Étude et optimisation de l'interaction processeurs-architectures reconfigurables dynamiquement

Ben Abdallah Faten

N° d'ordre : 3539

THÈSE
présentée
DEVANT L'UNIVERSITÉ DE RENNES 1
pour obtenir

le grade de : *DOCTEUR DE L'UNIVERSITÉ DE RENNES 1*
Mention : TRAITEMENT DU SIGNAL ET TÉLÉCOMMUNICATIONS

par
Faten BEN ABDALLAH MANAI

Équipe d'accueil : Institut de Recherche en Informatique et Systèmes Aléatoires - CAIRN

École Doctorale : Mathématiques, Télécommunications, Informatique, Signal, Systèmes et Électronique

Composante Universitaire : École Nationale Supérieure de Sciences Appliquées et de Technologie

---

ÉTUDE ET OPTIMISATION DE L'INTERACTION PROCESSEURS-ARCHITECTURES RECONFIGURABLES DYNAMIQUEMENT

Soutenue LE 20 octobre 2009 devant la commission d'Examen

COMPOSITION DU JURY :

Président du Jury :		
Guy GOGNIAT	Professeur des Universités	Université de Bretagne Sud
Rapporteurs :		
Michel PAINDAVOINE	Professeur des Universités	Université de Bourgogne
Ridha BOUALLEGUE	Professeur des Universités	Université du 7 novembre Carthage
Examinateurs :		
Sébastien PILLEMENT	Maitre de Conférences	Université de Rennes 1
Olivier SENTIEYS	Professeur des Universités	Université de Rennes 1
Ammar BOUALLEGUE	Professeur des Universités	Université de Tunis ElManar

Remerciements

C'est avec mon enthousiasme le plus vif et le plus sincère que je voudrais rendre hommage à tous ceux qui, à leurs manières, m'ont aidé à mener à bien cette thèse.

Je tiens tout d'abord à remercier mon directeur de thèse M. Olivier SENTIEYS, professeur à l'Université de Rennes 1, pour m'avoir accueilli au sein de son équipe de recherche et pour m'avoir fait découvrir et partager ses visions scientifiques. Je tiens à le remercier vivement pour sa sympathie, sa compréhension et le temps qu'il m'a consacré malgré ses charges nombreuses.

Mes remerciements vont également à mon directeur de thèse M. Ammar BOUALLEGUE, professeur à l'Université de Tunis El Manar, pour avoir accepté de me diriger et pour tous les conseils avisés qu'il m'a prodigués. Outre ses qualités scientifiques, j'ai beaucoup apprécié sa générosité et sa gentillesse hors pair.

Je tiens à exprimer ma profonde reconnaissance à M. Sébastien PILLEMENT, maitre de conférences à l'Université de Rennes 1, pour son encadrement précieux, ses encouragements et ses coups de booster qui ont favorisé le développement de cette thèse. Je le remercie sincèrement pour sa disponibilité et son suivi attentif de mes travaux, pour son esprit critique et pour sa convivialité.

Ma reconnaissance va aux membres du Jury pour avoir répondu volontiers et chaleu-

reusement présents et pour m'avoir fait l'honneur de participer à ma soutenance. Un merci particulier s'adresse à Monsieur Guy GOGNIAT, d'avoir accepté la présidence du jury. Sa participation est, pour moi, un honneur.
Je suis reconnaissante envers notamment Messieurs Michel PAINDAVOINE et Ridha BOUALLEGUE pour le temps qu'ils ont consacré à rapporter mes travaux et les différentes remarques et suggestions qu'ils ont faites.

Mes remerciements chaleureux vont aussi à toutes les personnes avec qui j'ai été amenée à discuter et à valider mes travaux de recherche et plus particulièrement, aux membres du groupe de travail CAIRN au sein de l'ENSSAT.

Une pensée particulière s'adresse à tous les amis qui ont contribué, de près ou de loin, à rendre mes séjours plus agréables et enrichissants ainsi que tout le personnel de l'ENSSAT et de l'ENIT pour le suivi pédagogique de ma thèse.

J'adresse toute mon affection à ma famille pour son amour, sa tendresse et ses encouragements depuis de nombreuses années. J'ai une pensée très tendre et très reconnaissante à l'égard de mes parents pour leur apport quotidien à la réalisation de ce travail. Ils m'ont toujours donné l'espoir d'aller de l'avant.

Le dernier et non le moindre des mercis, je le dédie à mon époux. Nejah les mots me manquent pour vous remercier. Grâce à vous, je suis non seulement une meilleure scientifique, mais je me considère aussi une meilleure femme.

Table des matières

Introduction

Contexte de l'étude

L'évolution des réseaux de télécommunications a permis, grâce à d'importants débits, l'apparition de nouveaux services orientés Traitement Du Signal et des Images (TDSI). Ces applications sont essentiellement orientées vers le traitement de flots de données et ont vu leurs besoins (notamment en termes de puissance de calcul) suivre une progression très importante, impliquant un volume important d'opérations arithmétiques à réaliser en temps réel.

Conjointement à l'évolution des normes de télécommunications, les concepteurs de circuits intégrés ont vu les capacités d'intégration des circuits augmenter de façon considérable leur offrant alors des possibilités d'implémentation de système complet sur une puce SOC pour (*System On Chip*). Le système sur puce est une solution technologique dédiée à l'intégration sur un même substrat de silicium d'un ensemble de composants de natures hétérogènes.
Un exemple de SoC hétérogène est présenté sur la figure 0-1 où chaque macro-bloc réalise une fonction spécifique. Généralement, le microprocesseur est chargé de la gestion des différentes tâches de l'application, le *DSP* (*Digital Signal processor*) est responsable de décharger le microprocesseur en assurant le traitement des motifs de calculs liés aux traitements du signal. Les composants spécifiques *ASIC* (*Application Specific Integrated Circuits*), intégrant des IP (Intellectual Property), sont mis en œuvre pour traiter efficacement les fonctions critiques (chronophages). Enfin, les composants re-

configurables de type *Field Programmable Gate Array, FPGA* assurent au circuit des possibilités d'évolution en redéfinissant complètement, ou partiellement, des caractéristiques de son architecture. La mémoire assure le stockage des données de l'application. La communication entre les différents composants du système est assurée par un réseau d'interconnexions (bus, bus hiérarchiques, crossbar, NoC (*Network on Chip*)...).

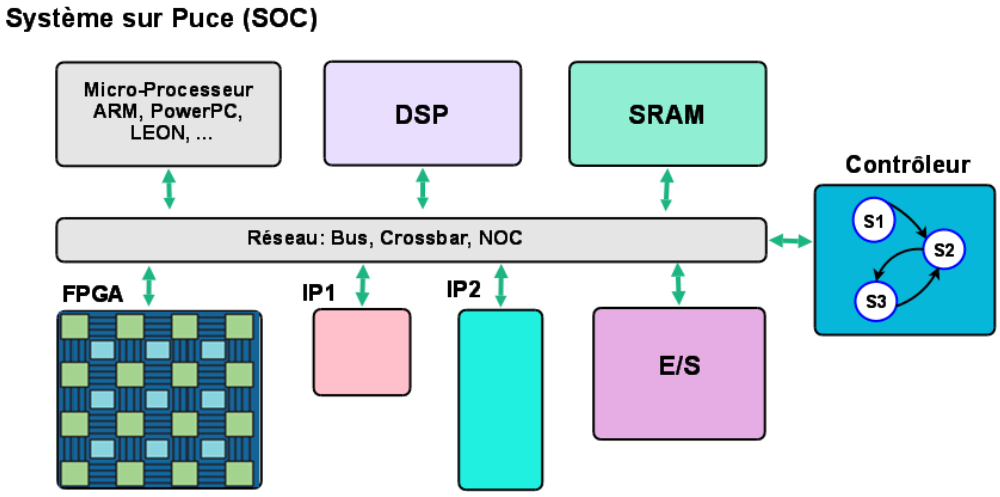

Figure 0-1 – **Exemple d'un système sur puce (SOC) hétérogène**. *Le microprocesseur est chargé de l'agencement des différentes tâches de l'application. Le DSP est responsable de décharger le microprocesseur en assurant le traitement des motifs de calculs liés au traitement du signal. Les IP, sont mis en œuvre pour traiter efficacement les fonctions critiques et les composants reconfigurables de type FPGA assurent au circuit des possibilités d'évolution en redéfinissant complètement, ou partiellement, des caractéristiques de son architecture. La mémoire assure le stockage des données de l'application. La communication entre les différents composants du système est assurée par un réseau d'interconnexions (bus, bus hiérarchiques, crossbar, NoC...)*

L'ITRS (*International Technology Roadmap for Semiconductors*) [1] prévoit que les performances de calcul des SoC pour les applications embarquées vont être multipliées par 100 entre 2009 et 2022 avec des besoins en puissance de calcul pouvant dépasser en 2022 les 70 TFLOPS. La croissance des besoins en puissance de calcul est soutenue par l'augmentation du nombre de processeurs et de composants calculatoires DPE (*Data Processing Engine*) dans le SoC (Figure 0-2).

La combinaison de ces deux évolutions (puissance de calcul et capacité d'intégration) place les concepteurs devant une multitude de solutions architecturales créant un espace de recherche de conception très grand.

D'autre part, les nouvelles contraintes de l'embarqué nécessitent d'autres caractéristiques du SoC tout aussi contraignantes. Les plus communes parmi celles ci sont le

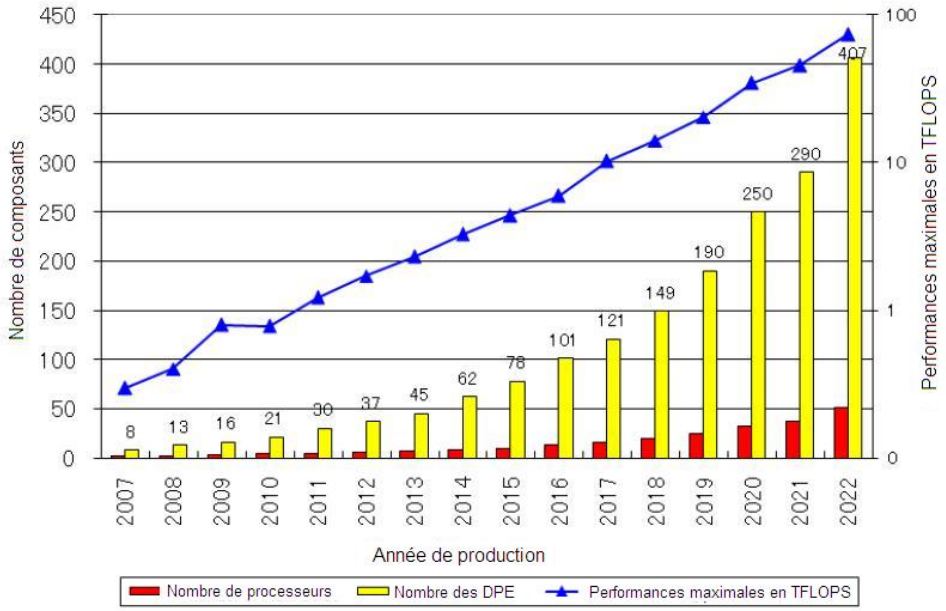

Figure 0-2 – **Prévisions des performances de calcul des SoC [1]**. *Les performances de calcul des SoC pour les applications embarquées vont être multipliées par 100 entre 2009 et 2022 avec des besoins en puissance de calcul pouvant dépasser en 2022 les 70 TFLOPS. La croissance des besoins de calcul est soutenue par l'augmentation du nombre des processeurs et des composants calculatoires DPE dans les SoC.*

faible coût du circuit et la nécessité de concevoir des circuits ayant une faible consommation d'énergie.

Problématique de l'étude

Dans le domaine des télécommunications si les SoC, tel que représentés sur la figure 0-1, sont parfaitement adaptés aux traitements des réseaux de télécommunications actuels, il en est tout autrement pour les réseaux de télécommunications futurs. En effet, la diversité des traitements devant être implantés impose l'intégration d'un très grand nombre de blocs dédiés. Le coût de ces systèmes devient dès lors très prohibitif, d'autant plus que la faible flexibilité des systèmes dédiés restreint leur durée de vie.
Afin de pouvoir supporter les exigences des normes actuelles et futures, de nouvelles solutions architecturales ont vu le jour : les architectures hybrides reconfigurables. Il s'agit d'utiliser conjointement des processeurs d'usage général ou spécifiques, dont les performances atteignent aujourd'hui des niveaux très élevés (en million d'opérations

par seconde (MOPS) mais aussi en fonctionnalités d'entrées-sorties), et un ou plusieurs circuits reconfigurables dynamiquement. La reconfiguration dynamique permet de modifier, en cours d'exécution, partiellement ou complètement la fonctionnalité (sa configuration) du circuit.

Le besoin de caractérisation de ces architectures est alors important afin que le concepteur puisse identifier, suivant un certain nombre de critères, l'architecture en adéquation avec son domaine applicatif. Nous détaillons ci-après quelques challenges liés à ce domaine.
Quels sont les mécanismes qui permettent un échange optimal d'informations entre le processeur et l'ARD (*Architecture Reconfigurable Dynamiquement*) ?
Comme les architectures hybrides reconfigurables se révèlent être une alternative pour l'accélération des applications TDSI, que peut-on espérer de l'amélioration des performances ? Peut-on définir des métriques mesurant cette accélération ? Existe-t-il une limite architecturale ou algorithmique à l'accélération ?
Si ce type d'architectures permet une réduction de l'énergie dissipée par rapport à celle dissipée par le traitement sur le processeur seul, peut-on avoir une mesure quantitative de cette réduction ? Si une application doit être implémentée sur ce type d'architectures, existe-t-il une méthode pour identifier les tâches qui doivent êtres mappées sur le composant reconfigurable dynamiquement ainsi que l'ordonnancement de leur exécution ?

Contributions

C'est à la problématique développée ci-dessus que nous nous proposons de répondre dans cette thèse.
Le contexte de l'étude regroupe trois axes de recherche représentés à la figure 0-3 : le traitement du signal et des images, l'adéquation algorithme architecture et les architectures hybrides reconfigurables. Nos travaux se situent à l'intersection de ces trois axes.

Dans ce contexte, une première étape dans nos travaux est de proposer une méthodologie d'exploration de l'espace de conception des architectures hybrides reconfigurables. La conception des systèmes hybrides souffre de l'absence de modèles et de méthodes capables de quantifier leurs performances. Pour cela, nous avons développé des modèles flexibles et paramétrables pour les architectures hybrides reconfigurables ainsi que des

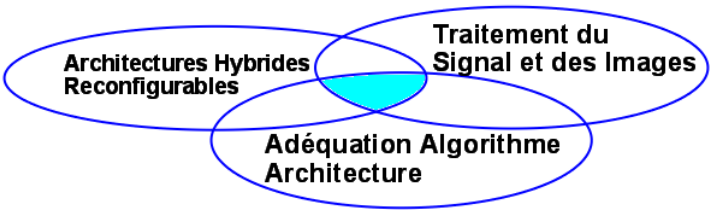

Figure 0-3 – **Contexte de l'étude**. *Nos travaux se situent a l'intersection du traitement du signal et des images, de l'adéquation algorithme architecture et des architectures hybrides reconfigurables.*

modèles d'estimation précis de leurs performances.
Par ailleurs, nous proposons une méthodologie d'adéquation algorithme architecture hybride (AAAH) qui permet de déterminer, pour une application donnée, le modèle de couplage CPU-architecture reconfigurable dynamiquement optimal.
En outre, nous proposons une méthodologie permettant d'exploiter efficacement les possibilités de reconfiguration dynamique. Nous intégrons cette méthodologie dans le flot de compilation d'une architecture hybride comportant le circuit reconfigurable dynamiquement DART conçu au sein de notre équipe de recherche.

Nous avons validé nos méthodologies par l'implémentation d'une application de calcul intensif : un démodulateur DVB-T/H multi-mode (c'est à dire basculant d'une configuration à une autre suivant les caractéristiques du canal de transmission et de l'état de la charge de la batterie) sur une architecture hybride couplant un cluster reconfigurable dynamiquement DART et un processeur de traitement de signal, le TMS320C6713.

Plan du mémoire

Le premier chapitre de ce document présente les systèmes hybrides reconfigurables. En explorant l'espace de conception de ces architectures, nous dressons un bilan qualitatif en mettant le point sur les paramètres ayant un impact sur les performances de ces systèmes. Ce chapitre commence par définir les concepts sous-jacent de ces architectures et montre les points d'intérêts des architecture reconfigurable dynamiquement.
Le chapitre II est consacré à la modélisation des architectures hybrides reconfigurables. Nous développons dans un premier temps les modèles d'estimation de leurs

performances ; Ensuite nous présentons un modèle de communication processeur - architecture reconfigurable dynamiquement permettant de représenter leurs comportements réalistes pour chaque type de couplage.

Au sein du troisième chapitre, une méthodologie d'adéquation algorithme architecture hybride est proposée. Cette méthodologie permet de déterminer, pour une application donnée, le modèle de couplage CPU-ARD optimal. Dans ce chapitre, nous montrons la généricité de notre méthode en l'appliquant à une implémentation d'un décodeur WCDMA (*Wideband Code Division Multiple Access*) sur un cluster reconfigurable dynamiquement de l'architecture DART.
Ce chapitre présente ensuite une méthodologie d'optimisation de l'implémentation sur l'architecture, visant à exploiter de façon optimale la reconfiguration dynamique. Cette méthodologie exploite les modèles d'estimation de performances démontrés dans le chapitre II.

La validation des approches proposées fait l'objet du quatrième chapitre où sont exposés les résultats de l'implémentation d'un démodulateur DVB-T/H multimode sur une architecture hybride reconfigurable composée d'un cluster reconfigurable dynamiquement DART et d'un DSP TMS320C6713. Ce démodulateur DVB-T/H permet de passer d'un mode DVB-T/H à un autre permettant d'accommoder la fiabilité et l'efficacité énergétique aux conditions de transmission du signal.

Finalement, en conclusion, nous effectuons une synthèse des contributions de ce travail de recherche et proposons différentes perspectives possibles.

Chapitre

1

Etat de l'art

Sommaire

Les architectures hybrides reconfigurables ("*Reconfigurable Computing*" en terminologie anglo-saxone) [2] se révèlent être des plate-formes particulièrement adaptées pour les applications embarquées qui combinent du contrôle et du calcul intensif. Ces plate-formes comprennent des unités de calculs logicielles (processeurs généralistes, DSP (*Digital Signal Processor*), microcontrôleur, etc.) et matérielles (unités de calcul reconfigurables, IP, etc.).

Dans ce chapitre, nous introduisons les systèmes hybrides reconfigurables et nous examinons l'espace de conception de ces architectures. A travers l'état de l'art des systèmes hybrides, nous dressons un bilan qualitatif en mettant le point sur les paramètres ayant un impact sur les performances de ces systèmes.

A Paradigme des systèmes hybrides reconfigurables

Les processeurs possèdent un paradigme de calcul se basant sur une implémentation temporelle des traitements qualifiée de logicielle (ou "Software"). Ils utilisent un faible nombre de ressources de calcul (en général une UAL (Unité Arithmétique et Logique)) qui sont réutilisés dans le temps, et présentent donc l'avantage d'occuper peu de surface de silicium au détriment de leur performance en temps et en consommation d'énergie.

A l'inverse les implémentations spatiales sont réalisées sur des architectures qualifiées de matérielles (ou "Hardware") dans lesquelles chaque opérateur existe en différents points de l'espace et traite des opérandes directement câblées sur ses entrées. Ainsi, les différentes opérations à effectuer ne sont pas réalisées de manière séquentielle comme dans le cas de l'implémentation temporelle, mais de manière parallèle. Ces architectures présentent l'avantage d'être performantes en temps et en consommation d'énergie, au détriment d'une surface de silicium plus importante et d'une flexibilité inexistante.
Le schéma de Horner (Figure 1-1) explicite le principe des implémentations temporelle et spatiale de l'équation de second degré $y = A \times X^2 + B \times X + C$.

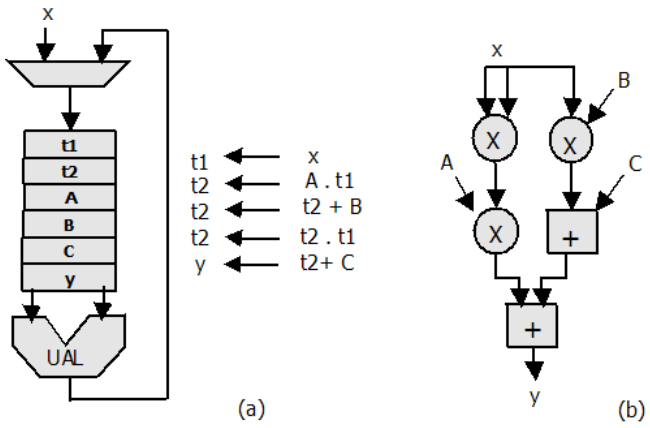

Figure 1-1 – **Exemple d'une implémentation temporelle (a) et spatiale (b) d'une équation second degré :** $y = A \times X^2 + B \times X + C$. *Dans le cas d'une implémentation temporelle, une ressource de calcul unique est partagée dans le temps entre les différentes opérations. A l'inverse, pour une implémentation spatiale, les opérations sont assignées à des opérateurs existants en différents points de l'espace.*

Le concept sur lequel repose les architectures hybrides consiste à mixer les implémentations spatiales et temporelles. Une implémentation logicielle d'une partie de l'application possède l'avantage de lui procurer une flexibilité liée à la possibilité de re-programmation, contrairement à une implémentation matérielle figée mais qui a l'avantage de satisfaire plus facilement les contraintes de performances. La flexibilité est mieux exploitée quand la partie matérielle est une architecture reconfigurable. En effet, un circuit reconfigurable dispose de mécanismes matériels capables de redéfinir complètement, ou partiellement, les caractéristiques de l'architecture qu'elle implémente. Chaque configuration temporelle distincte est appelée contexte de reconfiguration. Un processus appelé "reconfiguration", consistant à charger un nouveau contexte dans l'unité reconfigurable, permet de changer sa fonctionnalité, ce qui permet d'implémenter efficacement différents traitements à des instants disjoints.
Deux types de reconfiguration existent :

- la reconfiguration statique CTR pour (*Compile-Time Reconfiguration*) : où chaque application consiste en une ou plusieurs configurations (contextes). Pour reconfigurer un tel système, on doit l'interrompre pour procéder à sa reconfiguration et le redémarrer avec la nouvelle configuration. La reconfiguration statique peut être totale, c'est-à-dire que le contexte occupe la totalité des ressources matérielles du circuit, ou partielle dans le cas où les composants permettent de reconfigurer sélectivement une zone du circuit alors que le reste du système reste inactif tout en conservant ses informations de configuration. Le temps de reconfiguration dans ce cas dépend de la surface en termes d'éléments logiques du contexte à reconfigurer.

- la reconfiguration dynamique RTR pour (*Run-Time Reconfiguration*) : où les phases de reconfiguration et d'exécution sont concurrentes. Ceci permet un gain considérable en temps de traitement par rapport à la reconfigurabilité statique. Par contre l'association d'une architecture reconfigurable dynamiquement avec un processeur nécessite des mécanismes plus sophistiqués pour la gestion des flux de reconfiguration.

Cette thèse se place plus particulièrement dans le cas des architectures reconfigurables dynamiquement. En effet, afin de répondre à l'ensemble des contraintes inhérentes aux applications de télécommunications actuelles et futures, à savoir la flexibilité, la haute performance et la faible consommation d'énergie, les architectures hybrides intègrent des circuits reconfigurables dynamiquement.
On parle alors de RSoC pour (*Reconfigurable Systems on Chip*) qui sont des systèmes hybrides reconfigurables sur une même puce. Ils combinent des cœurs de processeur, des blocs de mémorisation, des modules d'entrée/sortie I/O, ainsi que des blocs recon-

figurables sur un même substrat reliés par un réseau d'interconnexion. La différence majeure entre l'architecture d'un RSoC et celle d'un système sur puce classique réside dans la présence, au sein même du RSOC, d'une architecture reconfigurable dynamiquement.

Comme son intégration dans une architecture hybride s'avère intéressante en termes de performances et de coût de circuit, le circuit reconfigurable est souvent nommé "accélérateur". Il est caractérisé par la granularité de son chemin de données et par le type de sa reconfiguration dynamique. L'interaction entre le processeur CPU (*CPU : Central Processing Unit*) et l'architecture reconfigurable dynamiquement dépend étroitement de ces deux paramètres.
Nous montrerons dans ce qui suit la manière dont les caractéristiques du circuit reconfigurable affecte la conception des architectures hybrides reconfigurables.

A.1 Reconfiguration dynamique

Nous distinguons deux modes de reconfiguration dynamique selon la manière de construire les contextes, total ou partiel, qui sont introduits dans les sous-sections suivantes.

A.1-1 Reconfiguration dynamique totale

Pour la reconfiguration dynamique totale (*Global RTR* en terminologie anglo-saxone), chaque application est divisée en un ensemble de contextes de reconfiguration dont chacun occupe la totalité des ressources matérielles du composant.
Tout changement de contexte implique la reconfiguration totale du composant. Un plan de configuration unique peut être actif à un certain moment sur le circuit reconfigurable, mais le passage à un autre plan (on parle de multi-contexte) peut être très rapide (de l'ordre de quelques nano-secondes). Par exemple, le projet de recherche industriel de NEC [3], basé sur le principe de multiplexage temporel, affiche des temps de reconfiguration de l'ordre de 5 à 30 nano-secondes.

Afin d'assurer la superposition entre les phases de calcul et de reconfiguration, le futur contexte à traiter est pré-chargé. Dans ce sens, des architectures comme GARP [4] et CS2000 [5] ont opté pour l'utilisation de la notion de cache de reconfiguration. Le cache de configuration consiste à stocker un nombre de configurations dans des mémoires (généralement en mémoire locale du composant) et de changer de contexte pendant l'exécution en un temps comparativement très faible par rapport à un chargement de

contexte à partir d'une mémoire externe. L'architecture CS2000 par exemple, possède un cache permettant de stocker une configuration et de reconfigurer le composant à partir de ce cache en un seul cycle. Le rapprochement entre les données de configuration et la ressource reconfigurable permet ainsi de réduire le temps de reconfiguration. Les performances sont alors nettement améliorées mais ceci se paie par une augmentation de la surface de silicium et de la consommation d'énergie.

A.1-2 Reconfiguration dynamique partielle

Dans le mode de reconfiguration dynamique partielle (*Local RTR* en terminologie anglosaxone), on reconfigure sélectivement une zone du composant, ce qui réduit considérablement le temps de reconfiguration.
Cette approche permet de reconfigurer une zone de l'architecture pendant que le reste du composant poursuit son fonctionnement. Elle est basée sur le concept de "matériel virtuel" qui est similaire à l'idée de mémoire virtuelle. Plusieurs contextes peuvent être actifs à la fois sur le composant selon le nombre de zones actives. Le temps de reconfiguration d'un contexte sur une zone de ce composant est masqué par l'exécution d'un autre contexte sur une autre zone. Afin de réduire la latence de reconfiguration totale, la notion de différence est appliquée lorsque les changements à faire entre l'ancienne et la nouvelle configuration sont minimes. Dans ce cas, on charge dans le composant un bitstream correspondant à la différence entre les deux versions, ce qui réduit le temps de reconfiguration.
Cette approche a été utilisée par la famille Virtex II pro de Xilinx [6]. L'organisation générale consiste en une matrice de CLBs (*Configurable Logic Block*) structurée en colonnes intercalées avec des colonnes de blocs dédiés (mémoire BlockRAM, des blocs DSP, des cœurs de processeurs) et entourée par des blocs d'entrée/sortie programmables. La reconfiguration dynamique est basée sur la notion de modules. Cette dernière permet de reconfigurer dynamiquement un module formé d'un minimum de quatre colonnes de *slices* (sur toute la hauteur du composant) et un maximum allant jusqu'à la largeur totale du composant avec chaque fois des incréments de quatre colonnes. Toutes les ressources logiques (IO-Blocks, ressources de routage, BlockRAM, blocs DSP...) contenues dans la largeur du module considéré font partie de son *bitstream*.

L'architecture Systolic Ring [7], présentée dans la figure 1-2, reprend le principe des FPGA (*Field Programmable Gate Array*) (grain fin), à savoir la présence d'une couche de configuration et d'une couche opérative. La brique de base de la couche opérative

est un composant nommé D-Node. Ce dernier consiste en un multiplieur et une UAL câblés travaillants sur des données de 16 bits stockées dans une file de registre. Le D-Node est apte, contrairement au CLB, à effectuer un éventail d'opérations principalement arithmétiques sur des flots de données. La couche de configuration est, comme pour les FPGA, une mémoire RAM permettant de stocker la configuration de tous les composants de la couche opérative. Une reconfiguration du D-Node est réalisée par une micro-instruction de 17 bits. La particularité ici réside dans la possibilité de modifier le contenu de cette mémoire en cours de traitement (c'est-à-dire de changer la fonctionnalité de tout ou d'une partie du réseau), grâce à l'utilisation d'un contrôleur de configuration.

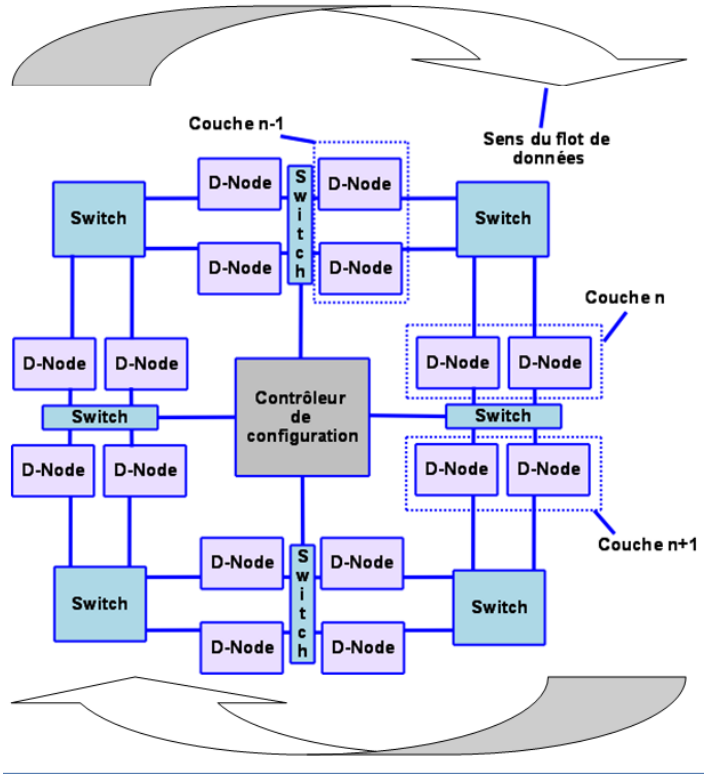

Figure 1-2 – ***Systolic Ring* est une architecture hybride à reconfiguration dynamique partielle**. *Elle est composée d'éléments de calcul appelés D-Nodes connectés en anneau. Le flot de données est propagé de D-Node en D-Node après reconfiguration dynamique. La reconfiguration d'un D-Node intervient pendant la phase de traitement des autres D-Node et permet une modification de sa fonctionnalité en un cycle d'horloge.*

A.1-3 Mécanismes d'optimisation de la reconfiguration dynamique

Lorsque les phases de reconfiguration se succèdent très rapidement, le temps alloué au chargement du flot de configuration peut ne pas être suffisant pour assurer la dynamicité de la reconfiguration. Il existe différentes techniques pour réduire la latence liée à la reconfiguration dynamique.

Une approche pour réduire cette latence de re-configuration est de réduire le volume du contexte de reconfiguration. Des techniques de compression peuvent être introduites pour diminuer le volume des données de configuration qui doivent être transférées à l'ARD [8] et réduire par conséquent considérablement la taille du cache de reconfiguration et le temps de chargement du contexte. En effet, Atmel a doté les architectures AT40K [9] et AT6000 [10] d'un cache logique associé à un algorithme de compression du *bitstream*.

Une autre technique pour réduire de manière sensible le volume de données de configuration est de chercher à exploiter le parallélisme de données, c'est-à-dire d'utiliser la même configuration pour plusieurs cellules reconfigurables. Cette technique est exploitée par l'architecture Morphosys [11].

Le bloc reconfigurable est ici un tableau de 8x8 cellules reconfigurables RC (*Reconfigurable Cell*). Cette architecture est considérée à gros grain puisqu'elle opère sur des données de largeur 8 à 16 bits, bien que chaque RC intègre une unité à grain fin FGB (*Fine Grain Bloc*) qui est une sorte de petit FPGA opérant au niveau bit. Le circuit reconfigurable est configuré d'une mémoire de contexte contrôlée par un cœur de processeur RISC (*Reduced Instruction Set Computer*)(Tiny Risc), se chargeant par ailleurs de l'exécution des traitements séquentiels et de la gestion des transferts de données entre la mémoire principale et le bloc reconfigurable. Ces transferts se font par l'intermédiaire d'un contrôleur DMA qui a par ailleurs la charge d'assurer les transferts de données entre la mémoire principale et une mémoire tampon de trame (*Frame Buffer*) stockant les données manipulées par le bloc reconfigurable.
Puisque cette architecture cible principalement les traitements par blocs disposant d'un parallélisme de données important (par exemple codage d'image ou de vidéo), les configurations des RC varient peu à l'intérieur du tableau. Dès lors, le choix retenu pour Morphosys est de configurer toutes les cellules d'une même ligne ou colonne de la même façon. La figure 1-3 présente la version M2 de l'architecture Morphosys qui a été optimisée pour les applications de télécommunication sans fil.

Une autre approche, présentée dans [12] utilise la notion de pipeline afin que le char-

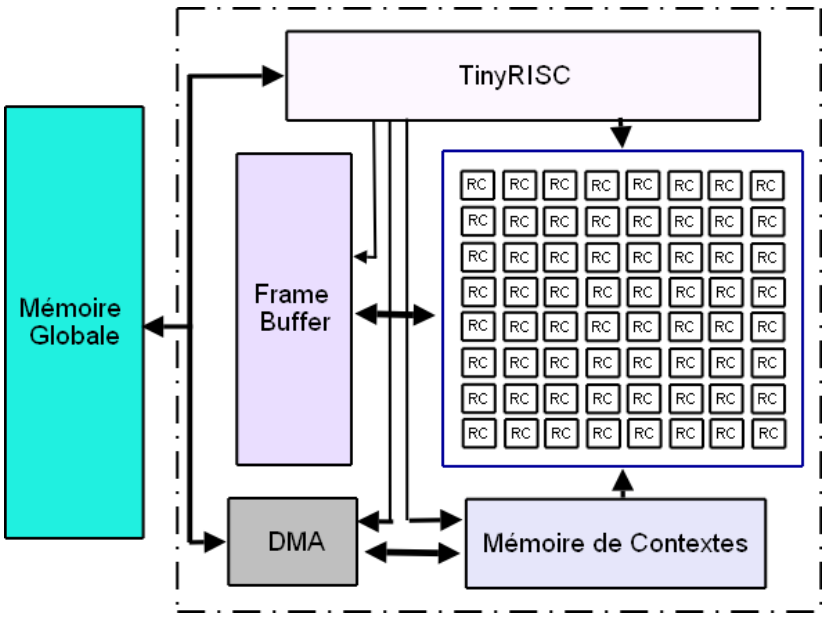

Figure 1-3 – **Architecture bloc de Morphosys (M2)**. *L'ARD, un tableau de cellules reconfigurables, est configuré par une mémoire de contexte contrôlée par un cœur TinyRISC.*

gement de la reconfiguration soit chronométré de façon à se chevaucher le plus souvent possible avec les phases d'exécution du circuit.

Finalement, le processus de transfert de données du processeur hôte au matériel reconfigurable peut être modifié de façon à inclure un cache de configuration qui peut fournir une reconfiguration rapide. Par ailleurs, la taille et les caractéristiques de ce cache dépendent essentiellement de la solution du couplage entre le CPU et l'ARD envisagée.

Outre le type de reconfiguration dynamique, l'étude de la granularité du motif de calcul du circuit reconfigurable est primordiale car elle permet de comprendre ses performances potentielles et sa flexibilité et agit par voie de fait sur les choix effectués par les concepteurs des architectures hybrides reconfigurables. L'étude de la notion de la granularité des motifs de calcul des ARD et son impact sur la conception des architectures hybrides fera l'objet de la section suivante.

A.2 Granularité de l'architecture reconfigurable dynamiquement

Par granularité en entend "taille en bits de la donnée élémentaire manipulée". Les architectures à faible granularité, typiquement les FPGAs, manipulent des données de 1 à 4 bits (bien qu'il existe des architectures de FPGA intégrant des multiplieurs sur des mots de 18 bits).
En effet, l'unité fonctionnelle (UF) de base d'une architecture reconfigurable à grain fin est la LUT (*Look-Up Table*). Il s'agit d'un bloc entièrement combinatoire programmable composé de n entrées $E_0 \ldots E_{n-1}$ d'une sortie et d'un tableau vertical de 2^n cellules mémoires connectées à un multiplexeur 2^n vers 1. La LUT est capable de réaliser toutes les fonctions possibles à n entrées. Pour cela, des cellules mémoires sont programmées selon la table de vérité de la fonction donnée. De plus, afin de permettre la mise en oeuvre des fonctions séquentielles, des bascules D sont intégrées aux éléments de calcul et un multiplexeur situé en sortie de l'élément permet de choisir ou non cette option (figure 1-4 (b)).

