blocked state, waiting to receive an external resource to enable the ready tasks to be executed in the reconfigurable unit. In the finite state machine, the longest execution time between two states must be considered in order to deduct the worst-case execution time.

3.1.3. RB-Model. At the physical level, tasks are presented as a set of reconfigurable resources called reconfigurable blocs (RB). RBs correspond to the physical resources in the reconfigurable hardware device required to achieve execution of the hardware task, and they define the RB model of the task as expressed by (1). Determination of the RB-model of hardware tasks is well detailed in our work described in [12]. The number of RB types is equal to the number of resource types in the chosen technology. The RBs are the smallest reconfigurable units in the hardware device. They are determined according to the available reconfigurable resources in the device, and they closely match its reconfiguration granularity. Each type of RB is characterized by a specified cost, $RBCost_k$, defined according to its frequency in the device, its power consumption, and the importance of its functionality,

$$A\_\mathrm{RB} = \{\alpha_{A,k}\mathrm{RB}_k\}, \quad \alpha_{A,k} \in \mathbb{N}, \quad 1 \le A \le \mathrm{NT}, \quad 1 \le k \le \mathrm{NP}. \tag{1}$$

The reconfigurable device is also characterized by its RB model as shown in our work described in [12] to enable the placement of hardware task clusters at a later stage.

The three main stages of the methodology used for static scheduling of DAGs on multi-reconfigurable-unit system with predefined constraints are described below.

3.2. Hardware Task Clustering. This stage comprises two steps that are performed consecutively: (i) reconfigurable zone type search and (ii) cost D computing. Bearing in mind that the concept of run-time partial reconfiguration had to be used, our main objective at this first stage was to partition tasks constituting the graph into cluster types determined according to their required RB types in order to enhance resource utilization.

3.2.1. Reconfigurable Zones Types Search. This step takes as input the RB model of each task in the DAG, and by performing Algorithm 1 of the worst-case complexity $O(\mathrm{NT}^2 * \mathrm{NP})$, it groups tasks sharing the same types of RBs under the same type of cluster by taking the maximum number of RBs between these tasks. With our methodology, the obtained types of clusters are denoted as reconfigurable zones (RZs). The upper bound of the possible RZs is NT. Thus, RZs are virtual units customized by Algorithm 1 to model the classes of hardware tasks in terms of RB types. RZs separate hardware tasks from their execution units on the reconfigurable device. In the last stage of our proposed methodology, RZs will be placed on their suitable reconfigurable units respecting the heterogeneity of the device and optimizing resource efficiency as well as configuration overhead. After the completion of Algorithm 1, each RZ is represented by its RB model as expressed by

$$\mathrm{RZ}_j\_\mathrm{RB} = \left\{\beta_{j,k}\mathrm{RB}_k\right\}, \quad \beta_{j,k} \in \mathbb{N}, \quad 1 \le j \le \mathrm{NZ}, \quad 1 \le k \le \mathrm{NP}. \tag{2}$$

Algorithm 1 processes the tasks of the DAG as follows. It scans the RB model of each hardware task and checks whether an already inserted type of RZ that closely matches the required types of RBs in the task exists in the RZ types list, *list-RZ* (line 6). Should this be the case, Algorithm 1 updates the number of RBs within this type of RZ by the maximum between the number of RBs in the task and that in the RZ (line 9).

If the required types of RBs in the task do not match any type of RZ included in the *list-RZ*, the algorithm of the search of RZ types decides on the creation of a new type of RZ as required by the task (lines 12, 13) and inserts it in *list-RZ* (line 14).

Figure 5 is an example of the execution of Algorithm 1 for the RZ types search for DAG comprising five tasks. Figure 5 illustrates the search for RZ types resulting from five tasks in a technology including four types of RBs (RB_1, RB_2, RB_3 and RB_4). A and C are grouped in the same type of RZ (RZ_1) as both need RB_1 and RB_2, and the number of each RB type within RZ_1 is adjusted by the maximum number of RBs between A and C. Similarly, RZ_2 is created by B and D, and E defines the third type of RZ (RZ_3).

After searching for the set of RZs, the configuration overhead for each obtained RZ is computed and denoted by $Config_j$. $Config_j$ corresponds to the configuration overhead to fit RZ_j in the target technology. This configuration overhead is computed by the floorplanning of each RZ_j on the chosen device and by conducting the whole partial reconfiguration flowup to the creation of the partial bitstream. $Config_j$ is determined by (3) according to configuration frequency and configuration port,

$$Config_j = \frac{\text{size of bit stream}}{(\text{Configuration frequency} \times \text{configuration port width})}. \tag{3}$$

3.2.2. Cost D Computing. This step commences by computing cost D between tasks and each RZ type resulting from the

(1) $RZ\text{-}type = 0$ // RZ types
(2) $List\text{-}RZ$ // list of RZ types
(3) n // natural
(4) **for all** tasks A **do**
(5) // $A\_\mathrm{RB} = \alpha_{A,k}\mathrm{RB}_k$
(6) **if** $((RZ\text{-}type \neq 0)$ **and** $(\exists n, 1 \leq n \leq RZ\text{-}type)/\forall k((\alpha_{A,k} \neq 0$ **and** $\beta_{n,k} \neq 0)$ **or** $(\alpha_{A,k} = 0$ **and** $\beta_{n,k} = 0)))$ **then**
(7) // this test checks whether the task matches with an RZ type that already exists in $list\text{-}RZ$
(8) **for all** k **do**
(9) $\beta_{n,k} = \max(\alpha_{A,k}, \beta_{n,k})$ // update RB number of RZ_n
(10) **end for**
(11) **else**
(12) Increment $RZ\text{-}type$
(13) $\mathrm{RZ}_{RZ\text{-}type} = Create\ new\ \mathrm{RZ}(\alpha_{A,k})$ // new type of RZ, $\mathrm{RZ}_{RZ\text{-}type} = \{\alpha_{A,k}\mathrm{RB}_k\}$
(14) Insert $(list\text{-}RZ, \mathrm{RZ}_{RZ\text{-}type})$
(15) **end if**
(16) **end for**

Algorithm 1: RZ types search or hardware task classes search.

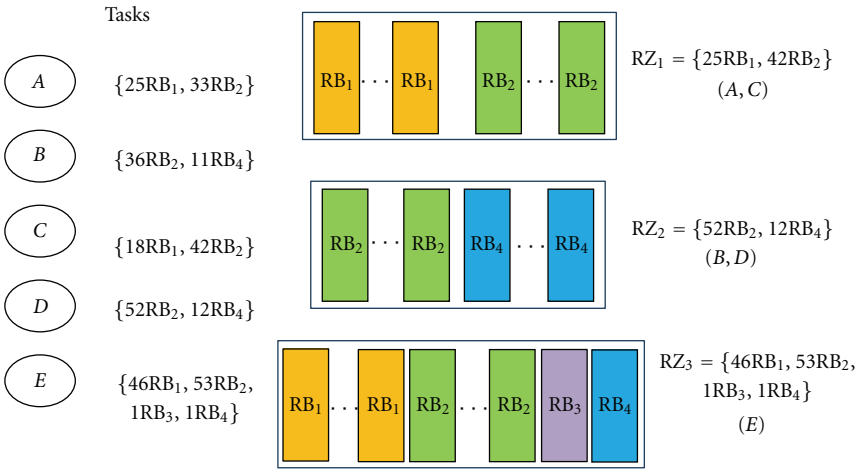


Figure 5: Example of the search for RZ types.

first step. Cost D represents the differences in RBs between tasks and RZs; consequently, it expresses resource wastage when a task is mapped to an RZ. Based on RB models of task A and RZ RZ_j, cost D is computed according to two cases as follows. Firstly, we define by

$$d_{A,j,k} = \alpha_{A,k} - \beta_{j,k}, \quad 1 \leq A \leq \mathrm{NT}, \quad 1 \leq j \leq \mathrm{NZ}, \quad 1 \leq k \leq \mathrm{NP}. \tag{4}$$

Case 1. For all k, $d_{A,j,k} \leq 0$, RZ_j contains a sufficient number of each type of RB (RB_k) required by A, and cost D is equal to the sum of the differences in the numbers of each RB type between A and RZ_j weighted by $RBCost_k$ as expressed in

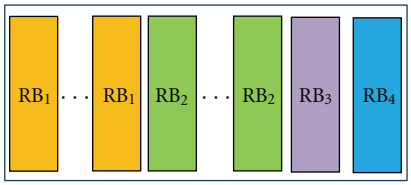
$$D\left(A, \mathrm{RZ}_j\right) = \sum_{1 \leq k \leq \mathrm{NP}} RBCost_k \times \left| d_{A,j,k} \right|. \tag{5}$$

Case 2. There exists k, $d_{A,j,k} > 0$, the number of RBs required by A exceeds the number of RBs in RZ_j or A needs RB_k which is not included in RZ_j. In this case, cost D is infinite,

$$D\left(A, \mathrm{RZ}_j\right) = \infty. \tag{6}$$

Cost D is exploited during the stage of task mapping and scheduling on RZs and the RZ placement stage to optimize the utilization of costly resources to the device. The execution of a given task in an RZ is allowed only when the cost D between them is finite. Figure 6 illustrates the computing of costs D between the five tasks and RZ_3 described in Figure 5.

3.3. Mapping and Scheduling of Hardware Tasks on RZs. It is well known that task mapping and scheduling are highly interdependent. The two issues therefore need to be addressed together for mapping and scheduling to be efficient. In order to analyze the scheduling of a given DAG



$RZ_3 = \{46RB_1, 53RB_2, 1RB_3, 1RB_4\}$ (E)
$RBCost_1 = 20$, $RBCost_2 = 80$, $RBCost_3 = 192$, $RBCost_4 = 340$
$D(A, RZ_3) = 20 \times |25–46| + 80 \times |33–53| + 192 \times |0\text{-}1| + 340 \times |0\text{-}1|$
$D(B, RZ_3) = \infty$ (Lack of $10RB_4$ in RZ_3 for B)
$D(C, RZ_3) = 20 \times |18–46| + 80 \times |42–53| + 192|0\text{-}1| + 340 \times |0\text{-}1|$
$D(D, RZ_3) = \infty$ (Lack of $11RB_4$ in RZ_3 for D)
$D(E, RZ_3) = 0$

Figure 6: Example of computing cost D with RZ_3.

of periodic tasks, it is sufficient to study its behavior for a time interval equal to the least common multiple of all the task periods, called the hyperperiod (HP). Consequently, the possible iterations of execution of each task A during the HP may be determined according to its period P_A, which is equal to $\lfloor HP/P_A \rfloor$. To resolve the subproblem of mapping and scheduling of hardware tasks on RZs, our methodology conducts three steps of spatial and temporal analyses. The first one checks the rightness of task precedences, dependences, and real-time functioning in the DAG by means of three essential constraints. Consequently, the DAG will be validated to launch the following analyses. The second analysis determines the lists of ready times of each task for each possible iteration during the hyperperiod. These lists take into account the periodicity, precedence, dependence, and deadline constraints. The third straightforward analysis takes as input the lists of ready times, searches at each iteration the possible execution intervals for each task, and therefore detects possible conflicts due to overlapping between execution intervals of parallel tasks on the same RZ. The third analysis pursues its processing to solve the detected overloads within RZs by performing either the migration of execution sections of some tasks respecting their predefined preemption points or by increasing the number of overloaded RZs.

3.3.1. Checking of Precedence, Dependence, and Real-Time Rightness in DAG. The first temporal analysis does not take into account spatial constraints and considers that there is an available RZ for each task. It also considers the periodicity of tasks. The main objective of this analysis is only to validate the correctness of task precedence and dependences and real-time constraints in the studied DAG. It is conducted by means of three essential constraints.

(a) Dependence Checking. More than precedence, we consider that the tasks related by an edge of precedence are also dependent. This means that the execution of a given task requires the data resulting from the execution of all its predecessors. This dependence is expressed by (7). Equation (7) focuses on periods and the amount of interchanged data between dependent tasks. It guarantees, when $P_A \geq P_B$ and $A \in \Pi_B$, that each data $(x_{A,B})$ produced by task A in its repetition A_i is consumed by its successor B during its iterations of execution B_i or $B_{j>i}$. When $P_A \leq P_B$ and $A \in \Pi_B$, at each iteration of execution of B, it ensures that there are the sufficient data $(y_{B,A})$ for B to be executed.

$$y_{B,A} \times \left\lfloor \frac{HP}{P_B} \right\rfloor = x_{A,B} \times \left\lfloor \frac{HP}{P_A} \right\rfloor, \quad \forall B, A \in \pi_B. \tag{7}$$

The previous equation eliminates the problem of data wastage and ensures that the data produced by all the iterations of each task are consumed by all the iterations of its successors. In this work, we focus on the case where $P_A \geq P_B$, for all $B, A \in \pi_B$.

(b) Precedence Checking. As we work in the case of $P_A \geq P_B$, for all $B, A \in \pi_B$, considering the periodicity constraint, this constraint claims that each iteration of execution A_i of a given task A may include only the iterations B_i or $\{B_{j<i}\}$ of its successors $\{B\}$ to ensure the correctness of the precedence constraint. During repetition A_i of a given task A, this constraint prohibits that B_i and $\{B_{j>i}\}$ repetitions of its successors $\{B\}$ coexist. In this way, during the HP, each ith execution of any task is preceded by the ith execution iterations of all its predecessors. To guarantee these rules, the following constraint expressed by (8) was developed to guide the selection of execution times and periods for tasks in the DAG:

$$\begin{aligned} &Tready_B + k \times P_B > Tready_A + (k-1) \times P_A, \\ &\forall B, A \in \pi_B, \quad k \in \mathbb{N}, \quad 1 \leq k \leq NbrIter_A. \end{aligned} \tag{8}$$

$NbrIter_A$ depicts the possible number of execution iterations of A during the HP. $Tready_A$ and $Tready_B$ are the ready times for tasks A and B without considering the spatial constraints. They are determined by searching the critical path corresponding to the task in the graph by using the ASAP (as soon as possible) technique. For each task B having a set of predecessors $\{A\}$, its ready time $Tready_B$ is computed as follows

$$\begin{aligned} Tready_B &= \max_{\{A \in \pi_B\}} \left(Tready_A + C_A \right), \\ Tready_B &= 0, \quad \text{if } \pi_B = \varnothing. \end{aligned} \tag{9}$$

(c) Real-Time Checking. Considering the periodicity constraint, this constraint analyses the respect of real-time constraints in the best case of spatial conditions in terms of number of RZs. By respecting the precedence and dependence constraints, (10) checks at each iteration whether each task may complete its execution before its strict absolute deadline. If the absolute deadline of the task for its last

iteration exceeds the HP, the deadline then turns into the HP. As we work in the case $P_A \geq P_B$, for all $A \in \pi_B$, the second expression of (10) satisfies the deadline constraint for the remaining repetitions of task B when its predecessors $\{A\}$ achieve all their execution iterations

$$\begin{aligned}
&\max\left(Tready_A + k \times P_A + C_A, Tready_B + k \times P_B\right) + C_B \\
&\quad \leq \min\left(Tready_B + (k+1) \times P_B, \mathrm{HP}\right), \quad \forall B, A \in \pi_B, \\
&\qquad k \in \mathbb{N}, \quad 0 \leq k \leq NbrIter_A - 1, \\
&Tready_B + k \times P_B + C_B \leq \min\left(Tready_B + (k+1) \times P_B, \mathrm{HP}\right) \\
&\quad \forall B, k \in \mathbb{N}, \quad \max_{A \in \pi_B}(NbrIter_A) \leq k \leq NbrIter_B - 1.
\end{aligned} \tag{10}$$

Respect of the three previous constraints validates the selection of periods and execution times for periodic tasks in the graph for the scheduling with precedence, dependence constraints, and under strict real-time constraints on an unlimited number of reconfigurable units. Nevertheless, the following temporal and spatial analyses will extract the corresponding number of RZs that will satisfy these constraints. When the previous constraints are unreal, the DAG is considered invalid and consequently it will be rejected or the features of its tasks will be modified and evaluated again till it respects the constraints expressed by (7), (8), and (10).

Consequently, with our methodology, we depart from the temporal analysis to construct a suitable physical architecture allowing a feasible schedule. As a final step at this stage, after fixing the physical architecture of the multi-reconfigurable-unit system, analytic resolution of mapping/scheduling provides the optimal scheduling of DAGs on target technology.

3.3.2. Determination of Lists of Ready Times. The temporal analysis does not consider the physical constraints and searches all the ready times in each execution iteration for all tasks in the graph by respecting the precedence, dependence, periodicity, and real-time constraints. This analysis yields, by means of (11), the lists $\{Treadymin_{B_i}, \ldots, Treadymax_{B_i}\}$ during which task B may start execution, where B_i denotes the ith iteration of task B within the HP. Outside this list, task B might not respect the predefined constraints, which would lead to unfeasible scheduling.

For each task B, $Treadymin_{B_i}$ is the lower bound time to start task execution at its iteration i in order to respect its precedence, dependence, and periodicity constraints. $Treadymax_{B_i}$ is the upper bound time from which the task can no longer start execution if its strict deadline and the data dependency and precedence of its successors are to be respected. To compute the *Treadymin* of tasks, we start computing from the top of the DAG. For the *Treadymax* calculation, we start from the bottom of the DAG, and for both, we proceed using the breadth-first search strategy

$$\begin{aligned}
&Treadymin_{B_1} \\
&\quad = 0 \quad \text{if } \pi_B = \varnothing, \\
&Treadymin_{B_i} \\
&\quad = \max\left(\max_{A \in \pi_B}\left(Treadymin_{A_i} + C_A\right), rmin_{B_i}\right), \\
&Treadymax_{B_1} \\
&\quad = \min\left(\min_{A \in \varphi_B}\left(Treadymax_{A_1}\right), P_B\right) - C_B, \\
&\qquad \varphi_B \text{ is the set of successors of } B, \\
&Treadymax_{B_{i/i>1}} \\
&\quad = \min\left(\min_{A \in \varphi_B}\left(Treadymax_{A_i}\right), rmax_{B_{i+1}}, \mathrm{HP}\right) - C_B.
\end{aligned} \tag{11}$$

$rmin_{B_i}$ and $rmax_{B_i}$ are the start times of the ith repetition of B according to its first ready times ($Treadymin_{B_1}$ and $Treadymax_{B_1}$). Hence, they are determined by incrementing $Treadymin_{B_1}$ and $Treadymax_{B_1}$ by P_B. For example, if we have a task B with a period = 8, a hyperperiod HP = 24, and we consider $Treadymin_{B_1} = 3$ and $Treadymax_{B_1} = 4$, then $rmin_{B_1} = Treadymin_{B_1} = 3$, $rmin_{B_2} = 3 + 8 = 11$, $rmin_{B_3} = 11 + 8 = 19$, $rmax_{B_1} = Treadymax_{B_1} = 4$, $rmax_{B_2} = 4 + 8 = 12$, $rmax_{B_3} = 12 + 8 = 20$.