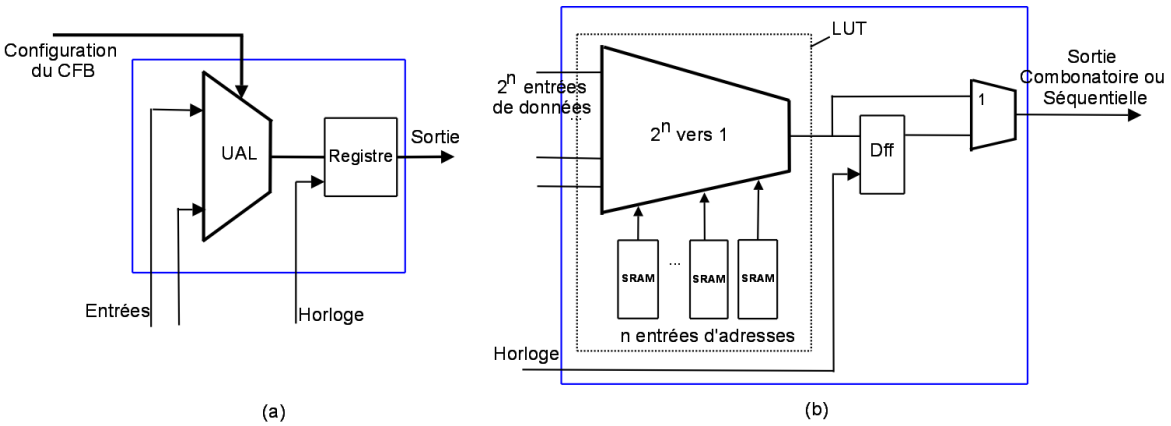

Figure 1-4 – **Architecture simplifiée d'un élément de calcul d'une architecture reconfigurable à grain épais (a) et à grain fin (b)**. *Le CFB est basé sur au minimum une UAL dont un mot de configuration permet de sélectionner la nature de l'opération à réaliser sur ses entrées. La sortie du CFB est synchronisée par une horloge. L'élément de calcul grain fin est composé d'une ou plusieurs LUT associées à une ou plusieurs bascules. Un multiplexeur permet de choisir la sortie séquentielle ou combinatoire.*

Les architectures à gros grain manipulent des unités fonctionnelles à 8 ou 16 bits voire même plus. L'unité fonctionnelle ou l'élément de calcul d'une architecture grain épais est le bloc fonctionnel configurable CFB (*Configurable Fonctionnel Bloc*). Il est

constitué au minimum d'une UAL et d'un registre de configuration (voir figure 1-4 (a)).

La granularité du bloc reconfigurable joue un rôle primordial pour les concepteurs au niveau des solutions proposées pour la conception d'une architecture hybride.

- Le temps de chargement d'une séquence de reconfiguration (appelée *bitstream* pour les architectures à grain fin) est directement proportionnel à la taille de la séquence. Les architectures à gros grain nécessitent des séquences plus courtes que les architectures à grain fin, donc un temps de configuration plus faible.
- Une faible granularité procure une plus grande flexibilité d'adaptation du matériel à la structure de calcul souhaitée. De plus, le grand nombre d'interconnexions entre les unités rajoute un délai conséquent à l'exécution du module implanté (malgré des structures permettant de réduire ce délai, comme les chaînes de propagation de retenue que l'on peut trouver dans les FPGA).
- Les circuits reconfigurables à gros grain utilisent des unités fonctionnelles qui ressemblent à ceux du processeur. Ceci représente donc une bonne opportunité pour le partage de ressources à faible coût avec le processeur.
- Du point de vue de la consommation d'énergie, les composants reconfigurables à grain fin utilisent des ressources de mémorisation locales plus importantes et dissipent donc plus d'énergie que celles à grosse granularité.

L'architecture lineDancer de Philips [13], une solution architecturale pour la radio logicielle et les applications sans fil, est une architecture hybride couplant un processeur SIMD (*Single Instruction Multiple Data*) avec une unité reconfigurable grain fin : le ASProCore. L'élément de calcul de base l'APE (*Associative Processing Element*) exécute la même opération logique ou arithmétique sur un bit. Si nous nous fixons l'APE comme brique de base du composant, la granularité est dans ce cas très fine et si nous considérons un nombre d'APE parallèles correspondants à la largeur du mot à traiter, la granularité est considérée comme adaptative (voir figure 1-5).

Un exemple d'architecture hybride implémentant des unités reconfigurables à une importante granularité est Chameleon Systems CS2000 [5]. Cette plate-forme intègre un processeur ARC (*Advanced RISC Compuing*) 32 bits et une unité de calcul reconfigurable RPF (*Reconfigurable Processing Fabric*) qui opère sur des données de largeur 32 bits. L'unité RPF est organisée en quatre tranches pouvant être reconfigurées indépendamment les unes des autres. Chaque tranche contient trois tuiles qui intègrent chacune sept unités de calcul sur 32 bits, deux multiplieurs 16*24 bits, quatre mémoires locales et une unité de contrôle logique (figure 1-6).

Afin de supporter la multi-granularité des applications, quelques architectures hybrides

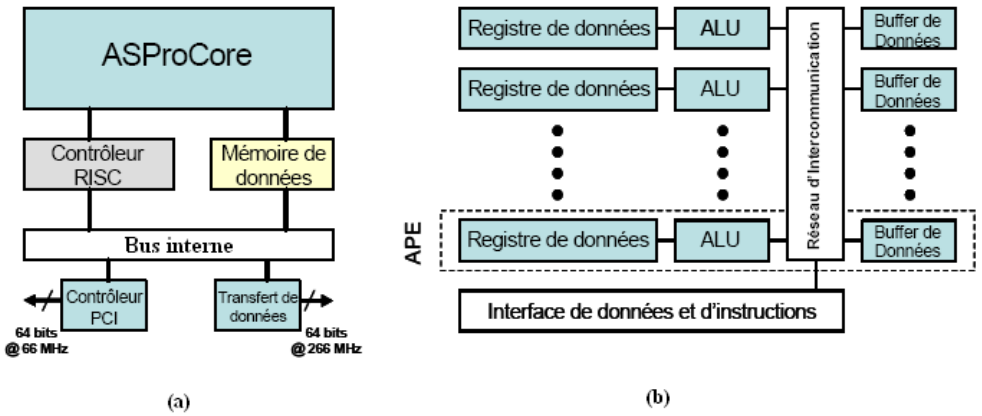

Figure 1-5 – **Architecture du *LineDancer***. *Cette architecture (a) couple un CPU SIMD avec une ARD grain fin le ASProCore. Le ASProCore (b) est composé d'éléments de calcul sur un bit appelés APE.*

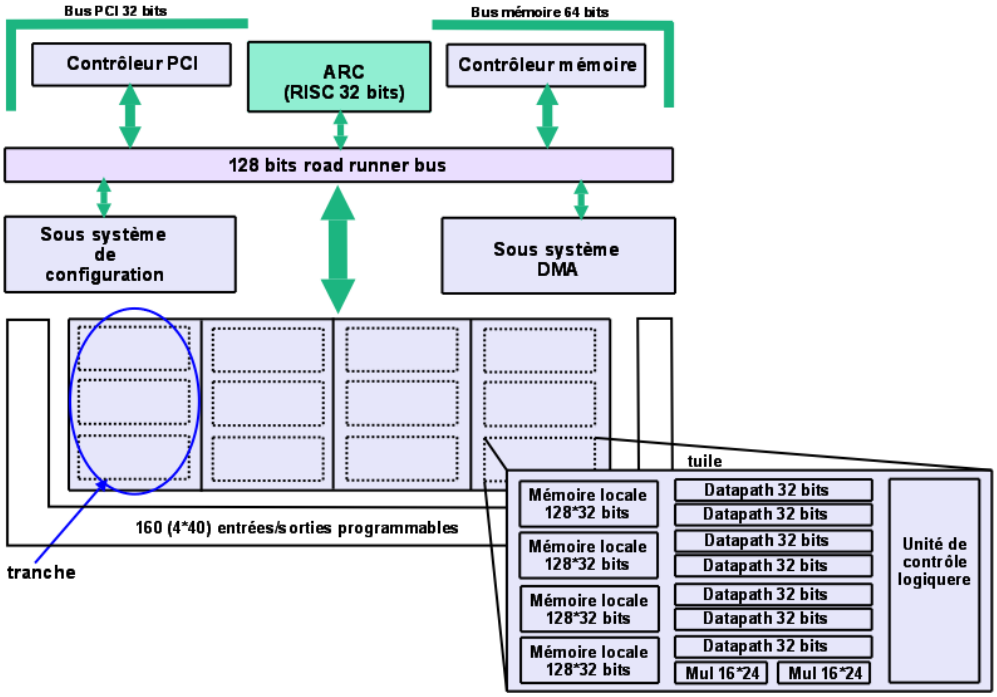

Figure 1-6 – **L'architecture hybride CS2000**. *Cette architecture opère sur des données de largeur 32 bits. Elle intègre un processeur ARC et une unité de calcul reconfigurable RPF (Reconfigurable Processor fabric). L'unité RPF est organisée en quatre tranches pouvant être reconfigurées indépendamment les unes des autres. Chaque tranche contient trois tuiles qui intègrent.*

comportent une architecture reconfigurable avec des unités fonctionnelles de différentes granularité. L'architecture Pleiades [14] conçue dans un souci de faible consommation

en est un exemple. Elle est basée sur un réseau d'interconnexion de type Crossbar qui relie un processeur hôte et des unités d'exécution de granularités différentes (satellites) ainsi que de la mémoire (figure 1-7). Cette architecture a été exploitée pour les terminaux mobiles 2.5 G et 3 G. En effet, les tâches qui cohabitent au sein d'un système mobile de ces générations manipulent des données de tailles très différentes avec des motifs de calculs et d'accès aux données eux aussi très variés. A titre d'exemple, un codeur source vidéo peut travailler par bloc sur des données arithmétiques peut être suivi d'un codage canal travaillant sur un flot de données binaire.

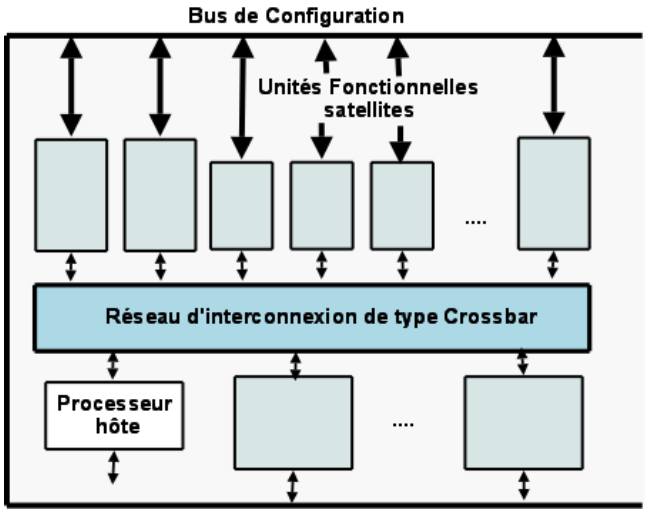

Figure 1-7 – **Architecture de Pleiade**. *Pleiade est une architecture hybride basée sur un réseau d'interconnexion crossbar reliant un processeur hôte et des unités d'exécution de granularités différentes (satellites).*

L'objectif des concepteurs des architectures hybrides est d'utiliser au mieux des ressources disponibles de la partie matérielle quel que soit le grain de son motif de calcul et en particulier celle de la reconfiguration dynamique afin de maximiser les performances.
Les performances d'une architecture hybride peuvent être analysées suivant plusieurs critères que nous nous proposons d'expliciter dans la section suivante.

A.3 Critères de performances

Afin de pouvoir évaluer les performances d'une architecture hybride reconfigurable, deux critères doivent être considérées : la puissance de calcul nécessaire à l'exécution d'une application et la dissipation d'énergie associée à cette exécution.

A.3-1 ACCÉLÉRATION DES TRAITEMENTS

Les besoins de traitement temps réel font que les temps d'exécution sur les architectures considérées doivent être inférieurs ou égaux à ceux imposées par les normes, appelés "contraintes temps réel".
Le temps d'exécution d'un tâche peut s'écrire comme une fonction de la latence L et de la cadence C. Pour un nombre d'échantillons traitées N, le temps d'exécution T_{exec} d'une tâche s'écrit :

$$T_{exec} = L + \frac{N}{C} \tag{1-1}$$

La latence L correspond au temps mis par l'architecture entre l'acquisition d'une entrée et la production de la première sortie. La cadence C caractérise le rythme auquel le système produit chacune de ses sorties (le nombre d'échantillons produits par seconde).

Les architectures hybrides trouvent principalement leur utilité dans l'accélération des traitements arithmétiques et logiques par rapport aux performances du processeur seul.
La loi d'Amdahl [15] définit l'accélération (le gain en vitesse d'une tâche) pouvant être obtenu par l'utilisation d'un dispositif d'amélioration donné. L'accélération sera le rapport :

$$Acceleration = \frac{T_{exec}\ sans\ amelioration}{T_{exec}\ avec\ amelioration} \tag{1-2}$$

La loi d'Amdhal reste incontournable dès qu'il s'agit de s'attaquer à l'optimisation d'un programme car cette loi s'applique dès que des portions de programme s'exécutent en parallèle sur différents processeur ou unités de calcul. La loi d'Amdhal s'exprime alors

$$Accleration = \frac{1}{P_s + \frac{1-P_s}{N_p}} \tag{1-3}$$

Avec P_s = pourcentage s'exécutant en mode scalaire et P_p = pourcentage s'exécutant en parallèle ($1 - P_s = P_p$), et N_p le nombre de processeurs.

A.3-2 CONSOMMATION D'ÉNERGIE

La maîtrise de la consommation d'énergie est un enjeu majeur dans le domaine de l'embarqué. En effet, cette consommation engendre une dissipation thermique qui provoque un vieillissement prématuré des composants, et augmente le coût des systèmes de refroidissement, leur taille ainsi que leur poids constituant un handicap de dimensionnement dans le cas des applications portables.

Puissance et énergie dissipée

L'énergie consommée dans un intervalle de temps [a,b] est par définition l'intégrale de la puissance dissipée : $E = \int_a^b P(t)dt$ où $P(t)dt$ est la puissance dissipée à l'instant t. Cette puissance dissipée dans un circuit électronique se compose de la puissance statique due aux courants de fuite et de la puissance dynamique due aux commutations des transistors (passage de l'état bloqué à passant et inversement). La puissance dynamique dissipée est définie par :

$$P_{dynamique} = \sum_{i \in E} \alpha_i.F_{clk}.C_{eq_i}.V_{DD}^2 \quad \forall i \in \{Opr;\ Ctr;\ Mem;\ inter\} \tag{1-4}$$

avec

$$C_{eq_i} = A_i.C_{s_i} \tag{1-5}$$

où E est l'ensemble des blocs du système (opérateurs,...), α_i est le taux d'activité c-à-d le nombre de transitions par cycle d'horloge, F_{clk} est la fréquence de fonctionnement, C_{eq_i} est la capacité équivalente. V_{DD}^2 est la tension d'alimentation. α_i dépend des données traitées et de la technique de codage utilisée. Le C_{s_i} caractérise le circuit utilisé, il est fonction de la capacité moyenne surfacique du composant i C_{s_i} et de la surface moyenne du composant i A_i.

On voit dans l'équation 1-4 qu'il existe quatre paramètres pour diminuer l'énergie consommée et toutes les techniques de réduction de la puissance dynamique s'attaquent à l'un ou l'autre de ces facteurs.

Dans les circuits CMOS avec des finesses de gravures supérieures à 0.13 μm la puissance dynamique représente 80-85 % de la puissance dissipée. Ainsi classiquement, la puissance statique est négligée. Du fait de la diminution de la technologie, la consommation statique devient de plus en plus importante et elle devrait même devenir prépondérante dans les années à venir. Il devient alors indispensable de prendre en compte cette dissipation statique qui dépend principalement des paramètres technologiques.

Efficacité énergétique

L'efficacité énergétique d'une architecture [16] est définie comme le nombre de millions d'opérations réalisées pour la consommation de 1 mW de puissance.

$$EE = \frac{N_{op}.F_{clk}}{P} \quad \left[\frac{MOPS}{mW}\right] \tag{1-6}$$

Dans l'équation 1-6, N_{op} représente le nombre moyen d'opérations réalisées par cycle. Le produit $N_{op}.F_{clk}$ représente donc la puissance de calcul de l'architecture considérée

et s'exprime en MOPS. P représente quant à elle la consommation de puissance sous-jacente à l'exécution de N_{op} opérations par cycle.

L'efficacité énergétique représente le compromis performance/consommation de l'architecture. Il est à noter que cette métrique ne présage pas de la qualité en performance d'une architecture mais permet de mettre en avant les optimisations souhaitables pour obtenir des architectures efficaces du point de vue de la consommation.

Pour les architectures hybrides reconfigurables, les performances présentées ici dépendent essentiellement de l'intégration architecturale de la ressource reconfigurable dans la plate-forme du processeur hôte.
Nous présentons dans la section suivante un état de l'art des différentes architectures selon la manière de coupler le processeur et l'architecture reconfigurable dynamiquement.

B Intégration architecturale

Le problème de l'intégration architecturale est le fait de déterminer comment la ressource reconfigurable peut-être intégrée avec le processeur, et en particulier le niveau de couplage entre ces deux ressources. Le processeur a généralement en charge les structures de contrôle qui configurent la logique, lancent les traitements, gèrent les transferts de données ainsi que les interfaces. La position relative du cœur du processeur et de l'ARD détermine les délais de communications et de reconfigurations desquels dépendent étroitement les performances de l'architecture hybride.
Dans cette section, nous proposons d'aborder les différentes stratégies d'intégration architecturale existantes dans la littérature, en suivant la taxonomie couramment utilisée.
La figure 1-8 présente les modes de couplage entre CPU et ARD variant de "faiblement couplés" à "fortement couplés". Le premier mode (a), correspondant au niveau de couplage le plus faible, connecte l'ARD comme un périphérique. Le deuxième mode (b) couple plus étroitement l'ARD en la connectant au CPU à travers une interface ou bus spécifique. Le troisième mode (c) connecte l'ARD au bus interne du CPU (en mode coprocesseur). Pour le mode le plus étroit (d), la ressource reconfigurable est située sur les chemins de données internes du processeur au même titre qu'une unité entière ou flottante.

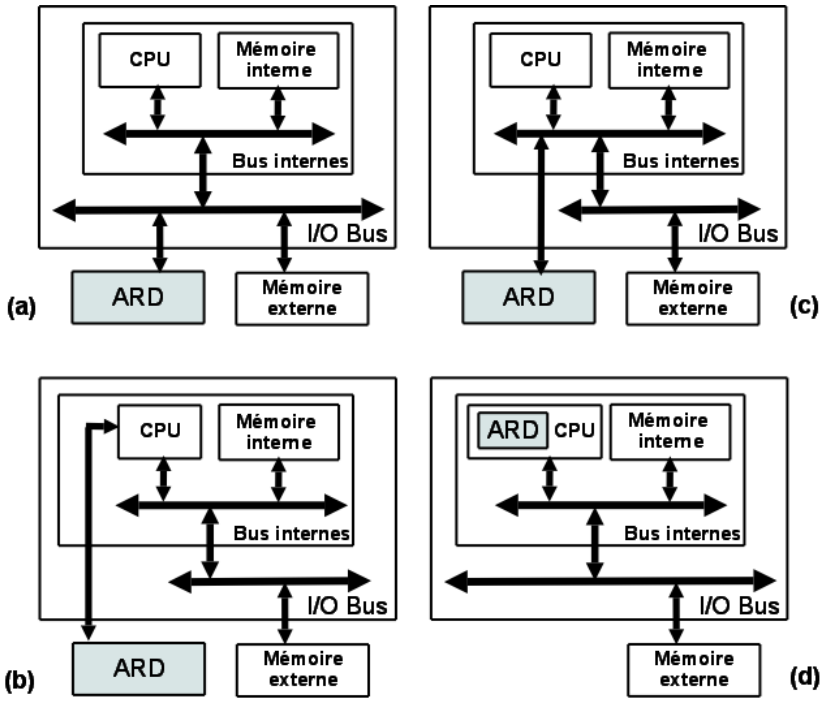

Figure 1-8 – **Types de couplage processeur (CPU)- architecture reconfigurable dynamiquement (ARD)**. *(a) couplage en mode périphérique ; (b) couplage en mode point à point ; (c) couplage en mode coprocesseur ; (d) couplage en mode unité fonctionnelle.*

Quelques travaux (exemple [17] et [18]) combinent même deux niveaux de couplages dans une architecture hybride afin de bénéficier des avantages de chaque type d'intégration architecturale. La figure 1-9 présente le principe sur lequel repose ces architectures. En effet, les accélérateurs qui sont fortement couplés au processeur hôte ont un accès direct à la mémoire ou cache interne. Ils doivent tous fonctionner à une seule fréquence d'horloge qui correspond nécessairement à la plus faible entre les fréquences respectives des différentes ARD. Par contre, les accélérateurs faiblement couplés accèdent à la mémoire à travers un pont et donc chacun utilise sa propre fréquence d'horloge. Ainsi, implémenter une application donnée sur les architectures fortement couplées permet d'effectuer un accès mémoire en un seul cycle d'horloge mais au coût d'un fonctionnement à une fréquence d'horloge probablement plus faible. Inversement, une implémentation sur les architectures faiblement couplées permet une exécution avec des fréquences individuelles plus rapide avec un accès en mémoire sur plusieurs cycles.

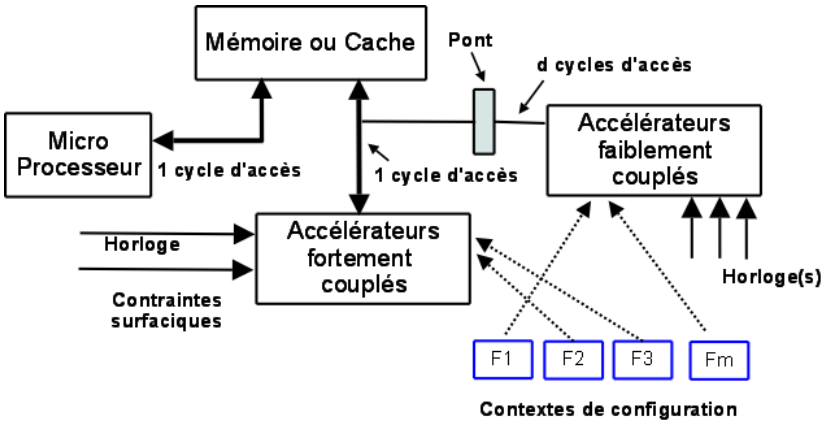

Figure 1-9 – **Principe d'une architecture combinant deux niveaux d'intégration architecturale**. *Les accélérateurs fortement couplés au processeur hôte ont un accès direct à la mémoire ou cache interne et doivent tous fonctionner à une seule fréquence d'horloge. Par contre, les accélérateurs faiblement couplés accèdent à la mémoire à travers un pont et utilise chacun sa propre fréquence d'horloge.*

B.1 UNITÉ PÉRIPHÉRIQUE RECONFIGURABLE

Les Unités Périphériques Reconfigurables sont les premiers types d'architectures d'accélérateurs matériels reconfigurables à être apparus dans la communauté du calcul reconfigurable. Cette approche propose l'intégration de la ressource reconfigurable en tant qu'unité périphérique du système hôte, en la connectant sur le bus périphérique (I/0) (voir figure 1-8 (a)). Dans ce type de couplage, le processeur peut donc communiquer avec le circuit reconfigurable, grâce à des instructions d'entrées-sorties permettant de déclencher les transactions de chargement des données et des contextes de reconfiguration. Ces transactions se font au moyen de transferts de blocs de données, dans lesquels la mémoire principale (externe) est partagée entre le processeur et le périphérique, la synchronisation se faisant au moyen d'interruptions matérielles.
De ce fait, ce type d'architecture offre de nombreux avantages.

- Un coût relativement faible, grâce à l'utilisation de composants matériels standardisés (circuit d'interfaces périphériques) et produits en gros volumes. Le coût final de la carte est en général imposé par le composant reconfigurable lui-même.
- Le fait d'interconnecter l'accélérateur matériel au travers d'un bus périphérique permet une grande flexibilité quant à son organisation matérielle interne (nombre et topologies des interconnexions, utilisation de mémoires locales, génération d'horloge flexible, etc.), permettant de dimensionner le système selon les besoins de l'application.
- En tant que périphérique, la reconfiguration dynamique ainsi que l'exécution de l'ap-

plication sur l'ARD est réalisée sans une supervision continue du CPU.

Dans ce type de topologie, l'accélérateur ne bénéficie pas de la hiérarchie mémoire du processeur (mémoire cache). Cependant, le fait qu'il soit connecté comme périphérique permet d'y intégrer ses ressources mémoire propres. Celles-ci sont en général formées par un ensemble de bancs mémoire indépendants qui permettent une latence d'accès réduite (<20ns) et offrent bande passante élevée (jusqu'à 800 Mbps).
La faiblesse du modèle de couplage en tant que périphérique réside dans leur système de communication : l'utilisation du bus d'entrées-sorties limite très fortement la performance des communications entre l'hôte et l'unité reconfigurable, pour les raisons suivantes :

- dans un système complet, les autres périphériques consomment également une part de la bande passante disponible, et de plus, les arbitrages entre périphériques, mémoire cache, et processeurs induisent des latences d'accès importantes, qui réduisent d'autant plus la bande passante disponible ;
- les caractéristiques des périphériques actuels ne justifient que rarement l'existence de bus périphériques à hautes performances (exception faite des accélérateurs matériels de rendus 3D temps-réels).

Afin de répondre aux besoins de la reconfiguration dynamique qui impose un chargement simultané des données nécessaires à l'exécution de l'application et des contextes de reconfiguration et donc une bande passante assez importante, et de palier aux limites de performances des bus périphériques standards (PCI, S-bus ou VME), des architectures hybrides avec un couplage en périphérique comme DAP-DNA [19] utilisent des bus spécifiques haute performance (Figure 1-10).

L'architecture conçue pour le traitement d'image intensif est composée d'un cœur RISC 32 bits DAP (*Digital Application Processor*) et d'une architecture reconfigurable dynamiquement nommée DNA (*Distributed Network Architecture*) contenant 955 PEs de 16 bits arrangés en forme de *2D Mesh* (maillage 2D). Le DNA contient trois bancs mémoire pour le stockage des configurations. La communication entre le DAP et le DNA se fait à travers un bus périphérique haute performance "PCI Express" donnant un débit jusqu'à 8Go/s. Par contre, le transfert de données du DNA vers la mémoire externe (et vice-versa) se fait à travers quatre canaux d'interface (DNA Direct I/O) à 200 MHz chacun.
L'environnement de développement DAP/DNA-FW II permet, en partant d'une spécification de haut niveau d'une application (de type MATLAB/Simulink de MathWorks), d'assister le concepteur dans les différentes étapes du processus de développement jusqu'à l'implémentation sur le composant.

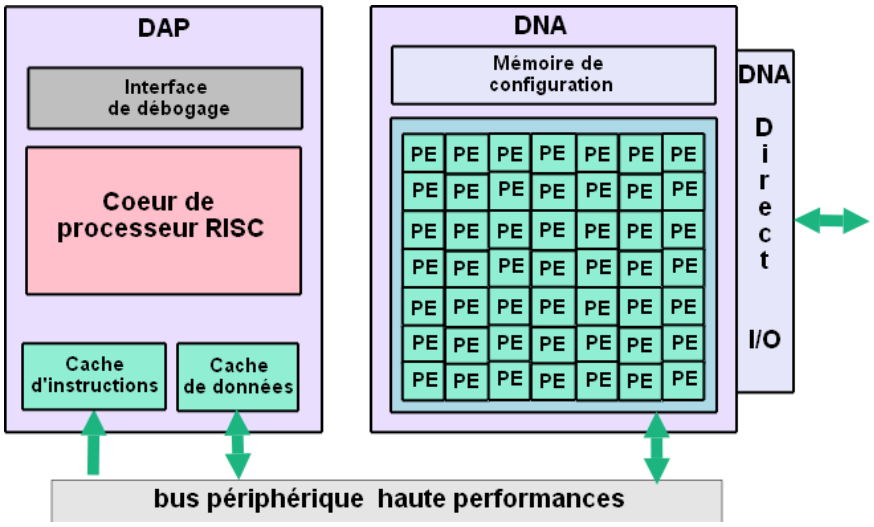

Figure 1-10 – **L'architecture hybride DAP-DNA**. *Dans cette architecture, l'ARD (DNA) est couplé au CPU (DAP) sur le bus périphérique haute performance donnant un débit jusqu'à 8 Go/s. Le transfert de données du DNA vers la mémoire externe (et vice-versa) se fait à travers quatre canaux d'interface (DNA Direct I/O) à 200 MHz chacun.*

B.2 UNITÉ RECONFIGURABLE EN COUPLAGE POINT-À-POINT

Dans un type de couplage point à point, le circuit reconfigurable est couplé directement au processeur à travers une interface ou un bus spécifique (Figure 1-8 (b)). Cette technique assure une bonne bande passante entre les deux ressources de calculs. De plus, il n'y a pas d'arbitrage à effectuer pour accéder au lien de communication. Le CPU a accès aux fonctionnalités qu'offre l'interface spécifique par l'intermédiaire de fonctions API (*Application Programming Interface*) qui permettent une utilisation transparente des ressources du reconfigurable.

Ce modèle permet de profiter des avantages que procure le modèle de couplage en périphérique aux architectures hybrides reconfigurable et en plus de s'affranchir de la latence élevée causée par le bus I/O.

Pour ce mode de couplage CPU/ARD, l'architecture Tartan [20] utilise un bus spécifique de largeur 96 bits. Les données et les contextes de reconfiguration sont acheminés entre un cœur RISC synchrone et une ressource reconfigurable asynchrone RF (*Reconfigurable Fabric*) à travers des FIFO. La RF est une architecture clusterisée gros grain asynchrone. Un cluster de la RF est formé de 4x4 pages, chacune formée de 16 PE. Chaque PE est une UAL de 8 bits. La communication intra-cluster est ba-

sée sur un réseau d'interconnexion NOC (*Network On Chip*) asynchrone (Figure 1-11).

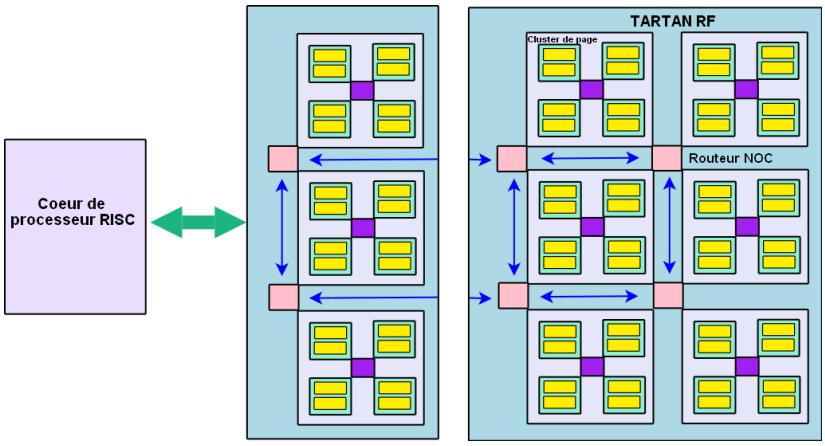

Figure 1-11 – **Tartan est une architecture hybride avec un couplage CPU/ARD point à point entre le cœur RISC et la Reconfigurable Fabric (RF)**. *La RF est une architecture clusterisée gros grain asynchrone. Un cluster de la RF est formé de* 4×4 *pages formée chacune de 16 PE. Chaque PE est une UAL de 8 bits. La communication intra-cluster est basée sur un réseau d'interconnexion NOC asynchrone.*

La solution de couplage en point à point souffre d'une latence d'acheminement des données non négligeable. Afin de palier à cette latence, quelques architectures hybrides utilisent des interfaces de communication spécifiques assurant l'accès de l'ARD à la mémoire externe de la puce. Dans ce cas, la mémoire doit être multi-ports. Pour d'autres, l'interface est formée par un ensemble de bancs mémoire indépendants (auxquels le CPU et l'ARD accèdent) permettant une latence d'accès réduite (<20ns) et offrant une bande passante élevée. C'est la cas pour l'architecture présentée dans [21]. En effet, le CPU et le reconfigurable sont connectés en point à point à travers ICURE une interface intelligente illustrée par la figure 1-12. L'interface ICURE comprend principalement une mémoire de données, un cache de contextes, une matrice de routage reconfigurable et un contrôleur de reconfiguration. Elle a pour rôle de faire communiquer de façon transparente un CPU avec une unité de calcul reconfigurable UCR (Unité de Calcul Reconfigurable) et de gérer intelligemment les processus de reconfiguration. Le CPU est connecté à ICURE par un bus d'adresses et un bus de données standards. L'interface ICURE est connectée à l'UCR au moyen de ses ports d'entrée/sortie (I/O) bidirectionnels et d'un lien de reconfiguration. La reconfiguration de l'UCR est effectuée par une unité de gestion dédiée située à l'intérieur de l'interface.

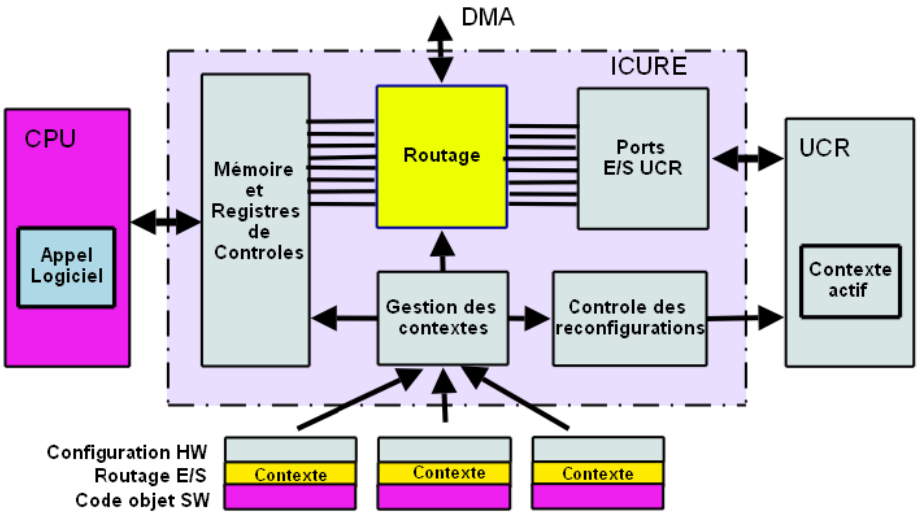

Figure 1-12 – **Un CPU et une UCR (Unité de Calcul Reconfigurable) connectés en point à point à travers l'interface spécifique ICURE**. *L'interface ICURE comprend principalement une mémoire de données, un cache de contextes, une matrice de routage reconfigurable.*

B.3 COPROCESSEUR RECONFIGURABLE

Dans le cas de couplage en coprocesseur reconfigurable, la zone reconfigurable dynamiquement est connectée au bus interne du processeur (Figure 1-8 (c)). Le bus interne peut être soit un bus système ou un bus spécifique sur puce tel que AMBA de ARM [22] ou CoreConnect d'IBM [23].
L'architecture présentée dans [24], conçue par STMicroelectronics [25] afin d'accélérer les traitements d'image, présente un processeur 32 bits couplé en mode coprocesseur à un e-FPGA (*embedded-FPGA*) à travers une interface spécifique sur bus AMBA (Figure 1-13). Cette architecture permet une accélération de 1.5 à 2 fois pour les algorithmes de reconnaissance de visage (*face recognition*).

Dans la reconfiguration dynamique, un chargement simultané des données nécessaires à l'exécution des tâches et des contextes de reconfiguration est possible. Pour le modèle de couplage en coprocesseur, ceci peut causer un goulot d'étranglement sur le bus interne. Afin de limiter l'accès à la hiérarchie mémoire au cours du chargement de la configuration, la plupart des architectures hybrides dotent l'ARD coprocesseur de sa propre mémoire de configuration (cache, SRAM...).

L'ARD est capable d'exécuter le traitement sans la supervision continue du processeur puisqu'il a accès à toute la hiérarchie mémoire du CPU (mémoire cache). Cependant,

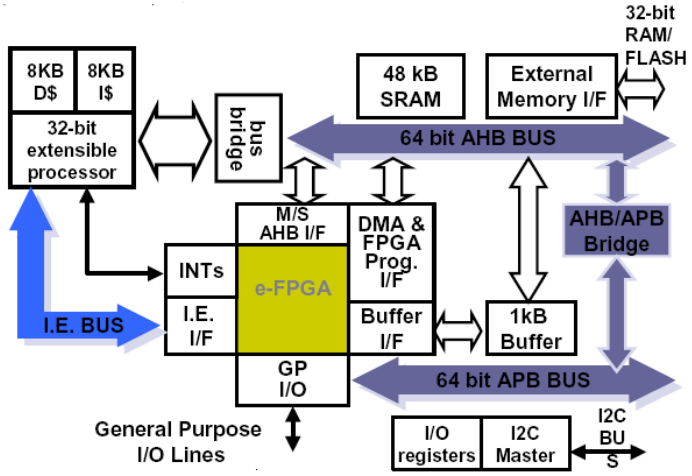

Figure 1-13 – **Une architecture hybride concue par STmicroelectronics avec coprocesseur reconfigurable**. *cette architecture présente un processeur 32 bits couplé à un e-FPGA à travers le bus sur puce AMBA*

elle doit être initialisée par le CPU. Ce dernier doit aussi contrôler les accès mémoire en lui référençant les adresses source et destination ainsi que la taille des blocs mémoire. Il faut toutefois prévenir les problèmes de cohérence mémoire en utilisant un modèle de programmation : "'mémoire partagée"' ou "'passage de message"'.
Le chargement et l'exécution des contextes de configuration sur la partie reconfigurable sont contrôlés par le processeur en utilisant différentes techniques.
Dans l'architecture GARP [4], le jeu d'instructions du processeur (MIPS II) a été étendu afin de contrôler l'accès du bloc reconfigurable à sa hiérarchie mémoire. L'unité reconfigurable est sous la forme d'une matrice de blocs logiques organisée en 32 colonnes de 24 blocs chacune. Un des 24 blocs est un bloc de contrôle et les blocs restants sont des blocs logiques opérant sur des données de largeur deux bits (granularité fine). La matrice reconfigurable est aussi dotée d'un cache de configuration pouvant supporter quatre configurations pour chaque colonne. La reconfiguration à partir de ce cache ne prend que quatre cycles (Figure 1-14).
Le compilateur [26] pour cette architecture (avec un front end SUIF C compiler [27]) met en oeuvre des techniques utilisées pour les compilateurs d'architectures VLIW afin d'identifier le parallélisme au niveau instruction dans le programme source. Il construit également un ordonnancement de l'exécution des parties de l'application sur l'unité reconfigurable.

L'architecture ZIPPY [28], développée par ETH Zurich réalise le contrôle des accès à

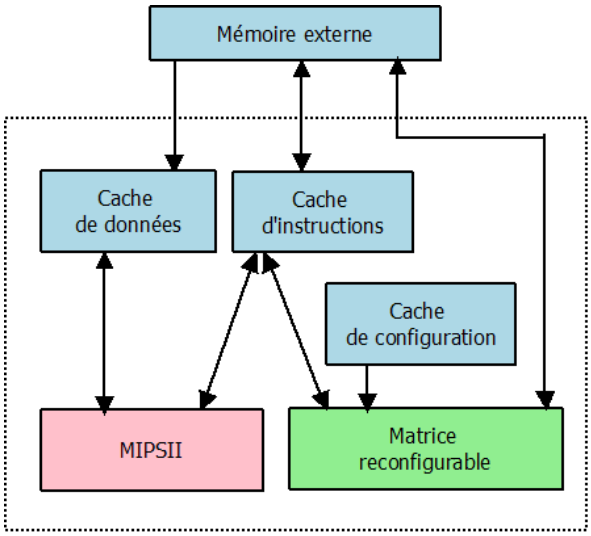

Figure 1-14 – **Architecture de Garp**. *Elle utilise le couplage en coprocesseur entre un processeur MIPS II et une matrice de blocs logiques dotée d'un cache de configuration.*

la hiérarchie mémoire à travers une exécution de type *multithread*. En effet, dans cette architecture (voir figure 1-15), le cœur de processeur exécute le programme jusqu'à rencontrer un traitement critique généralement reconnu par une directive assembleur. Dans ce cas, la zone reconfigurable RU (*Reconfigurable array Unit*), composée d'une matrice de CU (*Computing Unit*) organisées en mesh, est reconfigurée. Un séquenceur de contextes contrôle le chargement des différentes configurations à partir de la mémoire de configuration. Pour éviter de solliciter le bus de données du processeur, le CPU dispose d'instructions lui permettant de venir mettre des données dans une FiFo. L'accès à ces données lors de la mise en fonctionnement de la zone reconfigurable pourra alors être effectué sans solliciter le processeur.

Le mode d'intégration architecturale en "Coprocesseur reconfigurable" possède les avantages suivants.

- Le fait de bénéficier de toute la hiérarchie mémoire permet au reconfigurable d'avoir un accès rapide aux données et donc de limiter la latence de communication entre le CPU et l'ARD.
- Les calculs sur le reconfigurable peuvent là aussi s'effectuer en recouvrement avec ceux du processeur dans la mesure où il n'y pas de conflits au niveau des accès mémoire.

Néanmoins, ce modèle de couplage entre la ressource reconfigurable et le processeur possède les inconvénients suivants.

- Le coût de l'intégration de l'ARD sur le bus interne du CPU est relativement plus élevé que celui de son intégration sur le bus externe.

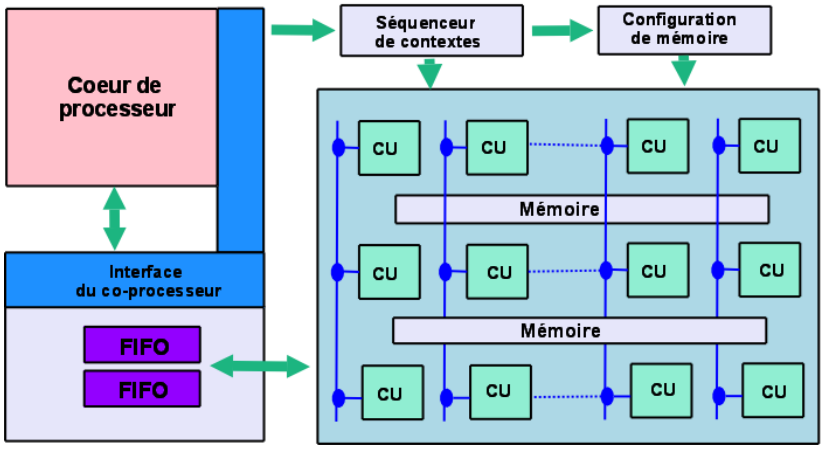

Figure 1-15 – **Architecture Zippy**. *L'ARD composée d'une matrice d'unités de calcul (CU) organisées en mesh est couplée à un cœur de processeur. Un séquenceur de contextes contrôle le chargement des différentes configurations à partir de la mémoire de configuration. Pour éviter de solliciter le bus de données du processeur, le CPU dispose d'instructions lui permettant de venir mettre des données dans une FiFo.*

- Du point de vue logiciel, au cas où l'ARD a un accès direct à la hiérarchie mémoire, il faut étendre le jeu d'instruction du processeur. Ce qui représente un coût supplémentaire dans la conception de l'architecture hybride.
- Pour le bus système, les performances en terme de bande passante dépendent en grande partie de l'arbitrage d'accès au bus et de la charge du bus.
- Dans le cas d'un bus spécifique sur puce, l'ARD tout comme les autres composants sur la puce, doit communiquer à travers un adaptateur (*Wrapper*) qui permettra de l'adapter au protocole de communication du bus. ce qui représente un coût supplémentaire pour la conception de ce type d'architectures hybrides.

B.4 UNITÉ FONCTIONNELLE RECONFIGURABLE

Cette alternative propose le couplage le plus fort entre le processeur et la ressource reconfigurable (voir figure 1-8 (d)). Les architectures hybrides utilisant cette approche sont appelées processeurs reconfigurables (*Reconfigurable Processor*). Dans ces systèmes, l'unité reconfigurable est directement intégrée au chemin de données du processeur. L'accélérateur est vu comme une unité fonctionnelle du CPU, c'est pourquoi il a été nommé par les concepteurs des architectures hybrides RFU (*Reconfigurable Functional Unit*). Ce dernier accède aux données directement à travers les registres du processeur. La file de registres doit être multi-ports afin de permettre un accès simultané à ou aux unité(s) de traitement du processeur et à l'ARD. Cette dernière est

contrôlée par un sous-ensemble d'instructions dédiées à la gestion de cette ressource. La figure 1-16 présente le principe d'intégration d'une ARD de type FPGA au chemin de données d'un processeur.

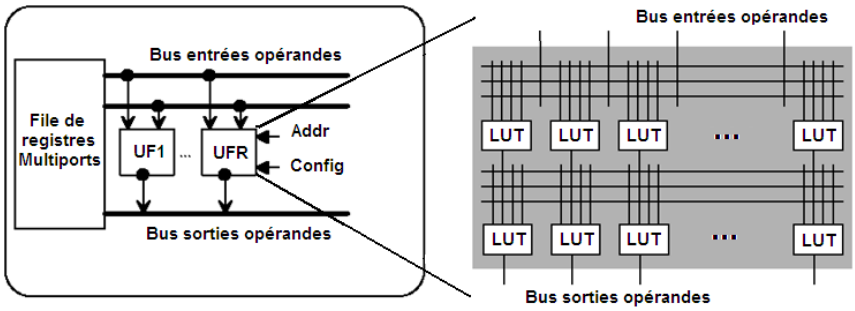

Figure 1-16 – **Principe des processeurs reconfigurables**. *Une unité de calcul reconfigurable est intégrée au chemin de données du processeur au même ordre qu'une unité fonctionnelle. Le RFU accède aux données directement à travers la file de registre du processeur.*

L'avantage de ce mode de couplage réside dans le fait que les temps de latence de chargement des données et de contextes de configuration sont minimes. Cependant, la latence du CPU perdue dans le contrôle du processus de chargement des contextes et des données est importante, comme c'est le CPU qui gère tous les accès mémoire et déclenche les processus de reconfiguration. Plus les reconfigurations sont fréquentes, plus la latence de contrôle augmente. Ainsi, si les reconfigurations sont très fréquentes, le temps de reconfiguration de l'unité reconfigurable doit être très faible pour que le système présente un réel intérêt en terme de performances.

En outre, le coût de l'intégration du reconfigurable sur le chemin de données du CPU est non négligeable. Les architectures de processeurs exploitant le parallélisme au niveau instruction ILP (*Instruction Level Parallelism*) (VLIW (*Very Long Instruction Word*) ou superscalaire) sont plus prédisposés pour l'intégration du reconfigurable sur leurs chemins de données. En effet, ils disposent déjà d'une file de registre multi-port à laquelle sont connectées plusieurs unités fonctionnelles UF. De point de vue logiciel, une extension du jeu d'instructions de processeur est à prévoir. Ce qui représente un coût supplémentaire.

L'architecture Onechip [29] peut être considérée comme représentative de ce couplage.

L'approche Onechip (Figure 1-17) propose de coupler un processeur RISC avec un DPGA (*Dynamically Programmable Gate Array*). Ce dernier permet de stocker plusieurs configurations sur un FPGA, mais un seul contexte est actif à un instant donné. En effet, le DPGA contient une unité locale de stockage qui joue le rôle d'un cache pour les prochains contextes de configurations à exécuter. Le contexte est chargé dans cette unité de stockage quand le FPGA est en train d'exécuter le contexte précédent. Ceci permet de limiter la latence de reconfiguration puisque le nouveau *bitstream* est déjà chargé de la mémoire.

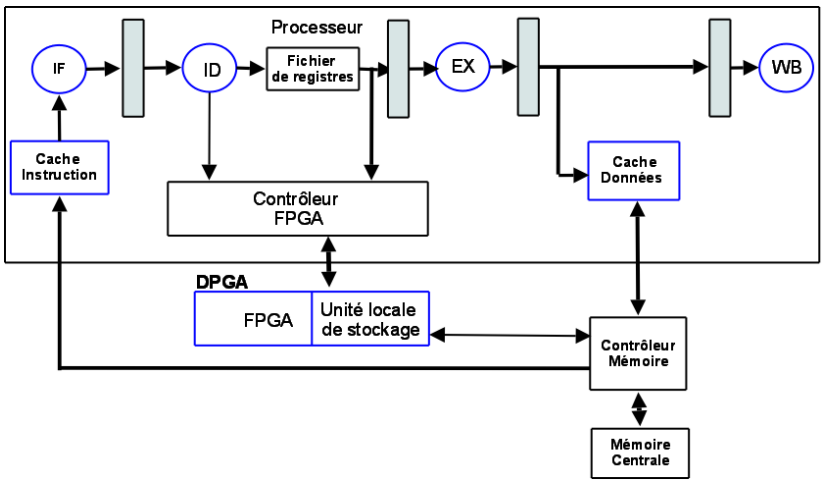

Figure 1-17 – **L'architecture Onechip98 est un processeur reconfigurable**. *Le DPGA est intégré en RFU à un processeur RISC. Le DPGA contient une unité locale de stockage qui joue le rôle d'un cache pour les prochains contextes de configurations à exécuter sur un FPGA.*

Le jeu d'instructions du CPU a été étendu afin de supporter des instructions indiquant que le FPGA doit se reconfigurer pour un traitement bien déterminé. Ces instructions comportent un *opcode* qui identifie le contexte de reconfiguration à charger sur le FPGA. En effet, le contrôleur FPGA vérifie si le contexte correspondant est chargé dans l'unité locale de stockage. S'il n'a pas été chargé auparavant, alors il déclenche le chargement à partir d'une adresse mémoire centrale qu'il fait correspondre à travers une "table de reconfiguration". Dans le cas contraire, le contrôleur FPGA déclenche le processus de chargement du bitstream de reconfiguration sur le FPGA.
Pour résoudre le problème d'aléas, Le DPGA notifie le CPU quand l'exécution de la configuration correspondante à l'instruction en cours est terminée et ainsi toutes les instructions bloquées qui ont des dépendances avec "l'instruction reconfigurable" (qui n'est autre que le traitement à exécuter sur le FPGA) peuvent reprendre.

Dans cette section, nous avons présenté l'état de l'art des architectures hybrides reconfigurables selon le couplage entre la ressource reconfigurable et le processeur hôte. A travers cet état de l'art, nous allons dresser dans la section suivante un bilan qualitatif de ces systèmes afin de déterminer les paramètres jouant sur les performances de celles-ci.

C Bilan qualitatif des systèmes hybrides

Durant cette dernière décennie, plusieurs architectures hybrides ont été commercialisées toutes aussi différentes d'un point de vue des caractéristiques architecturales que d'un point vue des outils et des méthodes de développement des applications.
Ces architectures ont toutes l'avantage de présenter une efficacité énergétique intéressante. Les architectures comme GARP et MORPHOSYS présentent une efficacité énergétique de l'ordre de 10 MOPS/mW. L'architecture Pleiades, en exploitant la multi-granularité, présente une efficacité énergétique de l'ordre de 32 MOPS/mW. Les architectures hybrides à gros grain comme le DAP-DNA et ZIPPY peuvent avoir plus que 40 MOPS/mW et représentent alors une grande opportunité pour une performance optimale.

C.1 Outils et méthodes de développement

Diverse méthodes de développement existent dans la littérature, elles permettent de procéder à la conception des applications sur les architectures hybrides et à la génération des flots de données de configuration qui permettront de mettre en oeuvre les processus de reconfiguration dynamique de la partie matérielle.

Une des approches se base sur deux chaînes de compilation distinctes, logicielle et matérielle, comme c'est le cas pour l'architecture CS2000. En effet, le flot de développement de cette architecture est basé sur deux chaînes d'outils de programmation distinctes. La première chaîne cible la ressource reconfigurable (RPF) et prend comme langage de spécification une description en Verilog. La deuxième chaîne cible le processeur ARC, elle est basée sur les outils GNU en ajoutant des bibliothèques, appelée eBIOS, qui tournent sur le processeur ARC pour contrôler la zone reconfigurable. Ces bibliothèques permettent au concepteur de charger les contextes de configuration, initier le calcul sur le RPF, contrôler son état...

Le modèle de développement le plus utilisé consiste à compiler une description haut niveau de l'application pour la transformer en un code exécutable sur le processeur hôte et en une succession de configurations pour le bloc reconfigurable. la description de haut niveau de l'application peut se faire en :

- langage procédural (e.g. C) ; pour quelques architectures le code (C) est ensuite traduit en une représentation intermédiaire comme c'est le cas pour l'architecture ZIPPY. En effet, l'environnement de développement de ZIPPY contient un compilateur C avec une bibliothèque de l'architecture matérielle. L'outil traduit l'application décrite en C en ZNF (*Zippy Netlist Format*) et génère ensuite l'exécutable pour le CPU et les contextes de configuration de la matrice reconfigurable.
  D'autres flots de compilation traduisent le code (C) en un graphe de flot de données auquel est appliqué le flot de synthèse matérielle de la ressource reconfigurable. Le flot de compilation de l'architecture XIRISC [30] en est un exemple. XIRISC est un DSP VLIW intégrant une unité fonctionnelle reconfigurable le PiCoGA (*Pipelined Configurable Gate Array*)(Voir figure 1-18). Pour cette approche, le code écrit auparavant pour le VLIW seul n'a pas besoin de re-écriture et est pris en main par le flot de compilation de l'architecture hybride. En effet, les procédures C à exécuter par la ressource reconfigurable sont traduites par le compilateur en des graphes de flots de données appelés *Pipelined Data Flow Graph* écrit à l'aide d'une syntaxe C proche du niveau RTL (*Register Transfer Level*) [31], utilisée pour les structures de compilation intermédiaires ;
- langage propriétaire de l'architecture comme le langage (NAPA-C) pour l'architecture hybride NAPA [32] et (RAPID-C) pour la programmation de l'architecture RAPID [33]. L'intérêt majeur de cette approche est de permettre d'expliciter le partitionnement et le parallélisme de l'application. Cependant, cette approche limite très nettement la portabilité du code ;

- langage pour la conception mixte tel que (CATAPULT-C) [34], (HANDLE-C) [35] et surtout le standard (SYSTEM-C) [36]. Ces langages s'appuient sur des librairies de classes construites à base du standard C++ et d'un noyau de simulation par événements. Ces librairies de classes permettent de modéliser des systèmes logiciels/matériels avec la possibilité de décrire la concurrence, la notion de temps et quelques types de données matériels spécifiques. Plusieurs outils commerciaux utilisent SYSTEM-C dans leurs flots de conception, parmi lesquels on trouve CoCentric de Synopsys et ConvergenSC de CoWare.

Du point de vue du flot de compilation, les approches basées sur un mapping de leur noyau autour du matériel reconfigurable (GARP et MORPHOSYS par exemple) réa-

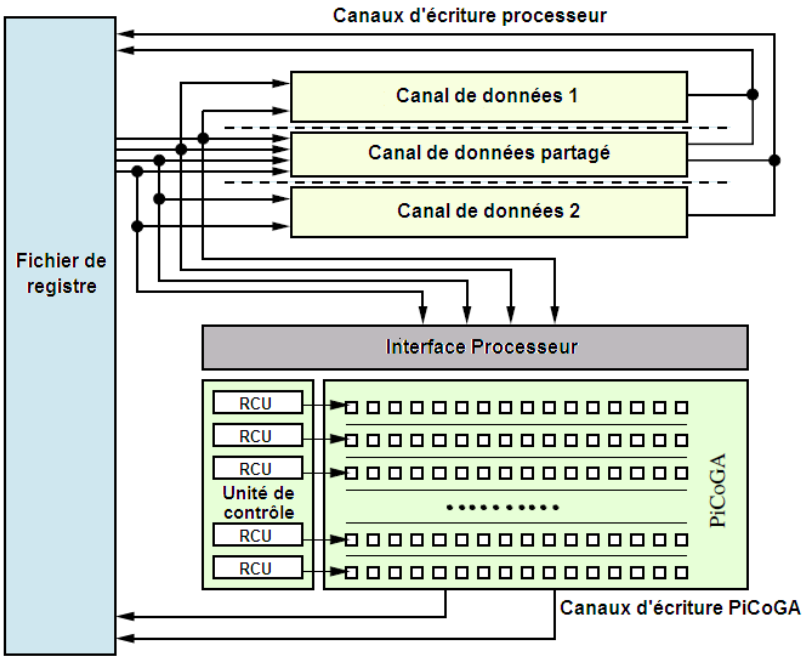

Figure 1-18 – **XIRISC une architecture hybride intégrant une unité reconfigurable (le PiCoGA) dans un processeur VLIW**.

lisent des accélérations significatives tandis que les approches qui étendent simplement le jeu d'instructions du processeur comme CS2000 voient simplement une amélioration marginale.

Les méthodes de conception des architectures hybrides reposent essentiellement sur l'étape du partitionnement logiciel/matériel. Dans le cas général, cette étape se subdivise en trois parties principales. La figure 1-19 reprend ces différentes étapes.

– Le partitionnement spatial (figure 1-19 (a)) : cette étape permet de prendre des décisions sur l'affectation (assignation) des tâches sur le matériel et le logiciel.
  Certaines approches, comme c'est le cas pour l'architecture XIRISC, identifient à travers un profilage du code les boucles candidates à être accélérées (nids de boucles : parties calculatoires consommant du temps dans l'application). Ces portions de code critiques en terme d'exécution seront implémentées sur le circuit reconfigurable.
  D'autres approches se basent sur une automatisation de la tâche de partitionnement spatiale en appliquant des algorithmes de partitionnement qui se basent sur la détermination d'une fonction de coût permettant de mesurer la qualité d'une solution donnée et de guider l'algorithme vers une meilleure solution (convergence).
  Dans [37], quatre types d'algorithme de partitionnement spatial sont présentés. Trois

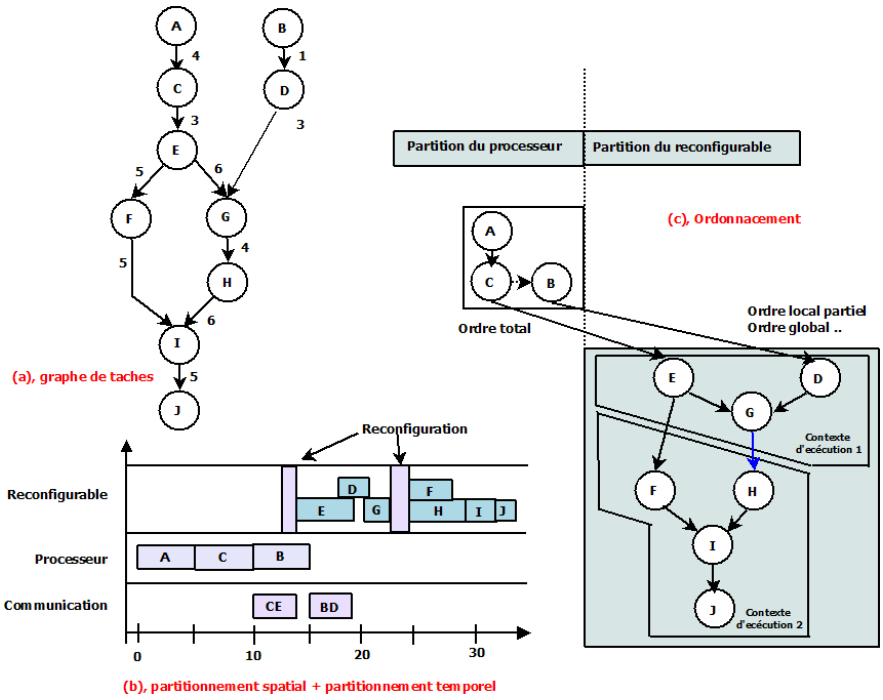

Figure 1-19 – **Etapes de partitionnement matériel/logiciel : (a) partitionnement spatial ; (b) partitionnement temporel ; (C) ordonnancement**.

sont classiques : le recuit simulé, Kernighan et Lin et un algorithme de clustering hiérarchique. Le quatrième algorithme est basé sur la notion de systèmes experts. Ces algorithmes ont été comparés par les auteurs sur la base des mêmes contraintes de temps, de surface et de mémoire et ceci en terme de qualité de la solution obtenue et de temps de calcul.

Dans [38], les auteurs présentent une méthode de partitionnement spatial basée sur une technique nommée VHEM (*Very Heavy Edge Matching*) cherchant à minimiser la fonction coût de communication global entre les différents nœuds d'un graphe de tâche.

– Le partitionnement temporel (figure 1-19 (b)) : cette étape n'existe dans le processus de partitionnement que dans le cas d'une reconfiguration dynamique de la ressource matérielle. Elle consiste donc à grouper les tâches déjà assignées à une implémentation matérielle sur le composant reconfigurable, au sein de segments temporels (contextes de configuration). Ces contextes vont être chargés puis déchargés de façon séquentielle sur le reconfigurable selon un ordre bien défini qui sera déterminé lors de l'étape d'ordonnancement.

Les auteurs de [39] proposent un algorithme de partitionnement temporel qui mini-

mise le coût de communication entre le CPU et le reconfigurable dynamiquement. Dans [40], la méthodologie de partitionnement temporel proposée vise à minimiser les ressources du reconfigurable utilisées pour implémenter un chemin de données sous une contrainte temps réel bien déterminée.

– L'ordonnancement des exécutions des tâches et des communications (figure 1-19 (c)) : le problème de l'ordonnancement est souvent couplé au problème de partitionnement temporel puisqu'ils ont de nombreuses similitudes si on considère les dépendances temporelles inter-contextes similaires à celles inter-tâches. Un contexte B, qui dépend du contexte A, ne peut être chargé sur le reconfigurable avant que A ne soit chargé et exécuté.
[41] a proposé une méthodologie d'ordonnancement complète ciblant la reconfiguration dynamique du XC-6200. Dans [42], les auteurs présentent une approche d'ordonnancement des contextes de configuration sur l'architecture hybride MorphoSys ainsi que des tâches au sein de ces contextes afin de minimiser le temps d'exécution global. Les travaux présentés dans [43] considèrent un modèle de type GdT (Graphe de Tâche) avec un ordonnancement sur plates-formes hétérogènes. [44] présente une méthode d'ordonnancement de GdT (basée sur les réseaux de neurones) pour architectures hétérogènes de type RSOC contenant une unité de calcul reconfigurable dynamiquement.

Pour ces différentes méthodes de conception des architectures hybrides, la qualité de l'intégration repose donc essentiellement sur la qualité du partitionnement. De plus, la définition de l'architecture ne tient pas compte directement des propriétés intrinsèques de l'application.
Une approche développée au Laboratoire d'Électronique des Systèmes TEmps Réel (LESTER) cherche au contraire à guider la construction de l'architecture en fonction des propriétés de l'application. Cette approche est appliquée à l'aide de l'outil *Design Trotter* [45].
La méthode est composée des étapes suivantes :

1. spécification de l'application dans un langage de haut niveau (langage C). Cette spécification est traduite dans la représentation interne HCDFG (*Hierarchical and Control Data Flow Graph*) grâce au parser C->HCDFG ;

2. caractérisation de l'application par un ensemble de métriques définissant son orientation (type d'opération dominant de celle-ci : traitement ou contrôle ou transfert mémoire) ainsi que sa criticité (parallélisme potentiel de ses sous-fonctions). Afin de procéder à l'estimation des fonctions (étape suivante), celles-ci sont classées par ordre de criticité décroissante. En effet, comme les fonctions les plus cri-

tiques ont un potentiel de parallélisme spatial important, elles se verront allouées un certain nombre de ressources. Il est alors possible d'envisager la réutilisation de ces ressources par les fonctions moins critiques ;

3. estimation intra-fonction dynamique consistant à utiliser pour plusieurs contraintes de temps un algorithme d'ordonnancement à temps contraint. L'ordonnancement permet, pour chacune des contraintes de temps, de trouver le nombre minimum de ressources nécessaires à l'exécution de la fonction dans le temps imparti. A cet effet, l'information d'orientation obtenue dans l'étape précédente permet de choisir au mieux le type d'algorithme d'ordonnancement à appliquer (trois types sont considérés : mémoire prioritaire, traitement prioritaire et mixte).
   Lorsque l'ordonnanceur n'arrive pas à trouver de solution respectant la contrainte du temps, il est possible de faire apparaître artificiellement du parallélisme à partir de différentes itérations d'une boucle donnée. Cette solution s'appelle le déroulage de boucle.
   Une boucle non déroulée est généralement modélisée par un nœud hiérarchique contenant une itération. Une boucle partiellement déroulée sera représentée par F fois le nombre de nœuds compris dans une itération avec F le facteur de déroulage de la boucle. La figure 1-20 présente un exemple de déroulage partiel d'une boucle. La boucle modélisée est déroulée d'un facteur 2.

   Le résultat de l'étape d'estimation intra-fonction est une courbe de compromis temps d'exécution/ressources qui étend les possibilités d'implantation de la fonction. Les bornes minimum et maximum de cette courbe correspondent à, d'une part l'exécution de la fonction en un temps égal à son chemin critique (exécution avec un haut degré de parallélisme), et d'autre part à l'exécution de la fonction dans le temps nécessitant le nombre de ressources minimum (exécution quasi séquentielle) ;

4. les résultats obtenus lors des étapes d'estimations vont permettre de procéder à un prépartitionnement : une étape de projection de chacune des fonctions de l'application sur une cible modélisée. Deux types de projections sont possibles dans l'outil DesignTrotter : la projection logicielle (cible de type microprocesseur) et la projection matérielle (cible de type composant programmable FPGA).
   Il est ainsi possible de classer les fonctions selon le type d'implantation qui leur semble le plus approprié à priori. Trois catégories sont considérées : les fonctions potentiellement implantables sous forme matérielle, logicielle et celles où il est encore trop tôt pour décider d'un type d'implantation.

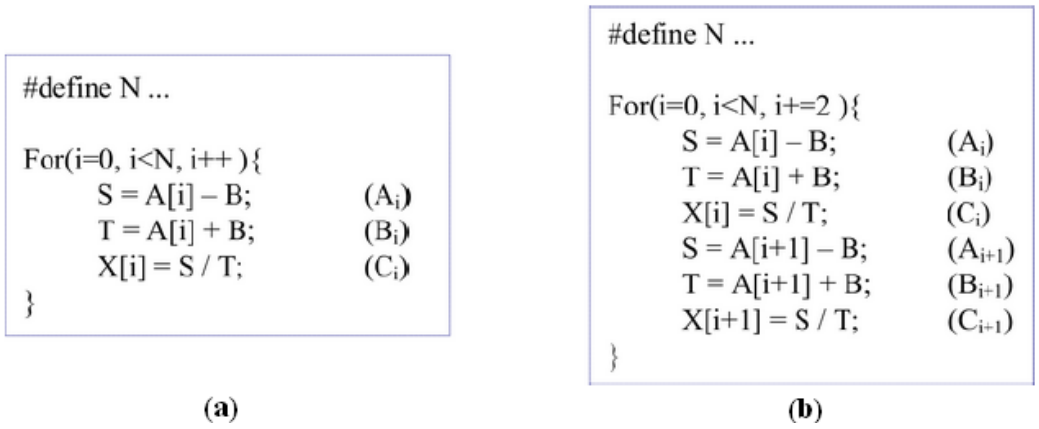

Figure 1-20 – **Exemple d'une boucle (a) non déroulée (b) déroulée partiellement d'un facteur de déroulage de boucle F = 2**. *La boucle partiellement déroulée est représentée par F fois le nombre de nœuds compris dans une itération.*

C.2 ASPECT ARCHITECTURAL

A partir de l'état de l'art des architectures hybrides, selon le couplage entre les ressources matérielles et logicielles, nous remarquons qu'en tendant vers un couplage plus fort, les modèles souffrent moins de la latence des communications et du chargement des données et des contextes de configuration. Par contre, ils tiennent compte de latences de contrôle plus élevées dans l'exécution du programme. Dans les applications qui exigent beaucoup de communications, ceci peut réduire ou enlever tous les avantages d'accélération gagnés par le matériel reconfigurable. Ce qui fait que cela reste dépendant des quantités respectives de communication et de contrôle requis pour l'exécution de l'application.

Le temps de traitement d'une application sur une architecture hybride reconfigurable est la somme des différentes latences dûes : au calcul sur l'ARD, à la reconfiguration de l'ARD, à la communication des données et au contrôle par le CPU du processus de communication des données et des contextes de configuration. L'énergie dissipée dans le traitement d'une application sur une architecture hybride est la somme des énergies dissipées par l'ARD dans le calcul et la reconfiguration, de celle dissipée dans le chargement des données et des contextes de configuration à partir du système mémoire et de celle dissipée par le CPU dans le contrôle de ce processus de chargement.

La latence et l'énergie dissipée dues au calcul de l'ARD dépendent des paramètres liées aux caractéristiques de celle ci à savoir la fréquence de fonctionnement, la granularité

du motif de calcul, le nombre d'unités de calcul, etc. ...
Cette latence de calcul dépend aussi du nombre d'opérations de l'application à exécuter sur l'ARD ainsi que de la contrainte temps réelle gérant cette application. En effet, la manière de *mapper* l'application sur le circuit reconfigurable est étroitement liée à la contrainte temps réel imposée.

La latence et l'énergie dissipée dues à la reconfiguration de l'ARD dépendent des mécanismes de gestion de la reconfiguration dynamique. En effet, selon ces mécanismes liés à la manière dont l'ARD est intégrée dans l'architecture hybride, que le taux de chevauchement entre le chargement d'un contexte de configuration et l'exécution du contexte précédent est plus ou moins important.

La latence et l'énergie dissipée dues à la communication des données sont étroitement liées à la topologie de l'interconnexion entre les ressources reconfigurables, le processeur hôte et les ressources de mémorisation. Par exemple, un accès en mémoire externe (off-chip) est plus coûteux que celui en mémoire interne (on-chip). D'un autre côté, cette latence dépend du volume des données à communiquer entre les différentes tâches et du nombre d'itérations d'exécution de cette tâche, c'est à dire le nombre de fois que ce volume de données doit être communiqué à l'ARD. En outre, les mécanismes mis en place pour le chargement de ces données du système de mémorisation influencent énormément cette latence. A savoir un accès en DMA (*Direct Memory Access*) par exemple diminue sensiblement le temps de chargement à partir du système mémoire.

Enfin, la latence et l'énergie dissipée dues au contrôle par le CPU des différents accès au système de mémorisation dépendent aussi bien de la manière dont la ressource reconfigurable est couplée au processeur hôte, que de la reconfiguration et de l'exécution de l'application sur l'ARD elles mêmes. En effet, le volume du contexte de configuration, le volume des données à exécuter, les taux d'accès en mémoires internes et externes à la puce ont tous un impact sur cette latence de contrôle.

Nous concluons qu'il est difficile de faire une étude qualitative des architectures hybrides puisque les performances de l'architecture dépendent non seulement des caractéristiques architecturales mais aussi des caractéristiques de l'application ainsi que de sa manière de s'exécuter sur la plate-forme hybride reconfigurable.
Il en découle une nécessité de modélisation précise de ces architectures et de caractérisation selon des paramètres aussi bien architecturaux que applicatifs. Cette modélisation est une des contributions de la présente thèse.

D Synthèse

Dans ce chapitre, nous avons présenté les caractéristiques définissant les architectures hybrides reconfigurables. Celles-ci associent un ou plusieurs processeurs à une ou plusieurs architectures reconfigurables.
Afin de répondre à l'ensemble des contraintes inhérentes aux applications de télécommunications actuelles et futures à savoir la flexibilité, la haute performance et la faible consommation d'énergie, la reconfiguration dynamique est essentiellement exploitée au niveau des architectures hybrides reconfigurables.

Nous avons ensuite réalisé un état de l'art de ces architectures selon le critère d'intégration architecturale définissant le couplage entre la ressource logicielle et la ressource matérielle reconfigurable.

A partir de cet état de l'art, nous avons dressé un bilan qualitatif des systèmes hybrides dans lequel nous avons ressorti les paramètres agissant sur les performances de ces architectures. Ces paramètres dépendent non seulement des caractéristiques architecturales mais aussi des caractéristiques de l'application ainsi que de sa manière de s'exécuter sur la plate-forme hybride reconfigurable.

Par ailleurs, la conception des systèmes hybride souffre de l'absence de modèles et de méthodes capables de quantifier les performances de ces architectures et d'exploiter efficacement les possibilités de reconfiguration dynamique qu'offrent les ressources reconfigurables.

Dans le chapitre suivant, nous allons présenter des modèles et des métriques qui permettent au concepteur d'analyser qualitativement les architectures hybrides reconfigurables dynamiquement.

Chapitre

2 Modélisation des architectures hybrides reconfigurables

Sommaire

Dans l'approche de conception des architectures hybrides, l'objectif est de minimiser le temps global d'exécution de l'application en exploitant :
– au mieux les ressources disponibles ;
– la reconfiguration dynamique de la partie matérielle ;
– la possibilité d'exécution concurrente des unités logicielles et matérielles.
La granularité choisie est importante pour pouvoir profiter au mieux de l'accélération apportée par l'accélérateur reconfigurable. L'accélération des calculs sera optimisée conjointement à la réduction de la consommation d'énergie. Aborder le problème de l'optimisation des performances d'une architecture hybride, impose de définir des modèles précis permettant de représenter des comportements réalistes.
Nous présentons dans ce chapitre une modélisation des architectures hybrides reconfigurables ainsi que des modèles d'estimation précis de leurs performances. Nous validons ces modèles sur deux architectures hybrides reconfigurables : GARP [4] et XIRISC [30].

A Modèles considérés

Dans l'objectif d'une modélisation précise de l'interaction entre les ressources logicielles et matérielles des architectures hybrides, il est nécessaire de modéliser l'application qui sera déployée sur celles-ci, de définir les métriques de performances utilisées pour l'analyse des performances des différents couplages CPU/ARD, ainsi que les paramètres agissant sur ces métriques de performance.

A.1 Modélisation de l'application

Les applications considérées (multimédia, télécommunication, réseau ...) présentent outre une grande complexité algorithmique, un très fort degré de parallélisme. En effet, les modèles d'applications offrent plusieurs niveaux de parallélisme qui peuvent être exploités comme présenté dans la figure 2-1.