3.3.3. Determination of Task Execution Intervals and the Number of RZs. This temporal and spatial analysis considers the RZ types resulting from the task clustering stage and searches the possible parallelism between tasks to study execution conflicts on the RZs. When an overload is detected in some RZs, the analysis starts by solving this problem through a migration mechanism; if migration does not produce a solution, it increments the number of overloaded RZs as required. This analysis searches first, by means of Algorithm 2, the execution intervals of each task for each possible iteration during the HP, then, using Algorithm 3, it deals with the overlapping execution intervals on the RZs to search the possible overloads, and finally, it uses Algorithm 4 to try to solve the found overloads by the migration mechanism; when this latter mechanism fails, additional RZs are inserted in the architecture of the multi-reconfigurable-unit system to solve the persisting overloads.

(a) Search of Execution Intervals. This step uses the functional model of tasks and determines their execution intervals by means of Algorithm 2 of worst-case temporal complexity equal to $O(\mathrm{NT} * \mathrm{HP}^3)$. An execution interval for a given task at a given iteration is an interval during which the task can be executed while satisfying all the predefined constraints.

For each task A in the DAG, Algorithm 2 produces the set of its possible execution intervals expressed by

(1) A, B // Tasks
(2) i // Natural, iterations of execution of task A
(3) $List\text{-}Tready_{A_i}$ // the list of ready times for task A during ith iteration
(4) $Tready_{A_i}$ // the ready times for task A during ith iteration
(5) $Tready_A$ // the ready times for task A from the current $Tready_{A_i}$
(6) $Execution\text{-}Interval_{A_i} = \oslash$ // the set of execution intervals for task A during ith iteration
(7) **for all** tasks A **do**
(8) **for all** execution iterations i of task A **do**
(9) $List\text{-}Tready_{A_i} = \{Treadymin_{A_i}, \ldots, Treadymax_{A_i}\}$
(10) **if** $i = 1$ **then**
(11) $Execution\text{-}Interval_{A_i} = \{[Tready_{A_i}, \min(Tready_{A_i} + P_A, \min_{B \in \varphi_A}(Treadymax_{B_i}), \mathrm{HP})], \forall Tready_{A_i} \in List\text{-}Tready_{A_i}\}$
(12) **else**
(13) **for all** $Tready_{A_1} \in List\text{-}Tready_{A_1}$ **do**
(14) $Tready_{A_i} = \mathrm{First}\ (List\text{-}Tready_{A_i})$
(15) **for all** $Tready_A \in \{Tready_{A_i}, \ldots, Treadymax_{A_i}\}$ **do**
(16) $Execution\text{-}Interval_{A_i} = Execution\text{-}Interval_{A_i} \bigcup \{[Tready_A, \min(Tready_{A_1} + i * P_A, \min_{B \in \varphi_A}(Treadymax_{B_i}), \mathrm{HP})]$, if $Tready_{A_1} + i * P_A$ is chosen as upper bound for this interval then it must respect: $Tready_A + C_A \leq Tready_{A_1} + i * P_A$, if HP is chosen then it must respect: $Tready_A + C_A \leq \mathrm{HP}\}$
(17) **end for**
(18) $List\text{-}Tready_{A_i} = \mathrm{next}\ (List\text{-}Tready_{A_i})$
(19) **end for**
(20) **end if**
(21) **end for**
(22) **end for**

Algorithm 2: Search of execution intervals.

$Execution\text{-}Interval_{A_i}$ at each possible iteration i during the HP. When this algorithm processes the first iteration of task A, the set of its possible execution intervals is determined directly (line 11) considering the precalculated ready times in $List\text{-}Tready_{A_1}$, and the periodicity and dependence constraints. For the next iterations, considering each possible $Tready_{A_1}$ for the purpose of respecting the periodicity constraint (line 13), at each iteration i, Algorithm 2 searches all the corresponding execution intervals starting with each possible $Tready_{A_i}$ (line 14, line 15), and considering the dependence and the deadline constraints, and the HP to determine the upper bound for each execution interval (line 16). In line 18, in order to guarantee the periodicity and dependence constraints, progressing from a $Tready_{A_1}$ to another in list $List\text{-}Tready_{A_1}$, Algorithm 2 must shift the list $List\text{-}Tready_{A_i}$ by omitting its first element, since ready times in each iteration i are also linked to the ready times of the first iteration.

(b) Search of Overlapping between Execution Intervals of Tasks in the Same RZ. For the first iteration, the parallel tasks on a given RZ are extracted directly from the DAG; hence, there is no parallel execution between a given task and its successors. However, in the next iterations i, searching parallel tasks must respect some rules. For the next iterations i, for parallel efficiency, a given task could overlap with its successors during their iterations j ($j < i$) on the same RZ. Moreover, this step must also consider the execution conflicts on the same RZ for a given task in its iteration i and other tasks that are in the iterations j ($j \leq i$) when there are no dependence constraints between them. This step prohibits simultaneous execution of several iterations of the same task.

The step defines for each task the possible RZs allowing its execution in terms of types and number of RBs based on computed cost D and then searches the execution conflicts on the RZs using Algorithm 3. Algorithm 3 has a worst-case temporal complexity equal to $O(\mathrm{NZ} * 2^{\mathrm{NT} * \mathrm{HP}})$.

During each iteration i, for each RZ, in order to find the parallel tasks with a finite cost D with the RZ, Algorithm 3 searches all the possible combinations of sets of execution intervals $\{Execution\text{-}Interval_{A_j}\}$ of all the tasks that can be executed on the current RZ and produce overlapping execution intervals respecting the rules described above (line 11). Then, from the resulting sets of execution intervals, Algorithm 3 extracts the execution intervals causing the conflicts in the current RZ which will result in *Comb* (line 12). *Comb* may contain two or more tasks. Each obtained *Comb* is processed individually to study the load of the RZ (line 13–line 24). For each *Comb*, we start the study only at the time at which all the tasks coexist to check for possible conflict and its consequence on the current RZ. Thus, Algorithm 3 searches the latest tasks $\{B\}$ in *Comb* (line 14) and for tasks $\{D\}$ that either have already started their executions and still running after $\{B\}$ arriving or have been ready before $\{B\}$ arriving; it searches their remaining execution times and periods (line 15–line 17) by promoting the tasks with the earliest deadlines and using the ASAP technique, especially in the case there are more than two tasks within the *Comb*. Finally, Algorithm 3 computes the

(1) A, B, D // Tasks
(2) i, j, k, m // Natural, iterations of execution of tasks
(3) $Execution\text{-}Interval_{A_i}$ // The set of execution intervals for task A during i^{th} iteration
(4) *Crossing-Combination* // All the possible combinations of sets $Execution\text{-}Interval_{A_i}$ of distinct tasks that give overlapping execution intervals on a given RZ, during HP
(5) *Comb* // Combination of overlapping execution intervals depicted by [A-min, A-max]
(6) C_D^r // The remaining execution time within task D
(7) P_D^r The remaining period within task D
(8) *Pmax* // The period of conflict for a combination of overlapping execution intervals
(9) **for all** iteration i **do**
(10) **for all** RZ **do**
(11) *Crossing-Combination* = $\{\{Execution\text{-}Interval_{A_j}\}/\bigcap_{j\leq i} Execution\text{-}Interval_{A_j} \neq \varnothing\}$
(12) *Comb* = one combination of overlapping execution intervals extracted from sets in *Crossing-Combination*.
(13) **for all** *Comb* in *Crossing-Combination* **do**
(14) In *Comb*, search the latest tasks $\{B\}$ in starting execution
(15) In *Comb*, search all the remaining tasks $\{D\}$ that are ready or start execution before $\{B\}$ and determine their remaining execution times: C_D^r and their remaining periods: P_D^r
(16) $C_D^r = C_D - (B\text{-min} - D\text{-min})$
(17) $P_D^r = P_D - (B\text{-min} - D\text{-min})$
(18) **if** ($\forall D\text{-min}$, $D\text{-min} = Tready_{D_1} + (k-1) * P_D$, $\forall D\text{-max}$, $D\text{-max} = Tready_{D_1} + k * P_D$, $\forall B\text{-min}$, $B\text{-min} = Tready_{B_1} + (m-1) * P_B$ and $\forall B\text{-max}$, $B\text{-max} = Tready_{B_1} + m * P_B / Tready_{D_1} \in \{Treadymin_{D_1}, \ldots, Treadymax_{D_1}\}$, $Tready_{B_1} \in \{Treadymin_{B_1}, \ldots, Treadymax_{B_1}\}$, and k and m are the iterations from which the execution intervals are taken respectively for tasks $\{D\}$ and $\{B\}$) **then**
(19) $RZ\text{-}Load = \sum_D C_D^r/P_D^r + \sum_B C_B/P_B$
(20) **else**
(21) $Pmax = \max_D(D\text{-max} - B\text{-min})$
(22) $RZ\text{-}Load = \sum_D C_D^r/Pmax + \sum_B C_B/Pmax$
(23) **end if**
(24) **end for**
(25) **end for**
(26) **end for**

Algorithm 3: Search of overlapping execution intervals and RZ loads.

load of the current RZ for this current *Comb* according to two cases. In the first case (Case 1, line 18-line 19), the remaining tasks with their new execution time values and periods are considered as virtual new tasks, and if their execution intervals in *Comb* corresponds to their total periods, Algorithm 3 intuitively computes the load of the RZ as mentioned in line 19 by considering the occupation rate (C_A/P_A) of each virtual new task A and each latest task $\{B\}$ in *Comb* on the current RZ.

In the second case (Case 2), the longest period of time (*Pmax*) that could be shared by the tasks in *Comb* is determined in line 21 and the load of the RZ is studied during *Pmax* (line 22).

This step deals with all possible cases of execution conflicts on all the RZs. At the end of this step, we obtain at each iteration during the HP, the loads of each RZ produced by each combination of tasks giving overlapping execution intervals. Consequently, we can detect the possible overloads in the RZs (*RZ-load* $>$ 1). Some combinations might be included within other ones. When overloads are detected on some RZs, the next step resolves the problem either by performing migration of tasks respecting their preemption points or by incrementing the number of overloaded RZs until the overloads are covered.

(c) Task Migration or Addition of RZs. Migration of tasks causing an overload on a given RZ during a combination of simultaneous executions might be total or partial. Total migration consists in replacing the entire execution of one or many tasks on another RZs. Partial migration consists in the reallocation of some execution sections within tasks predefined by their preemption points to nonoverloaded RZs. The migration is performed as explained by Algorithm 4. The worst-case temporal complexity of Algorithm 4 is $O(2^{NT*HP} * NT * NZ)$. Algorithm 4 searches the combinations producing overloaded RZs obtained by Algorithm 3 (line 6). Firstly, Algorithm 4 starts by total migration (line 8–line 15) to avoid the preemption of tasks resulting in difficulties related to context switches. During each *Comb* causing overload, the algorithm extracts the execution interval of the task that provides the largest occupation rate in the current *Comb* (line 10). In the case of equality between tasks, the task producing the fewest RZs for total migration during the *Comb* should be selected. The algorithm then determines the execution iteration of the task corresponding to this extracted interval and checks whether the iteration is also studied in another nonoverloaded RZs. Should this be the case, total migration of the task is allowed, and the task is eliminated from the overloaded RZ (line 11). If there are several RZs accepting total migration of the selected

```
(1) Comb // Combination of overlapping execution intervals
(2) Non-overload-comb // Boolean controlling after migration whether RZ is no longer overloaded by Comb
(3) RZ-Load // Load of the RZ corresponding to Comb
(4) RZ-migration // The set of RZs helpful for partial migration
(5) Task-migration // The set of tasks that might perform partial migration
(6) for all Comb giving overloaded RZ do
(7)       Non-overload-comb = False
(8)       while (Non-overload-comb = False) and (Comb ≠ ∅) do
(9)          // Total migration of tasks
(10)         Select the interval from Comb that gives the most heavy occupation rate and discard it from Comb.
(11)         Check whether the iteration, corresponding to the execution interval of the selected task, is studied on another
              non-overload RZ and update the load of the overloaded RZ after the elimination of the selected task.
(12)          if RZ-Load ≤ 1 then
(13)             Non-overload-comb = True
(14)          end if
(15)       end while
(16)    if Non-overload-comb = False then
(17)       Reinitialize RZ-Load and Comb with its tasks
(18)       Task-migration = ∅
(19)       RZ-migration = ∅
(20)       // Partial migration of tasks
(21)       Omit the tasks from the overloaded RZ, corresponding to Comb, that are also acceptable by another RZs (D ≠ ∞) and
           reduce their occupation rates from the loads of these RZs. These latter RZs with the overloaded RZ corresponding to
           Comb are included in the set RZ-migration. The omitted tasks are included in Task-migration
(22)       while Task-migration ≠ ∅ do
(23)          In Task-migration set, start by the task that gives the best trade-off between least number of RZs in RZ-migration
              where it could migrate and heaviest occupation rate in the overloaded RZ.
(24)          During Comb, within the selected task, choose the biggest execution sections that could be placed in RZs from the
               set RZ-migration without overloading them.
(25)          Update the load of RZs receiving execution sections from the selected task
(26)          if Some execution sections of the selected task are not placed then
(27)             Reinitialize the loads of RZs in RZ-migration to values before processing Comb
(28)             Increment the number of RZ corresponding to Comb up to ⌈RZ-Load⌉, go to 6
(29)          else
(30)             Discard the selected task from Task-migration.
(31)          end if
(32)       end while
(33)       // All the execution sections of tasks are placed
(34)       Non-overload-comb = True
(35)    end if
(36) end for
```

Algorithm 4: Total and partial migration or addition of RZs.

task, the RZ which is least required by tasks in the *Comb* is chosen. In cases of equality, the least loaded RZ is kept. After each total migration of a task, Algorithm 4 updates the load of the RZ corresponding to *Comb* and checks whether it is no longer overloaded (line 12–line 14). When total migration fails to resolve the overload in the RZ, partial migration takes place (line 16–line 35). Partial migration reinitializes the load of the current RZ corresponding to *Comb*. It searches the tasks in *Comb* that give finite D with other RZs and omits their occupation rates from the combinations on these latter RZs and from the current overloaded RZ considering the iteration of each task in *Comb* and includes the omitted tasks in the set *Task-migration*. The RZs accepting the tasks of *Comb* and the RZ corresponding to *Comb* are inserted within the set *RZ-migration* (line 21). Algorithm 4 attributes weights to tasks within *Task-migration* according to their occupation rates in *Comb* and according to the numbers of RZs in *RZ-migration* producing finite D with them. The task yielding the best trade-off between the two parameters is selected (line 23). Within the selected task, partial migration tries to reallocate its predefined execution sections in RZs from *RZ-migration* without causing an overload (line 24). Partial migration promotes the biggest execution sections respecting this rule. It starts by placing the selected execution section on the RZ which is least required by the other tasks in *Task-migration* waiting for partial migration. In cases of equality, it starts with the least loaded RZ in *RZ-migration*. Algorithm 4 pursues the processing of these partial migrations until the set *Task-migration* is empty. If partial migration does not successfully map all the execution sections of a given task in *Task-migration* (line 26), Algorithm 4 reinitializes the loads of the RZs to

their initial values before processing the *Comb* (line 27), stops its processing for the current *Comb*, and increments the number of the corresponding RZ to ⌈*RZ-Load*⌉ (line 28). When a given overloaded *Comb* is resolved by partial migration, Algorithm 4 takes into account the update of loads of all altered RZs (line 25) to be considered for the next *Comb*. Although migration might resolve many overloads for several combinations, it is still very difficult to perform as it is exhaustive and depends on the initial choices of tasks and RZs. One could consider the best case of studied migrations to minimize the number of added RZs. After each increment of RZ, the step must consider the added RZs to deal with the overloads of the remaining *Comb* not yet processed to avoid an excessive number of unusable RZs. As the proposed algorithm of migration is not exact, it might lead to an excessive number of RZs. This problem will be covered during resolution of mapping/scheduling which also optimizes the number of used RZs for the purpose of resource efficiency. However, an excess of RZs is very useful as it guarantees elimination of the infeasibility of analytic resolution and consequently, and it guarantees the feasibility of scheduling of the DAG.

At the end of this step, the number of RZs required to perform the scheduling of the chosen DAG on the FPGA is obtained. The resulting RZs constitute the target multi-reconfigurable-unit system where the scheduling of DAG will be conducted. The next step focuses essentially on determining the optimal valid scheduling that respects the predefined constraints.

3.3.4. Mapping and Scheduling Resolution. In the last step of the current stage, we concentrate on the resolution of mapping and scheduling tasks in the DAG on the resulting multi-reconfigurable-unit system. Mapping and scheduling are highly interlinked. It is well known that static multiprocessor scheduling of DAGs is performed by means of two actions: (i) assignment of an execution order expressed by temporal scheduling and (ii) assignment of processors expressed by mapping, for a set of tasks characterized by precedence and real-time constraints. With our methodology, based on a preemptive model, mapping consists in assigning each task to the most suitable RZs in terms of utilization of costly resources. Mapping is considered as spatial scheduling to a limited number of heterogeneous RZs. The scheduling searches the optimal scenario for task execution on RZs during the HP. At each execution iteration for a given task, it assigns for its execution sections specific times to launch their executions on the corresponding RZs. This scheduling is valid only when it satisfies predefined temporal constraints, and it should optimize the makespan of the graph, parallel efficiency, waiting time, and schedule response time. The proposed resolution leads to global static pipelined scheduling on a heterogeneous multi-reconfigurable-unit system because it is constructed at compile time, and it allows overlapping between execution iterations of distinct tasks on distinct RZs. Moreover, the problem of mapping/scheduling is a combinatorial optimization problem as it uses a discrete solution set and chooses the best combination of feasible assignments by optimizing a multiobjective function.