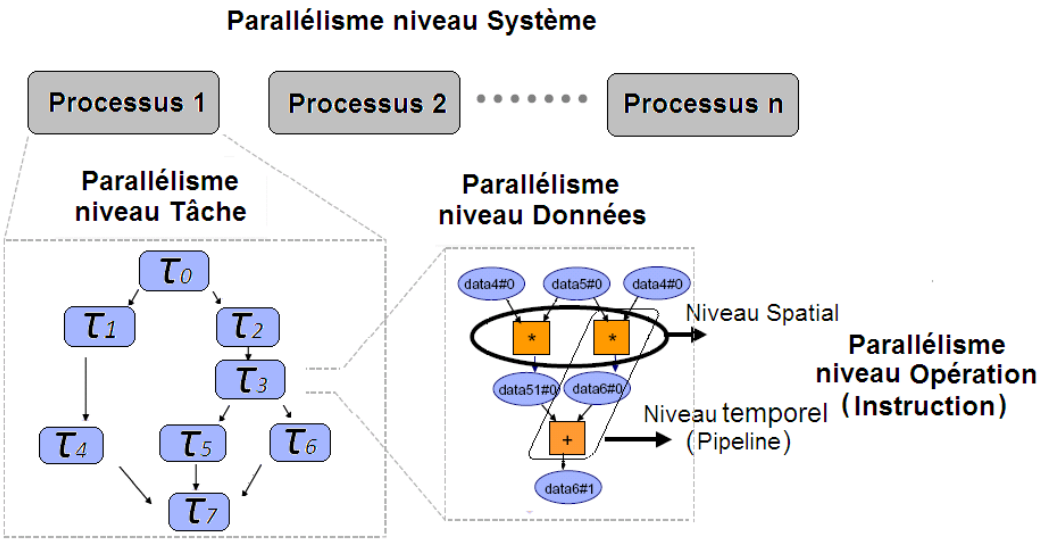

Figure 2-1 – **Niveaux de parallélisme exploitable pour l'optimisation de performances**. *Dans le cas de parallélisme au niveau système, le système d'exploitation répartit des processus que ce soit sur le CPU ou sur les accélérateurs de traitement. Un processus consiste en plusieurs tâches qui peuvent s'exécuter en parallèle, c'est le parallélisme de tâches. Le parallélisme au niveau données repose sur la duplication des unités de calcul UFs afin d'avoir les moyens matériels de prendre en charge les opérations parallèle. Le parallélisme au niveau instruction se manifeste au niveau spatial ou au niveau pipeline.*

Quand le système d'exploitation répartit des processus que ce soit sur le CPU ou sur les accélérateurs de traitement, nous parlons de parallélisme au niveau système, ou encore de parallélisme au niveau processus PLP (*Process Level Parallelism*) ou pour une granularité plus fine, de parallélisme au niveau "*thread*".

Un processus consiste en plusieurs tâches (représentées chacune par un code source) qui peuvent s'exécuter en parallèle. Chaque tâche représente un traitement à réaliser sans connaissance a priori de la cible d'exécution qui peut être des ressources logicielles ou matérielles compte tenu de l'aspect hétérogène de l'architecture cible. À ce niveau nous parlons de parallélisme de tâches TLP (*Task Level Parallelism*).

Le parallélisme au niveau données DLP (*Data Level Parallelism*) repose sur la duplication des unités de calcul UF afin d'avoir les moyens matériels de prendre en charge les opérations parallèles. Mais il est parfois impossible d'avoir autant d'opérations en parallèle que d'UF disponibles, ce qui implique une sous-utilisation des ressources. une solution consiste alors à utiliser un déroulage de boucle afin d'exploiter au mieux le parallélisme potentiel.

Le parallélisme au niveau opération, connu pour les CPU comme le parallélisme niveau instruction ILP (*Instruction Level Parallelism*), se manifeste à deux niveaux :
- *le parallélisme spatial* où deux opérations s'exécutent en même temps et n'ont pas de dépendances de données ;
- *le parallélisme temporel* ou pipeline où deux opérations qui s'exécutent en même temps, ont une dépendance de données mais les opérateurs ne travaillent pas sur les mêmes partitions temporelles.

Dans notre étude, nous considérons que l'application, à traiter sur l'architecture hybride, est modélisée au niveau parallélisme de tâches. Á ce niveau, l'application est découpée en tâches. Le graphe de tâches (GdT) décrit chacune de ces tâches ainsi que les dépendances qui existent entre elles. Une application est ainsi décrite par un graphe orienté acyclique $G = (N, \epsilon)$ où $N = \{\tau_i / i = 1, \ldots, N\}$ est un ensemble de noeuds (représentent les tâches) et $\epsilon = \{E_{i,j} / (i,j) \subset 1, \ldots, N \times 1, \ldots, N\}$ est un ensemble d'arcs définissant les relations de précédence (dépendances de flots ou de données) entre les tâches. On associe à chaque arc $E_{i,j}$, la quantité de données que la tâche τ_i doit transférer à la tâche τ_j. Par définition, une tâche d'entrée du graphe n'a aucun prédécesseur et une tâche de sortie est sans successeur. Une tâche est prête lorsque tous ses prédécesseurs ont terminé leur exécution.
Pour l'implémentation sur l'ARD, à chaque tâche τ_i correspond un ou plusieurs contextes

de reconfiguration. Chaque contexte permet de changer la fonctionnalité de l'ARD pour un traitement donné. Ces contextes sont définis selon les ressources disponibles de l'ARD ou selon un découpage optimisant les performances de celles-ci.

A.2 Paramètres du modèle

Le comportement d'une architecture hybride reconfigurable dépend d'un certain nombre de paramètres. Parmi ces paramètres, nous trouvons ceux caractéristiques de l'application, ceux dépendants de l'architecture et ceux fonction de l'exécution du code.

A.2-1 Paramètres applicatifs

Une application donnée est modélisée par un GdT. Chaque tâche τ_i est caractérisée par les paramètres suivants :
- Nb^i_{op}, le nombre d'opérations arithmétiques et logiques qui doivent être exécutées par la tâche τ_i ;
- ρ^i, le volume de données en octet, à charger depuis les ressources de mémorisation, en entrée de la tâche τ_i ;
- n^i, le nombre d'itérations d'exécution de la tâche τ_i.

Pour les applications de TDSI visées par ce type d'architectures, les normes sont très strictes et un non respect de la contrainte temps réel engendre des conséquences désastreuses sur le signal. Nous définissons T^i_{TR} comme étant la contrainte temps réel relative à la tâche τ_i.
Si la puissance calculatoire d'une architecture est inférieure à la complexité algorithmique de la tâche, son exécution sur cette architecture conduira à une violation de la contrainte temps réel. La complexité algorithmique C^i en MOPS (Mega Operations Per Second) d'une tâche τ_i représente le nombre d'opérations qu'il faut exécuter en une seconde. Elle est calculée en utilisant la formule suivante :

$$C^i = \frac{Nb^i_{op}}{T^i_{TR}} \times 10^{-6}[MOPS] \tag{2-1}$$

A.2-2 Paramètres architecturaux

Les paramètres architecturaux sont les caractéristiques définies par les constructeurs des différentes ressources mis en jeu dans l'architecture hybride.
Les paramètres architecturaux qui caractérisent le CPU sont sa fréquence d'exécution F_{CPU} et son IPC (*Instruction Per Cycle*) définissant le nombre d'instructions pouvant

être exécutées par cycle. Ce paramètre illustre le degré de parallélisme ILP exploité par le CPU. Un autre paramètre architectural du CPU est l'efficacité énergétique du CPU EE_{CPU}. Ce paramètre nous permettra d'estimer la puissance consommée lors de l'exécution d'une tâche sur le CPU en fonction du nombre d'opérations que ce dernier traite par seconde.
Les puissance P_{int} et P_{ext} dissipées lors d'un transfert vers ou depuis respectivement la mémoire interne et la mémoire externe du CPU sont aussi des paramètres architecturaux du CPU.

L'architecture reconfigurable dynamiquement est caractérisée par :
- le nombre d'unités fonctionnelles $N_{UF_{total}}$ composant l'ARD. L'unité fonctionnelle peut être un CLB dans le cas d'une ARD grain fin ou un CFB dans le cas d'une ARD gros grain (se référer au paragraphe A.2) ;
- η, le volume de reconfiguration en octet d'un contexte de configuration à charger depuis les ressources mémoire ;
- la fréquence F_{config} pour laquelle le composant reconfigurable se configure totalement (donnée généralement par le constructeur). La latence de reconfiguration du chemin de données $T^i_{confARD}$ dépend de nature de la reconfiguration dynamique : totale ou partielle. Elle est déduite par :

$$T^i_{confARD} = \frac{N_{UF}}{N_{UF_{total}}.F_{config}} \tag{2-2}$$

  N_{UF} est le nombre d'unités fonctionnelles qu'il faut configurer pour un contexte de reconfiguration donné.
  Dans le cas où une tâche à implémenter sur l'ARD possède plusieurs contextes de configuration, la latence de reconfiguration totale est la somme des latences relatives à la reconfiguration de l'ARD avec chacun des contextes.

Les mémoires (caches, SRAM ...) sont caractérisées par l_{mem} la latence d'un accès à la mémoire. Il s'agit de la latence s'écoulant entre l'envoi de la commande de lecture et l'arrivée effective de la donnée (paquet).
Un autre paramètre caractérisant les mémoires est ε_{mem} la largeur en octet du mot mémorisable par un accès à la mémoire.

Les interconnexions sont caractérisés par leurs latences de communication d'un paquet sur l'interconnexion ($L_{interconnect}$) ainsi que du nombre d'octets par paquet supportés par l'interconnexion (λ_{int}).

A.2-3 Paramètres algorithmiques

Nous définissons les paramètres algorithmiques comme étant les paramètres dont les valeurs évoluent en fonction du code exécuté sur l'architecture hybride. Un profilage de l'application, nous indiquera les paramètres algorithmiques suivants :

- pour un contexte de configuration d'une tâche τ_i, le taux de chevauchement χ^i entre le chargement du contexte, dont la latence est définie par T^i_{CHconf}, et le calcul du contexte précédent.
  Pour une tâche à laquelle correspond plus d'un contexte de configuration, χ^i est la moyenne des taux de chevauchements relatifs aux différents contextes ;
- le taux d'accès en mémoire externe ou taux de défaut de cache (s'il existe)μ^i pour la tâche τ_i. Ce paramètre est algorithmique puisque sa valeur dépend du nombre de fois où des données externes sont accédées durant l'exécution du code. La valeur du taux d'accès en mémoire externe sera donc fonction de l'exécution du code. Cette valeur est calculée en utilisant la formule suivante :

$$\mu^i = \frac{Nb_{AccesExt}}{Nb_{AccesTotal}} \tag{2-3}$$

  avec $Nb_{AccesExt}$ le nombre total d'accès en mémoire externe et $Nb_{AccesTotal}$ le nombre total d'accès au système mémoire ;
- le taux d'accès en DMA α^i pour la tâche τ_i qui est lié au paramètre μ^i. La valeur de ce paramètre est calculée en utilisant la formule suivante :

$$\alpha^i = \frac{Nb_{AccesDMA}}{Nb_{AccesExt}} = \frac{Nb_{AccesDMA}}{\mu^i.Nb_{AccesTotal}} \tag{2-4}$$

  $Nb_{AccesDMA}$ est le nombre d'accès via le DMA ;
- le taux de recouvrement δ^i entre le temps où la tâche τ_i est en train d'être exécutée et T^i_{com} le temps de chargement des données de celle ci. La portion de latence perdue dans la communication des données pour une tâche τ_i est égale à $(1-\delta^i)T^i_{com}$;
- $ch_{I/O}$ la charge du bus d'E/S dépend de la façon avec laquelle les données transitent entre le système mémoire, le CPU et l'ARD.

A.3 Métriques de performances

Nous définissons dans cette sous-section les métriques qui permettent d'évaluer les performances calculatoires et énergétiques de l'architecture hybride reconfigurable.

A.3-1 Accélération apportée par l'architecture hybride

Nous discutons le gain en performances calculatoires en re-visitant la fameuse loi d'Amdhal. En effet, afin d'évaluer les performances de l'architecture hybride en regard de celles du processeur seul, nous définissons la métrique accélération comme l'accélération du traitement apportée par l'architecture hybride par rapport à l'exécution du calcul par le CPU seul. Pour une tâche τ_i, c'est le rapport entre la latence de traitement sur le CPU $T^i_{execCPU}$ et la latence de traitement sur l'architecture hybride.

La figure 2-2 montre les diagrammes temporels pour l'ARD et l'interconnexion dans le cas d'une reconfiguration dynamique totale et partielle pour un contexte de reconfiguration donné.

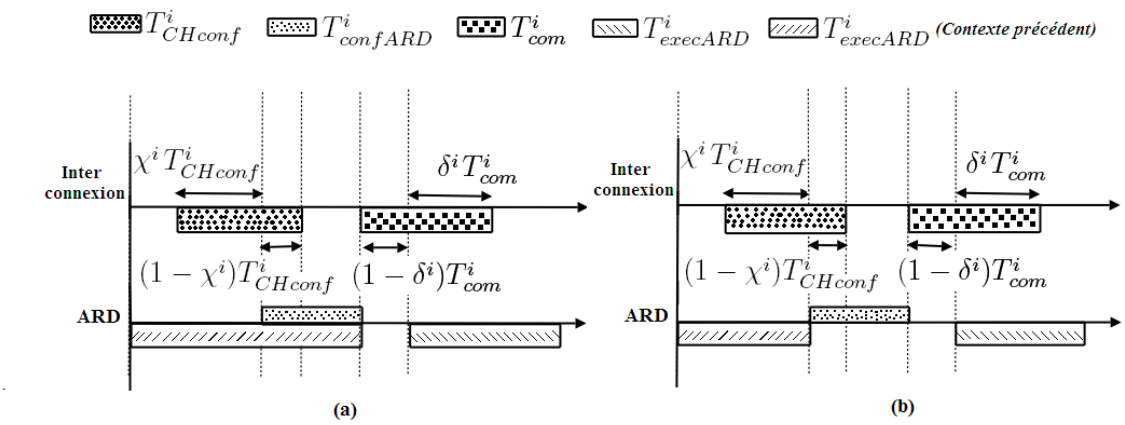

Figure 2-2 – **Diagrammes temporels dans le cas d'une reconfiguration dynamique partielle (a) et totale (b) pour un contexte de reconfiguration donné.** *Selon le principe de la reconfiguration dynamique, la phase de chargement du contexte de reconfiguration, dont la latence est T^i_{CHconf}, commence avant que le contexte précédent ne termine son exécution. Dans le cas de la reconfiguration dynamique partielle, il y a un recouvrement entre les phases d'exécution $T^i_{execARD}$(du contexte précédent) et de reconfiguration $T^i_{confARD}$. Cependant, ceci n'est pas le cas pour la reconfiguration totale. La phase de chargement de données de la tâche τ_i dont la latence est T^i_{com} ne doit commencer que si la phase reconfiguration est achevée.*

La phase de chargement du contexte de reconfiguration, dont la latence est T^i_{CHconf}, commence avant que le contexte précédent ne termine son exécution. C'est le principe même de la reconfiguration dynamique. Pour le premier contexte de reconfiguration de la tâche, $\chi^i T^i_{CHconf}$ représente une latence perdue.
Selon le type de reconfiguration dynamique correspondante à l'ARD, la latence de

reconfiguration affecte plus ou moins la performance de l'architecture hybride. En effet, dans le cas de la reconfiguration dynamique partielle, les cycles perdus à chaque reconfiguration de l'ARD ne nuisent pas à la performance puisqu'il y a un recouvrement entre les phases d'exécution (du contexte précédent) et de reconfiguration. Cependant, ceci n'est pas le cas pour la reconfigurations totale. La phase de chargement de données de la tâche τ_i dont la latence est T^i_{com} ne doit commencer que si la phase reconfiguration est achevée.

La latence de traitement totale sur l'architecture hybride $T^i_{totalHybride}$ est la somme entre la latence d'exécution de la tâche sur l'ARD $T^i_{execARD}$ et les latences perdues dans :

- la reconfiguration de l'ARD $T^i_{confARD}$ si la reconfigutation dynamique est totale ;
- le chargement du premier contexte de configuration. Cette latence est égale à $\chi^i T^i_{CHconf}$ $(1^{er}\ contexte)$;
- le chargement de données. Cette latence est égale à $(1-\delta^i)T^i_{com}$;
- le contrôle du processus de chargement des contextes de reconfiguration et des données. Cette latence est référencée par $T^i_{controlCPU}$.

L'expression de la latence de traitement totale sur l'architecture hybride s'écrit alors :

$$T^i_{totalHybride} = T^i_{controlCPU} + T^i_{execARD} + \theta T^i_{confARD} + \chi^i T^i_{CHconf}(1^{er}\ contexte) + (1-\delta^i)T^i_{com} \tag{2-5}$$

θ est un paramètre qui exprime le mode de reconfiguration dynamique : si totale $\theta = 1$, sinon si partielle $\theta = 0$.

Cette latence de traitement totale de l'architecture hybride reconfigurable diffère d'un modèle de couplage CPU/ARD à un autre. Cette différence réside dans les valeurs de latence des processus de transfert de données et du premier contexte de reconfiguration depuis ou vers le système mémoire à l'ARD ainsi que dans la latence perdue par le CPU dans le contrôle de ces processus.

L'accélération du calcul apportée par l'architecture hybride reconfigurable par rapport à l'exécution du calcul par le CPU seul A^i s'exprime par l'équation suivante :

$$A^i = \frac{T^i_{execCPU}}{T^i_{totalHybride}} \tag{2-6}$$

Soit $Taux_{cycles\_perdus}$, le pourcentage de cycles perdus (dans le chargement des données, la configuration de l'ARD et le contrôle de ces deux processus par le CPU) par rapport au nombre de cycles total de traitement sur l'architecture hybride reconfigurable et est

défini par :

$$Taux_{cycles\_perdus} = \frac{T^i_{controlCPU} + \theta T^i_{confARD} + \chi^i T^i_{CHconf}(1^{er}\ contexte) + (1-\delta^i)T^i_{com}}{T^i_{totalHybride}} \tag{2-7}$$

L'accélération apportée par l'architecture hybride peut alors être simplifiée à la formule suivante :

$$A^i = \frac{T^i_{execCPU}}{T^i_{execARD}}(1 - Taux_{cycles\_perdus}) \tag{2-8}$$

Le rapport $\frac{T^i_{execCPU}}{T^i_{execARD}}$ dans l'expression de A^i représente le gain en accélération apporté par l'exécution de l'application sur l'ARD par rapport à l'exécution sur le CPU sans prise en compte des cycles perdus dans le contrôle, la reconfiguration et le transfert des données et des contextes de reconfiguration. Nous remarquons que l'accélération a une valeur maximale (de saturation) qui correspond à cette valeur du gain pour un $Taux_{cycles\_perdus}$ tendant vers 0.
Ce gain est le même pour tous les types de couplage entre le CPU et l'ARD. Ainsi, $Taux_{cycles\_perdus}$ permet de caractériser le couplage le plus adéquat entre le CPU et l'ARD pour une classe d'applications donnée.

A.3-2 Taux de réduction de la puissance dissipée et efficacité énergétique

Puissance dissipée par une architecture hybride
Nous définissons par l'équation 2-9 la puissance dissipée pour l'exécution d'une tâche τ_i dans une architecture hybride reconfigurable par la somme des puissances $P^i_{controlCPU}$ et P^i_{ARD} dissipées respectivement dans le contrôle par le CPU et le calcul par l'ARD de la tâche τ_i et $P^i_{transfert}$ celle dissipée dans le transfert des données et des contextes de re-configuration.

$$P^i_{architecture\_hybride} = P^i_{controlCPU} + P^i_{ARD} + P^i_{transfert} \tag{2-9}$$

Cette puissance totale dissipée dans le traitement par l'architecture hybride reconfigurable diffère d'un modèle de couplage CPU/ARD à un autre. Cette différence réside dans les valeurs de puissance dissipée par le processus de transfert de données et des contextes de reconfiguration depuis ou vers le système mémoire à l'ARD, ainsi que dans la puissance perdue par le CPU dans le contrôle de ces processus.

Taux de réduction de la puissance dissipée
Nous exprimons le gain potentiel en performances énergétiques par rapport à celles

du CPU seul par $Taux_{Red\_E}$, le taux de réduction de la puissance dissipée par une architecture hybride pour l'exécution de la tâche τ_i par rapport à celle dissipée dans l'exécution de la tâche par le CPU seul. Il est définit par la formule suivante :

$$Taux^{i}_{Red\_E} = \frac{P^{i}_{CPU} - P^{i}_{architecture\_hybride}}{P^{i}_{CPU}} \tag{2-10}$$

P^{i}_{CPU} est la puissance dissipée lors de l'exécution de la tâche τ_i par le CPU seul.

Efficacité énergétique d'une architecture hybride
L'équation 2-11 définit l'efficacité énergétique d'une architecture hybride pour une tâche τ_i avec une complexité algorithmique C^i exprimée en MOPS et une puissance dissipée $P^{i}_{architecture\_hybride}$.

$$EE = \frac{C^i}{P^{i}_{architecture\_hybride}} \left[\frac{MOPS}{W}\right] \tag{2-11}$$

La latence totale et la puissance dissipée totale dans le traitement sur une architecture hybride différent d'un mode couplage CPU/ARD à un autre par les valeurs de latence et de puissance dissipée dans processus de transfert de données et des contextes de reconfiguration depuis ou vers le système mémoire à l'ARD ainsi que celles de latence et puissance perdue par le CPU dans le contrôle de ces processus.
Afin de pouvoir quantifier la latence et la puissance dissipée dans communication des données et des contextes de reconfiguration, nous allons définir des modèles de communication généralistes qui peuvent être appliqués à toutes les architectures hybrides présentant une interaction entre le processeur et l'architecture reconfigurable dynamiquement.

B Modélisation de la communication dans une architecture hybride

Dans cette section, nous expliquons les différentes phases de la construction du modèle de communication processeur-ARD. Nous définissons en premier lieu des lois générales permettant de quantifier la latence et la dissipation d'énergie due aux transferts entre les différents éléments du modèle général. En deuxième lieu, nous spécifions les différentes lois de communications pour chaque modèle de couplage CPU/ARD et étudions l'impact des différents paramètres applicatifs sur les performances de chaque modèle.

B.1 Lois de communication

Afin de pouvoir caractériser les architectures hybrides reconfigurables, nous avons défini des lois de communication qui permettent de quantifier les latences et la dissipation d'énergie dues aux transferts de données et de contextes de configuration pour une application donnée.
Les lois de communication sont des fonctions mathématiques utilisant les différents paramètres applicatifs, architecturaux et algorithmiques définies dans ce chapitre.

Tout au long de notre étude, nous allons prendre en compte l'hypothèse suivante : *il n'y a pas de recouvrement entre traitement et communication sur le CPU. Par contre, ceci ne s'applique pas pour l'ARD. C'est à dire que les traitements effectués sur une partie d'un contexte peuvent se faire en parallèle avec les communications effectuées par une autre partie du contexte.*

B.1-1 Latence de communication

Soit une tâche τ_i de l'application et une tâche τ_j lui étant directement successive et dépendante. Les données calculées par la tâche τ_i doivent être sauvegardées dans la mémoire puis transférées de la mémoire vers la tâche consommatrice.

Nous définissons γ^i_{com} la latence de communication d'un octet vers le système mémoire (ou depuis le système mémoire) pour une tâche τ_i. Elle est définie par une somme des latences respectives dues aux communications d'un octet vers les mémoires internes (*on chip*) et externes (*off chip*) :

$$\gamma^i_{com} = \gamma^i_{com_{int}} + \gamma^i_{com_{ext}} \tag{2-12}$$

La latence de communication depuis ou vers la mémoire interne du CPU $\gamma^i_{com_{int}}$ est calculée par la somme de la latence de l'acheminement d'un octet sur le bus interne et la latence d'accès à la mémoire interne :

$$\gamma^i_{com_{int}} = (1 - \mu^i)(\frac{L_{int}}{\lambda_{int}} + \frac{l_{int}}{\varepsilon_{int}}) \tag{2-13}$$

μ^i est un paramètre algorithmique définissant le taux d'accès en mémoire externe ou taux de défaut de cache pour la tâche τ_i.
L_{int} représente la latence de communication d'un paquet transitant sur les bus internes

du CPU et λ_{int} représente le nombre d'octets supportés par un paquet sur ce bus. l_{int} représente la latence d'accès à la mémoire interne et ε_{int} est la largeur en octet du mot mémorisable sur cette dernière.

Pour le modèle d'architecture hybride que nous avons considéré, deux ressources de mémorisation externes sont à considérer : la mémoire principale externe au processeur et les ressources mémoires spécifiques qui peuvent être rajoutées, dans le cas d'un couplage point-à-point entre le processeur et l'ARD à travers une interface ou bus spécifique.

L'ARD accède à la mémoire principale externe à travers le bus I/O. Dans la plupart des architectures des CPU actuels, l'accès à cette mémoire est en DMA. La latence d'accès à cette mémoire est alors égale à $\frac{\alpha^i l_{extDMA}+(1-\alpha^i)l_{ext}}{\varepsilon_{ext}}$ avec α^i le taux d'accès en DMA. l_{extDMA} et l_{ext} représentent respectivement les latences d'un accès à la mémoire externe avec et sans DMA.

La latence de transfert sur le bus périphérique dépend de l'arbitrage du bus entre les différents périphériques et donc du taux de charge $ch_{I/O}$ du bus. En effet, le bus I/O constitue une ressource partagée, ce qui fait qu'un haut degré de contention pour le bus réduit l'efficacité de la largeur de bande disponible pour tous les périphériques sur le bus. Un bus SCSI doté de plusieurs disques durs très actifs illustre bien cette situation. Les disques durs très actifs saturent le bus SCSI, ne laissant disponible qu'une largeur de bande très faible pour tout autre périphérique sur le même bus. En conséquence, toutes les activités d'E/S vers tout autre périphérique seront lentes, et ce, même si aucun périphérique du bus n'est extrêmement actif.
La latence de transfert sur le bus I/O est alors définie par $\frac{1}{ch_{I/O}}\frac{L_{I/O}}{\lambda_{I/O}}$. $L_{I/O}$ est la latence de communication d'un paquet transitant sur les bus I/O et $\lambda_{I/O}$ représente le nombre d'octets supportés par un paquet sur ce bus.

La latence d'accès d'un octet aux ressources mémoires spécifiques externes est $\frac{l_{spec}}{\varepsilon_{spec}}$. l_{spec} représente la latence d'un accès du mot mémorisable composé de ε_{spec} octets. Quelques architectures hybrides en modèle point-à-point assurent l'accès de l'ARD à la mémoire principale du processeur à travers l'interface spécifique. La latence de transfert d'un paquet sur cet interface est L_{spec} et la taille du paquet transitant sur ce bus est de λ_{spec}.

La latence d'acheminement d'un octet depuis ou vers les mémoires externes s'écrit

alors :

$$\gamma^i_{com_{ext}} = \mu^i \left(\beta \frac{1}{ch_{I/O}} \frac{L_{I/O}}{\lambda_{I/O}} + (1-\beta)\frac{L_{spec}}{\lambda_{spec}} + \sigma \frac{\alpha^i l_{extDMA} + (1-\alpha^i) l_{ext}}{\varepsilon_{ext}} + \Omega \frac{l_{spec}}{\varepsilon_{spec}} \right) \tag{2-14}$$

β vaut 1 si l'accès à la mémoire externe du CPU se fait à travers le bus périphérique et 0 si à travers l'interfaceou bus spécifique.
σ vaut 1 si l'ARD utilise la mémoire externe au CPU pour le chargement des données et éventuellement les contextes de configuration et 0 sinon.
Ω vaut 1 si l'ARD utilise des ressources mémoires propres à l'interface spécifique d'interconnexion et 0 sinon.

T^i_{com}, la latence de transfert des données est alors calculée en multipliant ρ^i le volume de données de données en octet, à charger depuis les ressources de mémorisation, en entrée de la tâche τ_i par γ^i_{com}.

$$T^i_{com} = \rho^i \gamma^i_{com} \tag{2-15}$$

T^i_{CHconf}, la latence de chargement du contexte de reconfiguration, de taille η, correspondant à la tâche τ_i est définie par :

$$T^i_{CHconf} = \eta \left(a \frac{l_{conf}}{\varepsilon_{conf}} + b\gamma^i_{com} \right) \tag{2-16}$$

l_{conf} représente la latence d'un accès à un cache de configuration et ε_{conf} est la largeur en octet du mot mémorisable par accès à ce cache. $a = 1$ et $b = 0$ si les contextes sont chargés depuis ce cache de configuration. Les contextes de configuration peuvent aussi être chargés depuis la hiérarchie mémoire mais avec un retard de transfert et dans ce cas $a = 0$ et $b = 1$.

B.1-2 Dissipation de puissance due aux communications

L'estimation de la consommation d'énergie pour les architectures hybrides passe nécessairement par l'estimation de l'énergie dissipée lors des phases de calcul sur le CPU et l'ARD ainsi que celle dissipée lors de leur interaction.
Dans cette partie, nous modélisons la consommation dissipée pour la communication des données et des contextes de reconfiguration. Celle ci dépend étroitement de la position de la mémoire. En effet, placer une mémoire sur le même substrat silicium que la partie opérative (on chip) limite de façon importante le courant à fournir pour réaliser cet accès [46]. Par contre, si la mémoire est externe, l'accès devra alors être réalisé au travers d'un buffer de sortie, dans ce cas une amplification devra être réalisée et la

consommation deviendra importante.
La puissance dissipée dans le transfert des données et des contextes de re-configuration $P^i_{transfert}$ d'une tâche τ_i s'écrit :

$$P^i_{transfert} = N^i_{acces} P^i_{com} \tag{2-17}$$

N^i_{acces} représente le nombre d'accès à la hiérarchie mémoire pour la tâche τ_i.
P^i_{com}, la puissance dissipée pour la communication lors d'un seul transfert depuis ou vers le système mémoire est définie par l'équation 2-18.

$$P^i_{Com} = (1 - \mu^i) P_{int} + \mu^i P_{ext} \tag{2-18}$$

μ^i est le taux d'accès mémoire externe ou taux de défaut de cache. P_{int} est la puissance dissipée lors d'un transfert vers la mémoire interne et qui n'est autre que la puissance consommée par la mémoire interne dite P_{memInt}. En effet, nous négligeons la puissance consommée par le transfert à l'intérieur de la puce puis qu'elle est très réduite par rapport à P_{memInt}. P_{ext} est la puissance dissipée lors d'un transfert vers la mémoire externe (off chip).
Á partir de cette équation, nous notons qu'en limitant le taux d'accès en mémoire externe μ^i (paramètre algorithmique), la puissance dissipée par le transfert des données de la mémoire diminue.
En nous appuyant sur les phases de consommation lors d'un accès entre un processeur et sa mémoire externe spécifiés dans [47], nous présentons une modélisation des consommations mis en jeu par les architectures hybrides lors d'un accès à une mémoire externe.

En effet, si l'ARD est intégrée dans le chemin de donnée du processeur. L'architecture hybride (ou processeur reconfigurable) accède à la mémoire externe de la même façon qu'un accès fait par le CPU seul. Pour les autres cas d'intéraction processeur-ARD, deux cas de figures sont possibles :
– le CPU initie les accès mémoires de l'ARD à la mémoire externe en produisant l'adresse de la donnée à accéder (Fig.2-3 (a)) puis la donnée est transférée à l'ARD ;
– l'ARD initie lui même ces accès à la mémoire externe (Fig.2-3 (b)). C'est le cas des couplages en périphérique et en point-à-point.

Chaque accès à la mémoire externe comporte les phases de consommation suivantes :

1. production de l'adresse (P@) de la donnée à accéder soit par le CPU soit par l'ARD selon le type d'interaction entre le CPU et l'ARD, la consommation qui découle de cette phase dépend beaucoup du générateur d'adresses mis en place ;

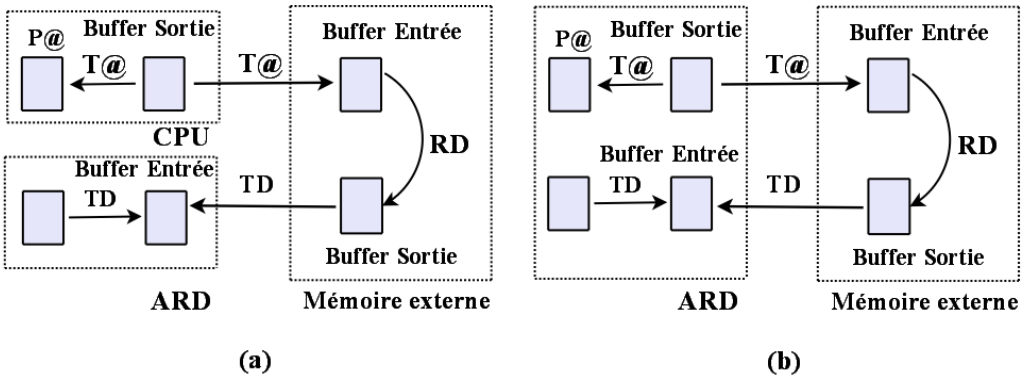

Figure 2-3 – **Modélisation de la consommation des architectures hybrides lors d'un accès à une mémoire externe**. *(a) le CPU initie les accès mémoires de l'ARD à la mémoire externe en produisant l'adresse de la donnée à accéder. (b) L'ARD initie lui même ces accès à la mémoire externe.*

2. transport de l'adresse (T@) jusqu'au buffer de sortie ;
3. passage au travers du buffer de sortie, la puissance du buffer dépend des caractéristiques du bus d'adresses (notamment sa longueur) ;
4. transport de l'adresse sur le bus ;
5. passage au travers du buffer d'entrée de la mémoire, la taille de la mémoire définit de manière logarithmique la complexité du circuit de décodage et donc sa consommation ;
6. recherche de la donnée, décodage (RD) ;
7. passage au travers du buffer de sortie de la mémoire ;
8. transport de la donnée sur le bus de données (TD) ;
9. passage au travers du buffer d'entrée de l'ARD ;
10. transport de la donnée vers sa destination (TD).

Évidemment, la consommation d'un accès mémoire dépend largement des tailles des différents bus mis en jeu : bus de données et bus d'adresses.

Nous déduisons que la puissance consommée par l'architecture hybride lors d'un accès en mémoire externe est spécifiée par l'équation 2-19. $P^{I/O}_{CPU}$, $P^{I/O}_{memExt}$ et $P^{I/O}_{ARD}$ représentent les puissances dissipées par les buffers d'entrées/sortie de chaque composant. $P_{interconnexion}$ est la puissance dissipée par l'interconnexion liant l'architecture hybride à la mémoire externe et P_{memExt} est celle dissipée par la mémoire externe.

$$P_{Ext} = \varphi P^{I/O}_{CPU} + P^{I/O}_{memExt} + \psi P^{I/O}_{ARD} + P_{interconnexion} + P_{memExt} \quad (2\text{-}19)$$

φ et ψ prennent la valeur 1 si le CPU initie l'accès en mémoire externe de l'ARD et prennent respectivement les valeurs 0 et 2 si l'ARD initie lui même son accès à la mémoire externe.

Dans la sous section suivante, nous exprimerons les lois de communication pour les différents modèles de couplage CPU/ARD.

B.2 Application des lois de communication aux architectures hybrides reconfigurables

Au niveau de l'organisation générale de l'architecture hybride, le concepteur doit construire le réseau d'interconnexion entre les différents composants et dimensionner les liens. Les lois de communication définies ci-dessus permettent de quantifier les délais perdus dans les transferts des données et des contextes de configuration. Ce qui permet au concepteur de choisir le modèle de couplage qui optimise les performances de l'architecture.

B.2-1 Unité Fonctionnelle Reconfigurable Dynamiquement

Le couplage en UFRD (Unité Fonctionnelle Reconfigurable Dynamiquement) permet une haute intégration architecturale (Fig.2-4). L'UFRD reçoit les données, comme une unité fonctionnelle du CPU à travers la file de registre. Les CPUs VLIWs et superscalaires sont prédisposés pour ce type de couplage puisque ils possèdent déjà plusieurs UFs.
Le transfert de données de la mémoire vers l'UFRD ou vice versa n'est autre que le transfert entre la mémoire et le CPU.

Comme l'interface externe spécifique ne rentre pas en jeu dans l'interaction entre le CPU et l'ARD pour ce mode de couplage, β vaut donc 1, Ω vaut 0 et σ vaut 1. Ainsi pour le modèle de couplage en UFRD, la latence de communication d'un octet à la hiérarchie mémoire se résume à :

$$\begin{aligned} \gamma^i_{com} &= (1-\mu^i)\left(\frac{L_{sys}}{\lambda_{sys}} + \frac{l_{int}}{\varepsilon_{int}}\right) \\ &+ \mu^i\left(\frac{1}{ch_{I/O}}\frac{L_{I/O}}{\lambda_{I/O}} + \frac{\alpha^i l_{extDMA} + (1-\alpha^i) l_{ext}}{\varepsilon_{ext}}\right) \end{aligned} \quad (2\text{-}20)$$

Pour ce mode de couplage, il est trivial d'utiliser un cache de configuration. L'UFRD charge les contextes de reconfiguration directement à partir de la mémoire interne du

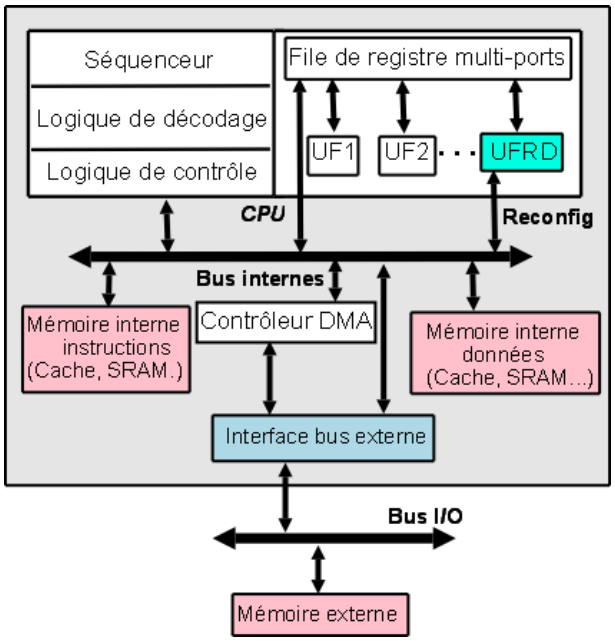

Figure 2-4 – **ARD couplée en unité fonctionnelle reconfigurable dynamiquement au CPU**. *L'UFRD reçoit les données, comme une unité fonctionnelle du CPU à travers la file de registre et il charge les contextes de reconfiguration directement à partir de la mémoire interne du CPU. Pour les applications à gros volume de données à transférer entre les tâches successives, ce modèle de couplage perd son avantage consistant à limiter les latences de transfert des données. En même temps, comme le CPU gère les accès mémoire de l'ARD ainsi que le processus de reconfiguration, la latence et la puissance dissipée dues à ce contrôle deviennent prépondérantes et font chuter les performances de l'architectures hybride.*

CPU. la latence de chargement du premier contexte pour une tâche τ_i à exécuter sur l'UFRD vaut :

$$T^i_{CHconf} = \eta\gamma^i_{com} \tag{2-21}$$

Dans la limite du nombre de registres du CPU, le chargement de données nécessaires à l'exécution d'une tâche τ_i se fait à travers les registres du CPU. Ceci diminue considérablement la latence de transfert de données T^i_{com} qui pour ce type de couplage devient négligeable devant la latence de reconfiguration et la latence due au contrôle du CPU. En effet, pour ce mode de couplage, le jeu d'instructions du processeur est étendu afin de contrôler le processus de chargement des contextes de reconfiguration et des transferts de données depuis la hiérarchie mémoire vers sa file de registres. Ainsi, le nombre de cycles perdus dûs au contrôle du CPU occupe ainsi une proportion assez large dans le nombre de cycles perdus total. Cette latence due au contrôle du CPU ($T^i_{controlCPU}$) a tendance à augmenter avec le volume de données à charger depuis la hiérarchie mémoire.

Comme l'UFRD est intégrée au CPU, l'accès en mémoire externe dissipe la même puissance qu'un accès fait par un CPU seul. La loi de dissipation de puissance due à la communication s'écrit alors :

$$P^i_{Com} = (1 - \mu^i)P_{int} + \mu^i(2P^{I/O}_{CPU} + P_{bus\_I/O} + P_{memExt}) \quad (2\text{-}22)$$

La puissance dissipée par le CPU dans la gestion de l'exécution sur l'ARD $P^i_{controlCPU}$ devient prépondérante sur P^i_{Com} si de gros volumes de données doivent être transférés entre les tâches ce qui nuit considérablement aux performances énergétiques de l'architecture hybride.

B.2-2 Coprocesseur Reconfigurable Dynamiquement

Le CRD (Coprocesseur Reconfigurable Dynamiquement) est connecté aux bus internes du CPU, β vaut donc 1, Ω vaut 0 et σ vaut 1. Le bus interne peut être soit un bus système ou un bus spécifique sur puce tel que AMBA de ARM ou CoreConnect d'IBM. Dans le cas d'un bus spécifique sur puce, le CRD, tout comme les autres composants sur la puce, doit communiquer à travers un adaptateur "wrapper" qui permettra d'adapter le CRD au protocole de communication du bus (Fig.2-5).
Comme les performances du bus interne, en termes de bande passante, dépendent en grande partie de l'arbitrage d'accès au bus, ce couplage utilise pour les transferts de données depuis la hiérarchie mémoire, des instructions de type "wait for interruption" qui mettent le CPU en suspension jusqu'à ce que l'interruption appropriée arrive du CRD indiquant que l'opération de chargement de données du CRD est achevée. Ceci permet au CRD d'utiliser l'intégralité de la bande passante nécessaire du bus interne $\frac{L_{sys}}{\lambda_{sys}}$ en minimisant le trafic CPU.
L'expression de la latence de communication d'un octet sur la hiérarchie mémoire γ^i_{com} pour le couplage CRD pour une tâche τ_i est la même que pour le couplage en UFRD. Cependant, la latence de communication des données est beaucoup plus importante pour le modèle CRD puisque le circuit reconfigurable doit charger tout le volume de données ρ^i à partir de la hiérarchie mémoire.

Un chargement simultané des données nécessaires à l'exécution des tâches et des contextes de reconfiguration, peut causer un goulot d'étranglement au bus interne. Le CRD doit disposer alors de sa propre mémoire de configuration (cache, SRAM...) afin de limiter l'accès à la hiérarchie mémoire au cours du chargement de la configuration. Le CPU commande le chargement sur le CRD des contextes de configuration à partir de la mémoire de configuration en lui référençant les adresses source et destina-

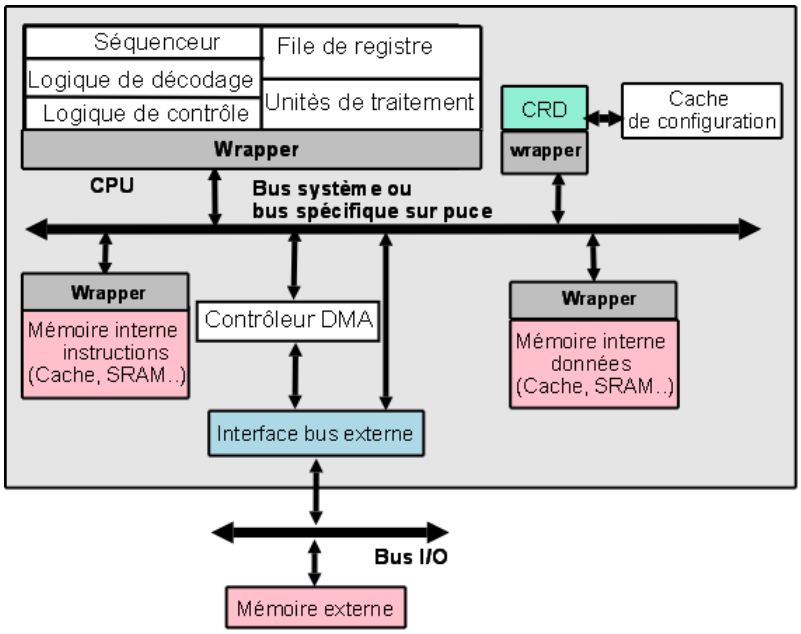

Figure 2-5 – **ARD couplée en coprocesseur reconfigurable dynamiquement au CPU**. *Le CRD est connecté aux bus internes du CPU. Le bus interne peut être soit un bus système soit un bus spécifique sur puce. Dans le cas d'un bus spécifique sur puce, le CRD, tout comme les autres composants sur la puce, doit communiquer à travers un adaptateur "wrapper" qui permettra d'adapter le CRD au protocole de communication du bus.*

tion ainsi que la taille des blocs mémoires. Ainsi la latence de chargement d'un contexte (exprimée par T_{CHconf} ne dépend que des caractéristiques de ce cache de configuration et de la taille des contextes.

$$T_{CHconf} = \eta \frac{l_{conf}}{\varepsilon_{conf}} \tag{2-23}$$

Pour le modèle de couplage en CRD, le CPU initie tous les accès mémoire pour le transfert de données à l'ARD en produisant l'adresse de la donnée à accéder. Ainsi, l'expression de la loi de dissipation de la puissance due au transfert des données pour un accès à la hiérarchie mémoire est :

$$P^i_{Com} = (1-\mu^i)P_{int} + \mu^i(P^{I/O}_{CPU} + P^{I/O}_{memExt} + P^{I/O}_{ARD} + P_{bus_{I/O}} + P_{memExt}) \tag{2-24}$$

Le deuxième terme de cette équation correspond à la puissance dissipée pour un accès en mémoire externe définie à l'équation 2-19 avec φ et ψ prenant la valeur 1.

Le processeur étant l'initiateur des transactions pour le transfert des données et des configurations, dissipe de la puissance dans ce processus de contrôle ($P^i_{controlCPU}$). Pour une tâche τ^i, cette dissipation est augmente avec le nombre d'accès mémoire.

B.2-3 Périphérique Reconfigurable Dynamiquement

Le PRD (Périphérique Reconfigurable Dynamiquement) est connecté "off chip" au bus externe (fig.2-6), β vaut donc 1, Ω vaut 0 et σ vaut 1. Il charge ses données et ses contextes de configuration directement de la mémoire externe. Ainsi le taux d'accès à la mémoire externe μ^i est de 100%.

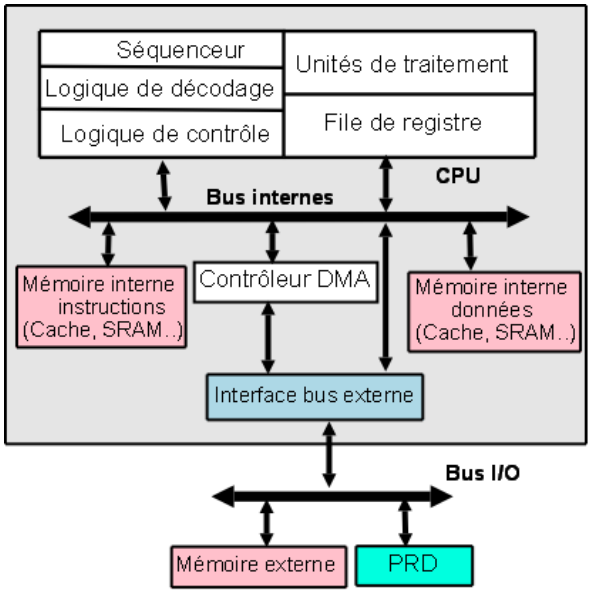

Figure 2-6 – **Architecture reconfigurable dynamiqument couplée en périphérique reconfigurable dynamiquement au CPU**. *Le PRD charge ses données et ses contextes de configuration directement de la mémoire externe. Le CPU communique avec le PRD aux travers d'instructions d'entrées-sorties pour le transfert des contextes de configuration sur l'ARD. Une fois configuré, le PRD devient maître du bus externe et l'utilise afin d'accéder à la mémoire principale partagée entre le processeur et le périphérique. La synchronisation se fait au moyen d'interruptions matérielles.*

Le CPU communique avec le PRD aux travers d'instructions d'entrée-sortie pour le transfert des contextes de configuration sur l'ARD. Une fois configuré, le PRD devient maître du bus externe et l'utilise afin d'accéder à la mémoire principale partagée entre le processeur et le périphérique. La synchronisation se fait au moyen d'interruptions matérielles.
L'utilisation du bus d'entrée-sortie limite très fortement la performance des communications entre l'hôte et le PRD. En effet, les autres périphériques consomment une part de la bande passante disponible du bus externe, et de plus, les arbitrages entre périphériques, mémoire cache, et processeurs induisent des latences d'accès importantes, qui réduisent d'autant plus la bande passante disponible.

Pour le modèle de couplage en PRD, la latence de communication γ^i_{com} pour une tâche τ_i se résume à :

$$\gamma^i_{com} = \frac{1}{ch_{I/O}} \frac{L_{I/O}}{\lambda_{I/O}} + \frac{\alpha^i l_{extDMA} + (1-\alpha^i) l_{ext}}{\varepsilon_{ext}} \quad (2\text{-}25)$$

Comme les chargements des contextes de reconfiguration et des données sont réalisés entièrement depuis la mémoire externe, les latences de ces chargements sont importantes, tout comme la dissipation d'énergie dues à ces transferts. Cependant, ce modèle de couplage en PRD présente l'avantage que les accès mémoires ne sont pas sous le contrôle du CPU. Ce qui fait que la latence due au contrôle du CPU ($T^i_{controlCPU}$) est faible et que dans l'expression de l'équation 2-19 $\varphi = 0$ et $\psi = 0$. Ainsi, la dissipation d'énergie due à un accès à la mémoire n'est autre que la puissance d'un accès de l'ARD à la mémoire externe :

$$P^i_{Com} = P^{I/O}_{memExt} + 2P^{I/O}_{ARD} + P_{bus_{I/O}} + P_{memExt} \quad (2\text{-}26)$$

Bien que la puissance dissipée due au chargement pour ce mode de couplage est importante $P^i_{transfert} = N^i_{acces} P^i_{com}$, elle est plus faible que pour les autres couplages car la puissance dissipée par le CPU dans le contrôle des accès mémoires est presque nulle. Cette proportion devient très avantageuse pour les applications avec un volume important de données à charger depuis le système mémoire.

B.2-4 Couplage Point-à-Point Reconfigurable Dynamiquement

Dans le modèle point-à-point reconfigurable dynamiquement (PPRD), l'ARD est connectée "off chip" au CPU à travers une interface ou un bus spécifique (figure 2-7) qui a pour rôle de faire communiquer de façon transparente le CPU et le PPRD (d'où la nomination "point-à-point" du modèle) et de gérer intelligemment les processus de reconfiguration, β vaut donc 0.

L'interface spécifique peut intégrer ses propres ressources mémoires, auxquelles peuvent accéder le CPU et l'ARD, ou assurer la communication entre le CPU et le PPRD à travers la mémoire externe de la puce.
Ce modèle permet de s'affranchir de la latence élevée causée par le bus I/O et son arbitrage. Il permet également de connecter l'ARD off-chip et ainsi de s'affranchir de la supervision du CPU.
Pour le modèle de couplage en PPRD, la latence d'un accès au système mémoire se

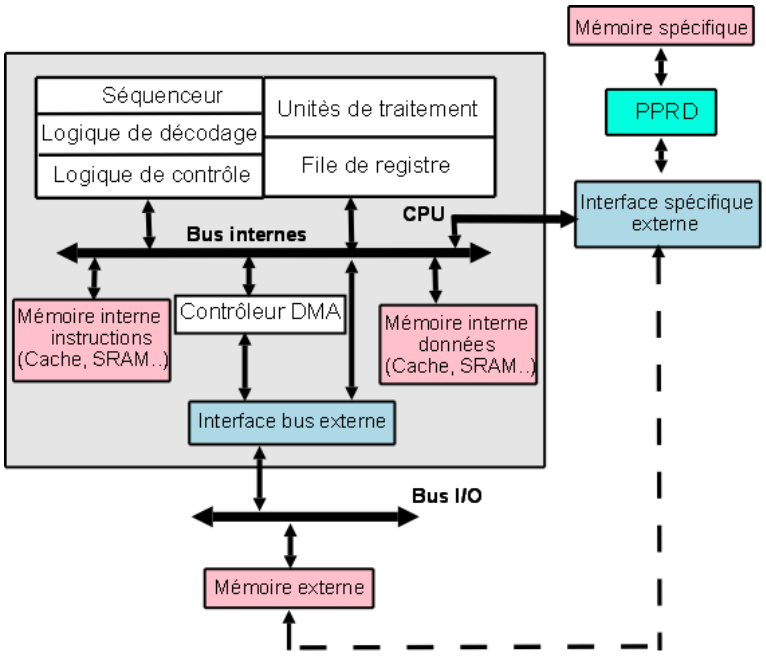

Figure 2-7 – **Architecture reconfigurable dynamiquement couplée en point-à-point au CPU**. *L'interface spécifique peut intégrer ses propres ressources mémoires, auxquelles peuvent accéder le CPU et l'ARD, ou assurer la communication entre le CPU et le PPRD à travers la mémoire externe de la puce.*

résume pour une tâche τ_i à :

$$\gamma_{com}^{i} = \frac{L_{spec}}{\lambda_{spec}} + \sigma \frac{\alpha^i l_{extDMA} + (1-\alpha^i) l_{ext}}{\varepsilon_{ext}} + \Omega \frac{l_{spec}}{\varepsilon_{spec}} \tag{2-27}$$

Pour ce modèle de couplage, l'expression de la dissipation d'énergie due à un accès mémoire n'est autre que celle dissipée lors d'un accès en mémoire externe et est donc définie par :

$$P_{Com}^{i} = P_{memExt}^{I/O} + 2P_{ARD}^{I/O} + P_{spec} + P_{memExt} \tag{2-28}$$

P_{spec} est la puissance dissipée par un transfert sur l'interface spécifique entre CPU et ARD.
Afin que ce modèle de couplage présente un intérêt du point de vue des performances, un compromis est à trouver entre le fait que la latence et la dissipation d'énergie du CPU dues au processus de contrôle sont nulles, et la latence et la dissipation d'énergie dues aux transferts depuis les mémoires externe.

Dans la section suivante, nous allons appliquer ces modèles de performances sur deux architectures hybrides, une en couplage CRD et la deuxième en couplage UFRD.

C Validation du modèle

Afin de valider nos modèles de détermination de l'accélération des calculs et de la réduction de la dissipation d'énergie pour une architecture hybride de type CPU/ARD, nous les appliquons aux architectures GARP et XIRISC pour une application de filtrage d'image médian.

Le filtre d'image médian est un filtre spatial qui calcule en chaque pixel la médiane des niveaux de gris des pixels de sa fenêtre, ce qui donnera le niveau de gris du pixel dans l'image filtrée.

Dans [26], il a été relevé une accélération de 43 de GARP (avec une fréquence d'exécution de 133 MHz) par rapport au processeur superscalaire ULTRASPARC (tournant à 167 MHz, soit plus rapide de 73 % que le processeur MIPS de GARP), pour ce benchmark et ce pour une image de taille 640×480 pixels. Dans [30], les auteurs présentent un chiffre d'accélération de 7.7 et une réduction de dissipation d'énergie de 60% pour la même application.

Nous avons appliqué notre modèle d'accélération à l'architecture hybride GARP par rapport au processeur ULTRASPARC, et ce pour différentes tailles de l'image.
L'architecture hybride GARP, que nous avons présenté à la sous section B.2 du chapitre I, correspond au modèle de couplage CRD entre un processeur MIPS II et une matrice de blocs logiques dotée d'un cache de configuration. La matrice de blocs logiques est composée de 32 lignes de 24 blocs. Chaque bloc a besoin de 8 octets pour se re-configurer. Ainsi, la taille d'un contexte de reconfiguration pour l'ARD est de $\eta = 6144$ octets.
La reconfiguration à partir du cache de configuration ne prend que 4 cycles GARP. Cependant, si le contexte de configuration n'est pas dans le cache de configuration, le chargement à partir de la mémoire externe dure $50\mu s = 6144 \times (7.19ns + 0.74ns)$. En effet, la mémoire externe de GARP possède une latence d'accès de 155 ns avec une largeur du mot mémorisable de 128 bits (16 octets). Ainsi, la latence d'accès d'un octet à cette mémoire externe est de 7.19 ns. Le bus I/O est un bus PCI classique (133 MHz/128 bits), sa bande passante est alors de 1.06 Go/s. La latence de transfert d'un octet à travers ce bus est de 0.74 ns.

La latence de transfert des données est définie par le produit entre le volume de données à l'entrée de la tâche par la latence de communication d'un octet depuis le système mémoire. L'expression de cette dernière est $\gamma^i_{com} = 2.5(1 - \mu^i) + (0.74 + 7.19)\mu^i$. En

effet, GARP possède un bus système avec une bande passante de 3.2 Go/s ; ce qui fait que la latence de transfert d'un octet sur le bus système est de 0.3125 ns. L'unité reconfigurable accède au cache de données à travers 4 bus de 32 bits chacun. Ce cache possède une latence d'accès de 8.75 ns (mot mémorisable de 32 bits).
Sachant qu'un pixel est codé sur 8 bits, le volume de donnée à l'entrée de la tâche du filtrage d'image median n'est autre que la traille en pixels de l'image.
Un accès au système mémoire est réalisé en deux phases : la première est la demande d'accès à la mémoire (que ce soit interne ou externe) "*memory access request*" initiée par le CPU et la deuxième phase est le transfert de la donnée. Pour le cache de données, ces deux phases sont pipelinées. Ainsi un nouvel accès mémoire est initié chaque cycle. De plus, quatre mots de 32 bits peuvent être lus du cache avec un seul "*memory access request*".

Les résultats de l'application du modèle du couplage CRD à GARP sont présentés dans la figure 2-8.

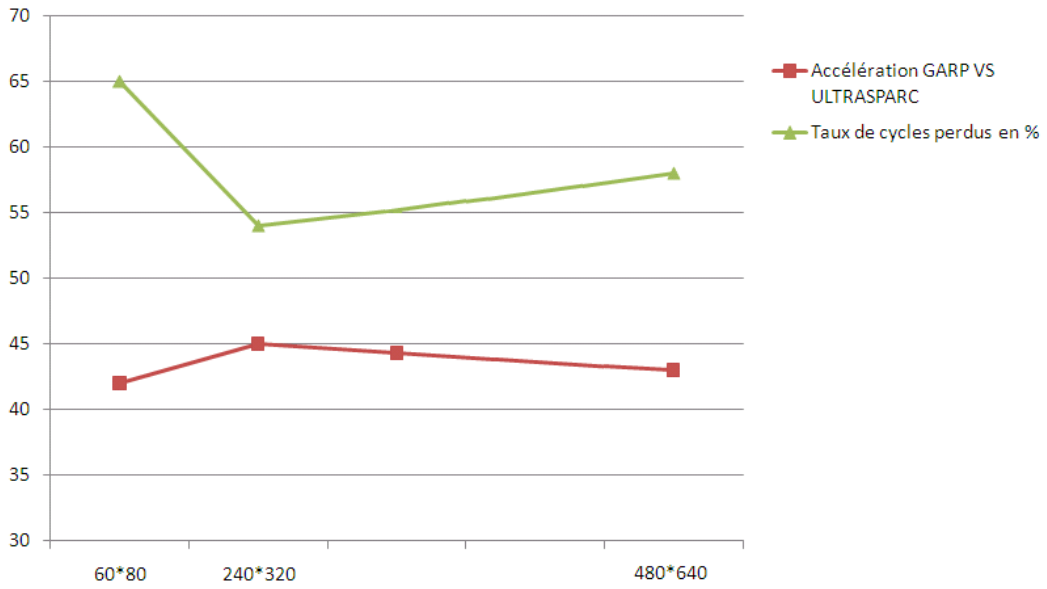

Figure 2-8 – **Taux de cycles perdus et accélération de l'architecture hybride GARP (par rapport au processeur ULTRASPARC) en fonction de la taille d'image en pixels**. *Le taux de cycle perdus diminue d'autant que la taille d'image en pixels est plus grande et il sature à 58%. Une accélératon entre 35 et 42 par rapport aux performances de l'ULTRASPARC est à souligner.*

Le taux de cycles perdus diminue d'autant que la taille d'image en pixels est plus grande et il sature autour de 55%. Cette diminution du taux de cycles perdus est due

au fait que la latence de chargement des données et des contextes de configuration est amortie sur la latence d'exécution d'un nombre de pixels plus grand.
En effet, sur le processeur ULTRASPARC, le filtrage median d'une image de taille 60×80 pixels prend 7 ms. Alors que sur l'architecture hybride reconfigurable GARP et pour la même taille d'image en pixel, l'exécution de ce benchmark ne prend que 0.2 ms.
Une accélération de 35 fois par rapport aux performances de l'ULTRASPARC est à souligner. Cette accélération augmente d'autant que la taille de l'image augmente et sature à une valeur de 42. Il est à noter que le processeur ULTRASPARC tourne 73% plus rapidement que le processeur MIPS II de GARP.
L'application de notre modèle d'étude de performances de l'architecture hybride est bien conforme à ce qui a été présenté dans [26] pour une image de taille 640×480 pixels.

L'architecture XIRISC correspond au modèle de couplage CPU/ARD en UFRD. Les résultats de l'application du modèle du couplage UFRD à l'architecture XIRISC sont présentés dans la figure 2-9.

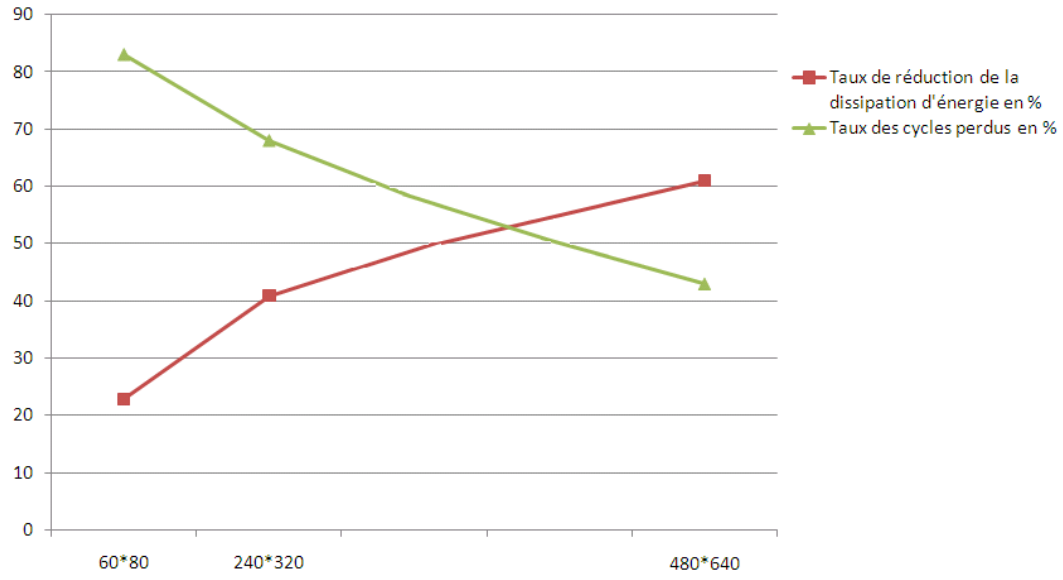

Figure 2-9 – **Taux de cycles perdus et taux de réduction de la dissipation d'énergie de l'architecture hybride XIRISC en fonction de la taille d'image en pixels**. *Avec un taux de cycles perdus variant de 83% à 42%, l'architecture XIRISC présente une accélération variant de 6.01 à 7.8 pour le filtrage d'image médian. Le taux de réduction de la dissipation d'énergie varie entre 21% et 58%.*

Avec un taux de cycles perdus variant de 83% à 42%, l'architecture XIRISC présente une accélération variant de 6.01 à 7.8 pour le filtrage d'image médian. Le taux de

réduction de la dissipation d'énergie varie entre 21% et 58%.

D Synthèse

Dans ce chapitre nous avons présenté une modélisation des architectures hybrides reconfigurables. Ces modèles nous ont permis de caractériser ces architectures et de quantifier leur performances. Cette quantification est basée sur une approche fonctionnelle et paramétrique qui permet pour un CPU et une ARD donnée de déterminer le gain en performances calculatoires et énergétiques par rapport à celles du CPU seul, et ce pour les différents types de couplages entre le processeur et l'architecture reconfigurable. L'application de ce modèle aux architectures hybrides GARP et XIRISC pour le benchmark filtrage d'image median, nous a permis de valider notre modèle en comparant nos résultats à ceux fournis par les publications relatifs à ces architectures. Nos résultats montrent que l'erreur maximale de prédiction par rapport aux mesures est de 8%.

Dans le chapitre suivant, nous présentons une méthodologie d'adéquation algorithme architecture permettant suivant les paramètres de l'application de déterminer le couplage adéquat CPU/ARD et ce en utilisant les modèles que nous avons définis dans ce chapitre.
Nous présentons aussi comment nous avons exploité ces modèles afin de proposer une méthodologie de partitionnement temporelle pour les architectures hybrides.

Chapitre

3

Méthodologies d'optimisation des architectures hybrides reconfigurables

Sommaire

Ce chapitre présente tout d'abord une méthodologie d'adéquation algorithme architecture hybride qui permet de déterminer, pour une application donnée, le modèle de couplage CPU-Architecture Reconfigurable Dynamiquement optimal.
Nous avons par ailleurs étudié l'impact des paramètres applicatifs et algorithmiques sur les performances d'une architecture hybride. Pour cela, nous avons porté un récepteur WCDMA dynamique sur un modèle générique d'architecture hybride en appliquant notre méthodologie.
Ce chapitre présente ensuite une méthodologie d'optimisation de l'implémentation sur l'architecture visant à exploiter de façon optimale la reconfiguration dynamique. Cette méthodologie exploite les modèles d'estimation de performances démontrés dans le chapitre II. Dans cette partie, nous présentons l'architecture reconfigurable dynamiquement DART qui nous servira de support de validation dans le chapitre suivant.

Nous allons montrer comment nous avons adapté le flot de compilation classique de l'architecture DART afin de l'intégrer dans une architecture hybride.

A Méthodologie d'Adéquation Algorithme Architecture Hybride AAAH

Dans cette section, nous proposons une méthodologie d'AAAH (Adéquation Algorithme Architecture Hybride reconfigurable) de type processeur (CPU) associé à une ARD. Cette méthodologie, représentée par la figure 3-1, permet de faire correspondre, pour une application donnée, la façon optimale (en termes de performances) de mettre en interaction un CPU et une ARD. Dans cette méthodologie, nous nous sommes appuyés sur les modèles généraux présentés dans le chapitre précédent.

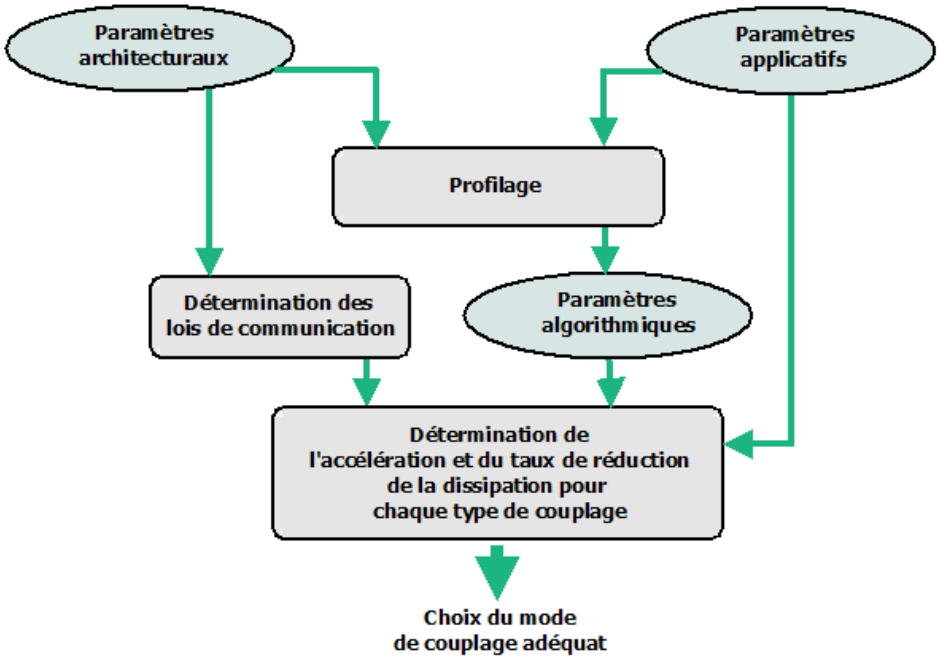

Figure 3-1 – **Méthodologie d'adéquation algorithme architecture hybride reconfigurable dynamiquement**. *Après vérification des contraintes temps réel, un profilage permet de fournir les paramètres algorithmiques à partir des paramètres applicatifs et architecturaux. La deuxième étape consiste en la détermination des lois de communications à partir des paramètres architecturaux. Enfin, l'application de ces différents paramètres aux modèles d'accélération et de réduction de la consommation d'énergie permet de déterminer le couplage CPU/ARD adéquat pour l'application considérée.*

La première étape de notre méthodologie est le profilage qui permet de fournir les paramètres algorithmiques à partir des paramètres applicatifs et architecturaux. La deuxième étape consiste en la détermination des lois de communications à partir des paramètres architecturaux. Enfin, l'application de ces différents paramètres aux modèles d'accélération et de réduction de la consommation d'énergie permet de déterminer le couplage CPU/ARD adéquat pour l'application considérée.

Nous présentons dans la suite de ce chapitre une application de télécommunications mobiles de troisième génération que nous utiliserons comme exemple de portage sur notre méthodologie AAAH. Nous expliquons les principaux éléments qui composent cette application. Enfin, nous étudions les retombées des différents paramètres applicatifs et algorithmiques sur les performances du modèle hybride CPU/ARD.

A.1 Exemple d'application de la méthodologie AAAH

Pour illustrer les travaux que nous avons développés dans la partie précédente, nous avons retenu une application de télécommunication liée aux systèmes de radiocommunication de troisième génération (systèmes 3G) et plus particulièrement un récepteur WCDMA (*Wideband Code Division Multiple Access*) embarqué sur un terminal mobile.

A.1-1 Présentation générale de WCDMA

Au niveau des communications mobiles cellulaires, le canal de transmission entre l'émetteur et le récepteur possède des propriétés particulières. Le signal reçu est composé de plusieurs répliques du signal émis, liées à la présence de multiples chemins de propagation de l'onde radioélectrique entre l'émetteur et le récepteur. Chaque trajet est caractérisé par un retard et une amplitude complexe. Le déplacement du terminal mobile peut, par ailleurs, conduire à une variation de la puissance du signal reçu. Ce phénomène, dénommé *fading*, produit des évanouissements du signal lorsque la combinaison des multi-trajets est destructrice.
Le WCDMA utilise une technique d'accès par étalement de spectre, en attribuant un code unique à chaque utilisateur. Les signaux sont alors mélangés sur le canal. Les propriétés mathématiques des codes utilisés permettent à la réception d'extraire du signal étalé les seules informations de l'utilisateur concerné. L'élément correspondant à un symbole du code est dénommé *chip* et le débit du code D_c est fixé à 3.84 Mbits/s. Le synoptique d'un récepteur WCDMA est représenté sur la figure 3-2. Dans un tel récepteur, le signal capté par l'antenne est tout d'abord translaté en bande de base,

puis converti en une forme numérique. L'exploitation d'une modulation de phase à quatre états se traduit par la représentation des signaux sous une forme complexe. La numérisation du signal opère à une fréquence F_{chip} multiple de celle d'un *chip* (3.84 MHz). Le facteur de sur-échantillonnage N_{se} ne doit pas être trop élevé afin de limiter la complexité des calculs réalisés par la suite. En effet, pour chaque bloc de traitement τ_i la complexité algorithmique C^i est égale à

$$C^i = N_{se} \times F_{chip} \times Nb^i_{op} \times 10^{-6}[MOPS] \tag{3-1}$$

Nb^i_{op}, est le nombre d'opérations arithmétiques et logiques qui doivent être exécutées par la tâche τ_i. Dans le cadre de notre expérimentation, le N_{se} a été fixé à 4.

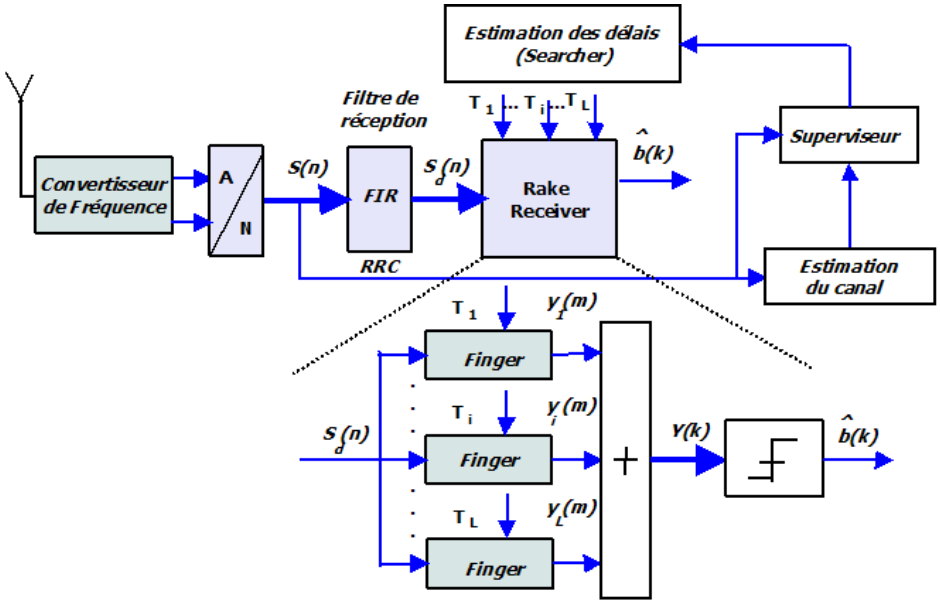

Figure 3-2 – **Synoptique d'un récepteur WCDMA (dynamique) de terminal mobile**. *Une fois numérisé, le signal complexe attaque un filtre de Nyquist (de réception) qui permet d'éliminer l'interférence entre symboles. Le Rake Receiver est destiné à exploiter la diversité induite par la présence des trajets multiples, il est composé de L fingers. Le rôle de l'estimateur de délais (*searcher*) d'estimer les retards introduits par le canal sur les répliques du signal original. Le superviseur détermine le nombre L de trajets multiples et par conséquent des* fingers *suivant la valeur du RSB du signal.*

Une fois numérisé, le signal complexe attaque un filtre de Nyquist, dit de réception. Celui-ci permet d'éliminer l'interférence entre symboles. Les parties réelles et imaginaires de ce filtre sont traitées indépendamment par un filtre FIR à coefficients réels composé de 64 cellules. La complexité de ce filtre est égale à $C^{fir} = (N_{se} \times F_{chip}) \times 2 \times (64 + 63) \times 10^{-6} = 3901 MOPS$. Soit M le nombre d'utilisateurs,

L le nombre de répliques du signal transmis disponibles en entrée du récepteur et w le bruit introduit par le canal et présent en sortie du filtre de Nyquist, le signal $S_{d(n)}$ en sortie du filtre de réception satisfait l'expression suivante :

$$S_{d(n)} = \sum_{m=1}^{M} \sum_{l=1}^{L} S_m(n - T_l) + w(n) \tag{3-2}$$

La composante $S_m(n-T_l)$ correspond au signal reçu de l'utilisateur m affecté du retard T_l.

Un bénéfice supplémentaire est ainsi obtenu en combinant les signaux issus des différents multi-trajets après traitement de ceux-ci au sein d'un récepteur en râteau (*Rake Receiver*). Le *Rake Receiver* est composé de L fingers. Chaque *finger* réalise le décodage des données, l'estimation de l'amplitude complexe du canal et la synchronisation des codes générés en interne. La complexité algorithmique d'un *finger* est de 331 MOPS. Ainsi la complexité algorithmique du *Rake Receiver* est $C^{rake} = L * 331MOPS$.
Afin de bénéficier du décodage des multi-trajets, il est nécessaire d'estimer les retards introduits par le canal sur les répliques du signal original. C'est le rôle de l'estimateur de délais (*searcher*).