In this paper, resolution of mapping/scheduling is performed by mixed integer nonlinear programming solver as it is well adapted for this kind of problem. The mapping/scheduling problem is modeled by the quadruplet (constants, variables, constraints, and objective function).

Constants

NT:	Number of tasks constituting the DAG
NZ:	Number of RZs resulting from the task migration or addition of RZs analysis
NP:	Number of RB types existing in the target technology
i, o:	The references of iterations of executions during the HP
j:	The references of RZs
A, B:	The references of tasks
k:	The references of RB types
l, e:	The references of preemption points in tasks
t:	The references of times values, $t \in \{0, \ldots, \infty\}$
$D(A, \mathrm{RZ}_j)$:	The cost D between task A and RZ RZ_j
$RBCost_{\mathrm{k}}$:	The cost of each RB type
HP:	The hyperperiod in the DAG
C_A:	The worst case execution time of task A
P_A:	The period of task A which is equal to its relative deadline
$NbrPreemp_A$:	The number of possible preemption points within task A
$Preemp_{A,l}$:	The set of possible preemption points of task A. The first preemption point for all tasks is equal to 0
$Section_{A,l}$:	The execution section within task A provided by the predefined preemption point l
$NbrIter_A$:	The number of execution iterations of task A during the HP
$TimesValues_t$:	The set of possible times assigned to preemption points during the HP. It is equal to $\{0, \ldots, \infty\}$
$Pred_{A,B}$:	Binary constant takes 1 when task A has a precedence constraint with task B in the DAG
$Depend_A$:	Binary constant takes 1 when task A has data dependence constraints with tasks in the DAG
$Config_j$:	The configuration overhead of RZ_j
Com:	The maximum value of time for transmitting a data of unit length between two dependent tasks
$y_{A,B}$:	The amount of data sent by B for A execution.

Variables

$TUnicity_{j,A,l,t,i}$. Binary variable takes 1 when the preemption point l of task A is mapped to RZ_j at time t at iteration i. This

variable ensures the link between mapping and scheduling. In our resolution, the mapping/scheduling problem is solved when binary values are assigned to all these variables.

$PTimeRZ_{j,A,l,i}$. This variable represents the time value assigned to preemption point l of task A on RZ_j at iteration i during the HP. This variable is not defined when RZ_j gives infinite D with task A. It is obtained as expressed by

$$PTimeRZ_{j,A,l,i} = \sum_{\substack{t \\ 0 \le TimeValues_t \le \mathrm{HP}}} TUnicity_{j,A,l,t,i} \times TimeValues_t,$$
$$\forall 1 \le j \le \mathrm{NZ}, \quad 1 \le A \le \mathrm{NT}, \quad 1 \le l \le NbrPreemp_A,$$
$$1 \le i \le NbrIter_A, \quad D\left(A, \mathrm{RZ}_j\right) \neq \infty. \tag{12}$$

$PTime_{A,l,i}$. This variable provides the time value assigned to preemption point l of task A at iteration i during the HP and is calculated by means of

$$PTime_{A,l,i} = \sum_{\substack{1 \le j \le \mathrm{NZ} \\ D(A,\mathrm{RZ}_j) \neq \infty}} PTimeRZ_{j,A,l,i}, \quad \forall 1 \le A \le \mathrm{NT},$$
$$1 \le l \le NbrPreemp_A, \quad 1 \le i \le NbrIter_A. \tag{13}$$

$Occupation_{j,A}$. The total duration of execution of task A on RZ_j, after achievement of mapping/scheduling; it is computed by summing up all the execution sections of A mapped to RZ_j. This variable is not defined when RZ_j gives infinite D with task A. It is obtained using

$$Occupation_{j,A} = \sum_{\substack{l,i \\ 1 \le l \le NbrPreemp_A \\ 1 \le i \le NbrIter_A}} \sum_{\substack{t \\ 0 \le TimeValues_t \le \mathrm{HP}}} \left(TUnicity_{j,A,l,t,i} \times Section_{A,l}\right)$$
$$\forall 1 \le j \le \mathrm{NZ}, \quad 1 \le A \le \mathrm{NT}, \quad D\left(A, \mathrm{RZ}_j\right) \neq \infty. \tag{14}$$

$Exist_{j,A,l,i}$. This Binary variable tests whether the preemption point l of task A during its execution iteration i is mapped to RZ_j. This variable is not defined when RZ_j gives infinite D with task A. It is obtained by means of

$$Exist_{j,A,l,i} = \sum_{\substack{t \\ 0 \le TimeValues_t \le \mathrm{HP}}} TUnicity_{j,A,l,t,i}, \quad \forall 1 \le j \le \mathrm{NZ},$$
$$1 \le A \le \mathrm{NT}, \quad 1 \le l \le NbrPreemp_A,$$
$$1 \le i \le NbrIter_A, \quad D\left(A, \mathrm{RZ}_j\right) \neq \infty. \tag{15}$$

Constraints

Infeasibility of Mapping for Preemption Points. The constraint expressed by (16) prohibits the mapping of preemption points of task A to RZ_j giving infinite D with the task. Indeed, as explained in the second step of the task clustering stage, infinite D between a task and an RZ means that there is a lack of RBs in the RZ preventing task execution or an absence of RB types which are required by the task in the RZ. In addition, this constraint asserts that the time values chosen during mapping/scheduling for preemption points of tasks must be within the set of integers $\{0, \ldots, \mathrm{HP}\}$,

$$TUnicity_{j,A,l,t,i} = 0,$$
$$\text{when } D\left(A, \mathrm{RZ}_j\right) = \infty \quad \text{or} \quad TimeValues_t > \mathrm{HP},$$
$$\forall 1 \le j \le \mathrm{NZ}, \quad 1 \le A \le \mathrm{NT},$$
$$1 \le l \le NbrPreemp_A, \quad 1 \le i \le NbrIter_A, \quad 0 \le t \le \infty. \tag{16}$$

Uniqueness of Mapping/Scheduling Preemption Points on RZs. As expressed by (17), at each possible execution iteration i, if some tasks $\{A\}$ need to be scheduled on RZ_j at a time referred to as t, one task can be executed on this RZ at this time,

$$\sum_{\substack{A,l,i \\ 1 \le A \le \mathrm{NT} \\ 1 \le l \le NbrPreemp_A \\ 1 \le i \le NbrIter_A \\ D(A,\mathrm{RZ}_j) \neq \infty}} TUnicity_{j,A,l,t,i} \le 1, \quad \forall 1 \le j \le \mathrm{NZ},$$
$$0 \le TimeValues_t \le \mathrm{HP}, \quad 0 \le t \le \infty. \tag{17}$$

Uniqueness of RZs for Preemption Points. This constraint asserts that at each execution iteration i, each preemption point l of A must exist on a unique RZ_j (see (18)) and must be scheduled at a unique time referred to as t. This constraint also guarantees the achievement of task execution at each repetition, as all the preemption points delimiting the execution sections of the task are fitted on RZs at specified time values,

$$\sum_{\substack{j,t \\ 1 \le j \le \mathrm{NZ} \\ 0 \le TimeValues_t \le \mathrm{HP} \\ D(A,\mathrm{RZ}_j) \neq \infty}} TUnicity_{j,A,l,t,i} = 1, \quad \forall 1 \le A \le \mathrm{NT},$$
$$1 \le l \le NbrPreemp_A, \quad 1 \le i \le NbrIter_A. \tag{18}$$

Order of Preemption Points of a Task. At each execution iteration i for a task A, the preemption points must be scheduled in order, that is, the time value assigned to a preemption point $l + 1$ must be superior to the completion of execution section specified by the preemption point l. Equation (19) guarantees a consistent order of scheduling

of preemption points for a given task and the completion of their corresponding execution sections,

$$PTime_{A,l+1,i} \geq PTime_{A,l,i} + Section_{A,l}, \quad \forall 1 \leq A \leq \mathrm{NT}, \quad 1 \leq l \leq NbrPreemp_A - 1, \quad 1 \leq i \leq NbrIter_A. \tag{19}$$

Order between Iterations. Based on (19), the constraint expressed by (20) requires that during each execution iteration i for a task A, the scheduling of tasks must lead to the completion of each execution section within A before the beginning of its next iteration $(i + 1)$. Simultaneous executions of distinct iterations for the same task are not permitted,

$$PTime_{A,NbrPreemp_A,i} + Section_{A,NbrPreemp_A} \leq PTime_{A,1,i+1}, \quad \forall 1 \leq A \leq \mathrm{NT}, \quad 1 \leq i \leq NbrIter_A - 1. \tag{20}$$

Upper Bound of Task Execution. Based on (19) and (20), (21) indicates that for each scheduled preemption point for a given task A, at each execution iteration i, its execution section must be completed before the HP. Otherwise, the resulting scenario is not considered a feasible scheduling,

$$PTime_{A,NbrPreemp_A,NbrIter_A} + Section_{A,NbrPreemp_A} \leq \mathrm{HP} \quad \forall 1 \leq A \leq \mathrm{NT}. \tag{21}$$

Nonoverlapping Execution. This constraint eliminates task conflicts on a given RZ. For all execution iterations, when several tasks are scheduled on the same RZ, their predefined execution sections must not overlap (see (22)). This constraint is also applicable within the same task, that is, overlapping running between execution sections is not allowed either during the same iteration, which is implicitly expressed by (19) by the imposed order for preemption point scheduling, or between distinct iterations, which is implicitly expressed by the imposed order for execution iterations of a given task in (20). Equation (22) describes this constraint between two tasks A and B existing on the same RZ_j,

$$Exist_{j,A,l,i} \times Exist_{j,B,e,o} \times \left(PTimeRZ_{j,A,l,i} + Section_{A,l}\right) \leq PTimeRZ_{j,B,e,o},$$

or

$$Exist_{j,A,l,i} \times Exist_{j,B,e,o} \times \left(PTimeRZ_{j,B,e,o} + Section_{B,e}\right) \leq PTimeRZ_{j,A,l,i},$$

$$\forall 1 \leq A, B \leq \mathrm{NT}, \quad 1 \leq j \leq \mathrm{NZ}, \quad 1 \leq l \leq NbrPreemp_A, \quad 1 \leq e \leq NbrPreemp_B, \quad 1 \leq i \leq NbrIter_A, \quad 1 \leq o \leq NbrIter_B. \tag{22}$$

Precedence and Dependence Constraints. Equation (23) defines the precedence and dependence constraints. At each possible execution iteration i of task A, if A has dependence links with tasks $\{B\}$, its execution for this iteration can be launched only if the running within ith execution iterations of all its predecessors $\{B\}$ has been completed and the required data for A execution have been received,

$$PTime_{A,1,i} \geq PTime_{B,NbrPreemp_B,i} + Section_{B,NbrPreemp_B} + Com \times y_{A,B} \quad \forall 1 \leq A, B \leq \mathrm{NT}, \quad 1 \leq i \leq NbrIter_B, \quad Pred_{B,A} = 1. \tag{23}$$

Periodicity Constraint without Dependence Constraints. This constraint focuses on the periodic execution of tasks without predecessors in the DAG. Each task A is repeated periodically according to its specified period P_A in its functional model. Each repetition defines an execution iteration for the task. Equation (24) imposes constraints only on the first preemption point of the task as (19) defines an order for scheduling the preemption points in the same iteration which consequently will take into account the periodicity constraint for the other remaining preemption points,

$$PTime_{A,1,i} \geq (i-1) \times P_A \quad \forall 1 \leq A \leq \mathrm{NT}, \quad 1 \leq i \leq NbrIter_A, \quad Depend_A = 0. \tag{24}$$

Periodicity Constraint with Dependence Constraints. Equation (25) addresses tasks with predecessors in the DAG. These predecessors assert a dependence of execution on their successors at each repetition. Like for tasks without predecessors, tasks having dependence constraints with other tasks in the DAG must be repeated periodically as specified by their functional model and especially taking into consideration the first instants of completion of their predecessors. Indeed, a task with dependence constraints begins its first iteration after the execution achievement of all its predecessors and their data sending. Consequently, the periodicity constraint must consider the beginning time of these tasks with dependence constraints. Similarly, by means of (19), this constraint will be considered for all the preemption points,

$$PTime_{A,1,i} \geq \max_{\substack{B \\ B \neq A \\ Pred_{B,A}=1}} \left(PTime_{B,NbrPreemp_B,1} + Section_{B,NbrPreemp_B} + Com \times y_{A,B}\right) + (i-1) \times P_A, \quad \forall 1 \leq A \leq \mathrm{NT}, \quad 1 \leq i \leq NbrIter_A, \quad Depend_A = 1. \tag{25}$$

Deadline Constraint without Dependence Constraints. The key idea behind the constraint explained by (26) is to respect the strict real-time constraints in the case of the absence of

dependence constraints. At each given execution iteration i for a task A without predecessors, the running of task A must be completed before its absolute deadline,

$$PTime_{A,NbrPreemp_A,i} + Section_{A,NbrPreemp_A} \leq i \times P_A \quad \forall 1 \leq A \leq \mathrm{NT}, \quad 1 \leq i \leq NbrIter_A, \quad Depend_A = 0. \tag{26}$$

Deadline Constraint with Dependence Constraints. Equation (27) adheres to the deadline constraint in the case of the existence of dependence constraints between tasks in the DAG. Knowing that the beginning of a given task A with predecessors $\{B\}$ in the DAG is considered since the end of running of all its predecessors within their first execution iterations and the receipt of required data, the absolute deadline of A at each execution iteration i must be met as follows:

$$\begin{aligned} &PTime_{A,NbrPreemp_A,i} + Section_{A,NbrPreemp_A} \\ &\leq \max_{\substack{B \\ B \neq A \\ Pred_{B,A}=1}} \left(PTime_{B,NbrPreemp_B,1} + Section_{B,NbrPreemp_B} + Com \times y_{A,B}\right) + i \times P_A, \quad \forall 1 \leq A \leq \mathrm{NT}, \\ &\qquad 1 \leq i \leq NbrIter_A, \quad Depend_A = 1. \end{aligned} \tag{27}$$

Objective Function (F). The objective function guides resolution and helps to converge to the optimal solution. The minimization objective function F in (28) defines the optimal solution for the mapping/scheduling sub-problem by means of six parameters. F promotes the solution that provides the best trade-off of lowest values for these parameters. The parameters are all considered important for our methodology and are weighted according to their ranges of values. However, F gives priority to the number of occupied RZs more than other parameters. In fact, F increases exponentially with the minimum growth of the number of used RZs. Preference for this parameter is explained by the fact that our methodology is strongly dependent on physical architecture; hence, the minimum number of RZs enabling scheduling and satisfying the strict predefined constraints must be determined in order to avoid resource wastage and its consequences,

$$\begin{aligned} F &= \ln(Param) + \exp(NumberOfOccupiedRZs), \\ Param &= \delta_1 \times MakeSpan + \delta_2 \times WaitingTime + \delta_3 \times ResourceOptimization + \delta_4 \times MigrationNumber + \delta_5 \times ConfigurationOverhead. \end{aligned} \tag{28}$$

MakeSpan. It is determined by the length of the obtained scheduling. This parameter is considered the most pertinent factor in multiprocessor scheduling of DAGs as it evaluates the efficiency of the performed scheduling in terms of parallelism on processors and execution speed. Equation (29) reduces the *MakeSpan* of the DAG by minimizing the time of finishing execution within the last execution iterations for all tasks as they are linked by precedence and dependence constraints. Consequently, as the execution iterations are also highly dependent, it is necessary to start the execution of a given task for each iteration as soon as possible in order to minimize the makespan of the DAG,

$$MakeSpan = \max_{\substack{A \\ 1 \leq A \leq \mathrm{NT}}} \left(PTime_{A,NbrPreemp_A,NbrIter_A} + Section_{A,NbrPreemp_A}\right). \tag{29}$$

WaitingTime. It is determined by the response time of the scheduling for task execution. By means of (30), *WaitingTime* is computed as the sum of differences between the obtained runtimes of the tasks during the scheduling span and their effective execution times. *WaitingTime* includes the waiting time of tasks in the ready queue of the scheduler waiting for execution when their dependence and periodicity constraints are satisfied and the blocking time when the task is preempted,

$$\begin{aligned} &WaitingTime = \sum_{\substack{A \\ 1 \leq A \leq \mathrm{NT}}} (RunTime_A - NbrIter_A \times C_A), \\ &RunTime_A = \sum_{\substack{i \\ 1 \leq i \leq NbrIter_A}} \left(\left(PTime_{A,NbrPreemp_A,i} + Section_{A,NbrPreemp_A}\right) - (i-1) \times P_A\right) \\ &\text{if } Depend_A = 0, \\ &RunTime_A = \sum_{\substack{i \\ 1 \leq i \leq NbrIter_A}} \left(\left(PTime_{A,NbrPreemp_A,i} + Section_{A,NbrPreemp_A}\right) - \max\left(\max_{\substack{B \\ B \neq A \\ Pred_{B,A}=1}} \left(PTime_{B,NbrPreemp_B,1} + Section_{B,NbrPreemp_B} + Com \times y_{A,B}\right) + (i-1) \times P_A, \right. \right. \\ &\qquad \left. \left. \max_{\substack{B \\ B \neq A \\ Pred_{B,A}=1}} \left(PTime_{B,NbrPreemp_B,i} + Section_{B,NbrPreemp_B} + Com \times y_{A,B}\right) \right) \right), \\ &\text{else} \end{aligned} \tag{30}$$

ResourceOptimization. It is related to the physical features of the chosen technology. The parameter considers resource wastage and the utilization of costly resources. Both issues are involved in cost D computation in the first stage. Thus, resource optimization, using (31), targets mapping tasks with high occupation rates to the RZs providing the lowest cost D with them,

$$\begin{aligned} &ResourceOptimization \\ &= \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}}} \left(\frac{D\left(A,\ \mathrm{RZ}_j\right)^2 \times Occupation_{j,A}{}^2}{4} \right. \\ &\qquad \left. - D\left(A, \mathrm{RZ}_j\right) \times Occupation_{j,A} \right). \end{aligned} \tag{31}$$

MigrationNumber. Although the migration concept optimizes scheduling length, it helps to guarantee real-time constraints and to increase resource efficiency; it also raises configuration overhead. Bearing this in mind, we constructed (32) that aims at minimizing the number of migrations required. For a given task A, the first expression of (32), on the left of the summation, searches the number of migrations within the same execution iteration, and the second part of the summation calculates the performed migrations between iterations,

$$\begin{aligned} MigrationNumber = &\sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}\\ 1\le i\le NbrIter_A}} \ \sum_{\substack{1\le l<NbrPreemp_A\\ Exist_{j,A,l,i}\neq 0,\ Exist_{j,A,l+1,i}=0}} 1 \\ &+ \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}}} \ \sum_{\substack{1\le i<NbrIter_A\\ Exist_{j,A,NbrPreemp_A,i}\neq 0,\ Exist_{j,A,1,i+1}=0}} 1. \end{aligned} \tag{32}$$

ConfigurationOverhead. During the scheduling span, we do not consider the configuration overhead before the task execution on the RZ. This configuration overhead is negligible, that is, according to several experimentations performed with Virtex 5 technology, which uses parallel high-speed configuration ports, it represents less than 10% of the computation time of the task and does not impact the real-time functioning. However, after scheduling has been achieved, the configuration overhead, that impacts scheduling performance and power consumption, is evaluated using (33). The configuration overhead parameter should be reduced during resolution of mapping/scheduling. Equation (33) includes four expressions in the summation. The first expression, *Exp*1, is the initial configuration when tasks are launched by the scheduler for their first execution iteration as they did not exist on the RZs. The second expression, *Exp*2, represents the configuration overhead required to configure a task that migrates from one RZ to another within the same iteration i; the third expression, *Exp*3, depicts the configuration overhead required by a task that has finished its execution on a given RZ for an iteration i and started its $(i+1)$th iteration by migrating to another RZ. The fourth expression, *Exp*4, computes the configuration overhead resulting from intermediate tasks preempting a given task running on the same RZ. The first part of *Exp*4 considers the configuration overhead required in cases where a task A, during iteration i, finishes its execution section delimited by preemption point l, then it is preempted by other tasks to perform execution sections, which is then followed by task A performing its $l+1$ execution section at a later stage on the same RZ. This event is detected by the binary variable $CurrentIter_{j,A,l,i}$. This binary variable takes 1 if two successive execution sections for the same task at the same iteration are separated by one or more tasks on the same RZ. By means of variable $InterIter_{j,A,NbrPreempA,i}$, the second part of *Exp*4 deals with situations during which a task finishes its last execution section for a given iteration i on an RZ and starts the first execution section of $(i+1)$th iteration on the same RZ after the execution of other execution sections of other tasks on the same RZ. This variable takes 1 when at least one other task runs on the same RZ between two successive execution iterations of the same task,

$$\begin{aligned} &ConfigurationOverhead = Exp1 + Exp2 + Exp3 + Exp4 \\ &Exp1 = \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}\\ 0\le TimeValues_t\le \mathrm{HP}}} TUnicity_{j,A,1,t,1} \times Config_j \\ &Exp2 = \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}\\ 1\le i\le NbrIter_A}} \ \sum_{\substack{1\le l<NbrPreemp_A\\ Exist_{j,A,l,i}\neq 0,\ Exist_{j,A,l+1,i}=0}} Config_j \\ &Exp3 = \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}}} \ \sum_{\substack{1\le i<NbrIter_A\\ Exist_{j,A,NbrPreemp_A,i}\neq 0,\ Exist_{j,A,1,i+1}=0}} Config_j \\ &Exp4 = \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}\\ 1\le i\le NbrIter_A\\ 1\le l<NbrPreemp_A}} Config_j \times CurrentIter_{j,A,l,i} \\ &\qquad + \sum_{\substack{1\le A\le \mathrm{NT}\\ 1\le j\le \mathrm{NZ}\\ 1\le i<NbrIter_A}} Config_j \times InterIter_{j,A,NbrPreemp_A,i}. \end{aligned} \tag{33}$$

NumberOfOccupiedRZs. As explained in the previous step (determination of task execution intervals and the number of RZs), the obtained number of RZs ensures the feasibility of scheduling but is not necessarily the optimal number for valid scheduling. Thus, for the purpose of resource efficiency, (34) searches the lowest possible number of RZs required for

feasible real-time scheduling for the chosen DAG on a multi-reconfigurable-unit system,

$$NumberOfOccupiedRZs = \sum_{\substack{1 \le j \le \mathrm{NZ} \\ RZOccupation_j \neq 0}} 1,$$
$$RZOccupation_j = \sum_{\substack{A \\ 1 \le A \le \mathrm{NT}}} Occupation_{j,A}. \quad (34)$$

Other optimization parameters are also integrated in objective function F, but they are not mentioned in the formulation above as they implicitly impact the makespan. These parameters optimize the solutions that launch the execution of tasks as soon as their predecessors terminate their computations. In addition, they aim to bring the beginning of the tasks closer to the start time of each repetition.

After resolution of the first sub-problem of mapping/scheduling tasks on RZs, the resulting RZs must be placed on the target device. The second sub-problem of task cluster placement is a well-known problem in reconfigurable computing systems, and it differentiates our problem of scheduling on reconfigurable devices from that on software multiprocessors, generally not constrained by the physical architecture of the homogeneous processors. The second sub-problem of RZ placement is described in the next and last stage of our proposed methodology and like mapping/scheduling; it is solved by means of mixed integer programming as it is considered to be a combinatorial optimization problem.

3.4. Partitioning of Reconfigurable Device and Fitting of RZs. In general, the problem of hardware task placement includes two primary subfunctions: (i) *partitioning*, which handles free resource space on the reconfigurable device to identify potential sites enabling hardware task execution and (ii) *fitting*, which, according to the chosen criteria, selects a feasible placement solution among sites provided by the partitioning to fit the hardware task on a suitable physical location. Several research works have been proposed for the placement of hardware tasks on reconfigurable devices, and placement can be divided into on-line and off-line placement. For on-line placement, most scenarios propose as a partitioning technique to search for the maximal empty physical rectangles in the free space to avoid the resource wastage. For example, the two techniques proposed by Bazargan et al. in [13], one referred to as "*Keeping All Maximal Empty Rectangles*" searches all overlapping maximal empty rectangles and the other referred to as "*Keeping Nonoverlapping Empty Rectangles*" keeps only the distinct holes in the free space. In [14], Walder et al. describe efficient partitioning algorithms, the main one being the *On-The-Fly* (*OTF*) partitioner which resizes empty rectangles only if the new arrived task overlaps them. Handa and Vemuri introduce staircase partitioning in [15]. Marconi et al. extend in [16] Bazargan's partitioner by means of an *Intelligent Merging* (*IM*) algorithm that dynamically combines three techniques for managing free resources. In [17], Ahmadinia et al. present a new method of on-line partitioning which manages occupied space on devices rather than free space, given the difficulty of managing empty space and the resulting vast increase in empty rectangles. Regarding the fitting subfunction, the works are essentially based on bin-packing rules, such as in [13] which describes the *First Fit*, *Best Fit*, and *Bottom left* bin-packing algorithms. In [14], Walder et al. employ a hash matrix to perform fitting based on *Best Fit*, *Worst Fit*, *Best Fit with Exact Fit*, and *Worst Fit with Exact Fit* algorithms. In [17], Ahmadinia et al. fit tasks on sites, reducing communication costs, by means of the *Nearest Possible Position* algorithm. For off-line placement, many research works deal with metaheuristics such as simulated annealing, greedy research, and *Keeping All Maximal Empty Rectangle-Best Fit* as described in [13]. In [18], ElGindy et al. describe task rearrangement by using a genetic algorithm approach with the *First Fit* strategy. Considering task placement as a 2D packing problem, researchers propose off-line heuristics such as Lodi et al. who describe the *Next-Fit Decreasing Height*, *First-Fit Decreasing Height*, and *Best-Fit Decreasing height* algorithms in [19], as well as the *Floor-Ceiling* and *Knapsack packing* algorithms in [20]. Integer linear programming is also proposed by Panainte et al. in [21] to model the problem of hardware task placement by minimizing the resources area that is reconfigured at runtime.

With our methodology, we focus on off-line placement of RZs on reconfigurable devices. As the reconfigurable devices afford a limited number of reconfigurable resources and DAG includes bounded numbers of tasks, analytic resolution of the placement sub-problem is the recommended method as it guarantees the optimal solution to perform efficient allocation of tasks on FPGA. The placement of RZs is a combinatorial mono-objective problem. It consists in searching for suitable coordinates for each RZ in the FPGA among all the possible combinations of discrete coordinates. In this section, similarly to the mapping/scheduling sub-problem, the resolution of RZ placement on the reconfigurable device is accomplished through mixed integer programming and optimizes resource efficiency. The achievement of RZ placement results in a 2D physical locations for each task cluster. These physical locations are depicted by reconfigurable physical blocs (RPBs) and are represented by their RB model as described in (35). RZs are abstractions of hardware task clusters, and RPBs are the final reconfigurable units where tasks may be executed. With the concept of run-time partial reconfiguration, after the mapping/scheduling results are obtained respecting the strict predefined constraints and optimizing several criteria, task execution sections are reconfigured and executed on RPBs as restricted by the schedule scenario. Thus, the task bitstreams on their associated RPBs are created at compile time. These bitstreams are used during the DAG running as indicated by scheduling which specifies the corresponding state for each task on each RPB,

$$\mathrm{RPB}_j\_\mathrm{RB} = \left\{\gamma_{j,k}\mathrm{RB}_k\right\}, \quad \gamma_{j,k} \in \mathbb{N},$$
$$1 \le j \le \mathrm{NZ}, \quad 1 \le k \le \mathrm{NP}. \quad (35)$$

As shown in Figure 7, the partitioned RPBs on a reconfigurable device for a given RZ must include all its required RB types to ensure the execution of the tasks. The partitioned RPBs for a given RZ might contain some RBs that are not required by the RZ. For instance, RB_4 is inserted within RPB_3 but is not used by the corresponding RZ. This resource inefficiency is explained by the rectangular shape of the RPBs. Thus, the number of RBs included in the RPB could exceed the required RBs in the RZ. Resource inefficiency is also due to the heterogeneity of the device; partitioning could book some RBs which are not used by the RZ. RB utilization efficiency is an important metric parameter for evaluating placement quality. During RZ placement resolution, we focus on fitting RZs as closely as possible to RPBs in terms of number and types of RBs.

By means of mixed integer linear programming solver, both the partitioning and fitting subfunctions are solved simultaneously. The RZ placement sub-problem is modeled by the quadruplets (Constants, Variables, Constraints, and Objective Function).

Constants

NZ: Number of RZs resulting from mapping/scheduling resolution

NP: Number of RB types existing in the target technology RZ features, RB features

Device features

Device_Width: The width of the device

Device_Height: The height of the device

Device_RB: The RB model of the device.

Variables

Placement resolution consists in assigning discrete values for the four coordinates specifying the RPB_j for each RZ_j. Each RPB_j is constructed by four coordinates:

X_j: The abscissa of the upper left vertex of RPB_j

Y_j: The ordinate of the upper left vertex of RPB_j

$WRPB_j$: The abscissa of the upper right vertex of RPB_j

$HRPB_j$: The ordinate of the bottom left vertex of RPB_j.

Constraints

Heterogeneity Constraint. As RZs are fitted on RPBs, during RPB partitioning, the number of each RB type within the RPBs (X_j, $WRPB_j$, Y_j, $HRPB_j$) must be greater than or equal to those in the RZs ($\beta_{j,k}$) as formulated by (36) in order to satisfy the RB requirements of the RZs. Because of the heterogeneity of RBs in the device and the rectangular shape of RPBs, the partitioned RPBs could include some RB types not required by the RZs. Moreover, the number of RB types in the RPBs and included in the RZs might exceed that

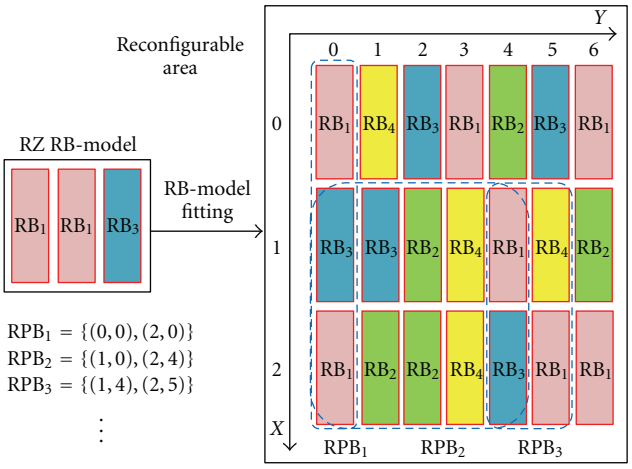


Figure 7: Example of RPBs for RZ.

required by the RZs. This resource inefficiency is minimized by means of the objective function,

$$\beta_{j,k} \leq \sum_{\substack{X_j \leq m \leq WRPB_j \\ Y_j \leq n \leq HRPB_j}} \sum_{device\_RB[m][n]=RB_k} 1,$$

$$\forall 1 \leq j \leq NZ, \quad 1 \leq k \leq NP,$$

$$RZ_j\_RB = \left\{\beta_{j,k}\ RB_k\right\}, \quad RPB_j\left(X_j, WRPB_j, Y_j, HRPB_j\right). \tag{36}$$

Nonoverlapping between RPBs. As expressed by (37), this constraint restricts the fitting of RZs on nonoverlapping RPBs

$$X_q > WRPB_j \quad \text{or} \quad X_j > WRPB_q$$
$$\text{or} \quad Y_q > HRPB_j \quad \text{or} \quad Y_j > HRPB_q, \tag{37}$$
$$\forall j \neq q, \quad 1 \leq j, \quad q \leq NZ.$$

Objective Function (F). Objective function F comprises one parameter which is *ResourceEfficiency*. This primordial parameter focuses on finding the closest RPB partitioned on the FPGA to fit each RZ in terms of number and types of RBs. By respecting both previous constraints, (38) assesses placement quality after RZ insertion on the selected RPBs in order to achieve the optimal solution. Increasing resource efficiency reduces configuration overhead,

$$ResourceEfficiency = \sum_{\substack{1 \leq j \leq NZ \\ 1 \leq k \leq NP}} RBCost_k \times \left(\gamma_{j,k} - \beta_{j,k}\right),$$
$$RPB_{j\,RB} = \left\{\gamma_{j,k}\ RB_k\right\}, \quad RZ_j\_RB = \left\{\beta_{j,k}\ RB_k\right\}, \tag{38}$$
$$1 \leq j \leq NZ, \quad 1 \leq k \leq NP.$$

Both subproblems, that is, (i) mapping/scheduling tasks on RZs and (ii) partitioning/fitting RZs on the FPGA, are successively solved by means of powerful solvers dedicated

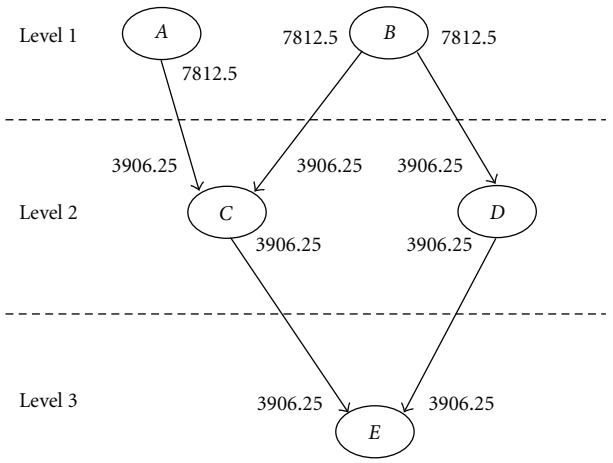


Figure 8: Tasks in the DAG.

by the AIMMS [22] environment. The chosen solvers for mixed integer programming satisfy predefined constraints and provide an optimal solution in an acceptable time frame. The following section shows how our proposed methodology may be applied to an example of DAG on Virtex 5 technology; the section also evaluates scheduling and placement quality.

4. Application and Results

The experiments were separated into two parts. The first part focuses on constructing the DAG as restricted by the rules of the mapping/scheduling stage, detailing the performed analysis and showing the results of task mapping/scheduling on the resulting RZs with a performance evaluation. The second part demonstrates placement resolution of the RZs on the reconfigurable device, Virtex 5 FX70T (ML507 board), and describes the metrics used to measure placement quality.

To illustrate the methodology proposed for static scheduling of DAGs on multi-reconfigurable-unit systems with precedence and deadline constraints, we designed the fully connected DAG shown in Figure 8. Tasks were selected from opencores (http://www.opencores.org/).

Each task is characterized by its worst-case execution time (WCET) which also integrates the communication latency for data sending to task successors, its period (relative deadline), and the set of preemption points. The number of needed slices is also synthesized for each task. In the slices, either only LUTs are used or only registers are used or both of them are used. The number of slices used by the task for LUTs and those for registers are mentioned in the third column of Table 2.

The configuration overheads were computed by (3) and by using our own IP based on the ICAP reconfiguration port having a width of 32 bits and a frequency of 100 MHz. Values on the DAG edges represent the average number of packets of 16 words of 32 bits for communication between tasks.

With the Virtex 5 technology [23], there are four main resource types, that is, CLBL, CLBM, BRAM, and DSP. Considering reconfiguration granularity, the RBs in Virtex 5 are vertical stacks composed of the same type of resources: RB_1 (20 CLBMs), RB_2 (20 CLBLs), RB_3 (4 BRAMs), and RB_4 (8 DSPs). In our application, for Virtex 5 FX70T, we considered that the lower the number of the RB types on the device and the higher its functioning speed, the more its cost would increase. We, respectively, assigned 16, 10, 168, and 194 as the *RBCost* for RB_1, RB_2, RB_3, and RB_4. We modeled the Virtex 5 FPGA and the hardware tasks with their RB-models. The task features are shown in Table 2.

The overview functioning of selected tasks is described below.

Reed-Solomon (*A*). It is an error correcting code that works by oversampling the Galouee's field polynomial constructed from the data to be coded. It is widely used to recover data from possible errors that occur during disk reading.

AES (*B*). Advanced encryption standard can decrypt input data using 256-bit key.

IIR (*C*). It performs infinite impulse response low-pass filter which cut off frequency in the range of 0.1 to 0.4 of the sampling frequency.

FFT (*D*). It performs 64 points Fast Fourier Transform where the data and coefficients are adjustable in the range 8 to 16 bits.