Le récepteur WCDMA trouve l'intérêt d'implémentation sur une architecture hybride reconfigurable dynamiquement si le nombre L de trajets multiples, et par conséquent des *fingers*, est déterminé dynamiquement suivant la valeur du RSB (Rapport Signal sur Bruit) du signal décodé déterminée par le bloc superviseur. Une reconfiguration dynamique de l'ARD est nécessaire suivant le nombre de trajets multiples L afin d'activer ou de désactiver des *fingers* supplémentaires.

A.1-2 Étude du portage du *Rake Receiver* dynamique sur une architecture hybride reconfigurable

La décomposition en graphe de tâches (GdT) du récepteur WCDMA est présentée par la figure 3-3. Un *slot* à l'entrée du filtre comporte 10 symboles et chaque symbole comporte 256 *chips*. En prenant en compte le sur-échantillonnage de 4, le *slot* représente alors 10240 échantillons, chacun de 8 bits. Avant l'exécution sur le *Rake Receiver*, un sous échantillonnage de 4 (choix d'un échantillon parmi 4) est effectué et par conséquent un *slot* représente 2560 échantillons.
Nous proposons de porter la tâche τ_3 (*Rake Receiver*) sur une architecture hybride

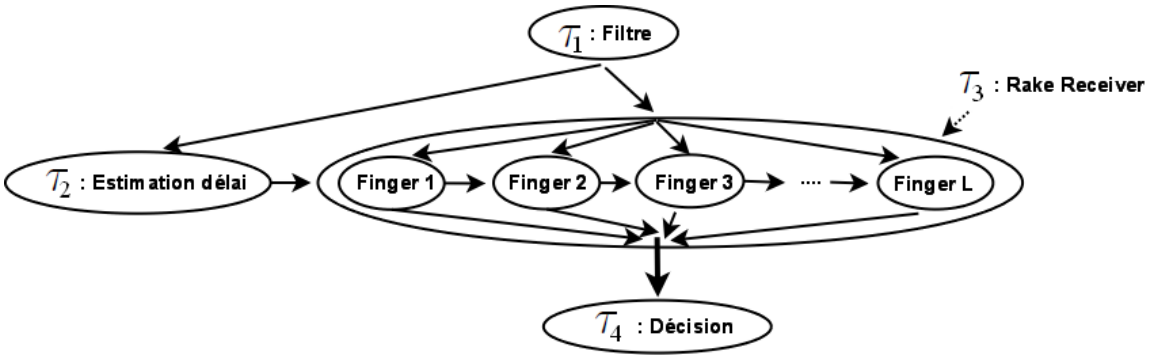

Figure 3-3 – **Décomposition en graphe de tâches du récepteur WCDMA**. *Les tâches les plus calculatoires sont le filtre et le* Rake Receiver. *La complexité de la tâche filtre est égale à 3901 MOPS. Celle du Rake Receiver est variable selon le nombre de* fingers. *La complexité algorithmique d'un* finger *est de 331 MOPS.*

générique. Ce *Rake Receiver* est composé de 1 à *L fingers*, suivant le nombre de multi-trajets déterminé par le superviseur.
La caractéristique de cette implémentation dynamique du *Rake Receiver* est que ses paramètres applicatifs sont variables. En effet, le volume de données en octet ρ^{Rake}, à charger depuis les ressources de mémorisation en entrée du *Rake Receiver*pour l'exécution sur un slot, est variable suivant le nombre de *fingers* à implémenter sur l'ARD.

Le nombre d'itérations d'exécution du *Rake Receiver* n^{Rake} est aussi variable selon le schéma d'exécution de l'application (c-à-d selon que l'on exécute moins d'un *slot* ou plus d'un *slot*) en entrée de chaque tâche.

Nous avons appliqué notre méthodologie afin de déterminer, suivant les besoins de l'application (paramètres applicatifs et algorithmiques), l'interaction la plus adéquate entre le CPU et l'ARD au sein de l'architecture permettant d'avoir une performance optimale. Nous étudions l'impact des différents paramètres applicatifs et algorithmiques sur les performances de l'architecture hybride dans chaque modèle de couplage entre le CPU et l'ARD.

Pour cela, nous considérons une architecture hybride reconfigurable dont la ressource logicielle est le DSP TMS320C64x. Le temps d'exécution par *slot (2560 chips)* d'un *finger* sur ce DSP (en exploitant le SWP) prend 23.29 μs [48]. L'ARD considérée présente un gain en temps de calcul d'un *finger* de 10 fois. Ce gain correspond au rapport $\frac{T^{finger}_{execCPU}}{T^{finger}_{execARD}}$ dans l'expression de l'accélération (équation 2-8).

Nous considérons que le taux de dissipation d'énergie de l'ARD pour l'exécution d'un *finger* par rapport au CPU $\frac{P_{ARD}^{finger}}{P_{CPU}^{finger}}$ est de 10%.
Ces gains ne comprennent donc pas la latence et la dissipation d'énergie dues au contrôle, à la reconfiguration et au transfert. Celles-ci dépendent du modèle de couplage dans l'architecture hybride.
Nous supposons que la reconfiguration est partielle et que les caractéristiques des interconnexions (bus internes, externes...) sont standards.

A.2 Impact des paramètres applicatifs et algorithmiques sur les performances d'une architecture hybride

A.2-1 Choix du couplage en fonction des paramètres applicatifs

Nous étudions l'impact des différents paramètres applicatifs sur les performances de l'architecture hybride. Nous faisons varier les paramètres applicatifs et les scénarios de couplages CPU/ARD, puis nous appliquons les modèles de performances relatifs à ces couplages.

Impact du volume de données

Le volume de données en octet ρ^{Rake} à charger depuis les ressources de mémorisation en entrée du *Rake Receiver*, pour une exécution sur un slot, est variable suivant le nombre de *fingers* à implémenter sur l'ARD.

La figure 3-4 présente la variation du taux de cycles perdus en fonction du volume de données en entrée du *Rake Receiver* pour les différents modèles de couplage CPU/ARD. Nous avons considéré une exécution sur un *slot* à l'entrée du *Rake Receiver*.
Le cas PPRD1 correspond au modèle de couplage point-a-point reconfigurable dynamiquement (PPRD), avec les deux tâches τ_{i-1} et τ_i s'exécutant toutes les deux sur l'ARD. Ce qui veut dire que, dans le cas du récepteur WCDMA, le filtre et le *rake receiver* sont implémentés tous les deux sur l'ARD.
Le cas PPRD2 est le cas contraire c'est à dire que le filtre est exécuté sur le CPU et le *Rake Receiver* sur l'ARD.

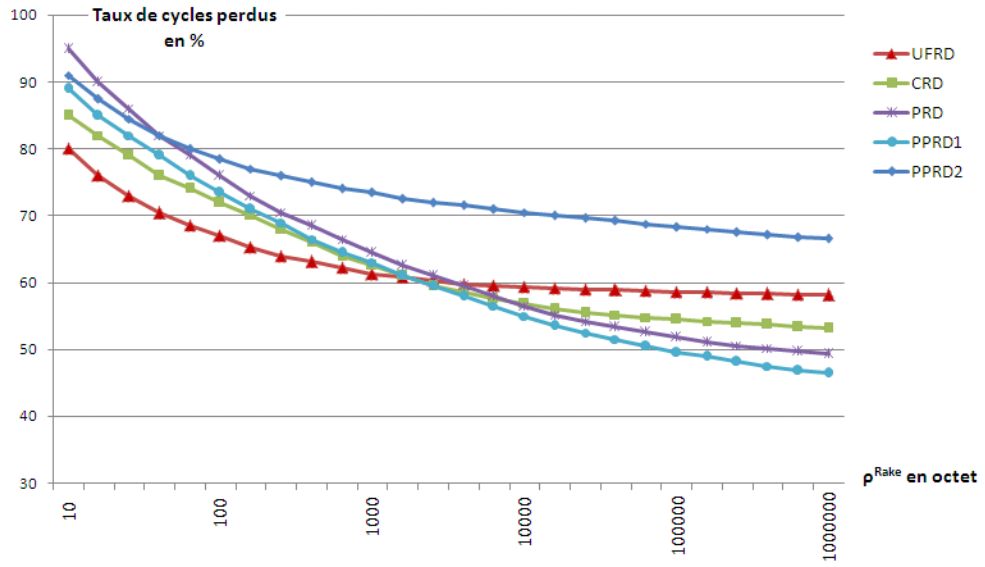

Figure 3-4 – **Taux de cycles perdus lors du traitement du *Rake Receiver* en fonction du volume de données échangées pour les différents modèles de couplage CPU/ARD**. *Le cas PPRD1 correspond au modèle de couplage PPRD avec les deux tâches filtre et* Rake Receiver *implémentées sur l'ARD. Le cas PPRD2 correspond au cas où le filtre est exécuté sur le CPU et le* Rake Receiver *sur l'ARD.*

Nous remarquons que, pour tous les modèles de couplage, le taux de cycles perdus diminue avec l'augmentation du volume de données en entrée de la tâche *rake* en saturant quand ρ^i tend vers 10^5. Ce qui veut dire qu'on accélère d'autant plus que la quantité de données échangées augmente jusqu'à saturer en une valeur limite d'accélération.
La figure 3-5 montre que cette valeur de saturation de l'accélération apportée par l'architecture hybride diffère d'un couplage CPU/ARD à un autre. Pour le traitement du *Rake Receiver*, un couplage en PPRD sature à une accélération de 7 fois alors qu'un couplage en UFRD (Unité Fonctionnelle Reconfigurable Dynamiquement) sature en une accélération de 4.5 fois. Cette diminution du taux de cycles perdus est due à ce que la latence de chargement et de reconfiguration est amortie si la latence de traitement est plus grande. En effet, en augmentant le nombre de fingers traitant les échantillons en entrée du Rake Receiver, nous augmentons les taux de chevauchement δ^{Rake} et χ^{Rake} entre le calcul et le chargement des données et des contextes de reconfiguration jusqu'à tendre vers des valeurs de 100%, ce qui explique que cela sature à partir d'un certain volume de données.
Cela correspond à ce qui a été démontré dans [49] par le critère nommé rémanence R défini par l'équation suivante :

$$R = \frac{N_a.F_e}{N_c.F_c} \tag{3-3}$$

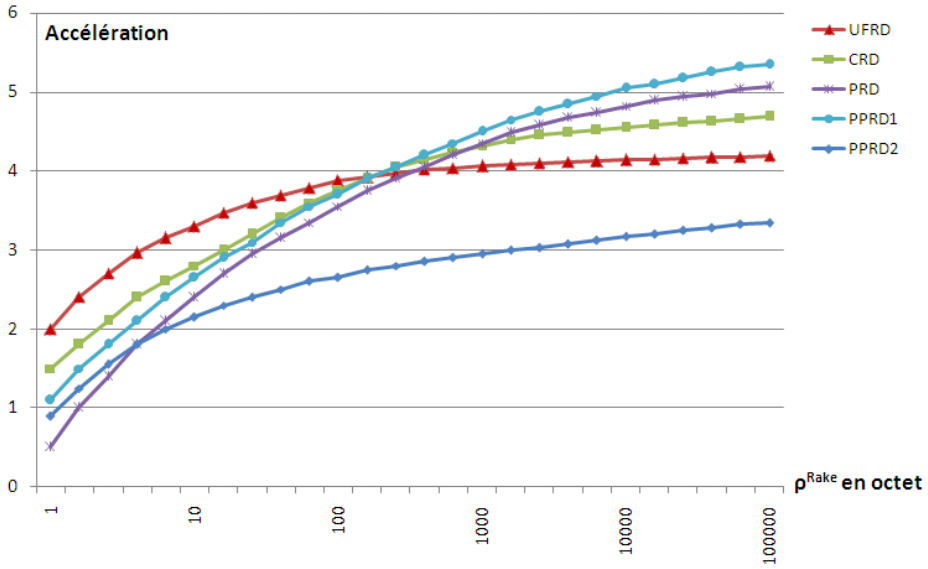

Figure 3-5 – **Accélération du Rake Receiver apportée par l'architecture hybride en fonction du volume de données échangées pour les différents modèles de couplage CPU/ARD**. *Le traitement est accéléré d'autant plus que la quantité de données échangées augmente jusqu'à saturer en une valeur limite d'accélération. Cette valeur de saturation diffère d'un couplage CPU/ARD à un autre. Par exemple, le couplage en PPRD sature à une accélération de 7 fois alors qu'un couplage en UFRD sature en une accélération de 4.5 fois.*

N_a est le nombre d'UFs utilisées dans une architecture et F_e sa fréquence d'exécution. N_c est le nombre d'UFs reconfigurées à chaque cycle d'une horloge de configuration de période $\frac{1}{F_c}$.
La rémanence R donne en effet une idée sur l'efficacité de la reconfiguration dynamique, c-à-d le nombre de données minimum devant être traitées entre deux reconfigurations pour que l'architecture exploite efficacement ces ressources de calcul. Les auteurs de [49] expliquent que les configurations doivent être maintenues pendant un nombre de cycles supérieur à 10 R.

Nous notons aussi que les modèles UFRD et CRD (Coprocesseur Reconfigurable Dynamiquement) présentent un taux de cycles perdus moins important que celui des modèles PRD (Peripherique Reconfigurable Dynamiquement) et PPRD pour les ρ^i moins que 10^4 octets et que, au delà de cette valeur le couplage PRD et PPRD deviennent plus intéressants. En effet, l'augmentation du volume de données en entrée de la tâche entraîne un nombre de lectures mémoires important. Comme pour les modèles UFRD et CRD, le CPU initie et contrôle continuellement tous les transferts ce qui entraine une latence $T_{controlCPU}$ importante.

Le modèle UFRD reste meilleur au niveau des performances que le modèle CRD mais ceci se paie par un coût supplémentaire pour la mise en œuvre de l'architecture qui n'est pas négligeable.

Le modèle PPRD est semblable au modèle PRD mais il trouve son intérêt dans l'allocation pour l'ARD d'une bande passante sur l'interconnexion plus importante que le modèle PRD où les autres périphériques consomment une part non négligeable de la bande passante. Toute fois, le modèle PPRD perd tous ces avantages quand les deux tâches τ_{i-1} et τ_i ne s'exécutent pas toutes les deux sur l'ARD puisque'une latence supplémentaire de transfert de données depuis la hiérarchie mémoire du CPU à la mémoire externe vient s'ajouter à la latence de transfert. Dans ce qui suit, nous allons considérer le modèle PPRD dans le cas où toutes les tâches sont exécutées sur l'ARD afin qu'il garde son intérêt.

La figure 3-6 montre que le taux de réduction d'énergie dissipée par l'architecture hybride pour l'exécution de la tache *rake* par rapport à la dissipation d'énergie lors de l'exécution de cette tâche sur le CPU seul, augmente d'autant plus que ρ^{Rake} et commence à saturer à partir d'un volume de données d'environ 10^5 octets.

Pour les modèles de couplage PRD et PPRD, tous les accès mémoires sont externes, ce qui dissipe une quantité d'énergie importante. Cependant, ces deux couplages donnent des taux de réduction d'énergie dissipée intéressants (jusqu'à 90%) pour des quantités de données manipulées importantes ($\rho^{Rake} = 5 \times 10^5$). En effet, quand le volume de données en entrée de la tâche augmente, les accès mémoires augmentent et par conséquent l'énergie dissipée dans le contrôle des transferts mémoire pour les modèles UFRD et CRD devient prépondérante.

Impact du nombre d'itérations

Le nombre d'itérations d'exécution du *Rake Receiver* n^{Rake} est un paramètre variable selon le nombre d'échantillons en entrée de cette tâche.
Pour un nombre de *fingers* fixé à 6 (donc un volume de données ρ^{Rake} fixe), l'exécution multiple de la tâche (exécution sur un nombre d'échantillons plus grand) amortit le coût des cycles perdus lors du chargement des contextes de configuration. Ceci explique la diminution du taux de cycles perdus quand le nombre d'itérations de la tâche augmente (voir figure 3-7).
Cependant, en augmentant le nombre d'itérations du traitement sur le *Rake Receiver*, nous diminuons le taux de chevauchement δ^{rake} entre le calcul et le chargement des

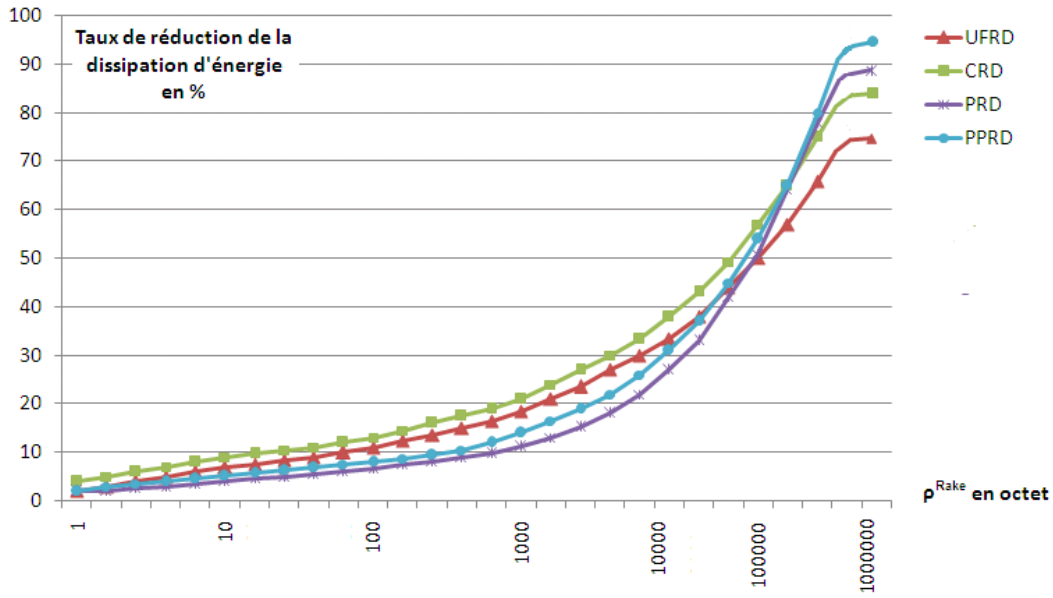

Figure 3-6 – **Taux de réduction en dissipation d'énergie lors du traitement du *Rake Receiver* par rapport à la dissipation d'énergie lors de l'exécution de cette tâche sur le CPU seul en fonction du volume de données échangées pour les différents modèles de couplage CPU/ARD**. *Les modèles de couplage PRD et PPRD dissipent plus d'énergie puisque tous les accès mémoires sont externes. Cependant, quand le volume de données en entrée de la tâche augmente, les accès mémoires augmentent et par conséquent l'énergie dissipée dans le contrôle des transferts mémoire pour les modèles UFRD et CRD devient importante.*

données. Ce qui fait que le taux de cycles perdus remonte à partir d'une certaine valeur du nombre d'itérations (10^3 pour les modèles UFRD et CRD et 10^4 pour les modèles PRD et PPRD) quand l'amortissement des cycles perdus dans le chargement de la configuration est dépassé par l'augmentation du nombre de cycles perdus dans le chargement de données. Le taux de cycles perdus sature quand δ^{Rake} tend vers 0 (correspondant à un nombre d'itérations de 10^5).

Les modèles PRD et PPRD sont intéressants à partir d'un nombre d'itérations de 10^3 à cause du coût de la latence due au contrôle CPU qui devient pénalisant en performances pour les modèles UFRD et CRD.

Le taux de réduction de la dissipation d'énergie de l'architecture hybride, par rapport à la dissipation du CPU seul lors du traitement du *Rake Receiver*, augmente avec le nombre d'itérations du traitement de cette tâche (Figure 3-8). Les architectures hybrides reconfigurables en modèles de couplage en UFRD et CRD dissipent moins d'énergie que celles en modèles de couplage PRD et PPRD sauf si le nombre d'itérations

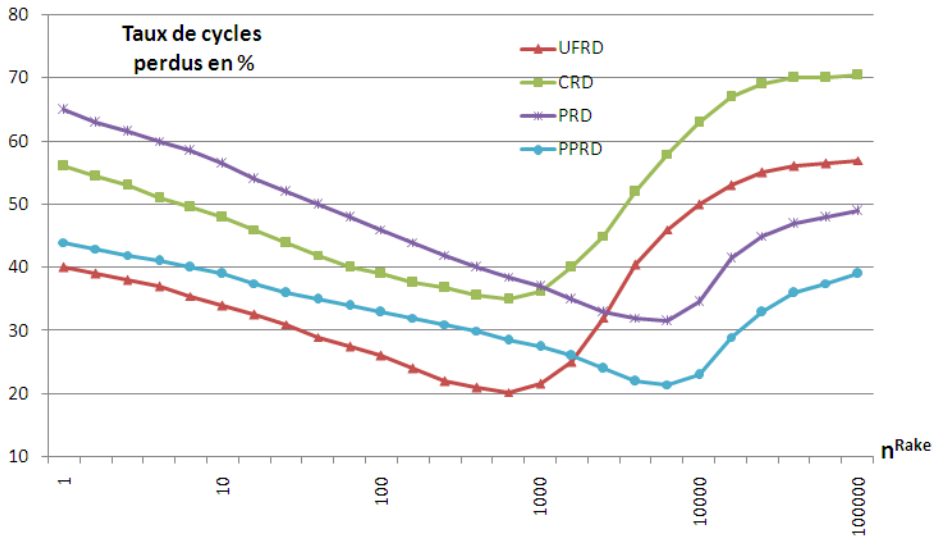

Figure 3-7 – **Taux de cycles perdus en fonction du nombre d'itérations de la tâche pour les différents modèles de couplage CPU/ARD**. *Les modèles PRD et PPRD sont intéressants à partir d'un nombre d'itération de* 10^3 *à cause du coût de la latence due au contrôle CPU qui devient pénalisant en performances pour les modèles UFRD et CRD.*

d'exécution du *Rake Receiver* dépasse environ les 10^4 itérations. En effet, dans ce cas la dissipation du CPU pour le contrôle des accès mémoire devient importante et pénalise ainsi les modèles de couplage UFRD et CRD.

A.2-2 Choix du couplage en fonction des paramètres algorithmiques

Un profilage de la tâche permet de connaître les paramètres algorithmiques de l'application. Nous avons étudié l'impact de la variation du paramètre algorithmique μ^{rake} (taux d'accès en mémoire externe pour la tache Rake Receiver) sur le taux de cycles perdus et de l'accélération et ceci pour un taux d'accès en DMA $\alpha^{Rake} = 100\%$.

La figure 3-9 montre que le taux de cycles perdus augmente (et inversement l'accélération diminue) linéairement en fonction du μ^{Rake} pour les modèles de couplage UFRD et CRD alors qu'ils ne sont pas affectés par la variation μ^{Rake} pour les modèles de couplage PRD et PPRD. Ainsi pour les applications dont le taux d'accès mémoire est important, les couplages PRD et PPRD s'avèrent intéressants. Généralement, le

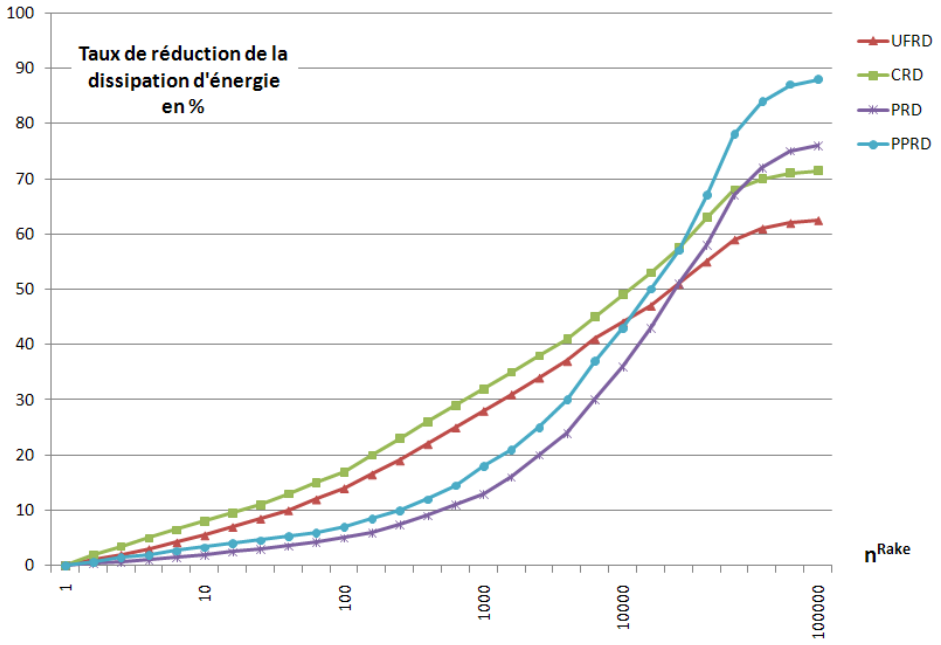

Figure 3-8 – **Taux de réduction de la dissipation d'énergie en fonction du nombre d'itérations du *Rake Receiver* pour les différents modèles de couplage CPU/ARD**. *Les architectures hybrides reconfigurables en modèles de couplage en UFRD et CRD dissipent moins d'énergie que celles en modèles de couplage PRD et PPRD sauf si le nombre d'itérations d'exécution du* Rake Receiver *dépasse environ les* 10^4 *itérations.*

taux important d'accès en mémoire externe est dû à un volume important de données à fournir en entrée de la tâche et nous retombons donc dans le cas ρ^{Rake} important étudié ci-dessus.

La variation du taux d'accès à la mémoire externe en DMA α^{Rake} a un impact sur les architectures hybrides à modèles CRD et UFRD.
Pour un taux d'accès en mémoire externe fixe (nous avons choisi un $\mu^{Rake} = 50\%$), le DMA permet d'augmenter le chevauchement entre le calcul et le chargement des données et de la reconfiguration. Ce qui fait que δ^{Rake} et χ^{Rake} vont augmenter quasi-linéairement avec α^{Rake} jusqu'à arriver à un seuil auquel l'accélération sature.
La dissipation d'énergie par le CPU dans le contrôle des accès en mémoire externe diminue avec un taux d'accès en DMA plus important. La figure 3-10 présente le taux de cycles perdus et le taux de réduction de la dissipation d'énergie en fonction de α^{Rake}.

Dans la section suivante, nous exploitons les modèles de performances dans une méthodologie d'optimisation de l'implémentation sur l'architecture visant à exploiter au

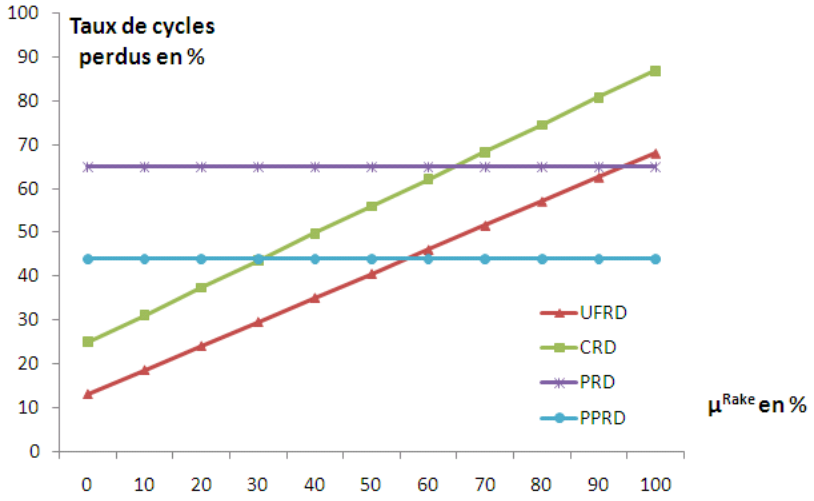

Figure 3-9 – **Taux de cycles perdus en fonction du taux d'accès en mémoire externe de la tâche *Rake Receiver* pour les différents modèles de couplage CPU/ARD**. *Le taux de cycles perdus augmente (et inversement l'accélération diminue) linéairement en fonction du μ^{Rake} pour les modèles de couplage UFRD et CRD alors qu'ils ne sont pas affectés par la variation μ^{Rake} pour les modèles de couplage PRD et PPRD.*

mieux la reconfiguration dynamique.

B Méthodologie d'optimisation de la reconfiguration dynamique dans une architecture hybride

Nous proposons dans cette section une méthodologie d'optimisation de l'implémentation sur une architecture hybride reconfigurable. L'objectif est de trouver la meilleure utilisation des ressources nécessaires à l'application en mettant en œuvre la reconfiguration dynamique. Cette méthodologie est une perspective pour la réalisation du partitionnement temporel sur une architecture hybride.
Nous appliquons la méthodologie proposée à l'architecture reconfigurable dynamiquement DART en l'intégrant dans son flot de compilation. Nous présentons ainsi l'ARD en question dans la sous-section suivante. Puis nous présentons la méthodologie d'optimisation et son intégration dans le flot de compilation d'une architecture hybride particulière couplant l'architecture DART au processeur TMS320C6713.

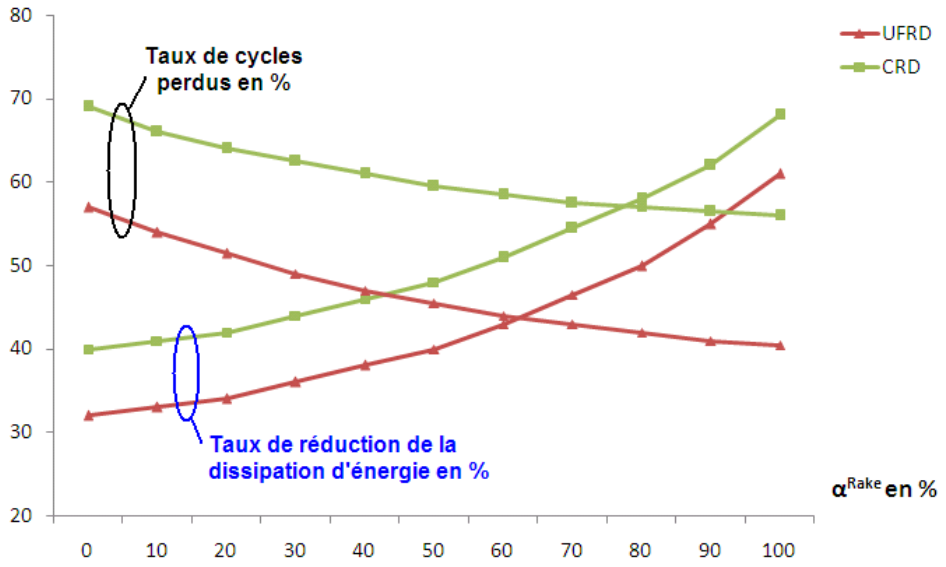

Figure 3-10 – **Taux de cycles perdus et taux de réduction de la dissipation d'énergie en fonction du taux d'accès en DMA à la mémoire externe pour l'exécution de la tâche *Rake Receiver***. *Pour les modèles de couplage CRD et UFRD, le DMA permet d'augmenter le chevauchement entre le calcul et le chargement des données et de la reconfiguration et de diminuer par conséquent le taux de cycles perdus. En outre la proportion de contrôle du CPU diminue ce qui augmente le taux de réduction de la dissipation d'énergie.*

B.1 Architecture DART

L'architecture **DART** [50] est une architecture reconfigurable dynamiquement à grain épais avec une fréquence d'exécution de 200 MHz. Elle possède une puissance calculatoire supérieure à 4 GOPS et exige une puissance d'alimentation de 709 mW dont 100 mW pour la puissance dissipée par les courants de fuite. Nous allons présenter dans ce qui suit les caractéristiques architecturales de DART et son flot de conception.

B.1-1 Présentation Générale de l'architecture

La figure 3-11 présente l'architecture d'un cluster DART. Il intègre 6 DPR (*Data Path Reconfigurable*) inter-connectés par un réseau segmenté. Les communications entre ces DPRs peuvent être globales (le cas, par exemple, d'une mémoire qui peut approvisionner en données des unités fonctionnelles appartenant à différentes DPR) ou locales (le

cas, par exemple, de l'échange d'un résultat temporaire entre deux unités fonctionnelles appartenant au même DPR).

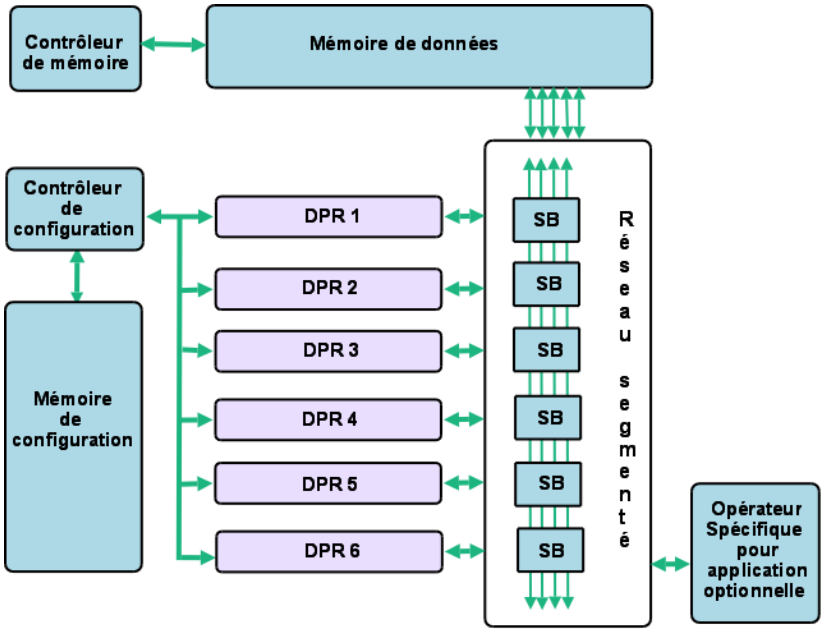

Figure 3-11 – **Architecture d'un cluster de DART**. *Il intègre 6 DPR interconnectés par un réseau segmenté. Toutes les ressources de calcul du cluster accèdent à une même mémoire de données de 16 Kmots de 16 bits. Le contrôleur de configuration accède à la mémoire de configuration et a pour rôle de séquencer les instructions de configuration.*

Chaque DPR (Figure 3-12) est constitué de 2 registres et 4 opérateurs ou UF (2 multiplieurs/additionneurs et 2 UAL) connectés à 4 mémoires locales de données à travers un réseau multi-bus.

Les UFs du cluster DART supportent un fonctionnement de type SWP (*Sub Word Parallelism*) permettant d'exploiter le parallélisme au niveau des données. La technique SWP (Figure 3-13) consiste à diviser les opérateurs arithmétiques (resp. un mot) de largeur N afin de pouvoir exécuter en parallèle p opérations de largeur N/p bits ($p \geq 2$).

Toutes les ressources de calcul du cluster accèdent à une même mémoire de données. Celle-ci a été dimensionnée à 16 Kmots de 16 bits afin d'assurer une certaine autonomie au cluster. Le cluster dispose encore de 24 mémoires locales distribuées entre les 6 DPRs. Ces mémoires, d'une capacité de 64 mots de 32 bits chacune, stockent les données manipulées au sein des DPRs. Chacune de ces mémoires est contrôlée par un

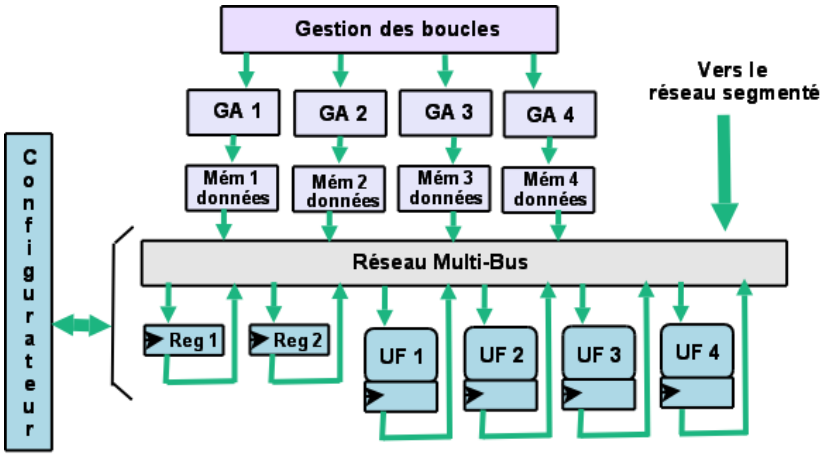

Figure 3-12 – **Architecture du DPR**. *Chaque DPR est constitué de 4 opérateurs (2 multiplieurs/additionneurs et 2 UAL) connectés à 4 mémoires locales de données à travers un réseau multi-bus. Ces mémoires, d'une capacité de 64 mots de 32 bits chacune, stockent les données manipulées au sein des DPRs. Chacune de ses mémoires est contrôlée par un GA local. Le configurateur détermine, à partir de la donnée de configuration, de quelle façon les UFs sont interconnectées, de quelle mémoire sont issues les données et dans quelle mémoire seront écrits les résultats.*

GA (Générateur d'Adresses) local. Tous ces mouvements de données sont programmés statiquement et déterminés au moment de la compilation.

Les contextes de reconfigurations des DPRs sont réalisés sous la forme de flots d'instructions de configuration (de 52 bits) qui forment le contexte de configuration. Ainsi, la principale tâche du contrôleur de configuration du cluster est de séquencer ces instructions de configuration. En effet, il doit accéder à la mémoire de configuration et spécifier, à partir de ces instructions, l'adresse de base et la taille de la donnée de configuration ainsi que le DPR cible. Il doit aussi spécifier quels GAs en sont concernées et leurs configurations des chemins de données ; c'est-à-dire qu'il doit charger leurs mémoires d'instructions.
Au niveau du DPR, le configurateur détermine à partir de la donnée de configuration de quelle façon les UFs sont inter-connectées, de quelle mémoire sont issues les données et dans quelle mémoire seront écrits les résultats.