Basic DES (*E*). It performs single Data Encryption Standard by processing 16 identical rounds. Each round encrypts 32-bit block using 64-bit key to provide 32-bit output block.

We applied the proposed methodology for placement and scheduling of DAGs on reconfigurable devices and obtained the following results.

4.1. Task Clustering Results. This stage executes Algorithm 1 which results in RZ types (RZ_1, RZ_2, RZ_3) represented by their RB models in the first column of Table 3. RZ_1 is inserted by *A* and *B*, RZ_2 is provided by *C* and *D*, and task *E* creates RZ_3.

When the maximal numbers of RBs within a constructed RZ are provided by several tasks, the configuration overhead of the RZ must be recomputed as described by (3). In our application, the maximum numbers of RBs within RZs are created by the same task. Thus, the RZ configuration overheads are provided by the predefined task features shown in Table 2. The second step of this stage computes the costs *D* between the DAG tasks and the resulting RZs. In Table 3, the bold numbers are the lowest costs *D* for tasks with the most suitable RZs.

The resolution of scheduling of the chosen DAG on Virtex 5 FX70T is detailed in Sections 4.2 and 4.3.

4.2. Mapping/Scheduling Results and Scheduling Quality Evaluation. DAG behavior is studied within the time interval (HP) of a period equal to the least common multiple of the

Table 2: Task features.

Reference	Name	Slices (LUTs/registers)	WCET (μs)	Period (ms)	Configuration overhead (μs)	Preemption points (μs)	RB model
A	Reed-Solomon	2234/1224	26576	500	1116	14770, 20200	$\{8RB_1, 7RB_2, 1RB_3, 1RB_4\}$
B	AES	1150/507	42733	500	675	5000, 11000, 23000, 30000	$\{5RB_1, 5RB_2, 1RB_3, 1RB_4\}$
C	IIR	678/565	11805	250	524	3334, 10000	$\{4RB_1, 1RB_2, 0RB_3, 1RB_4\}$
D	FFT	2333/2010	11806	250	1199	5000, 11667	$\{9RB_1, 7RB_2, 0RB_3, 1RB_4\}$
E	Basic DES	387/192	22280	250	188	7500, 15000, 19000	$\{2RB_1, 2RB_2, 0RB_3, 0RB_4\}$

Table 3: RZ types and *D* costs.

	A	*B*	*C*	*D*	*E*
$RZ_1\{8RB_1, 7RB_2, 1RB_3, 1RB_4\}$	**0**	**68**	292	∞	448
$RZ_2\{9RB_1, 7RB_2, 0RB_3, 1RB_4\}$	∞	∞	**140**	**0**	356
$RZ_3\{2RB_1, 2RB_2, 0RB_3, 0RB_4\}$	∞	∞	∞	∞	**0**

periods of its tasks which is equal to 500 ms. Consequently, the execution iterations of tasks during the HP are deducted as follows:

$$\begin{aligned} NbrIter_A &= \left\lfloor \frac{\mathrm{HP}}{P_A} \right\rfloor = \left\lfloor \frac{500}{500} \right\rfloor = 1, \\ NbrIter_B &= 1, \qquad NbrIter_C = 2, \\ NbrIter_D &= 2, \qquad NbrIter_E = 2. \end{aligned} \tag{39}$$

The three steps preceding mapping/scheduling resolution are detailed hereunder. In the remaining sections of the paper, the values are expressed in μs.

4.2.1. Checking of Precedence, Dependence, and Real-Time Rightness in the DAG. The constraints for checking precedence, dependence, and real-time rightness in the proposed DAG are detailed below.

Dependence Checking. In Figure 8, it is obvious that the periods of tasks between the levels are guarded or dubbed. Thus, the successors are more repetitive than the predecessors.

In this case, all the data produced on an edge linking a source task and its successor must be sufficient and consumed by the latter task. As can be seen in Figure 8, the constraint expressed by (7) is satisfied for all the edges. For example, task *B* produces at its unique execution iteration a data of size $x_{B,C}$ equal to 7812.5 packets on its outgoing edge. These data are consumed totally by its first successor *C* during its two iterations where it consumes $y_{C,B}$ (3906.25 packets) data at each iteration. A similar remark can be made for data interchanged on the edge linking *B* and *D*.

Precedence Checking. This constraint is satisfied by all the interdependent tasks in the DAG and is checked by means of (8) and (9). $Tready_A = 0$, $Tready_B = 0$, $Tready_C = \max(C_A, C_B) = 42733$, $Tready_D = C_B = 42733$, $Tready_E = \max(Tready_C + C_C, Tready_D + C_D) = \max(42733 + 11805, 42733 + 11806) = 54539$.

Task C and Task B Precedence:

$$K = 1\text{: } 42733 + 250000 = 292733 > 0. \tag{40}$$

Task C and Task A Precedence:

$$K = 1\text{: } 42733 + 250000 = 292733 > 0. \tag{41}$$

Task D and Task B Precedence:

$$K = 1\text{: } 42733 + 250000 = 292733 > 0. \tag{42}$$

Task C and Task E Precedence:

$$\begin{aligned} &K = 1\text{: } 54539 + 250000 = 304539 > 42733, \\ &K = 2\text{: } 54539 + 2 * 250000 \\ &\qquad = 554539 > 42733 + 250000 \\ &\qquad = 292733. \end{aligned} \tag{43}$$

Task D and Task E Precedence:

$$\begin{aligned} &K = 1\text{: } 54539 + 250000 = 304539 > 42733, \\ &K = 2\text{: } 54539 + 2 * 250000 \\ &\qquad = 554539 > 42733 + 250000 \\ &\qquad = 292733. \end{aligned} \tag{44}$$

The above tests prove that each execution iteration A_i of a given task *A* includes only execution iteration B_i or iterations preceding B_i of each successor *B*. Consequently, the precedences between tasks in the chosen DAG are correct.

Real-Time Checking. For the purpose of real-time functioning, (10) considers dependence constraints between tasks and makes it possible to verify, in the best case of spatial conditions and by providing an RZ for each task, the rightness of real-time functioning according to computation times and periods assigned to tasks in the DAG.

An example of testing real-time constraints for task *C* is shown hereunder.

Task C

$$\Pi_C = \{A, B\},$$
$$K = 0:\ \max(0 + 26576, 42733 + 0) + 11805 = 54538 \le \min(42733 + 250000, 500000) = 292733\ (A),$$
$$\max(0 + 42733, 42733 + 0) + 11805 = 54538 \le \min(42733 + 250000, 500000) = 292733(B)$$
$$K = 1:\ (42733 + 250000) + 11805 = 304538 \le \min(42733 + 2 * 250000, 500000) = 500000. \quad (45)$$

By providing an RZ for each task, the deadlines of the tasks are met taking into consideration the dependence and precedence constraints.

The above three tests validate the chosen graph in terms of dependence, precedence, and real-time constraints and in the following section; spatial/temporal analyses are performed in order to determine the required number of RZs allowing valid scheduling for the DAG.

4.2.2. Determination of Lists of Ready Times. The first temporal analysis uses (11) to search the ready times of tasks which are helpful to limit execution intervals. Without considering the number of available reconfigurable units and taking into account precedence, dependence, periodicity, and real-time constraints, the ready times are determined as shown in Figure 9.

4.2.3. Determination of Task Execution Intervals and the Number of RZs. The first step of this spatial/temporal analysis uses Algorithm 2 to determine the possible execution intervals of tasks based on lists of ready times and respecting specified constraints. The possible execution intervals for tasks in the DAG are shown below

$$\textit{Execution-Interval}_{A_1} = \{[0, 215915], [1, 215915], \ldots, [i, 215915], \ldots, [189339, 215915]\},$$
$$\textit{Execution-Interval}_{B_1} = \{[0, 215914], [1, 215914], \ldots, [i, 215914], \ldots [173181, 215914]\},$$
$$\textit{Execution-Interval}_{C_1} = \{[42733, 227720], [42734, 227720], \ldots, [215915, 227720]\},$$
$$\textit{Execution-Interval}_{C_2} = \{[292733, 477720], [292734, 477720], \ldots, [465915, 477720]\},$$
$$\textit{Execution-Interval}_{D_1} = \{[42733, 227720], [42734, 227720], \ldots, [215914, 227720]\},$$
$$\textit{Execution-Interval}_{D_2} = \{[292733, 477720], [292734, 477720], \ldots, [465914, 477720]\},$$
$$\textit{Execution-Interval}_{E_1} = \{[54539, 304539], [54540, 304540], \ldots, [i, i + P_E], \ldots, [227720, 477720]\},$$
$$\textit{Execution-Interval}_{E_2} = \{[304539, \mathrm{HP}], [304540, \mathrm{HP}], \ldots, [477720, \mathrm{HP}]\}. \quad (46)$$

Using Algorithm 3, the analysis therefore integrates the spatial aspect and studies possible conflicts between tasks in shared RZs to detect possible overloads. Algorithm 3 must respect the rules explained in Section 3.3.3(b) to search the possible overlapping execution intervals. At each iteration, for each RZ, Algorithm 3 searches all the combinations of tasks causing overlapping execution intervals and inserts them in *Crossing-Combination* and computes the loads of RZs according to two cases as detailed as follows.

Iteration 1

RZ_1. *Crossing-Combination* = {{*Execution-Interval*$_{A_1}$, *Execution-Interval*$_{B_1}$}}. This *Crossing-Combination* contains several combinations of overlapping execution intervals between A and B. In the worst case of overlapping, such as the overlapping between $[0, 215915]$ from *Execution-Interval*$_{A_1}$ and $[0, 215914]$ from *Execution-Interval*$_{B_1}$, the load of RZ_1 obtained by Case 2 is 32% satisfying the predefined constraints.

RZ_2. *Crossing-Combination* = {{*Execution-Interval*$_{C_1}$, *Execution-Interval*$_{D_1}$}}. Similarly, this *Crossing-Combination* provides several combinations of overlapping execution intervals between C and D. In the worst case of overlapping, such as when both tasks have confused execution intervals equal to $[42733, 227720]$, the load of RZ_2 obtained in Case 2 is 12%. Another possible solution consists in total migration, as expressed by Algorithm 4, of task C to the RZ RZ_1 to improve execution parallelism and to minimize

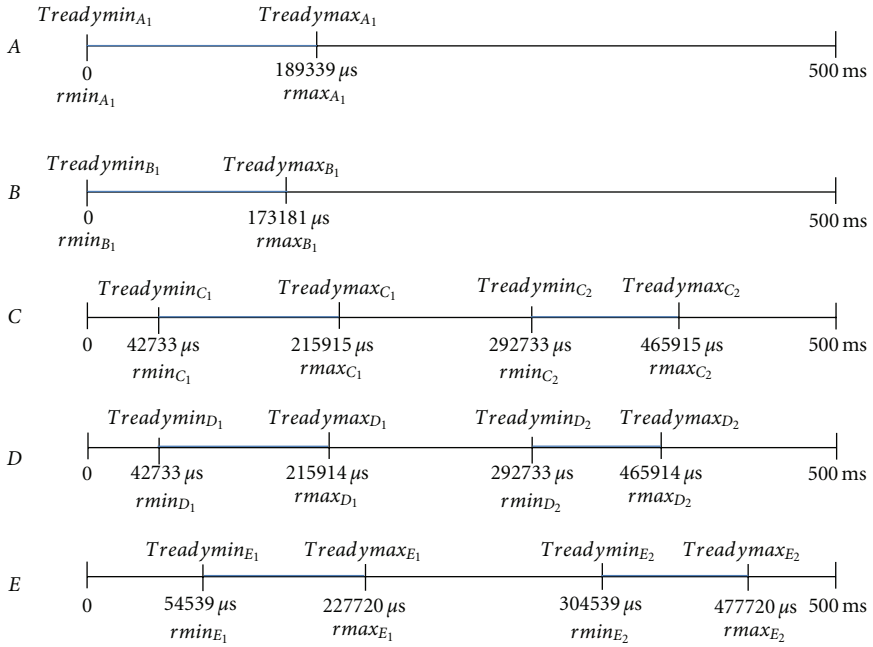


Figure 9: *Treadymin* and *Treadymax* of tasks in the DAG.

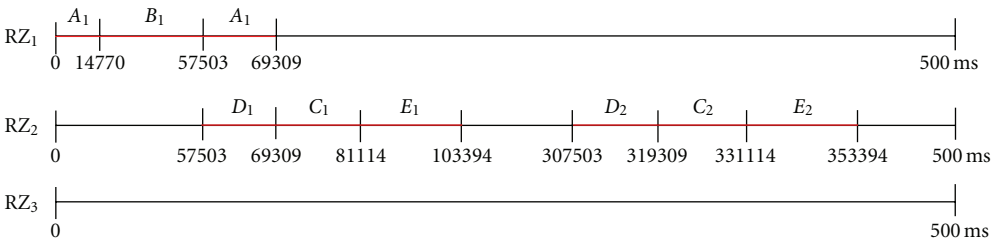


Figure 10: Mapping/scheduling resolution.

the makespan of the DAG. Thus, as soon as tasks *A*, *B* computations are achieved, tasks *C* and *D* may be launched, respectively, on RZ_1 and RZ_2.

During the first iteration, no combination of detected overlapping execution intervals causes an overload on RZs.

Iteration 2

RZ_2. *Crossing-Combination* = {{*Execution-Interval*$_{C_2}$, *Execution-Interval*$_{D_2}$}, {*Execution-Interval*$_{C_2}$, *Execution-Interval*$_{D_2}$, *Execution-Interval*$_{E_1}$}, {*Execution-Interval*$_{E_1}$, *Execution-Interval*$_{D_2}$}, {*Execution-Interval*$_{E_1}$, *Execution-Interval*$_{C_2}$}} {*Execution-Interval*$_{C_2}$, *Execution-Interval*$_{D_2}$}: As with the first iteration, all the combinations of overlapping execution intervals between tasks *C* and *D* do not cause an overload on RZ_2. The load of RZ_2, in the worst case of confused execution intervals, is equal to 12%. Hence, in the worst case, tasks *C* and *D* could be executed consequently on RZ_2 respecting all the predefined constraints.

As expressed by the rules set out in Section 3.3.3(b), detection of conflicting tasks must consider current and preceding iterations.

{*Execution-Interval*$_{C_2}$, *Execution-Interval*$_{D_2}$, *Execution-Interval*$_{E_1}$}: This set provides many combinations of overlapping execution intervals between tasks *C*, *D*, and *E*. However, following Algorithm 3 and the optimization parameters for reducing the makespan as explained at the end of mapping/scheduling resolution, task *E* starts its first iteration immediately after the achievement of its predecessors *C* and *D* which is equal, in the worst case, to 227720 μs, and RZ_2 is idle at this time. Consequent-ly, according to this start time of the first repetition, task *E* finishes its execution at 250 ms which is not attainable by the least start time of the second iterations of *C* and *D*, equal to 292733 μs. Thus, this set leads to no overlapping of execution between *C*, *D*, and *E* and becomes equivalent to the *Crossing-Combination* set: {*Execution-Interval*$_{C_2}$, *Execution-Interval*$_{D_2}$}. Another possible solution for this set is total migration of *C* to RZ_1 and *E* to RZ_3. This solution guarantees efficient parallelism between task computations.

{*Execution-Interval*$_{E_1}$, *Execution-Interval*$_{D_2}$}, {*Execution-Interval*$_{E_1}$, *Execution-Interval*$_{C_2}$}: As explained for the previous set, these sets of overlapping execution intervals do not cause an overload on RZ_2.

RZ_1

Crossing-Combination = {{*Execution-Interval*$_{E_1}$, *Execution-Interval*$_{C_2}$}}. As described for RZ_2 with the last three sets, tasks *C* and *E* do not cause overloads on RZ_1 during this *Crossing-Combination*. In fact, application of Algorithm 3 and the optimization parameters results in no remaining execution of the first repetition of *E* when *C* starts its second execution iteration.

After these spatial and temporal analyses, one can conclude there is no need to perform migration or to add other RZs. The three RZs resulting from the clustering stage are sufficient to perform valid scheduling of the chosen DAG. Thus, RZ_1, RZ_2, and RZ_3 represent the multi-reconfigurable-unit system for DAG scheduling. Based on powerful solvers dedicated by the AIMMS environment, we obtained the following mapping/scheduling which is evaluated in the next section.

4.2.4. Mapping and Scheduling Resolution. Figure 10 describes the mapping/scheduling obtained for five tasks in the DAG on three RZs. As the number of RZs is the highest priority parameter in the objective function, the resolution concludes that for scheduling this DAG, only two RZs are required. RZ_3 is not used, and hence it will not be placed on Virtex 5 FX70T in the third stage. The elimination of RZ_3 enhances resource efficiency and enables the DAG to be extended and performed on the FPGA for future needs. RZ_1 and RZ_2 are selected by the solver to remain in the multi-reconfigurable-unit system since *A* and *B* can execute only on RZ_1, task *D* can execute only on RZ_2, and tasks *C* and *E* can run on both RZs.

The obtained mapping/scheduling takes into account all the optimization parameters described in the sub-problem formulation and satisfies the specified constraints. Tasks *A* and *B* launch their computations first as they have no predecessors, and both can only execute on RZ_1. To reduce the makespan of the DAG, the scheduler decides the preemption of *A* at its second preemption point (14770 μs) to enable the execution of task *B*, the termination of which permits the execution of task *D* on RZ_2. The scheduler decides the execution of *A* before *B* in order to reduce the span between the executions of dependent tasks expressed by the dependence between *C*, *D*, and *E*. Hence, the scheduler promotes the solution that starts *C* immediately after completion of *A* and begins the execution of *E* once its predecessors *C* and *D* complete execution as reinforced by the optimization parameters explained at the end of mapping/scheduling resolution. These parameters aim at bringing closer the execution start of each task to the completion of its predecessors. *C* and *E* can execute on RZ_1 and RZ_2. The mapping assigns their executions to RZ_2 as this RZ provides the best utilization of costly resources guided by their costs *D*. For the purpose of fully exploiting the multi-reconfigurable-unit system and to increase parallel efficiency, task *A* resumes its execution simultaneously with the start time of the first repetition of task *D* at 57503 μs. In their second execution iterations, respecting the periodicity and precedence constraints tasks *C*, *D*, and *E* are performed on their optimal RZ, that is, RZ_2, to optimize resource utilization.

The resulting scheduling respects all the predefined constraints and during task running generates a small waiting time, equal to 42733 μs for task *A* when it is preempted and 14770 μs for task *B* in the ready queue. This waiting time represents only 16% of the overall running time. For the other tasks, the scheduler response is immediate, and the tasks are performed without preemption or migrations that substantially decrease the configuration overhead estimated in the following stage of resolution. The short response time of the scheduler is also reinforced by parallel computation.

Thanks to parallel computation and the optimization parameters, especially the launching of tasks whenever the start times of their repetitions are valid, the makespan is optimal and equal to 353394 μs. Compared to sequential execution of the DAG, the achieved speedup is 1.03.

This speedup refers to the amount by which the pipelined scheduling speeds up the sequential execution of the DAG. Consequently, the parallel efficiency of this scheduling indicates how efficiently the RZs are exploited to perform DAG execution and is obtained by dividing the achieved speedup by the number of used RZs in the multi-reconfigurable-unit system. In our resolution, the parallel efficiency is about 0.5 which is considered acceptable as tasks are heterogeneous and are not allowed to be executed in all the RZs. The resolution of this first sub-problem is conducted on a CPU of 2 GHz with 2 GB of RAM and lasts 6280 seconds.

4.3. Partitioning/Fitting RZs Results and Placement Quality Evaluation. Based on powerful solvers, RZ_1 and RZ_2 are fitted on their most suitable RPBs defined by the following coordinates after 80.63 seconds on Virtex 5 FX70T (8×47 RBs):

(i) RPB_1 for RZ_1: $X_1 = 14$, $Y_1 = 6$, $WRPB_1 = 32$, $HRPB_1 = 6$,

(ii) RPB_2 for RZ_2: $X_2 = 14$, $Y_2 = 3$, $WRPB_2 = 33$, $HRPB_2 = 3$.

Table 4 shows the comparison between the RBs of the obtained RPBs and their RZs expressed in the last column by cost Δ. The not null differences in RBs (Δ) are due to the rectangular shape of the RPBs and the heterogeneity of the device. Δ gives RB_3 in excess. In fact, both RPBs include RB_4, and in Virtex 5 FX70T there is no way to book RBs in a given RPB requiring DSP resources (RB_4) with CLBL and CLBM resources (RB_1 and RB_2) without crossing BRAM (RB_3) columns. Costs Δ also demonstrate how much the resulting placement of required RZs ensures resource efficiency. The low values found in Δ (1 RB_3 and 2 RB_3) show that the obtained RPBs are very close to their corresponding RZs, and consequently, resource efficiency is maintained during the conducted resolution.

Figure 11 depicts the floorplanning of RPB_1 and RPB_2 on Virtex 5 FX70T as obtained by their coordinates. Figure 11 also represents the placing/routing of tasks *A* and *D* named,

Table 4: Resource efficiency.

	RB_1	RB_2	RB_3	RB_4	Δ
RPB_1	8	7	2	1	1 RB_3
RPB_2	9	7	2	1	2 RB_3

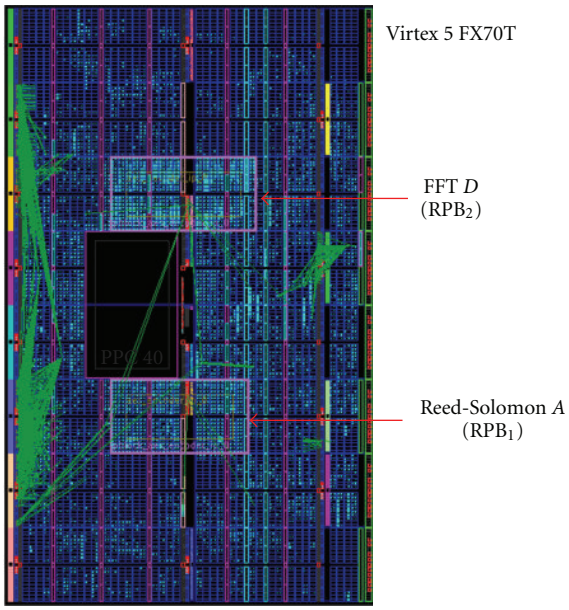


Figure 11: Floorplanning of RPBs and placing/routing of tasks *A* and *D* on Virtex 5 FX70T.

respectively, Reed-Solomon and FFT, which are running, respectively, on RZ_1 and RZ_2 during DAG execution in the time interval [57503, 69309]. The obtained results show an average resource utilization of 12.46% of the available resources on the reconfigurable device. This average is computed according to the number and the cost of each RB type. Optimization in the utilization of resources minimizes the area of the FPGA which is reconfigured at runtime. Due to dynamic partial reconfiguration, resource efficiency is improved by 17.3% compared to the static design of the chosen DAG. The static design is created by floorplanning each task in the DAG on its unique corresponding RPB without sharing any RPBs between different tasks. Once the RPBs are allocated on the reconfigurable device, their bitstreams are created, and their real configuration overheads are computed. The incurred reconfiguration overhead obtained after mapping/scheduling is 6729 μs and represents only 2% of the total running time of the DAG.

Although the experimental conditions are not the same in terms of DAG size and used architecture, we compared our results to attainable improvements in previous works of multiprocessor scheduling. Speedup in our resolution is modest and is about 1.03, compared to [6] which achieves 5.01 for FIR implementation and [3], which reaches a speedup of up to 2. Similarly, regarding parallel efficiency on processors, the results of [6] show a parallel efficiency of 1.3, [3] improves this parameter to 1, and the highest parallelism system is obtained in the work described in [10] which produces 9.3 degrees of parallelism. Nevertheless, our methodology ensures a parallel efficiency of 0.5. The modest improvement in speedup and parallelism results obtained with our methodology can be explained by the heterogeneity of the tasks, the non suitability of all the provided RZs to execute the tasks, and the small number of provided RZs in order to increase resource efficiency in reconfigurable devices. The highest improvement in speedup and parallel efficiency reached in previous works of multiprocessor scheduling does not take into account the resource efficiency and the processor heterogeneity. Effectively, the processors are considered homogeneous as they have identical features, and tasks are allowed to be executed on whatever processor whenever is idle.

However, compared to [18] where 80% of available resources are utilized, our methodology increases resource efficiency in heterogeneous devices by up to 17% compared to a static design and optimizes reconfigurable resource utilization by up to 12.46%. In contrast to [2] that targets an application of two or three tasks and attains a configuration overhead of 8% of total execution time, we immensely reduced the configuration overhead to 2% of the running time for a given DAG of five tasks. Similarly, in [24], Resano et al. attain 18% of configuration overhead to schedule only a JPEG decoder.

Concerning computational complexity, compared to the proposed heuristics to perform multiprocessor scheduling, the worst case temporal complexity is $O(n * (n + e))$, where n is the number of nodes, and e is the number of edges in the graph. We are aware that the efficiency and the optimality of our proposed methodology may be impaired by its temporal complexity in finding the optimal solution and which grows exponentially with the number of tasks in the DAG. It is estimated by: $O(2^{\mathrm{NT}*\mathrm{NZ}*NbrPreemp*\mathrm{HP}*NbrIter}) + O((Device_{width^2} \times Device_{height^2})^{\mathrm{NZ}})$, where *NbrPreemp* and *NbrIter* are, respectively, the maximum number of preemption points and the maximum number of execution iterations for a given task.

Thanks to the pertinent results obtained by the studied DAG, the efficiency of our proposed methodology in performing the placement and scheduling of small DAGs with few tasks is reinforced. However, due to the large size of search space containing the candidate solutions which will be checked for admission by constraints and evaluated for optimality by the objective function, our proposed approach is not capable to deal efficiently with intensive paralleled applications with hundreds of repetitive tasks. Effectively, using our approach in this class of application burdens the resolution time and the memory space, required for the computing of the several constraints and the different objective functions, and for searching the high number of assignments of possible values to the variables, parameters, and indexes. Currently, in our research project FOSFOR (Flexible Operating System FOr Reconfigurable platforms), the proposed methodology is efficiently employed as the target application is of type dataflow and contains small number of tasks.

5. Conclusion and Future Work

Under strict real-time constraints and from a parallel processing perspective, our paper deals with the problem of static scheduling of DAGs on multi-reconfigurable-unit system. In our opinion, most of the works proposed in this field do not consider resource efficiency or configuration overhead, and they are not applied to new heterogeneous technology. The approaches focus only on improving computation speedup and parallel efficiency for DAGs on homogeneous execution units. By means of a new methodology comprising three main stages and by selecting a preemptive model, we take into account periodicity, precedence, dependence, and real-time constraints, and we employ rigorous efficient analytic resolution in order to enhance quality of placement and scheduling in the most recent heterogeneous reconfigurable devices. Our methodology is illustrated in a realistic application, and the results obtained are encouraging. The resolution tries to find the trade-off between all the cited criteria since the performance of the DAG on reconfigurable devices is mainly defined by the degree of parallelism, the resource efficiency, and the amount of incurred reconfiguration. Our proposed approach is largely dependent on the physical features of tasks and technology as well as on temporal characteristics. During our tests, we concluded that our proposed methodology is efficient for small DAGs rather than for larger ones. In fact, solver processing is immensely delayed when the number of tasks in DAG exceeds five.

Static multiprocessor scheduling is a well-understood problem, and many efficient heuristics have been proposed to create compile-time scheduler scenarios. However, the approaches face difficulties in dealing with nondeterministic systems with run-time characteristics that are not well known before the DAG running. Thus, our future challenge is to define dynamic scheduling in heterogeneous reconfigurable devices to be applied for several DAGs of different sizes with nondeterministic behavior. We aim to consider intertask communication, all the specified constraints detailed throughout this paper, as well as to optimize all the cited criteria, especially degree of parallelism, resource efficiency, and configuration overheads.

Acknowledgments

The authors would like to thank the National Research Agency in France and the world-ranking "Secured Communicating Solutions" (SCS) cluster that sponsors our research project FOSFOR (flexible operating system for reconfigurable platforms). This work was also supported by AIMMS technical support and Xilinx tools.

References

[1] G. L. J. Djordjević and M. B. Tošić, "A heuristic for scheduling task graphs with communication delays onto multiprocessors," *Elsevier Parallel Computing*, vol. 22, no. 9, pp. 1197–1214, 1996.

[2] J. A. Clemente, C. Gonzalez, J. Resano, and D. Mozos, "A hardware task-graph scheduler for reconfigurable multitasking systems," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, pp. 79–84, Cancun, Mexico, December 2008.

[3] F. E. Sandnes and G. M. Megson, "Improved static multiprocessor scheduling using cyclic task graphs: a genetic approach," in *Parallel Computing: Fundamentals, Applications and New Directions*, vol. 12, pp. 703–710, North-Holland, 1998.

[4] Y. Abdeddaim, A. Kerbaa, and O. Maler, "Task graph scheduling using timed automata," in *Proceedings of the 17th IEEE International Symposium on Parallel and Distributed Processing (IPDPS '03)*, p. 237, Nice, France, April 2003.

[5] F. Redaelli, M. D. Santambrogio, and S. O. Memik, "An ILP formulation for the task graph scheduling problem tailored to bi-dimensional reconfigurable architectures," *International Journal for Reconfigurable Computing*, pp. 97–102, 2008.

[6] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan, "An ILP formulation for task mapping and scheduling on multi-core architectures," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '09)*, pp. 33–38, Nice, France, April 2009.

[7] F. E. Sandnes and O. Sinnen, "A new strategy for multiprocessor scheduling of cyclic task graphs," *International Journal High Performance Computing and Networking*, vol. 3, no. 1, pp. 62–71, 2005.

[8] M. Huang, H. Simmler, P. Saha, and T. El-Ghazawi, "Hardware task scheduling optimizations for reconfigurable computing," in *Proceedings of the 2nd International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA '08)*, pp. 1–10, Austin, Tex, USA, November 2008.

[9] S. Fekete, E. Kohler, P. Saha, and J. Teich, "Optimal FPGA module placement with temporal precedence constraints," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '01)*, pp. 658–665, Munich, Germany, March 2001.

[10] Z. Pan and B. E. Wells, "Hardware supported task scheduling on dynamically reconfigurable SoC architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, Article ID 4633696, pp. 1465–1474, 2008.

[11] W. H. Kohler, "A preliminary evaluation of the critical path method for scheduling tasks on multiprocessor systems," *IEEE Transactions on Computers*, vol. 24, no. 12, pp. 1235–1238, 1975.

[12] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of hardware tasks on FPGA," in *Proceedings of the 19th International Conference on Field Programmable Logic and Application (FPL '09)*, pp. 591–595, Prague, Czech Republic, September 2009.

[13] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[14] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 178, Nice, France, April 2003.

[15] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement," in *Proceedings of the Design Automation Conference (DAC '04)*, pp. 960–965, San Diego, Calif, USA, June 2004.

[16] T. Marconi, Y. Lu, K. Bertels, and G. Gaydadjiev, "Intelligent merging online task placement algorithm for partial reconfigurable systems," in *Proceedings of the Design Automation Test Europe Conference (DATE '08)*, pp. 1346–1351, Munich, Germany, March 2008.

[17] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '04)*, p. 134, Santa Fe, New Mexico, April 2004.

[18] H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, "Task rearrangement on partially reconfigurable FPGAs with restricted buffer," in *Proceedings of the International Conference on Field Programmable Logic and Application*, pp. 379–388, Villach, Austria, August 2000.

[19] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: a survey," *European Journal of Operational Research*, vol. 141, pp. 241–252, 2001.

[20] A. Lodi, S. Martello, and D. Vigo, "Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 125–139, Kluwer Academic Publishers, 1997.

[21] M. Panainte, K. Bertels, and S. Vassiliadis, "FPGA-area allocation for partial run-time reconfiguration," in *Proceedings of the Design Automation Test Europe Conference (DATE '05)*, pp. 100–105, Munich, Germany, March 2005.

[22] http://www.aimms.com/.

[23] "Virtex-5 FPGA Configuration User Guide," Xilinx white paper, August 2009.

[24] J. Resano, D. Mozos, D. Verkest, S. Vernalde, and F. Catthoor, "Run-time minimization of reconfiguration overhead in dynamically reconfigurable systems," in *Proceedings of the International Conference on Field Programmable Logic and Application*, pp. 585–594, Lisbon, Portugal, September 2003.

Reconfiguration Time Overhead on FPGA: Reduction and Cost Model

François Duhem, Fabrice Muller, Philippe Lorenzini
University of Nice-Sophia Antipolis - LEAT/CNRS
Bât.4, 250 rue Albert Einstein, 06560 Valbonne, France
e-mail: {Francois.Duhem, Fabrice.Muller, Philippe.Lorenzini}@unice.fr

Abstract

Partial reconfiguration suffers from low performance and thus its use is limited when the reconfiguration overhead is too high compared to the task execution time. To overcome this issue, we present a fast ICAP controller, FaRM, providing high-speed configuration and easy-to-use readback capabilities, reducing configuration overhead as much as possible. In order to enhance performance, FaRM uses techniques such as master accesses, ICAP overclocking, bitstream pre-load into a controller and our bitstream compression technique, Offset-RLE, which is an improvement of the Run Length Encoding algorithm. Combining these approaches allows us to achieve an ICAP theoretical throughput of 800 MB/S at 200 MHz. In order to complete our approach, we provide a cost model for the reconfiguration overhead for the system level that can be used during the early stages of development. We tested our approach on an AES encryption/decryption architecture.

1 INTRODUCTION

Partial Reconfiguration (PR) has been introduced in the late 90's to answer the need for increasing resources associated with Field Programmable Gate Arrays (FPGAs). Indeed, it allows for changes in the behaviour of reconfigurable partitions in the architecture while the remaining logic is still running. Partial reconfiguration can therefore be used to change the functionality of a system or to save FPGA resources and power [1], [2]. The first FPGAs to support this feature were the XC6200 series, programmed using the Java-based interface JBITS, introducing some concepts that are still used today [3].

Recent Xilinx FPGAs provide this feature via the Internal Configuration Access Port (ICAP) [4]. However, Xilinx's controller, *xps_hwicap* [5], shows low performance. In such case, PR adds non-neglictible time overhead which is seen as one of the big challenges of this technology. For instance, if the system performs scheduling on hardware tasks, the reconfiguration time to task execution time ratio might become unacceptable, thus dispensing developers with PR [6], [7].

It is possible to overcome this performance issue in many ways. First, by working on the configuration data, called bitstreams. These binary files are transferred from a memory to the ICAP macro.

Reducing bitstream size by using compression techniques therefore reduces reconfiguration overhead. Bitstream compression also lowers the memory requirements for the system. However, the system also needs to decompress the file, so resource overhead for the decompression part is a significant metric. Another way to improve performance is to work on the ICAP controller architecture, e.g. by integrating a Direct Memory Access (DMA), in order to directly increase the write throughput of the configuration interface [8]. Scheduling may also predict the future tasks to execute on a reconfigurable zone (RZ) [9], [10], [11]. Thus, pre-loading the corresponding bitstream into memory may significantly reduce reconfiguration overhead.

We propose to combine these approaches in our controller, FaRM (Fast Reconfiguration Manager). We present an architecture providing performance up to the ICAP limit. In order to reduce transfer time, our controller has an integrated smart master/slave interface and is able to handle bitstream compression and pre-load. In addition, we want FaRM to provide easy-to-use yet efficient readback capability (i.e. retrieving configuration data from the FPGA).

Moreover, in order to estimate the possible speedup introduced with partial reconfiguration using FaRM at the early stages of the development, we propose accurate cost models of the reconfiguration time overhead that may be used at system level.

This paper is structured as follows. In Section 2, we discuss works related to the acceleration of partial reconfiguration. Section 3 introduces our approach to speed up reconfiguration from the architecture to the bitstream optimization and Section 4 presents the associated cost model. Section 5 sums up our results while the final section presents some improvements we want to implement in our controller.

2 Related Works

Our approach focuses on optimizing time overhead by working on the effective configuration of the FPGA, i.e. when transmitting bitstreams to the ICAP macro. Therefore, we do not consider task placement in this paper which can also participate in improving reconfiguration overhead by choosing an optimal reconfigurable zone for a task [12].