La flexibilité introduite au sein des DPRs autorise la définition d'un chemin de données en adéquation avec le motif de calcul à exécuter. Deux modes de fonctionnement sont possibles :

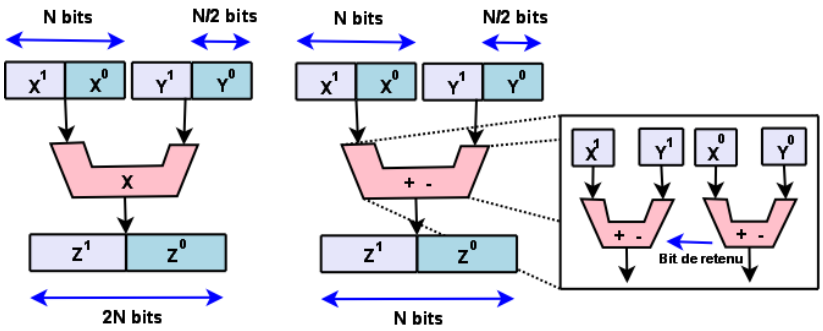

Figure 3-13 – **Opérateurs SWP**. *La technique SWP consiste à diviser les opérateurs arithmétiques (resp. un mot) de largeur N afin de pouvoir exécuter en parallèle p opérations de largeur N/p bits (p≥ 2).*

- Dans le mode SW, seules les fonctions des UFs et les sources et destination de données sont modifiées. Le mode de reconfiguration logicielle a été conçu afin de répondre aux besoins inhérents aux zones de calculs irrégulières (dans lesquelles les motifs de calculs se succèdent sans ordre particulier et de manière non répétitive). Un cycle suffit pour modifier en mode SW la configuration d'un DPR. Dans ce cas, le modèle de calcul est comparable à celui des DSP VLIW conventionnels. A chaque cycle, les données sont lues et traitées puis les résultats sont rangés en mémoire. Ceci justifie la qualification de logicielle de ce mode de reconfiguration ;
- En mode HW, les chemins reliant les opérateurs sont reconfigurés. Il faut alors 3 cycles pour reconfigurer un DPR et 1 cycle pour reconfigurer le réseau global. Ce mode répond aux besoins des traitements réguliers, tels que les cœurs de boucles dans lesquels un même motif de calcul est utilisé pendant de longues périodes de temps. La reconfiguration HW consiste à spécifier une configuration par le biais d'un flot d'instructions, puis d'adopter un modèle de calcul de type flot de données dans lequel une structure optimisée du chemin de donnée est figée. Une fois cette configuration spécifiée, le contrôleur de configuration du cluster est donc libéré du contrôle des DPRs et n'a plus à accéder à la mémoire de configuration et à décoder de nouvelles instructions de configuration. Les informations de configuration sont alors stockées dans des latchs au sein des DPRs. Avec ce modèle de calcul il ne reste qu'à approvisionner les unités fonctionnelles en données. Le contrôle est ainsi dévolu aux générateurs d'adresses.

Il est aussi possible de privilégier le parallélisme de données en appliquant le concept de SCMD (*Single Configuration Multiple Data*). Dans ce cas, la même configuration est "exécutée" simultanément dans tous les DPRs concernés. Ce concept est une évolution

du concept SIMD dans lequel plusieurs opérateurs exécutent une même opération sur des jeux de données différents. Dans le cadre du SCMD, le partage des configurations n'est plus limité aux seuls opérateurs mais est étendu aux DPRs.

B.1-2 Le flot de conception de DART

La chaîne de développement de DART, présentée sur la figure 3-14, est basée sur l'utilisation conjointe d'un front-end permettant la transformation et l'optimisation de code C (Suif [27]), d'un compilateur reciblable (environnement CALIFE [51] : outils CDART et ACG) et de l'outil GDART utilisant les techniques de synthèse de haut niveau issues de l'environnement BSS (*Breizh Synthesis System*) [52]).

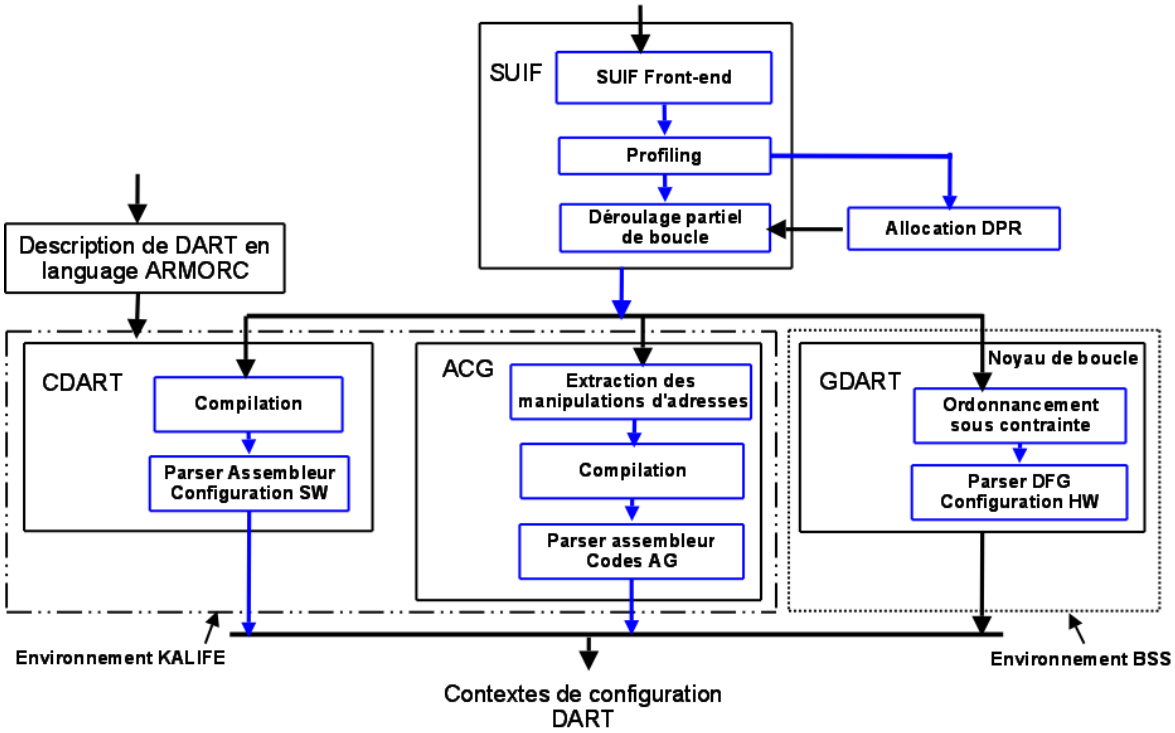

Figure 3-14 – **Flot de conception de DART**. *Ce flot utilise un code source qui est tout d'abord manipulé par un front-end SUIF. Celui-ci permet d'optimiser le code à haut niveau et d'extraire le parallélisme de l'application. Une passe de partitionnement permet de distinguer les traitements irréguliers, les manipulations de données et les traitements réguliers. Les deux premiers types de code seront traduits en codes binaires par les outils ACG et CDART. Les traitements réguliers sont transformés en configurations de DPRs par le biais de l'outil GDART.*

Le front-end SUIF permet d'optimiser le code à haut niveau, de générer un graphe de données et de contrôle (CDFG). Pour ce problème sont développées des passes spécifiques de déroulage de boucles et d'extraction de cœurs de boucles. Une passe de partitionnement permet ensuite de distinguer les traitements irréguliers, les manipulations de données et les traitements réguliers.

Les traitements irréguliers et les manipulations de données seront traduits en codes binaires via des passes classiques de compilation respectivement dans les outils ACG et CDART, à l'entrée desquels, une description en langage ARMOR de DART est fournie. Ces deux outils sont issus de l'environnement de compilation reciblable CALIFE.
Les traitements réguliers sont transformés en configurations de DPRs par le biais de techniques issues de la synthèse comportementale des circuits dédiées : GDART transforme les motifs de calculs en configurations de DPRs ; ce module se base sur l'infrastructure de l'outil de synthèse comportementale BSS pour transformer le code C d'entrée en un graphe flot de données GFD (Graphe de Flot de Données). La tâche de ce module se limite à déterminer la structure du chemin de données qui implémentera au mieux ce GFD et à transformer cette structure en une configuration matérielle des DPRs. Ceci est fait en plusieurs étapes :

- réduction de boucle ; en effet, la principale contrainte lorsqu'il s'agit d'ordonnancer un GFD en vue de l'implémenter sur DART est qu'il est préjudiciable de maintenir des résultats intermédiaires sur plus d'un cycle. En cassant les boucles critiques par permutation d'opérateurs, la réduction de boucle permet de réduire cette latence en un cycle. Les permutations d'opérateurs doivent cependant garantir le maintien de la fonctionnalité du GFD ;
- réduction de la profondeur du graphe ; dans le cadre des boucles imbriquées si le nombre d'itérations de la boucle de plus bas niveau est faible, un pourcentage important du temps d'exécution est consommé par les amorçages successifs des pipelines d'exécution. Une passe de parallélisation permet d'optimiser le GFD et d'en réduire la profondeur. L'algorithme est basé sur une fonction de recherche de suites d'addition ou de multiplications exécutées séquentiellement et candidates à la parallélisation. Ces dernières sont évalués suivant des critères de dates d'exécution. A ce stade, les opérations doivent être assignées aux opérateurs et les entrées/sorties doivent être assignés à des mémoires ;
- l'allocation mémoire gère un tableau de correspondance qui associe à chaque variable le numéro de la mémoire qui la stocke ;
- l'assignation des opérateurs a pour but de mettre à jour la table de correspondance et de générer le flot d'instructions permettant de spécifier la configuration matérielle

des chemins de données.

B.2 Formulation de la problématique et présentation de la méthodologie

La problématique que nous cherchons à résoudre est de trouver pour chaque tâche τ_i d'un graphe de tâche GdT, le nombre de partitions n^i qui minimise la surface nécessaire à l'application en créant des partitions homogènes, tout en garantissant une performance d'exécution optimale. Chaque tâche τ_i est modélisée par un graphe flot de données (GFD).

Il s'agit alors de calculer pour chaque tâche τ_i, la surface optimale (en nombre d'unités fonctionnelles) S^i permettant d'avoir les performances optimales. Ensuite l'accumulation des opérateurs successifs du GFD, en partant de son sommet jusqu'à l'obtention d'une surface supérieure ou égale à S^i, définit la première partition. Les suivantes sont réalisées de la même façon en réinitialisant le cumul de surface, ceci jusqu'à la fin du GFD. Le nombre de partitions n^i se calcule alors en utilisant la formule suivante :

$$n^i = \frac{Nb^i_{op}}{S^i} \tag{3-4}$$

Nb^i_{op} est un paramètre applicatif (voir section A.1 du chapitre II) exprimant le nombre d'opérations arithmétiques et logiques qui doivent être exécutées pour une tâche τ_i.
Soit une application modélisée en GdT à implémenter sur une architecture hybride reconfigurable. Nous supposons que le modèle de couplage CPU/ARD est déjà choisi. Pour chaque tâche τ_i du GdT, modélisée par un GFD, est associé le couple $\{Nb^i_{UF}, m^i\}$. Nb^i_{UF} est l'allocation en unités fonctionnelles de la tâche τ_i et m^i est le facteur de performance qui est un paramètre exprimant le compromis de performances calculatoires et énergétiques de l'exécution de la tâche τ_i calculé comme suit :

$$m^i = Taux^i_{Red\_E} \times A^i \tag{3-5}$$

$Taux^i_{Red\_E}$ est le taux de réduction de l'énergie dissipée et A^i est l'accélération de la tâche τ_i. Ce sont les métriques de performances que nous avons définies au paragraphe A.3 du chapitre II.
Notre objectif est de déterminer S^i qui n'est autre que le Nb^i_{UF} qui donne le m^i optimal en exploitant le parallélisme au niveau données. Nous allons décrire dans ce qui suit

l'algorithme servant à déterminer S^i.

Afin que l'implémentation de l'application soit efficace, pour chaque tâche τ_i une extraction du parallélisme intrinsèque au niveau données est effectuée par des passes de déroulage de boucle. Le déroulage partiel de boucles permet d'atteindre un large gain de performances à cause de l'extraction du cœur de boucle en éliminant la partie contrôle de la boucle, donnant ainsi la possibilité d'utiliser les techniques de parallélisation au niveau données. L'augmentation du parallélisme de l'application fait que le volume de configuration nécessaire à l'ARD augmente, ce qui conduira à une augmentation du nombre de contextes de configuration.
Par contre, à partir d'un certain degré de déroulage de boucle, la latence gagnée par l'exploitation du parallélisme de données dans l'exécution de la tâche est inférieure à la latence de reconfiguration supplémentaire causée par l'augmentation des contextes de configuration.
Le problème que rencontre la plupart des programmeurs est alors de savoir comment trouver le facteur de déroulage de boucle optimal qui mènera à la meilleure performance de l'architecture hybride.

Ce problème a été relevé dans [53] en cherchant pour un HCDFG donné, le facteur de déroulage F nécessaire pour obtenir le débit optimal. Celui-ci est obtenu à partir de l'équation 3-6 :

$$F = \left\lceil \frac{T_{max} \times PGCD(\Delta_{cr}, T_{cr})}{T_{cr}} \right\rceil \frac{\Delta_{cr}}{PGCD(\Delta_{cr}, T_{cr})} \tag{3-6}$$

Avec T_{max} la latence de l'opération la plus lente, Δ_{cr} le nombre de délais (repérés par les indices de tableaux dans le HCDFG) et T_{cr} la latence cumulée du cycle critique.

Dans le cadre de l'étape d'estimation intrafonction [45] (cf. sous section C.1 du chapitre I) et afin d'exploiter au mieux le parallélisme de l'application, Design-troter traite le problème différemment en cherchant le facteur de déroulage minimum pour ordonnancer une boucle avec une contrainte de temps T. Le problème peut-être reformulé comme suit : si une boucle nécessite T" cycles pour être ordonnancée sans déroulage, le facteur de déroulage permettant de gagner $G = T" - T$ cycles, est calculé par :

$$F = \frac{1}{1 - \frac{G}{T - d_{min}}} \tag{3-7}$$

$$d_{min} = max_{cycles}(\frac{T_{cr}}{\Delta_{cr}}) \tag{3-8}$$

Notre approche pour le calcul du facteur de déroulage de boucle F est de chercher celui qui permet des performances optimales en terme d'accélération de calcul et de taux de réduction d'énergie. Notre idée consiste alors à exploiter les modèles de performances des architectures hybrides reconfigurables définis dans le chapitre précédent. En effet, pour une architecture hybride donnée et après identification du couplage CPU/ARD exploité, nous faisons varier le facteur de déroulage de boucle et nous faisons un profilage de l'exécution de la tâche considérée et recalculons les métriques de performances A^i et $Taux^i_{Red\_E}$. Le facteur de déroulage de boucle F optimal correspond à celui où le facteur de performance m^i est maximal. Pour ce facteur de déroulage de boucle, nous déterminons par l'outil de synthèse architecturale la surface S^i (en nombre d'unités fonctionnelles de l'ARD) correspondante à ce facteur de déroulage de boucle.
Le pseudo algorithme 1 présente la structure de l'automatisation de la détermination du S^i pour chaque tâche τ_i.
Cet algorithme détermine pour chaque tâche τ_i le facteur de déroulage de boucle F

Algorithm 1 Pseudo algorithme pour chaque tâche

```
S^i ⇐ 0 ;
F ⇐ 0 ;// Facteur de déroulage de boucles
M ⇐ 0 ; // Facteur de performance
for j = 0 to N do
   Calcul de m^i(j) ; // Variation du facteur de déroulage
   if M < m^i(j) then
      M ⇐ m^i(j) ;
      F ⇐ j ;
   end if
end for
S^i ⇐ Nb^i_UF(F) ;// Détermination de la surface totale nécessaire
```

pour lequel les performances sont optimales. En effet, pour chaque facteur de déroulage de boucles j le facteur de performances $m^i(j)$ est recalculé à partir des paramètres issus du profilage de l'application. La surface optimale S^i (en nombre d'unités fonctionnelles) n'est autre que Nb^i_{UF} pour le facteur de profilage permettant d'avoir les performances optimales.

A titre d'exemple, pour l'implémentation du filtre de Nyquist (d'ordre 64) du récepteur WCDMA sur le cluster DART, nous pouvons dérouler la boucle du filtre 12 fois (ce qui correspond à une surface en nombre d'unités fonctionnelle de 24 UF) comme illustré

par la figure 3-15. Comme les échantillons à l'entrée du filtre sont complexes, nous exploitons la propriété SWP des opérateurs de DART. La puissance de calcul développée dans ce cas de figure par un cluster DART est de 6.2 GOPS et sa consommation de puissance lors de l'exécution avoisine les 143 mW.

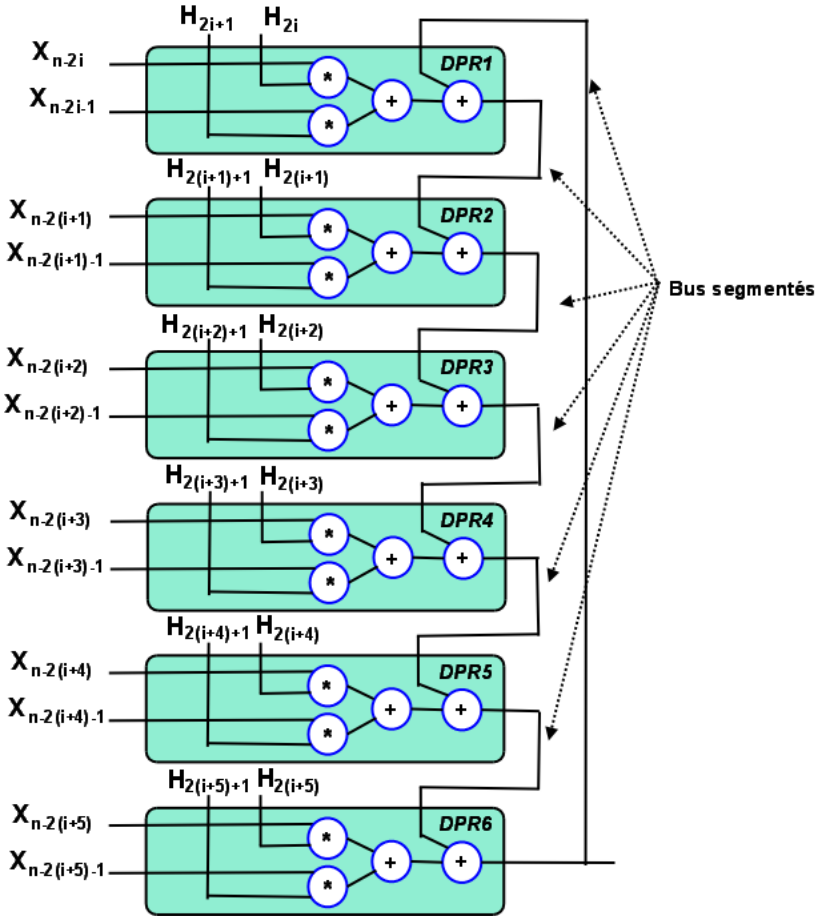

Figure 3-15 – **Implémentation du filtrage sur le cluster DART exploitant un déroulage de boucle (F=12)**. *Comme les échantillons à l'entrée du filtre sont complexes et que les coefficients sont réels, les opérations sont exécutées en exploitant le SWP. Chaque DPR est chargé de filtrer deux échantillons. Les accumulations intermédiaires sont passées d'un DPR à un autre à travers le réseau segmenté.*

B.3 Intégration de la méthodologie dans le flot de compilation d'une architecture hybride

La figure 3-16 présente l'intégration de cette méthodologie d'optimisation dans le flot de compilation d'une architecture hybride.

En effet, le flot commence par une représentation de l'application par un GdT. Chaque

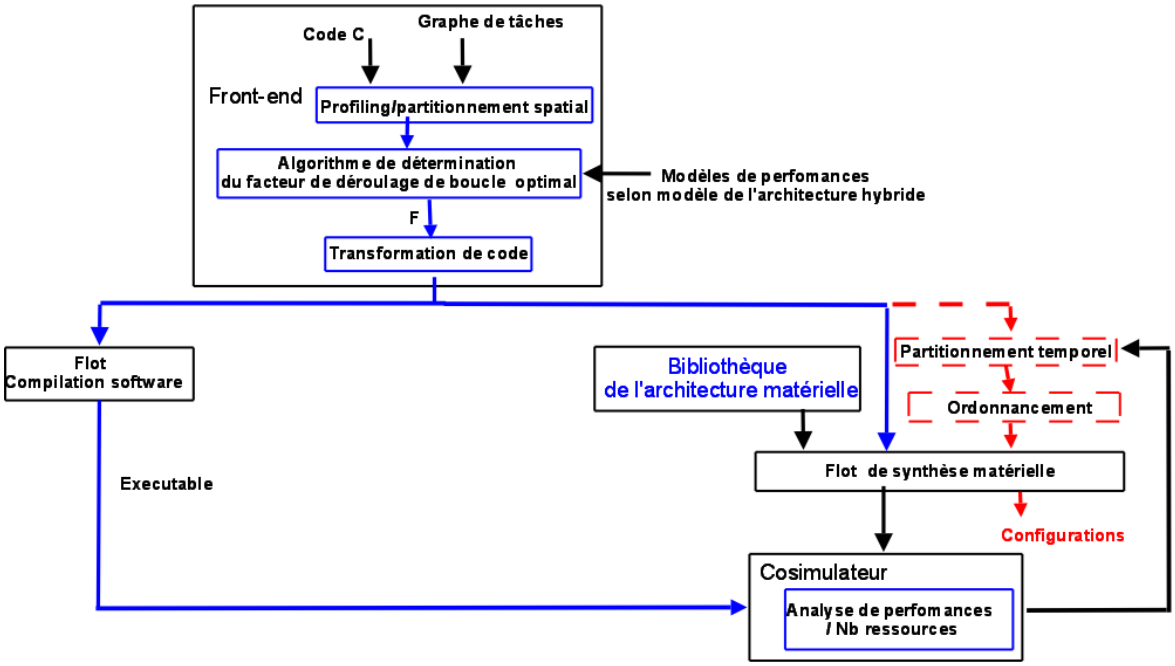

Figure 3-16 – **Flot de compilation d'une architecture hybride exploitant la méthodologie d'optimisation de la reconfiguration dynamique**.

tâche du GdT est décrite en C. Ce code source est manipulé par un front-end qui réalise les étapes suivantes.

1. *Profilage/partitionnement spatial*
   Cette étape vise à identifier pour chaque tâche les boucles candidates à être accélérées (nids de boucles : parties calculatoires consommant du temps dans l'application). Ces portions de code critiques en terme d'exécution seront "mappées" sur l'architecture reconfigurable dynamiquement et sont généralement des traitements réguliers auxquels il est souhaitable d'appliquer différentes techniques de parallélisation afin d'augmenter fortement les performances.

2. *Détermination du facteur de déroulage de boucle F*
   Cette étape fournit le facteur de déroulage de boucle optimal et ce en appliquant les modèles de performances de l'architecture hybride. En effet, pour chaque facteur de déroulage de boucle, A^i et $Taux^i_{Red\_E}$ sont recalculés. Le facteur de déroulage de boucle F optimal correspond à celui où le facteur de performance

m^i est maximal.

3. *Transformation du code*
   Le front-end assure les transformations du code afin d'exploiter le potentiel offert par les architectures matérielles que ce soit à grain fin ou à gros grain (diminution du format des données, transformation des boucles...).
   Dans ce flot de compilation, la transformation des boucles se fait par des passes de déroulage partiel de boucles avec le facteur de déroulage de boucle optimal déterminé dans la phase antérieure.

En sortie du front-end, le flot classique de compilation du CPU est appliqué. Pour la partie matérielle, le flot de synthèse sur l'ARD est appliqué directement (sans passer par les étapes de partitionnement temporel/ordonnancement) pour la première itération. Ceci permet de déterminer, à travers l'outil de synthèse, la surface S^i en nombre de ressources ARD qui correspond au facteur de déroulage de boucle optimal. S^i est fournie en entrée du bloc partitionnement temporel/ordonnancement. Le flot de synthèse matérielle est ré-appliqué et les contextes de configurations sont générés.

Nous avons adapté le flot de conception de DART présenté dans la section B.1-2 du chapitre III pour les besoins d'une implémentation sur une architecture hybride. Comme pour le flot classique de DART, le flot commence par une description en langage C des tâches de l'application (modélisée par un graphe de tâches). En premier lieu, le code complet est compilé, débogué et implémenté sur le CPU choisi. Ceci permet d'évaluer les performances temporelles du CPU quand il implémente seul l'application en question.

Ensuite, ce code source est manipulé par le front-end SUIF. Nous avons ajouté une première étape de profilage/partitionnement qui identifie les boucles candidates à être accélérées par DART, c'est à dire les parties calculatoires consommant du temps de l'application. Ces portions de code seront "mappées" sur le cluster DART et sont généralement des traitements réguliers.

Selon le couplage appliqué entre le CPU et DART, des instructions de contrôle traduisant les opérations que doit réaliser le CPU afin de gérer les traitements réalisés par DART, seront exécutées par le CPU.

Ensuite, des passes de déroulage partiel de boucle permettent de pipeliner le code et d'extraire les cœurs de boucles. Pour le code qui sera exécuté sur DART, un graphe

CDFG est généré et ensuite le flot classique de DART lui sera appliqué. Alors que pour le code à exécuter sur le CPU, le flot de compilation software classique lui est appliqué.

A l'aide des outils Code Composer Studio et Soft explorer [54], nous obtenons les performances temporelles et énergétiques du CPU (ici un DSP TMS320C6713). Pour DART, les codes binaires exécutables sur DART générés sont simulés au moyen de l'outil SCDART [50] permettant d'estimer les latences de traitement et l'énergie consommée lors de l'exécution de l'application.

A ce stade, une analyse de performances est réalisée en appliquant les modèles d'accélération et de réduction d'énergie dissipées que nous avons définis.

La méthodologie de découpage des contextes de configuration est basée sur l'algorithme de détermination du facteur de déroulage de boucle pour lequel les performances sont optimales. En effet, nous appliquons pour chaque tâche de l'application ce flot de conception plusieurs fois en faisant varier le facteur de déroulage de boucle et nous analysons à chaque reprise les performances. Pour le facteur de déroulage qui maximise les performances de cette tâche c'est à dire l'accélération et le taux de réduction de l'énergie dissipée, la simulation sur SCDART du code binaire généré par les outils ACG, CDART et GDART nous donne le nombre d'unités fonctionnelles pour chaque configuration DART. La plus petite entre ces valeurs d'UFs correspond à la surface associée à ce facteur de déroulage de boucle.
A ce stade, nous faisons générer les contextes de configuration de l'application en tenant compte de cette contrainte de surface.

C Synthèse

Dans ce chapitre, nous avons présenté une méthodologie d'adéquation algorithme architecture hybride reconfigurable permettant, suivant les paramètres de l'application, de déterminer le couplage CPU/ARD adéquat. Nous avons appliqué cette méthodologie sur une architecture hybride générique pour une implémentation d'un récepteur WCDMA dynamique. Cette implémentation a la particularité d'avoir des paramètres applicatifs variables. Ce qui nous a permis d'étudier l'impact des différents paramètres applicatifs et aussi algorithmiques sur le choix du couplage adéquat entre le CPU et l'ARD.

Comme perspectives pour réaliser du partitionnement temporel sur une architecture hybride, nous avons présenté une méthodologie d'optimisation de l'implémentation sur une architecture hybride exploitant au mieux la reconfiguration dynamique, basée sur la détermination de la surface (en nombre d'unités fonctionnelles) de l'ARD nécessaire pour avoir des performances optimales et ce en trouvant le facteur de déroulage de boucle qui assure le maximum de performances pour l'architecture hybride.
En intégrant cette méthodologie, nous avons adapté le flot de conception de l'architecture reconfigurable dynamiquement DART pour les besoins d'une implémentation sur une architecture hybride.
Dans le chapitre suivant, nous allons valider ces méthodologies en les appliquant sur le support de validation composée d'un CPU VLIW, le TMS320C6713 et du cluster DART. Sur cette architecture hybride, nous allons implémenter un démodulateur multimode DVB-T/H.

Chapitre

4

Validation

Sommaire

Pour la télévision numérique (DVB – Digital Video Broadcasting), la puissance de calcul requise par les traitements situés dans le décodeur dépasse la dizaine de milliards d'opérations par seconde (GOPS). Or, à partir de quelques milliards d'opérations par seconde, des solutions logicielles utilisant des processeurs de traitement du signal n'ont plus la puissance suffisante pour effectuer les traitements nécessaires ou consomment trop d'énergie. Aussi, l'utilisation d'un circuit intégré spécifique ASIC permet d'atteindre la puissance de calcul requise avec un coût raisonnable pour le marché grand public. Ainsi, chaque nouvelle application, chaque évolution dans les normes ou chaque amélioration entraîne la conception d'un nouveau circuit en mobilisant des ressources importantes.

L'utilisation des architectures hybrides reconfigurables dynamiquement, représente alors une opportunité pour implémenter cette application en temps réel tout en mainte-

nant la flexibilité. Dans cette perspective, nous nous sommes intéressés à l'implémentation d'un dé-modulateur DVB-T/H multimode sur une architecture hybride composée d'un cluster reconfigurable dynamiquement DART et d'un DSP TMS320C6713.

Nous présentons dans ce chapitre le système DVB-T/H ainsi que les architectures cibles pour la conception d'une architecture hybride. Nous appliquons ensuite les modèles et la méthode AAAH que nous avons présentés dans les chapitres précédents afin de trouver le couplage adéquat pour l'implémentation optimale du démodulateur DVB-T/H.
Une optimisation de l'implémentation sur l'architecture visant à exploiter de façon optimale la reconfiguration dynamique sera effectuée.
Enfin, nous présentons les résultats de l'implémentation du dé-modulateur DVB-T/H et nous mettons en exergue l'aspect multi-mode permettant d'accommoder la fiabilité et l'efficacité énergétique.

A Présentation et analyse du système DVB-T/H

A.1 Système DVB-T/H

DVB (*Digital Video Broadcasting*) [55] est le nom du projet européen associant plus de 180 structures de plus de 20 pays en Europe, qui a défini les standards de diffusion numérique Satellite (DVB-S), Câble (DVB-C) et Terrestre ou Hertzien (DVB-T). Il a également défini un système de cryptage pour le contrôle d'accès. D'autre standards DVB ont également été définis, comme le DVB-TXT (télétexte), DVB-ISC (services interactifs), DVB-MHP (moteur d'interactivité) et DVB-H (*DVB-Handled*) (mobile) qui est une technologie de diffusion de contenus audiovisuels vers des récepteurs mobiles ce qui permet la portabilité.

En fait, la combinaison des technologies Internet et de radiodiffusion ont créé de nouveaux services multimédia pour les utilisateurs de téléphones mobiles : DVB-H est désormais la base des applications datacasting mobiles sur IP (*Internet Protocol*). Le video-streaming DVB-H complète le DVB terrestre (DVB-T) sur des terminaux fonctionnant sur batterie, mais ne peut pas le remplacer et donc la réception de ces services est possible sur les réseaux terrestres existants DVB-T et en mobile même à grandes vitesses, grâce au procédé DVB-H consommant peu d'énergie. Ainsi, l'implémentation

d'un démodulateur DVB-T/H a suscité récemment beaucoup d'intérêts.

La compression des signaux audio et vidéo, la constitution du multiplex (multiplexage) et l'embrouillage sont communs à tous les supports de diffusion (terrestre, mobile, câble, satellite). Il n'y a que les techniques de transmission qui sont spécifiquement adaptées. Pour la compression des signaux audio et vidéo, DVB a retenu le standard MPEG-2. Pour le multiplexage, DVB a retenu le flux de transport MPEG. En ce qui concerne les systèmes de contrôle d'accès, seul l'embrouillage a été normalisé.

Contrairement au câble et aux voies satellitaires, les réseaux hertziens sont déjà en place et doivent être adaptés pour répondre à la qualité de service (QoS (*Quality Of Service*)) demandée par la diffusion de programmes de télévision numérique. En effet cette diffusion demande un Taux d'Erreur Binaire (TEB) très faible de l'ordre de TEB=10^{-11} soit 1 bit erroné pour 1 milliard de bits reçus.

Toutefois, le canal de transmission n'étant pas exempt d'erreurs qui viennent perturber le signal utile (bruit, interférences, échos...), il est nécessaire de prendre des mesures avant la modulation pour permettre la détection et la correction d'erreurs au niveau du récepteur. Ces mesures, dont la principale consiste à apporter de la redondance au flux de multiplexage, constituent l'essentiel du codage canal.

D'un autre côté, les échos (propagation par trajets multiples due à la présence d'obstacles) peuvent engendrer des fadings (évanouissements) qui sont des trous de transmission résultants de l'annulation du signal à un instant et une fréquence donnés. Par conséquent, lorsqu'on est en réception fixe, portable ou mobile, la probabilité de recevoir uniquement une onde directe provenant d'un émetteur est très faible. On va donc recevoir le signal émis par l'émetteur ainsi qu'une multitude de signaux atténués et retardés provenant des différents échos.
Le standard DVB adopte la modulation OFDM (*Orthogonal Frequency Division Multiplexing*) afin de s'affranchir des évanouissements dus aux échos en répartissant l'information sur un grand nombre de porteuses orthogonales modulées à bas débit.

La qualité de la transmission est gérée par l'adaptation de plusieurs paramètres du système DVB-T/H.
- Premièrement, le nombre de porteuses considérées permet de gérer la taille des cellules de transmission du réseau d'émetteurs isofréquences SFN (*Single Frequency*

Network) (dans lesquels tous les émetteurs transmettent simultanément le même contenu de programme sur la même fréquence) de large couverture (illimitée) ou à couverture locale (limitée). Deux valeurs de nombre de porteuses sont définies pour la télévision numérique terrestre et mobile, l'OFDM étant une modulation multi-porteuses, 1705 et 6817, connus sous les noms 2k et 8k, car ils sont réalisés via des transformées de Fourier rapides à 2048 et 8192 points.

Un autre mode (4k) est défini exclusivement pour les système DVB-H [56] et fournit un degré supplémentaire de flexibilité entre la taille de cellules et la capacité de réception mobile présentant ainsi des caractéristiques optimales pour la planification des cellules de radio mobile. Ce mode améliore les performances RF et permet la réception mobile à grande vitesse.

- Le principe de la modulation OFDM consiste à répartir des symboles de durée T_s (temps symbole utile) sur différentes porteuses modulées. Un intervalle de garde (GI (*Guard Interval*)), appelé aussi préfixe cyclique, dont la durée est paramétrable est inséré entre les symboles OFDM afin d'éliminer l'interférence entre deux symboles OFDM successifs ou entre les porteuses d'un même symbole OFDM. Le préfixe cyclique consiste simplement en une copie de la dernière partie du symbole OFDM avant le symbole considéré. A la démodulation, ce préfixe est simplement enlevé du signal reçu. Chacune des porteuses est modulée individuellement à bas débit pour que la durée utile T_s d'un symbole soit grande devant l'étalement des échos. La fréquence d'échantillonnage est fixée à 9,14 MHz.

  Quatre valeurs d'intervalle de garde (1/4 ; 1/8 ; 1/16 ; 1/32) sont utilisées. Plus le GI est élevé, plus la réception est robuste à des échos longs.
- Des modulations variant entre 4, 16 ou 64 QAM correspondent à différents compromis entre robustesse au bruit et débit utile.
- La protection des données est effectuée par ajout de redondance au signal transmis et ce à travers les codecs de Viterbi et de RS (*Reed Solomon*). Afin d'adapter la protection du signal contre les erreurs aux conditions du canal, plusieurs rendements de code convolutif "Inner Code Rate" (Δ) peuvent être choisis. L'augmentation du rendement du code augmente la redondance fournissant plus de protection contre les erreurs et ainsi plus de robustesse au signal. Ceci se paie par une augmentation de la complexité algorithmique et de la dissipation d'énergie.

Le tableau 4-1 donne les paramètres de configuration d'un système DVB-T/H pour un canal 8 MHz.

Un système DVB-T/H possède plusieurs modes proposant différents débits utiles. Le débit utile est calculé avec l'équation suivante [57] :

$$debitutile = K \times V \times \frac{1}{1+GI} \times \frac{1}{T} \times \Delta \times \frac{188}{204} \qquad (4\text{-}1)$$

Mode	2K (1705 porteuses) 4K (3409 porteuses) 8K (6817 porteuses)
Préfixe cyclique *GI*	1/4, 1/8, 1/16, 1/32
Modulation	QPSK (2 bits par porteuse) 16-QAM (4 bits par porteuse) 64-QAM (6 bits par porteuse)
Rendement du code convolutif Δ	1/2, 2/3, 3/4, 5/6, 7/8
Débit utile	4.98 - 31.67 Mbps

Tableau 4-1 – *Paramètres de configuration d'un système DVB-T/H.*

où K est le rapport entre le nombre de porteuses utiles et le nombre total des porteuses, V est le nombre de bits/symbole déterminé par la modulation utilisée (V=2 pour QPSK, V=4 pour 16-QAM et V=6 pour 64-QAM), T est la durée totale du symbole OFDM, *GI* est l'intervalle de garde et Δ est le rendement du codeur convolutif. Le rapport $\frac{188}{204}$ représente le rendement du codeur RS. En effet, Le flux de transport MPEG-2 TS d'entrée d'un émetteur DVB-T/H est organisé en paquets de longueur fixe de 188 octets. Le codeur RS est un codeur en bloc qui va ajouter 16 octets de parité (de redondance) derrière chaque paquet TS.