As mentioned previously, compression is key for improving performance. In paper [13], the authors discuss different bitstream compression techniques. They compare their own compression algorithms, based on Run Length Encoding (RLE) and LZW, with common algorithms such as RLE, ZIP or GZIP applied to different bitstreams. Although the authors provide compressed bitstream sizes, they do not give their initial size or the achieved compression ratio for each algorithm. It appears that RLE results in an extremely poor compression ratio compared to ZIP algorithms and that its custom algorithm gives ratios slightly lower than ZIP algorithms. Note that these bitstreams program a virtual FPGA layer so that these results may not be representative for physical FPGAs bitstreams.

The work presented in [14] tends to be more exhaustive in terms of compression techniques, not only considering the compression ratio of each algorithm but also key metrics such as throughput and resource overhead. Compression reduces the transfers on the bus, thus saving costly memory

accesses, but the decompression part embedded in the FPGA also has to send data to the ICAP efficiently to achieve good reconfiguration time.

In [15], the authors present a bitstreams repository hierarchy for partially reconfigurable systems reducing the need for local memory. Considering only the first level of the hierarchy, i.e. when the bitstreams are stored locally, they introduce an architecture based on a DMA writing bitstreams to the ICAP through an OPB bus. A similar approach is presented in [8]. In this work, the authors compare several architectures, from Xilinx controllers to their high-speed controller, using BRAM as a local cache memory. Even if they are close to theoretical maximum throughput, the IP requires a large amount of BRAM (nearly half the BRAM resource of a Virtex-4 FX20 FPGA).

The authors in [16] use virtual configurations to decrease reconfiguration time. The method consists in implementing two configuration contexts: one running in the foreground and the other one in the background. PR may influence the background context while the foreground context is still running, thus reducing reconfiguration overhead. The major drawback of this method appears when working with a mono-context FPGA: there have to be as many RZs as configuration contexts, resulting in significant size overhead.

To the best of our knowledge, there is a lack of resources for the estimation of the reconfiguration time overhead. The work presented in [17] offers a cost model for reconfiguration overhead. The authors propose a model for reconfiguration using a microprocessor as the configuration controller. The key point of this approach is that it not only considers the actual ICAP configuration time (which is given by Xilinx considering only the configuration interface throughput [18]) but also transfer times from the bitstream location in the memory (Compact Flash card or DDR memory) into the processor cache and then into the ICAP controller cache. Nevertheless, they obtain a difference between their estimation and the actual value that varies between 30 and 60%. Moreover, this approach cannot be applied in our case since we do not use a microprocessor to transfer data.

3 OUR APPROACH

3.1 FaRM architecture

FaRM is based on the architecture depicted in Fig. 1. It is built upon a smart interface, with both master and slave interfaces that can be plugged to a wrapper (in our case, we use a PLB wrapper), whereas Xilinx's IP only has a slave interface. The slave interface deals with the IP registers while the master interface is responsible for accesses to the memory where the bitstreams and readback contents should be stored. It handles the different operating modes described later. The master interface allows the IP to be independent from the local microprocessor and act like a DMA, since the microprocessor only works on registers to control the scheduling of hardware tasks and does not perform data transfers like Xilinx's IP does. The two interfaces may be connected on two different buses, separating fast accesses needed by read/write accesses on the bus and slower register accesses for configuration purposes. Moreover, it should be noted that FaRM is completely independent from the targeted application: it only needs a light microprocessor (e.g. MicroBlaze) or an hardware IP to control FaRM and an access

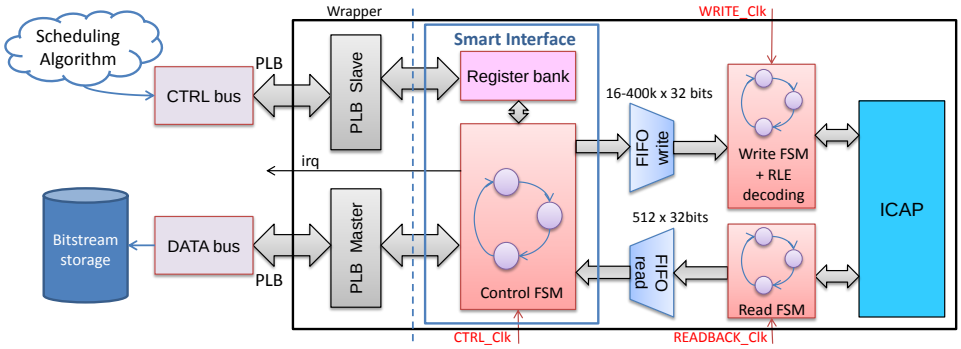


Fig. 1. FaRM architecture

to the bitstream repository (in our case, on-board DDR2). The application may be using a different PLB bus (or another type of bus) and has no constraint inherited from the configuration subsystem.

The ICAP macro is driven by two FSMs and interfaced with two FIFOs, respectively for read and write accesses, to separate clock domains. The write FIFO can hold from 16 to 400k 32-bit words, depending on bitstream sizes and performance requirements discussed later. Like Xilinx's IP, the ICAP and the bus are on different clock domains in order to deal with the ICAP clock limit of 100 MHz without limiting bus frequency. However, we were able to go beyond this limitation for the ICAP clock. We also chose to differentiate the clocks for read or write access to the ICAP because we achieved much higher frequencies with the write access compared to the read access, as explained in Section 5.

3.2 Compression technique

3.2.1 About bitstreams

Bitstreams are binary files used to configure the FPGA. In the case of endo-configuration (i.e. internally configuring the FPGA, which is the case of PR), bitstreams have to be transferred from a bitstream repository (in our case, DDR memory) to the ICAP hard macro. They are composed of a command part initializing the ICAP, and of a data part used for actual configuration of the FPGA (e.g. configuring Look-Up Tables and interconnections) [4]. After analyzing some bitstreams, we noticed that there were some redundancy in the data part that could easily be compressed, particularly when configuring BRAM contents, which are often zero-initialized. Therefore, we chose to use a RLE-based compression technique that seems to be a good compromise between efficiency and complexity.

3.2.2 The RLE principle

Run Length Encoding is a lossless data compression algorithm in which sequences of identical data are coded as the data combined with the number of times it is being repeated. The algorithm is therefore perfectly suitable for redundant data (e.g. memory initialization). As the RLE principle is relatively simple, it results in efficient hardware decompression, thereby meeting requirements.

Let us consider the data sequence presented in Fig. 3. The word 0x87654321 appears five times in a row in the original bitstream. After compression, it will appear twice to indicate that we are starting

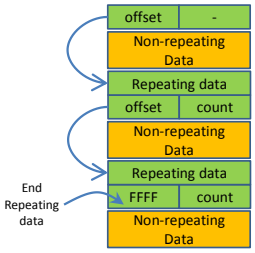


Fig. 2. O-RLE principle

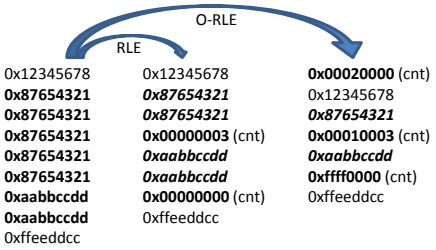


Fig. 3. Canonical RLE vs. O-RLE

a sequence of redundant data. The next word is a counter representing the number of times the word is repeated again (three times here). A clear drawback of this method is that the repeating word still appears twice in the compressed bitstream. In our example, the word 0xaabbccdd repeats twice but will be coded in three words within the compressed bitstream: twice the data and a counter of 0.

3.2.3 Offset RLE (O-RLE)

As an improvement, we tried to remove the extra word in the compressed bitstream. Our approach is illustrated in Fig. 2. In order to indicate repeating data, we use a pointer mechanism. At the beginning of the compressed bitstream, we insert a word containing the offset to the next repeating data. It is no longer useful to repeat the data twice like canonical RLE does, since the offset already gives us this information. Just after the data to repeat, we insert a word containing two pieces of information: the number of times the data is repeated and the offset to the next repeating word. If the data is the last repeating data of the bitstream, we use the reserved word 0xffff as an offset. Note that the counter and the offset are coded on two bytes each. This is possible because we work on 32-bit words, thus the counter and the offset are coded on 16 bits. In our experience, this is a good compromise since the maximum value of 65536 is not likely to be achieved for the bitstreams we are working on. If we need to code more than this step value, we just start another sequence. Figure 3 shows an example of the O-RLE coding scheme.

3.3 Operating modes

3.3.1 Pre-load mode

This mode allows the user to pre-load a bitstream or at least the beginning depending on the size of the write FIFO. Using this fast memory near the ICAP controller allows us to approach the ICAP theoretical throughput. The FIFO size does not need to be greater than the bitstream size (most of the time, it may not be relevant). In fact, using compression and pre-load mode significantly reduces the memory needs of the system for the same performance. Figure 4 shows a timing sequence for write mode with and without pre-load. We can see that with pre-load, the configuration overhead for the second IP is reduced since bitstream transfer begins while the first IP is running.

This mode is very useful if scheduling can predict whether reconfiguration is required. If so, the reconfiguration overhead might be reduced to its lower bound if all the data are present in the FIFO

so that only the ICAP write part of the transaction remains.

3.3.2 Auto-readback

The major feature of FaRM is auto-readback. It provides an easy yet efficient way to read the configuration data from any part of the FPGA. This feature can be used for context saving purpose in two ways. First, by reading back the entire partition on which the task is executed. This solution is very expensive in terms of time, but it may be used to relocate the task on another partition, with respect to some conditions as mentioned in [19]. The alternative is to directly read back registers from CLB columns containing relevant information, a much more efficient solution if no relocation is needed.

The readback is composed of three parts. First, a command part written to the ICAP which prepares it for the readback (transfer size, frame addressing). This is step 1 in Fig. 4. Then, there is a data part where the ICAP fills the read FIFO and the control FSM transfers it onto the bus (step 2). Finally, there is a last command part written to the ICAP which ends the transaction (step 3).

With the auto-readback feature, the entire readback process is automated. It is carried out in three steps: first, preparation of the readback. Both command parts are written into the memory. Then, the IP needs to be configured. Three registers contain the addresses for each part of the readback (write header, readback bitstream and write header end). The auto-readback is then started. Finally, the microprocessor waits for the end of the process, either in polling or interrupt mode.

Both read and write accesses use burst transfers on the bus to improve performance. Burst lengths range from 2 to 16. It should be noted that our controller may handle any readback bitstream size, in contrast to Xilinx's IP which can only readback one frame at a time (a frame being the smallest amount of data that can be configured), resulting in a certain amount of control overhead. Moreover, for each read operation, independently of its size, an extra padding frame and an extra word has to be read from the ICAP macro. Since it does not contain any relevant data, we choose not to transmit this data on the bus. While this overhead can be neglected for large readbacks, it doubles the data read for a frame-by-frame readback.

Finally, FaRM instantiates the readback capture macro [4]. In addition to reading all configuration memory cells, this macro reads the current state of all CLB and IOB registers, allowing the capture of the exact state of a reconfigurable partition.

3.3.3 CRC32 calculation

At the end of each bitstream generated by PlanAhead, there is a CRC (Cyclic Redundancy Check) calculated to check the integrity of the bitstream during the FPGA programming. Therefore, if one wants to modify the bitstream manually, the CRC has to be re-calculated and modified in the bitstream too. For the present, we can calculate the CRC for Virtex-5 partial bitstreams in software, which is useful when using bitstream relocation. The technique consists in modifying an existing bitstream so that it may be used to configure another RZ. In this case, a new CRC32 needs to be calculated to pass the check carried out by the ICAP.

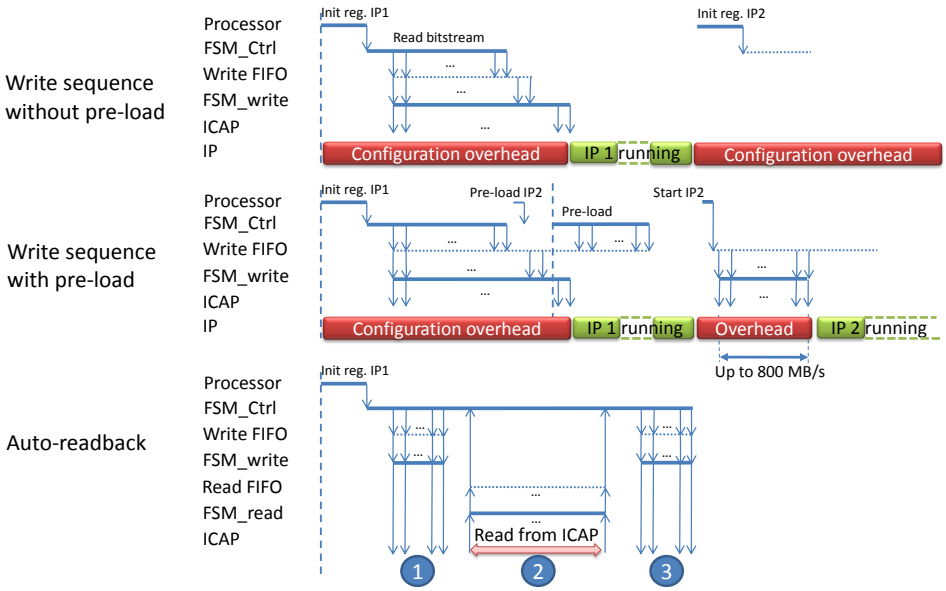


Fig. 4. Operating modes

4 COST MODEL

Reconfiguration overhead was estimated for a canonical write operation. Using Chipscope, it is possible to monitor some of our controller's signals, including the PLB interface. In our experience, the ICAP does not limit the transfer: once the access is finished on the bus, the ICAP can process all the words before the next burst, so that the overhead only reflects the bitstream bus transfers. The reconfiguration overhead is estimated by separating full burst transfers and remainder transfer that uses single accesses. Let $N_{bitstream}$ be the bitstream length in 32-bit words, T_{bus} the bus period in ns, $latency$ the initial latency in cycles, t_{burst} the average time for a burst in cycles and N_{burst} the number of words in a burst. Equation (1) gives an estimation of the reconfiguration overhead for an uncompressed bitstream.

$$t_{\text{write}}\,(\text{ns}) = (latency + t_{\text{burst}} * \lfloor \frac{N_{\text{bitstream}}}{N_{\text{burst}}} \rfloor + (t_{\text{burst}} - N_{\text{bitstream}} \bmod N_{\text{burst}})) * T_{\text{bus}} \quad (1)$$

In the case of a compressed bitstream, reconfiguration time cannot be estimated properly because it is highly dependent on compression regularity. If compression is uniform and equally distributed on the bitstream, then (1) can be used to compute time overhead. Introducing compression into (1) gives us the lower bound. In the worst case scenario, compressed data are located at the end of the bitstream so that there is an extra ICAP write overhead after all burst transfers are done. Let T_{icap} be the ICAP period in ns and $ratio$ the achieved compression rate (1 if using uncompressed bitstreams). Equations (2) and (3) show the lower bound t_{min} and the upper bound t_{max} of the reconfiguration overhead for a compressed bitstream.

$$t_{\text{min}}^{compressed}\,(\text{ns}) = (latency + t_{\text{burst}} * \lfloor \frac{ratio * N_{\text{bitstream}}}{N_{\text{burst}}} \rfloor + (t_{\text{burst}} - (ratio * N_{\text{bitstream}}) \bmod N_{\text{burst}})) * T_{\text{bus}} \quad (2)$$

$$t_{\text{max}}^{compressed}\,(\text{ns}) = t_{\text{min}}^{compressed} + N_{\text{bitstream}} * (1 - ratio) * T_{\text{icap}} \quad (3)$$

When using pre-load mode, it is more complicated to properly estimate reconfiguration time, mostly because the FIFO can be filled during an ICAP write operation. The operation can be divided into two phases. During phase 1, the FIFO, previously filled during pre-load, empties itself: the ICAP throughput reaches its upper bound. The number of words written and the time spent at maximum throughput fully depends on compression rate and FIFO size. Algorithm 1 provides an estimation of this information. It consists of several iterations during which words previously put into the write FIFO are written to the ICAP. During this time, the FIFO can be filled by the control FSM. We iterate until the FIFO is empty, which ends the first phase of the transfer. The second phase consists in writing the remaining words into the ICAP. It is done in the same fashion as if there were no pre-load.

Algorithm 1 Number of words written and time spent in Pre-load phase 1

1: **if** $N_{bitstream} * ratio < fifo\_depth$ **then**
2: $N_{word} \leftarrow N_{bitstream}$
3: **else**
4: $N_{word} \leftarrow fifo\_depth/ratio$
5: **end if**
6: **repeat**
7: $t_{iter} \leftarrow T_{icap} * N_{word}$
8: $N_{wordNext} \leftarrow ((t_{iter}/(t_{burst} * T_{bus})) * N_{burst})/ratio$
9: $t_{total} \leftarrow t_{total} + t_{iter}$
10: $N_{wordTotal} \leftarrow N_{wordTotal} + N_{word}$
11: **if** $N_{wordTotal} + N_{wordNext} > N_{bitstream}$ **then**
12: $sat \leftarrow 1$
13: $N_{wordNext} \leftarrow N_{bitstream} - N_{wordTotal}$
14: **end if**
15: $N_{word} \leftarrow N_{wordNext}$
16: **until** $N_{word} > 1$ **and** $\lceil N_{wordTotal} \rceil \neq N_{bitstream}$ **and** $(N_{wordNext} > N_{burst}$ **or** $sat = 1)$
17: $N_{wordLeft} \leftarrow N_{bitstream} - N_{wordTotal}$

Let us see in detail how Algorithm 1 operates. First of all, the algorithm calculates the number of words in the FIFO based on its depth and the compression ratio. We assume that the FIFO is full when the user starts the reconfiguration. The algorithm then enters the main loop, calculating the time needed to write all the words currently in the FIFO (t_{iter}). The time is used to calculate the number of words that can be written into the FIFO during this iteration $N_{wordNext}$, taking into account burst accesses and compression (line 8). This amount of data can be corrected if the bitstream length is exceeded (line 13). The loop ends when there are no more words to write to the FIFO. The algorithm shows the time spent during this phase t_{total} and in the number of words to be written during the second phase $N_{wordLeft}$.

To estimate the reconfiguration overhead with pre-load, we combine Algorithm 1 with (2) and (3). Without compression, Algorithm 1 gives us the time spent to write pre-loaded data t_{load} and the number of words remaining $N_{wordLeft}$, used as an input in (2). Equation (4) gives the reconfiguration overhead without compression.

$$t_{\text{preload}}^{uncompressed}\,(\text{ns}) = t_{\text{load}} + (latency + t_{\text{burst}} * \lfloor \frac{N_{\text{wordLeft}}}{N_{\text{burst}}} \rfloor + (t_{\text{burst}} - N_{\text{wordLeft}} \bmod N_{\text{burst}})) * T_{\text{bus}} \quad (4)$$

When using compression, the estimation cannot be carried out in the same way. In fact, we cannot pretend that the compression rate is constant throughout the bitstream since it leads to a really poor overhead estimation. In order to refine our estimation, we introduce a different compression rate for both phases of the pre-load. Let $t_{load}^{compressed}$ be the time spent at maximum throughput in pre-load mode, found with Algorithm 1 with a compression rate $ratio_1$, and let $ratio_2$ be the compression rate for the second part of the transfer. Equations (5) and (6) give an estimation of the overhead lower and upper bounds when using pre-load and compression.

$$t_{\text{preload,min}}^{compressed}\,(\text{ns}) = t_{\text{load}}^{compressed} + (latency + t_{\text{burst}} * \lfloor \frac{ratio_2 * N_{\text{wordLeft}}}{N_{\text{burst}}} \rfloor + (t_{\text{burst}} - (ratio_2 * N_{\text{wordLeft}}) \bmod N_{\text{burst}})) * T_{\text{bus}} \quad (5)$$

$$t_{\text{preload,max}}^{compressed}\,(\text{ns}) = t_{\text{preload,min}}^{compressed} + N_{\text{wordLeft}} * (1 - ratio_2) * T_{\text{icap}} \quad (6)$$

The inputs of (5) and (6), $ratio_1$ and $ratio_2$, are unknown at first since their computation requires the number of words written during phase 1 of the pre-load, which depends on $ratio_1$. We therefore have to define an arbitrary initial value for $ratio_1$ and $ratio_2$: they are set to the global compression rate. We then iterate Algorithm 1 and the computation of the partial compression rates, converging to the actual value. We continue to iterate until a stable value is obtained.

5 FaRM COMPARISON

5.1 FaRM versus xps_hwicap

Table 1 compares FaRM to Xilinx's solution in terms of functionalities and area usage on a Virtex-5 LX70T FPGA. It also presents the area required to implement O-RLE decompression on this FPGA. Note that Xilinx's IP in its version 1.00.a does not allow FIFO depth to be changed: write FIFO is 1024 words deep and read FIFO is 128 words deep whereas with FaRM, both read and write FIFOs are 512 words deep. We can see that FaRM does not require more resources than Xilinx's solution, but provides significant improvements as we will see in Sect. 6.

5.2 Bitstream compression

Table 2 compares different bitstream compression algorithms. We tested each algorithm on heterogeneous IPs in terms of size, complexity and resources (CLB, BRAM, DSP48).

With an average compression rate of 26.7% (using geometric mean to smooth the result), the FaRM compression algorithm is slightly more efficient than the 16- and 32-bit versions of canonical

TABLE 1

FaRM vs. xps_hwicap

	xps_hwicap		FaRM	
Write	Poor performance		Maximum throughput	
Readback	Frame-by-frame only		Any length, auto-readback	
Readback capture	By register		Automatic capture	
Bitstream compression	No		Yes	
Bitstream pre-loading	Using slave interface		Automatic	
Interfaces	Slave only		Master/Slave	
DMA	No		Yes	
Resources	Slave only	Slave + DMA	Full	O-RLE only
- Slice registers	427	991	1130	243
- Slice LUTs	1748	2455	1342	357
- LUT Flip Flop	1216	2072	1786	390
- BRAM	-	-	2	-

TABLE 2

Compression ratios in % for different algorithms and bitstreams

IP	Size (byte)	FaRM	RLE 16	RLE 32	ZIP	7z
AES decoder	97676	23.4	21.6	20.9	57	60.8
AES encoder	97676	31	31.6	26.6	67.6	69.7
AES Black Box	97676	90.8	93	88.1	97.3	97.5
Basic DES	24348	7	4.4	4.1	48.3	50.9
Basic RSA	68156	36.9	35.4	35.8	61.8	65.4
DCT	66844	41.1	40.5	38.8	65.3	67.9
FFT64	161308	32.9	31.3	32.2	57.3	60.8
FIR_7_16_8	17132	11.7	7.6	7.3	64.3	65
FIR_7_16_16	23036	12.8	10.3	7.5	65.3	67.1
Hibert	24348	19.9	18.7	14.8	65.4	66.7
Count 1	6632	51.2	48.9	44.8	78.3	78.5
Count 2	6632	51	48.8	44.6	78.2	78.3
Geometric mean		**26.7**	**23.9**	**21.5**	**66.1**	**68.1**

RLE. Although we cannot compete with state-of-the-art algorithms such as ZIP and 7z, our solution does not require a great deal of resources for decompression hardware implementation and has a throughput of one word per cycle.

6 APPLICATION

FaRM was tested on a ML507 board with both a AES encryption/decryption and FFT64 system, as depicted in Fig. 5. It contains one RZ hosting either the encoder or the decoder and one RZ for FFT64. The RZs are connected to the PLB bus through a wrapper. The system is driven by a MicroBlaze soft

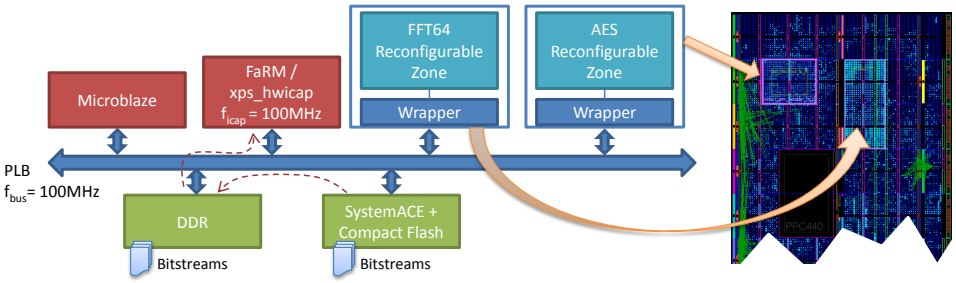


Fig. 5. AES test system

processor. The bitstreams are stored in the Compact Flash and then copied by the MicroBlaze into the DDR2 memory.

The AES partition has the following resources: 12 CLB columns and one BRAM column, representing a total of 590 frames. It has been designed to be as close as possible to the application needs. Each frame consists of 41x32-bit words. However, correct readback from BRAM columns is currently impossible so for readback operations, we worked on a subset of the partition comprising 462 frames. Note that the AES decoder needs a little more resources than the AES encoder so that there are some "wasted" resources when the decoder is loaded into the partition. However, FaRM achieves better compression with this bitstream which compensates area inefficiency. The FFT64 partition is composed of 2 BRAM columns, 2 DSP columns and 16 CLB columns (952 frames).

We used different ICAP controllers for the tests. We first used Xilinx's controller and its drivers. In this case, the microprocessor performs all the operations by itself. We enhanced the controller a little by coupling it to a DMA taking care of data transfers. Finally, we used our controller, with and without bitstream compression.

6.1 Performance evaluation

Our results are listed in Table 3. We measured the times taken for read or write operations and calculated the bandwidth (BW) for a reconfiguration, a frame-by-frame readback and a global readback (reading all frames at once). The frame-by-frame readback, far less efficient than global readback, was only tested with FaRM in order to compare it to Xilinx's IP, which cannot read more than one frame at a time. The results correspond to the partial bitstream AES encoder from Table 2.

We had to differentiate between communication throughput on the bus and ICAP throughput. Indeed, our controller does not transmit the extra padding frame read from ICAP when performing readback. In write mode, both throughputs are also different when considering reconfiguration with compressed bitstreams.

When using our IP, we obtain similar results for a readback with or without RLE compression. The set of commands to be sent to the ICAP requesting a readback is prepared on-line so it does not have to be compressed. It is not very relevant to compress this part since it contains about 20 words, which can be neglected most of the time.

TABLE 3
Reconfiguration and readback times

$f_{\text{bus}} = 100\text{MHz}$ $f_{\text{icap}} = 100\text{MHz}$	xps_hwicap			xps_hwicap + DMA		
	Time (μs)	COM BW (MB/s)	ICAP BW (MB/s)	Time (μs)	COM BW (MB/s)	ICAP BW (MB/s)
Reconfiguration	72800	1.29	1.29	1550	60.5	**60.5**
Frame/frame readback	159000	0.92	0.92	49200	2.97	2.97
Global readback	-	-	-	-	-	-
	FaRM			FaRM + RLE		
Reconfiguration	731	128	**128**	538	127	**174**
Frame/frame readback	1536	47	95	1600	45	91
Global readback	560	129	129	560	129	129

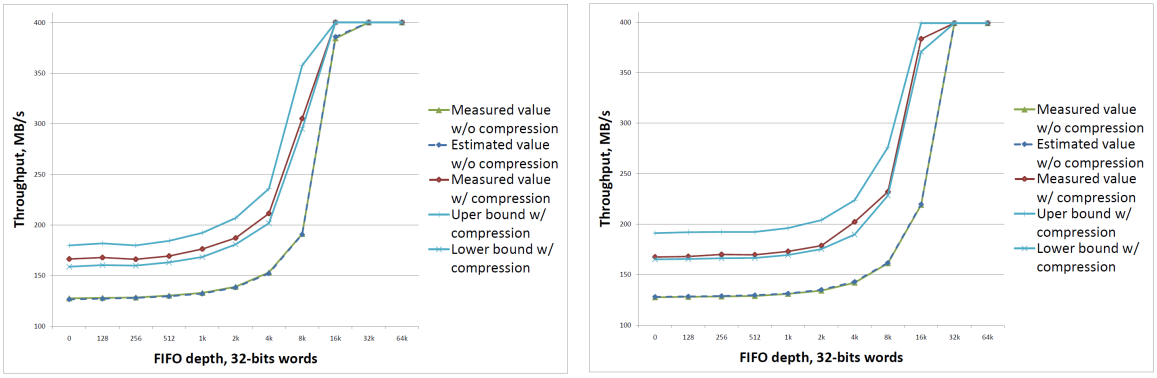
Compression plays an important role in reconfiguration time. In fact, results may vary depending on reconfigurable partition topology. In our case, the reconfigurable partition contained a BRAM column. A BRAM column is divided into interconnect frames and content frames that belong to different frame addressing spaces, so that BRAM contents are always initialized at the end of the configuration bitstream. This results in extensive compression at the end of the bitstream: the bus is free for a long period while the ICAP remains busy, lowering average performance.

We tested ICAP overclocking with the pre-load mode. The PLB bus ran at 125 MHz while the ICAP ran at 200 MHz without any problem. The bitstreams used were Count1 and Count2 from Table 2). Reconfiguration took 8304 ns while the ICAP needs at least 8290 ns (1658 ICAP cycles needed to write 6632 bytes), meaning that we achieved the maximum ICAP throughput of 800 MB/s. For the moment, we could not exceed the 100 MHz limit in readback mode.

6.2 FIFO depth influence on pre-load mode

In an attempt to estimate the benefit that can be achieved using pre-load, we measured ICAP throughput in this operating mode. Figure 6 shows the effect of FIFO depth on ICAP throughput for both AES and FFT64 applications. We were still using the partial bitstream AES encoder from Table 2, which is 24419 words long while FFT64 bitstream was 39218 words long. We obtain a maximum throughput of 400MB/s for AES when the FIFO is deeper than 16k words (64 kB), with or without compression. Of course, the result is highly dependent on bitstream size and compression. For instance, FFT64 is a bigger IP and thus requires a 32k FIFO to reach this throughput. The user might have to choose the best trade-off between performance and memory usage for his system. We also compared the actual results with the estimations we made. We took a latency of 10 cycles, 16 words per burst, 50 cycles per burst access (obtained with ChipScope Analyzer) and a period of 10 ns to comply with our benchmark. We can see that our estimation is excellent for both applications without compression (both plots coincide).

To help the user choose the FIFO depth, Fig. 7 shows the performance obtained for a given bitstream



(a) AES results (b) FFT results

Fig. 6. FIFO depth influence on ICAP throughput

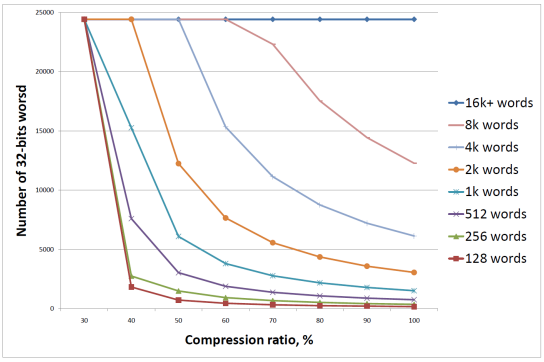


Fig. 7. Number of words written at maximum throughput

size and compression ratio with different FIFO depths. This performance is provided by the number of words FaRM writes to the ICAP at maximum throughput before needing some memory accesses (this is phase 1 of the pre-load). Here, the step value corresponds to bitstream size. We can see that the amount of data grows with compression since one word in the FIFO will generate more words to be written to the ICAP. We find that for a FIFO depth exceeding 16k words (64 kB), the whole bitstream can be transferred at maximum throughput, irrespective of its compression rate. This means that all the bitstreams from Table 2 except *FFT64* can be used at maximum throughput with a FIFO of this size. We may put this result to general use and say that a local memory of 64kB may be sufficient most of the time to reach the ICAP theoretical throughput, with a relatively low drawback in terms of memory use, considering the latest FPGAs series such as Virtex-6.

7 FUTURE WORKS

Up until now, we only focused on the performance of our ICAP controller and how to improve this performance. It may be interesting to study the power footprint of the IP. This is possible with Virtex-5 and Virtex-6 FPGAs which have some power monitoring functionalities (the Virtex-6 being much more accurate). We wish to develop power models for dynamic reconfiguration like we did for reconfiguration overhead.

In order to complete our approach, we would also like our reconfiguration manager to provide the user with services at a higher level of abstraction. It may gather information about the FPGA and RZs, such as available resources or the current module implemented for each zone, and use the information to perform on-line placement and scheduling, determining whether or not certain zones need reconfiguring based on pre-defined strategies (e.g. favouring power consumption or performance). Scheduling will use the models we presented for the reconfiguration overhead estimation and the power models we will be working on.

8 Conclusion

In order to reduce reconfiguration overhead, we propose an approach based on high-speed architecture combined with a sophisticated compression technique, Offset-RLE. The controller was tested with an AES encryption/decryption architecture. Enormous speedup was obtained compared to Xilinx's controller, resulting in a throughput of 800 MB/s for a write operation to the ICAP while working at 200 MHz and using the pre-load mode. We also propose an accurate reconfiguration overhead calculation model which can be used for developing scheduling strategies. The controller also has high speed auto-readback capability for retrieving configuration data from the FPGA.

Acknowledgements

This work was carried out in the framework of the ARDMAHN project [20] sponsored by the ANR, which aims at developing methodologies for home gateways integrating dynamic reconfiguration. We are also grateful to Xilinx, inc. for their support in the development of the FaRM readback capability.

References

[1] C. Kao, "Benefits of Partial Reconfiguration," *Xcell Journal*, vol. 55, pp. 65–67, 2005.

[2] K. Paulsson, M. Hübner, S. Bayar, and J. Becker, "Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III-based Systems," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Sep. 2008, pp. 699–700.

[3] S. Guccione, D. Levi, and P. Sundararajan, "JBits: Java based interface for reconfigurable computing," 1999.

[4] Xilinx Inc., *Virtex-5 Configuration User Guide*, 2010.

[5] ——, *LogiCORE IP XPS HWICAP datasheet*, 2010.

[6] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Comput. Surv.*, vol. 34, pp. 171–210, June 2002. [Online]. Available: http://doi.acm.org/10.1145/508352.508353

[7] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *J. VLSI Signal Process. Syst.*, vol. 28, pp. 7–27, May 2001. [Online]. Available: http://0-portal.acm.org.millennium.lib.cyut.edu.tw/citation.cfm?id=598544.598597

[8] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Aug. 2009.

[9] I. Belaid, F. Muller, and M. Benjemaa, "New Three-level Resource Management Enhancing Quality of Off-line Hardware Task Placement on FPGA," *International Journal of Reconfigurable Computing (IJRC)*, pp. 65–67, 2010.

[10] J. A. Clemente, C. Gonzalez, J. Resano, and D. Mozos, "A hardware task-graph scheduler for reconfigurable multi-tasking systems," *Reconfigurable Computing and FPGAs, International Conference on*, vol. 0, pp. 79–84, 2008.

[11] F. Redaelli, M. D. Santambrogio, and D. Sciuto, "Task scheduling with configuration prefetching and anti-fragmentation techniques on dynamically reconfigurable systems," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '08. New York, NY, USA: ACM, 2008, pp. 519–522. [Online]. Available: http://doi.acm.org/10.1145/1403375.1403500

[12] B. Ouni, I. Belaid, F. Muller, and M. Benjemaa, "Placement of Hardware Tasks on FPGA using the BEE Algorithm," in *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS11)*, 2011.

[13] K. Siozios, G. Koutroumpezis, K. Tatas, D. Soudris, and A. Thanailakis, "DAGGER: A Novel Generic Methodology for FPGA Bitstream Generation and Its Software Tool Implementation," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, Apr. 2005, pp. 165b – 165b.

[14] D. Koch, C. Beckhoff, and J. Teich, "Bitstream Decompression for High Speed FPGA Configuration from Slow Memories," in *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, Dec. 2007, pp. 161 –168.

[15] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-Fast Downloading of Partial Bitstreams through Ethernet," in *Proceedings of the 22nd International Conference on Architecture of Computing Systems*, ser. ARCS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 72–83.

[16] M. Liu, Z. Lu, W. Kuehn, and A. Jantsch, "Reducing FPGA Reconfiguration Time Overhead using Virtual Configurations," *ReCoSoC*, 2010.

[17] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model," *ACM Transactions on Reconfigurable Technology and Systems*, 2010.

[18] Xilinx Inc., "Partial Reconfiguration User Guide," p. 100, 2010.

[19] A. Flynn, A. Gordon-Ross, and A. D. George, "Bitstream relocation with local clock domains for partially reconfigurable fpgas," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 300–303. [Online]. Available: http://portal.acm.org/citation.cfm?id=1874620.1874691

[20] ARDMAHN consortium, "ARDMAHN project," http://ARDMAHN.org/.