L'équation 4-1 montre l'impact des différents paramètres du système DVB-T/H :
- un intervalle de garde important (224 μs en mode 8K) améliorera la protection contre les interférences co-canales mais détériorera le débit net d'information ;
- l'augmentation du rendement du code améliorera le débit net d'information mais diminuera la protection contre les erreurs et la robustesse du signal ;
- la modulation de la porteuse permet de choisir le nombre de bits par porteuse. Plus ce nombre est élevé, plus le débit net d'information sera important mais la transmission est plus sensible aux interférences (la séparation entre les états admissibles dans l'espace de phase est moins importante). Par conséquent, la couverture s'en trouve réduite ;
- le mode 8k, par rapport au mode 2k, présente l'avantage d'accroître les intervalles de garde et donc la robustesse de la réception aux échos longs. Par exemple, le mode 8k avec des valeurs d'intervalles de garde de 1/32 et 1/16, conduit à des échos de 28 μs et 56 μs respectivement. Le mode 2k conduit également à des échos identiques mais avec des valeurs d'intervalles de garde de 1/8 et 1/4.

Ainsi, le choix des différents paramètres doit être un compromis entre le débit net d'information à transmettre et la protection contre les erreurs du canal de transmission. Dans [58], les auteurs ont démontré que pour la DVB-T la configuration qui correspond à la transmission réalisant un compromis entre bande passante et robustesse estle mode 8k, 64-QAM, $GI = \frac{1}{4}$, $\Delta = \frac{1}{2}$. Le débit utile dans ce cas est de 14.93 Mbps.

Afin de répondre aux exigences de la qualité de service du réseau hertzien et d'assurer un TEB minimum de 10^{-11}, il faut choisir le mode DVB-T/H le plus approprié aux conditions du canal et le plus efficace de point de vue débit. D'un autre côté, pour les terminaux DVB-T/H fonctionnant sur batterie, il faut prendre en considération la dissipation d'énergie dans le choix du mode de fonctionnement DVB-T/H adéquat.
L'implémentation d'un démodulateur DVB-T/H trouve donc son intérêt sur une architecture hybride reconfigurable dynamiquement s'il est multi-modes c'est à dire qu'il bascule dynamiquement d'un mode à l'autre selon les caractéristiques du canal de transmission et la charge de la batterie. Une reconfiguration dynamique de l'ARD est nécessaire afin de supporter ces changements de mode.

A.2 Analyse du démodulateur DVB-T/H

Le signal DVB-T/H est organisé en trames (frames) successives de 68 symboles. Quatre trames successives forment une super-trame (superframe) de 272 symboles qui permet la transmission d'un nombre entier de paquets protégés RS de 204 octets.
Aussitôt après la mise en marche, après un "Reset" ou après une perte de synchronisation, le récepteur se place en phase "acquisition". La figure 4-1 présente le diagramme d'exécution des différentes sous-tâches de cette phase.
La première étape est la correction de l'offset fréquentiel entre l'émetteur et le récepteur. Deux flots I et Q (parties réelles et complexes), démultiplexés et codés sur 8 bits, sont à l'entrée de ce bloc.
Pour chaque symbole OFDM, la démodulation OFDM est réalisée en appliquant une FFT. Ensuite, une correction temporelle permet de déterminer le début d'un symbole grâce aux échantillons contenus dans l'intervalle de garde. L'étape de récupération de la porteuse, permet de déterminer précisément les positions des différentes porteuses pilotes d'un symbole. Ces porteuses pilotes seront utilisées afin d'estimer la réponse du canal.
Après 22 symboles OFDM (24,6 ms), le démodulateur DVB-T/H passe en phase "tracking". Le profilage sur MATLAB du démodulateur DVB-T/H, nous a permis de découper la phase "tracking" en 3 tâches successives. La figure 4-2 détaille les dif-

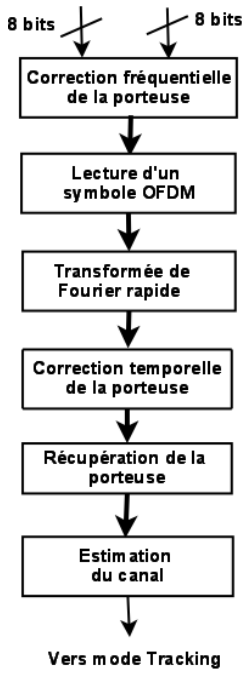

Figure 4-1 – **Diagramme des tâches du mode acquisition**. *La phase "acquisition" dure 22 symboles OFDM. La première étape est la correction de l'offset fréquentiel entre l'émetteur et le récepteur. Pour chaque symbole, la démodulation OFDM est réalisée en appliquant une FFT. Ensuite une correction temporelle permet de déterminer le début d'un symbole grâce aux échantillons contenus dans l'intervalle de garde. L'étape de récupération de la porteuse, permet de déterminer précisément les positions des différentes porteuses pilotes d'un symbole. Ces porteuses pilotes seront utilisés afin d'estimer la réponse du canal.*

férentes tâches du mode "tracking" ainsi que le format de données et la cadence de fonctionnement de chaque bloc.

La tâche 1 de la phase "tracking" a pour rôle de démoduler les symboles OFDM et d'égaliser la réponse fréquentielle du canal. Le de-mapper convertit chaque symbole complexe reçu en un mot de V bits selon la constellation utilisée par le modulateur. La tâche 1 met en sortie 8 symboles (de V bits chacun) d'un coup. Donc la tâche 2, constituée du décodeur et du désentrelaceur internes, doit attendre chaque 8 symboles afin de les traiter. Par contre la tâche 3, constituée du décodeur et du désentrelaceur externes ainsi que d'un débrasseur, ne peut s'exécuter que si elle a, en entrée, 204 octets c'est-à-dire une trame entière.

Nous avons analysé la complexité algorithmique des différents modes du dé-modulateur DVB-T/H. En nous basant sur un profilage MATLAB, le tableau 4-2 présente la répartition de la complexité algorithmique sur les différents bloc et ce pour les 3 modes du décodeur DVB-T/H suivants :

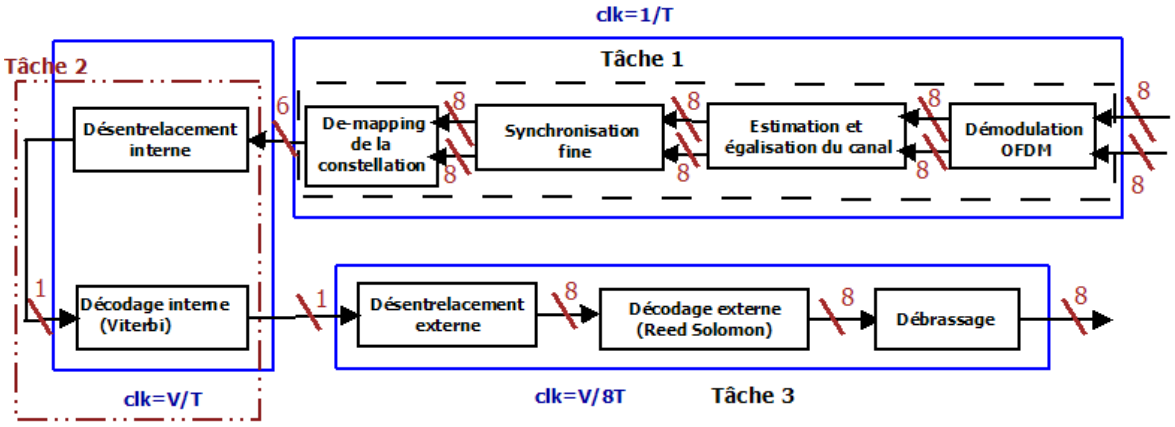

Figure 4-2 – **Phase "tracking" : tâches, format de données et cadence**. *La tâche 1 de la phase "tracking" a pour rôle de démoduler les symboles OFDM et d'égaliser la réponse fréquentielle du canal. Le de-mapper convertit chaque symbole complexe reçu en un mot de V bits selon la constellation utilisée par le modulateur. La tâche 2 est constituée du décodeur et du désentrelaceur internes. Par contre la tâche 3 est constituée du décodeur et du désentrelaceur externes ainsi que du débrasseur.*

- le premier mode (C_1)correspond à la transmission avec un débit maximal : Mode 8K, 64-QAM, $GI = \frac{1}{32}$ et $\Delta = \frac{7}{8}$. Le débit utile = 31,67 Mbps correspondant à la transmission **la moins bien protégée** ;
- le deuxième mode (C_2) correspond à la transmission avec un débit minimal : Mode 2K, QPSK, $GI = \frac{1}{4}$ et $\Delta = \frac{1}{2}$. Le débit utile = 4,98 Mbps correspondant à la transmission **la mieux protégée** ;
- le troisième mode (C_3) correspond à la transmission qui réalise un compromis entre bande passante et robustesse, nous nous sommes basés pour faire un tel choix sur [58] : Mode 8K, 16-QAM, $GI = \frac{1}{4}$ et $\Delta = \frac{1}{2}$. Le débit utile = 14,93 Mbps.

Le mode le plus complexe correspond aux paramètres suivants : mode 8k, 64 QAM, $GI = \frac{1}{32}$, $\Delta = \frac{7}{8}$. Pour ce mode, la complexité algorithmique totale du dé-modulateur DVB-T/H est d'environ 4 GOPS.

A.3 Portage du démodulateur DVB-T/H sur le cluster DART

Pour les différentes tâches du démodulateur DVB-T/H, nous vérifions leur portage sur le cluster DART en garantissant le respect de la contrainte temps réel pour le mode DVB-T/H dont la complexité algorithmique est la plus élevée (voir le tableau 4-2). En effet, le mapping garantissant un fonctionnement temps réel de ce mode sur

Configuration	C_1	C_2	C_3
Mode acquisition	720	491	595
Mode Tracking			
Démodulation OFDM	536	365	443
Décodage de Viterbi	3147	1207	2495
Décodage RS	26	13	26
Débrassage	5	5	5
Total	4434	2081	3568

Tableau 4-2 – *Complexité algorithmique en MOPS pour 3 modes du dé-modulateur DVB-T/H.*

l'architecture garantie le portage de n'importe quel autre mode moins complexe.

A.3-1 MAPPING DE LA PHASE ACQUISITION SUR LE CLUSTER DART

En plus de sa complexité algorithmique et des contraintes temps réels assez dures, toutes les opérations arithmétiques de la phase acquisition s'effectuent sur des nombres complexes. Le volume de la reconfiguration en octet du contexte de configuration de l'acquisition sur le cluster DART $\eta^{acquisition} = 104$ octets.

Correction fréquentielle de la porteuse

La correction de l'offset fréquentiel entre l'émetteur et le récepteur est effectuée par une multiplication complexe des échantillons dans le domaine temporel (avant la FFT) par $e^{2\pi kdf}$, $df = \frac{R_f}{N}$, R_f est la valeur de la fréquence de correction (normalisée par l'espacement entre les porteuses), N est le nombre de porteuses et k est l'index du pas. Comme les flots I et Q sont démultiplexés et les parties réelles et complexes sont codées sur 8 bits, nous avons la possibilité d'exploiter le SWP de DART (chaque opérateur traite en parallèle les parties hautes et basses des données qu'il reçoit sur ses entrées). La figure 4-3(a) présente l'implémentation de la multiplication complexe sur un DPR de DART demandant deux cycles de calcul.

Transformée de Fourier rapide

La démodulation OFDM pour la DVB-T/H est réalisée en appliquant une (FFT (*Fast Fourier Transform*)) 2k ou 4K ou 8k aux échantillons après correction fréquentielle. Pour le mode que nous avons considéré, le cluster DART doit exécuter une FFT sur

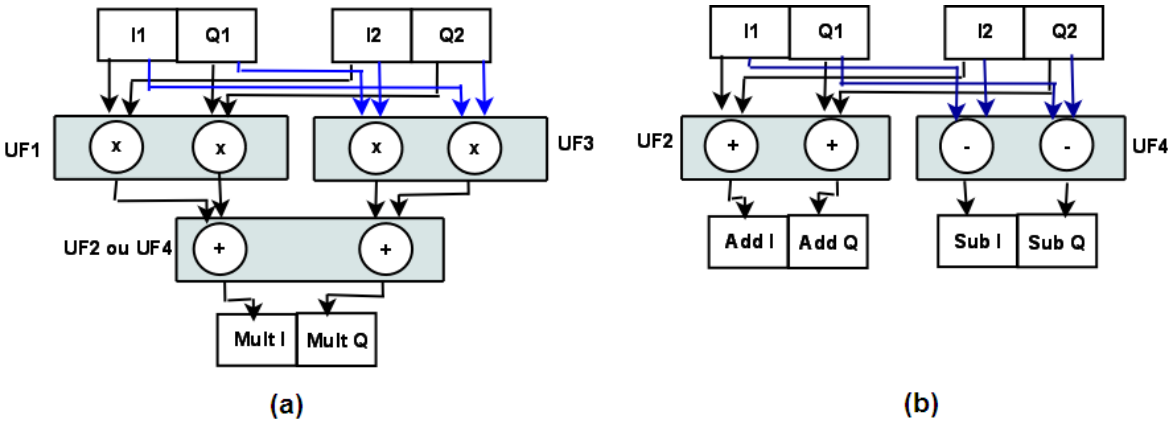

Figure 4-3 – **Implémentation de la multiplication complexe(a) et de l'addition/soutsraction complexe (b) en mode SWP sur un DPR de DART.** *L'exécution de la multiplication complexe sur un DPR de DART demande deux cycles de calcul. Par contre, une opération d'addition/soustraction est exécutée en un cycle.*

8192 points en une période symbole. Ce qui veut dire exécuter 53248 papillons chaque 896 μs. La contrainte temps réel est de 59.4 Méga papillons par seconde. Par conséquent, le cluster DART (dont la de fréquence d'exécution est de 200 MHz) doit exécuter au minimum un papillon tous les 3 cycles. Ceci peut être réalisé en utilisant le SWP pour le calcul des multiplications et des additions/soustractions complexes d'un papillon (Voir figure 4-3).

Correction temporelle

L'objectif de la correction temporelle est de déterminer le début d'un symbole grâce aux échantillons contenus dans l'intervalle de garde. L'estimation de l'offset temporel dépend de la longueur du préfixe cyclique et de la réponse impulsionelle du canal. Le préfixe cyclique est enlevé du signal reçu en ajustant la fenêtre FFT. Si la fenêtre FFT est positionnée dans la partie du préfixe cyclique non affectée par l'interférence avec le symbole OFDM précédent, le seul effet des symboles en sortie de la FFT sera un déphasage proportionnel à l'offset temporel. Le déphasage entre deux sous-porteuses est cependant constant pour toutes les sous-porteuses.
La correction temporelle est une multiplication complexe de chaque symbole complexe par $e^{j2\pi k\frac{T_{sh}}{N}}$, N est la taille de la FFT et T_{sh} est le déphasage temporel grossier qui ne doit pas dépasser la moitié de la durée du préfixe cyclique.
Pour le calcul de T_{sh}, nous utilisons l'algorithme introduit dans [59]. Cet algorithme calcule pour chaque symbole n (d'une porteuse k) une métrique $\lambda(n)$ correspondant au maximum de la corrélation entre les échantillons. Ensuite, le déphasage temporel

grossier T_{sh} correspond à $max_n \lambda(n)$. Pour chaque symbole OFDM, le cluster DART doit exécuter un calcul de corrélation avant de recevoir le symbole OFDM suivant. L'implémentation de la corrélation en mode SWP sur un seul DPR, (Figure 4-4(a)), permet d'exécuter quatre corrélations pour la contrainte temps réel imposée.

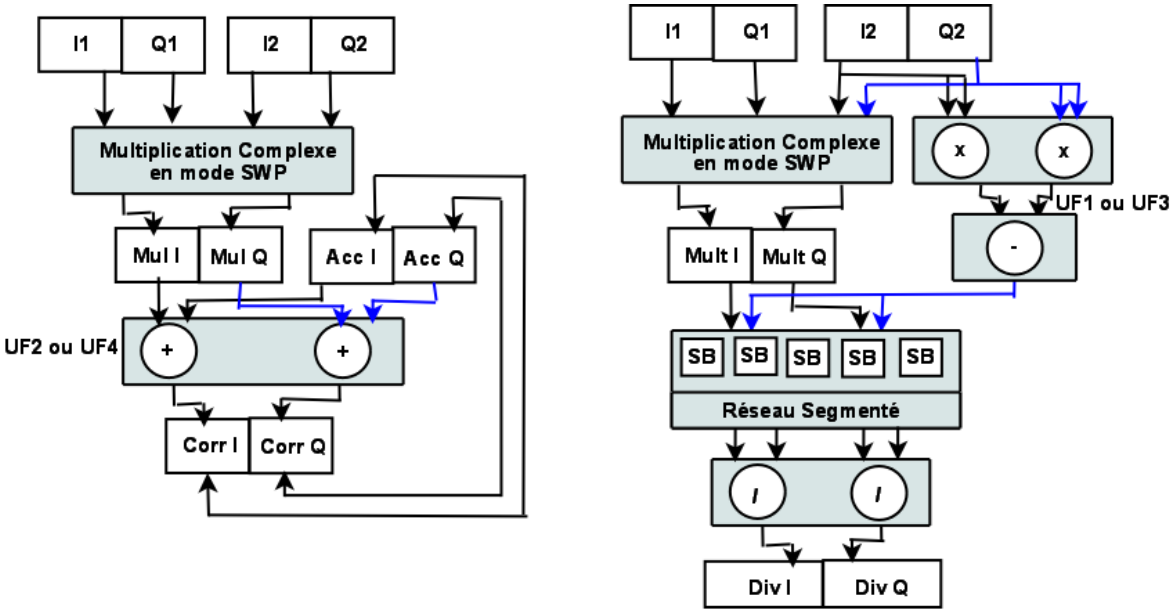

Figure 4-4 – **Implémentation d'une corrélation complexe (a) et d'une division complexe (b) en mode SWP sur le cluster DART**. *L'implémentation de la corrélation en mode SWP sur un seul DPR permet d'exécuter quatre corrélations pour la contrainte temps réel imposée. Pour l'implémentation de la division complexe, l'opérateur spécifique est utilisé afin de réaliser les divisions.*

Récupération de la porteuse

De manière à aider le récepteur à acquérir le signal et à l'informer des paramètres de modulation et de codage de canal, le signal OFDM inclut des porteuses qui ne sont pas modulées par le bitstream utile.
On distingue les porteuses pilotes continues (*Continual Pilot Carriers*) dont la position est fixe, les porteuses pilotes dispersées (*Scattered Pilot Carriers*) qui se décalent de trois positions à chaque symbole et les porteuses TPS (*Transmission Parameter Signalling*).
Les porteuses pilotes [60] sont transmises avec une puissance supérieure aux autres porteuses. Elles permettent au récepteur de se synchroniser et de faire une estimation du canal en vue de sa correction.

Les porteuses TPS transportent à faible cadence tous les paramètres de transmission au moyen d'une modulation BPSK différentielle (1bit/symbole) très robuste. Elles permettent une acquisition accélérée du signal au récepteur, ainsi qu'une réponse rapide à un éventuel changement de mode.

L'étape de récupération de la porteuse, "Carrier Tracking", permet de déterminer précisément les positions des différentes porteuses pilotes d'un symbole. Soit $R_{n,k}$ la k-ième donnée du n-ième symbole d'une trame OFDM. Nous calculons :

$$AutoCor(n) = \sum_{k \in E} R_{n,k} \times R^*_{n-1,k} \tag{4-2}$$

où E est un ensemble de porteuses pilotes et $AutoCor(n)$ l'autocorrélation des pilotes correspondant au symbole n. Si les calculs s'exécutent sur le bon ensemble, le résultat de la somme atteindra un pic. Le nombre de porteuses pilotes est $N_{pp} = 769$ en mode 8k, $N_{pp} = 193$ en mode 2K et $N_{pp} = 385$ en mode 4k.
En tant que corrélation, une auto-corrélation est mappée en mode SWP sur un DPR du cluster DART et nécessite ainsi $2 \times N_{pp}$ cycles (Figure 4-4(a)).

Estimation de canal

Soit $X_{n,k}$ et $R_{n,k}$ les n-ièmes symbole respectivement transmis et reçu de la k-ième porteuse. La relation entre $X_{n,k}$, $R_{n,k}$ et la fonction de transfert du canal $H_{n,k}$ est formulée comme suit :

$$R_{n,k} = X_{n,k} \times H_{n,k} + W_{n,k} \tag{4-3}$$

$W_{n,k}$ est le bruit additif gaussien du canal.

Après récupération de la porteuse, les échantillons correspondant aux sous-porteuses pilotes reçus sont extraits. Pour ces dernières, l'information transmise X(n_p,k) est connue du récepteur donc il est possible d'estimer la réponse du canal pour ces porteuses par une division complexe :

$$H_{n_p,k} = \frac{R_{n_p,k}}{X_{n_p,k}} \tag{4-4}$$

La figure 4-4(b) présente comment la division complexe est implémentée sur le cluster DART. Pour ce faire, l'opérateur spécifique du cluster est utilisé.

Ensuite, il faut interpoler cette réponse sur l'ensemble des sous-porteuses. Pour ce faire, l'estimateur LPI (*Low-Pass Interpolation*) présenté dans [61] est utilisé. Cet estimateur

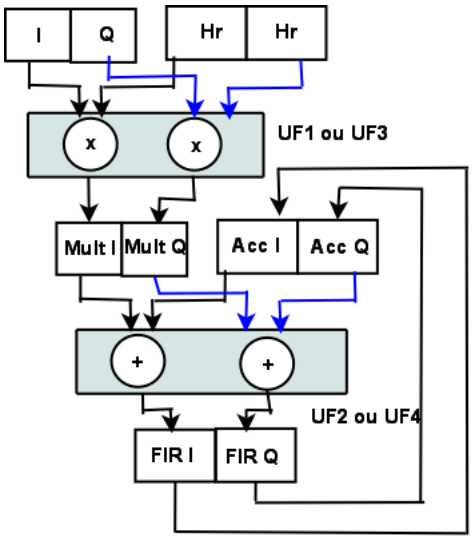

Figure 4-5 – **Implémentation du filtre FIR en mode SWP sur les DPRs de DART**. *Les cœfficients du filtre H_r sont réels et codés sur 8 bits. L'exécution d'une MAC est réalisée en deux cycles.*

est basé sur la méthode des "moindres carrés" avec une interpolation 1D ayant une complexité algorithmique modérée.
L'estimation des cœfficients du canal pour l'ensemble des sous-porteuses est réalisé par l'insertion de zéros dans la séquence, de longueur N_p des cœfficients H^p_{LS}, estimée pour les sous-porteuses pilotes afin d'obtenir un vecteur de longueur N. Un filtre FIR (*Finite Impulse Response*) passe-bas est ensuite appliqué. Les cœfficients du filtre H_r sont réels et codés sur 8 bits. La figure 4-5 présente l'implémentation d'un filtre FIR sur un cluster DART. L'exécution d'une MAC (*Multiplication-Accumulation*) est réalisée en deux cycles.

A.3-2 MAPPING DE LA PHASE "TRACKING" SUR LE CLUSTER DART

La phase "tracking" du démodulateur DVB-T/H est composée de trois tâches successives (voir figure 4-2). Ces tâches sont portées sur le cluster DART en vérifiant le respect des contraintes temps réel.

Tâche 1

Les sous-tâches démodulation OFDM et estimation du canal sont les mêmes que celles de la phase acquisition. Un égaliseur FEQ (*Frequency-domain EQualizer*) du canal est utilisé afin de compenser l'influence de la réponse impulsionnelle du canal. L'égaliseur du canal est globalement basé sur un diviseur complexe.

L'algorithme de synchronisation temporelle fine est ensuite appliqué. Cet algorithme [59] est exécuté tout les 8 symboles OFDM. Il est basé sur une corrélation des signaux reçus dans le domaine temporel avec les porteuses pilotes (dispersées et continues) dans le domaine temporel. Cette corrélation est mappée sur un DPR de DART en mode SWP.
Un de-mapper convertit chaque symbole complexe reçu en un mot de V bits. Afin de déterminer la constellation utilisée par le modulateur, le de-mapper utilise le facteur de modulation α et V. Ces informations sont transportées par les porteuses pilotes TPS. Nous utilisons le de-mapper à haute vitesse basé sur l'algorithme MUSCOD présenté dans [62]. Les opérations du de-mapper sont réels et exigent deux UALs.

Tâche 2

Cette tâche est décomposée en deux sous-tâches.

- Le désentrelaceur interne : l'opération d'entrelacement interne consiste à écrire un symbole OFDM dans une mémoire puis de les relire avec une loi d'adressage $H(q)$. Le désentrelaceur fait l'opération d'adressage inverse. La taille mémoire RAM nécessaire pour le désentrelacement est de 20 Ko.
  Le design d'un générateur d'adresses basé sur la fonction $H(q)$ peut être réalisé avec un registre à décalage de 6 bits.
- Le décodeur interne utilise l'algorithme de Viterbi qui cherche le chemin optimal à travers 2^{k-1} ($k = 7$ pour la DVB-T/H) nœuds (états) d'un diagramme en treillis de N étages. Cet algorithme consiste en trois unités :
  - L'unité pour le calcul des métriques de branches (BMU (*Branch Metric Unit*)) calcule la distance de hamming entre les symboles codés reçus et les symboles idéalement transmis. Ce calcul se réduit à 2 additions complexes et est ainsi mappé sur un DPR ;

  - L'ACSU (*Add-Compare-Select Unit*) : Cette unité permet de calculer quel nœud est le plus probable. En effet, à chaque nœud des 2^{k-1} nœuds du treillis arrivent deux branches venant de deux nœuds antécédents (a et b). A chacune de ces branches correspond une métrique de branche $MB(a)$ et $MB(b)$ et à chacun des nœuds correspondent deux métriques de chemin $MC(a)$ et $MC(b)$. Afin de calculer les probabilités cumulées pour chaque nœud de faire partie de la séquence émise, il faut faire l'addition des métriques de chemin et des métriques de branche puis la comparaison des deux métriques. D'où le nom d'ACS (Addition Comparaison et Sélection) pour ce bloc. La décision prise lors du choix entre les deux branches est conservée dans une mémoire. Cette décision se réduit à un bit appelé survivant. Ce survivant servira à reconstituer la séquence émise.

L'ACSU constitue le cœur de cet algorithme de Viterbi où le processeur élémentaire (PE), sera le bloc ACS. Chaque ACS exécute deux additions, une comparaison et une sélection. Il faut 2^{k-1} processeurs élémentaire interconnectés entre eux (64 ACS, ou 32 ACS doubles pour $k = 7$).

Pour un débit utile à l'entrée du décodeur de Viterbi de 39.28 Mbps, nous devons avoir une fréquence d'exécution minimale d'un ACS de 40 MHz. Implémenté sur un DPR de DART comme présenté sur la figure 4-6, l'ACS possède une fréquence d'exécution de 66 MHz ;

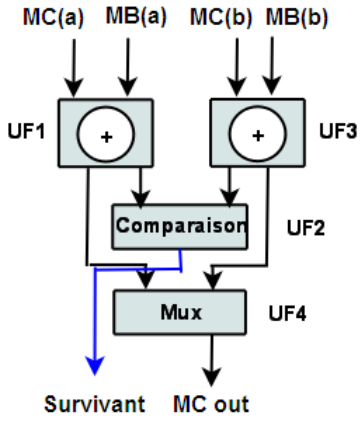

Figure 4-6 – **Implémentation d'un ACS en mode SWP sur les DPRs de DART**. *L'exécution d'un ACS est réalisée en trois cycles DART.*

– L'unité de Stockage et remontée des survivants (SMU (*Survivor Management Unit*)) : Pour chaque bit décodé en sortie du décodeur Viterbi, il faut une récurrence ACS (64 ACS) à exécuter. Donc pour une trame (204 octets) en sortie du décodeur, nous aurons $204 \times 8 = 1632$ récurrences ACS = 104 448 ACS. Pour chaque état, le survivant est codé sur 1 bit. Les survivants d'une récurrence ACS sont stockés sur quatre mots de 16 bits. Ce qui fait pour une trame DVB-T/H, un besoin mémoire de 105 Kbits = 14 Ko.
Nous avons choisi de stocker les vecteurs de survivants dans la mémoire principale (*dual port*) du cluster DART. Après n (profondeur de décision) itérations, la séquence émise est reconstituée par la relecture des survivants stockés. La fréquence nécessaire pour la reconstitution de la séquence émise est 2×66 MHz (fréquence d'exécution sur DART de l'ACS) soit 132 MHz.

Tâche 3

A l'émission, l'entrelaceur externe disperse les octets d'un paquet dans d'autres paquets. A la réception, l'ordre initial des échantillons est rétabli ce qui a pour effet de diviser les paquets d'erreurs en erreurs isolées et de faciliter la correction. L'entrelacement n'augmente pas la capacité de correction mais seulement son efficacité.
Le désentrelaceur externe utilisé dans le système DVB-T/H [55] consiste, au moyen d'une banque de 12 FIFOs et d'un dispositif d'aiguillage à I=12 branches, à transmettre 12 octets successifs (d'indices $j = 011$) chacun à travers une FIFO de longueur $M \times j$ avec M=17=$\frac{L}{I} = \frac{204}{12}$, où L est la longueur du paquet protégé et I est la profondeur d'entrelacement.
La mémoire nécessaire est égale à $\frac{I \times (I-1)}{2} \times J =$ 1122 octets (5,5 trames complètes). L'implémentation de la FIFO basée sur les registres à décalage occupe une surface large et dissipe une énergie de commutation importante. A cet effet, nous avons implémenté la FIFO sur une mémoire double port.

Pour chaque trame (204 octets) désentrelacée, un décodage RS (204, 188, T=8) est appliqué. Ce qui veut dire que pour 204 octets en entrée du décodeur, il y aura 188 octets en sortie et 8 octets sur 188 peuvent être corrigés. Si plus de 8 octets sont détectés comme erronés, le paquet est marqué comme défectueux.
Toutes les opérations du décodeur s'effectuent dans le corps de Galois GF(2^8) si bien que les opérations de multiplication et d'addition nécessitent soit un accès dans une mémoire, soit un opérateur spécifique. Le décodage Reed-Solomon s'effectue en cinq étapes [63].

- La première étape consiste à calculer le syndrome $S(x)$ correspondant au message. Ce syndrome se compose de $2t$ éléments (octets) et dépend uniquement des erreurs modifiant le mot de code (le bloc de 204 octets). Si tous les éléments du syndrome sont nuls, alors il n'y a aucune erreur à corriger. Le calcul du syndrome peut être fait à l'aide d'une FFT sur 256 points.
- La deuxième étape localise et évalue les erreurs. Pour cela, il faut résoudre l'équation clé :

$$S(x)\sigma(x) = \omega(x) \mod x^{16} \tag{4-5}$$

  où $S(x)$ est le syndrome, $\sigma(x)$ le polynôme localisateur d'erreurs et $\omega(x)$ l'évaluateur d'erreurs.
  Nous utilisons l'algorithme "Berlekamp-Massey" utilisant les registres de décalage afin de résoudre cette équation. Pour ce faire et pour un syndrome donné, $204 \times 4 =$ 816 cycles d'horloge DART sont nécessaires.

- Une fois le polynôme $\sigma(x)$ calculé, ses racines α^i sont évaluées à l'aide de l'algorithme "*Chien Search*".
- L'algorithme "*Forney*" est ensuite utilisé afin de calculer les valeurs des erreurs e_i,(i=0, 1,. . ., 15) telles que :

$$e_i = \frac{\omega(\alpha^i)}{\sigma'(\alpha^i)} \tag{4-6}$$

  où $\sigma'(\alpha^i)$ est le différentiel du localisateur d'erreur $\sigma(\alpha^i)$, une table ROM est utilisée pour calculer sa valeur inverse.
- La sortie de l'algorithme de "*Forney*" est le vecteur d'erreur de même longueur que le signal codé reçu. Ce vecteur contient des valeurs différentes de zéro aux endroits correspondants aux bits erronés. Ces erreurs sont corrigées par une addition dans le corps de Galois avec le mot de code qui a été conservé en mémoire pendant le temps des calculs.

La tâche 3 se termine par un dé-brassage (appelé aussi désembrouillage) correspondant à l'action inverse du brassage "*scrambling*" au niveau du démodulateur. En effet, ce dernier sert à effectuer une dispersion d'énergie, c'est à dire une répartition uniforme de l'énergie dans le canal d'émission afin d'éviter les longues suites de 1 ou de 0 qui créeraient des raies parasites dans le spectre du signal et qui empêcheraient la récupération de la porteuse. Le débrassage emploie le même circuit que celui utilisé par le brasseur en émission. C'est un générateur pseudo aléatoire PRBS (Pseudo Random Binary Sequence) de polynôme générateur $1+X^{14}+X^{15}$. Le schéma correspondant au générateur pseudo-aléatoire qui est le même pour l'embrouillage et le désembrouillage, présenté sur la figure 4-7, est un dispositif simple formé de 15 registres à décalage et d'un OU exclusif. Un octet ayant le mot de synchronisation 0xB8 est attendu pour initialiser le générateur pseudo-aléatoire en chargeant le mot <100101010000000> puis débrasser huit paquets d'affilée. La porte AND est activée à chaque octet de synchronisation pour ne pas les brasser et ainsi conserver ces points de repère.

La latence d'exécution totale de la tâche 3 sur le cluster DART est de 5600 cycles = 28 μs, valeur très inférieure à la contrainte temps réel équivalente à la durée de la trame OFDM= 60.928 ms.

A ce stade, nous avons vérifié que le démodulateur DVB-T/H peut être mappé sur le cluster DART tout en respectant les contraintes temps réel. Ceci a été rendu possible par l'utilisation du SWP des unités fonctionnelles pour les opérations complexes.
Dans la section suivante, nous allons varier les scénarios de couplage en appliquant les modèles d'accélération de calcul et de réduction d'énergie dissipée pour l'implémentation du démodulateur DVB-T/H multimode. Ceci nous permettra de déterminer le

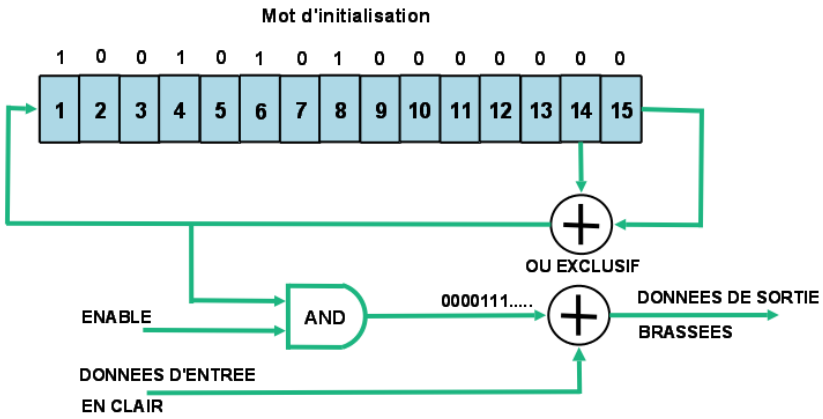

Figure 4-7 – **Principe du débrassage en DVB**. *C'est un dispositif formé de 15 registres à décalage et d'un OU exclusif. Un octet ayant le mot de synchronisation 0xB8 est attendu pour initialiser le générateur pseudo-aléatoire en chargeant le mot <100101010000000> puis débrasser huit paquets d'affilée. La porte AND est activée à chaque octet de synchronisation pour ne pas les brasser et ainsi conserver ces points de repère.*

couplage adéquat entre le cluster DART et le TMS320C6713.

Dans la section suivante, nous appliquons notre méthodologie AAAH présentée dans le chapitre 3 afin de déterminer le couplage adéquat entre le cluster DART et le CPU pour implémenter le démodulateur DVB-T/H.

B Application de la méthodologie AAAH

Nous nous proposons d'étudier le couplage entre le cluster DART et un processeur de traitement de signal le TMS320C6713 afin d'implémenter un démodulateur DVB-T/H multimode.
Le TMS320C6713 [64] est un DSP cadencé à 225 MHz et basé sur une architecture VLIW ce qui lui permet de traiter jusqu'à 8 instructions 32 bits par cycle (*IPC* maximum égal à 8) toutes les 4.44 ns (1/225MHz). La puissance calculatoire de ce DSP est de 1350 MFLOPS. Ce processeur possède 8Ko de mémoire cache (L1P et L1D) et notamment une mémoire interne (mémoire L2) pour permettre une rapidité d'exécution du code non négligeable. Un accès en DMA à la mémoire externe (SDRAM de 16 Mo) permet de palier à la latence provoquée par le bus PCI.

Outre l'avantage de l'utilisation d'une unité reconfigurable gros grain, notre architecture hybride possède un autre avantage l'utilisation d'un VLIW alors que la plupart

des architectures hybrides comportent un CPU de type RISC. Le VLIW permettra d'exploiter l'ILP afin d'accélérer le code. En plus, le CPU choisi est un DSP, donc adapté aux applications TDSI, qui sont généralement de type flot de données et gourmandes en puissance calculatoire.

Nous proposons de porter le démodulateur DVB-T/H sur notre architecture hybride reconfigurable dynamiquement et d'appliquer la méthodologie AAAH définie au chapitre 3 afin de déterminer suivant les besoins de l'application (paramètres applicatifs et algorithmiques) le couplage le plus adéquat entre le CPU et l'ARD permettant d'avoir une performance optimale du démodulateur DVB-T/H. L'objectif est de pouvoir envisager une commutation dynamique d'un mode à un autre.

Afin d'appliquer les différents modèles de couplages aux différentes tâches du démodulateur DVB-T/H, nous définissons tout d'abord les lois de communication associés à l'architecture hybride DART/TMS320C6713. Ensuite, nous faisons varier les scénarios de couplage et nous appliquons les modèles d'accélération et du taux de réduction de la dissipation d'énergie développés au chapitre 2 et nous analysons les performances.

B.1 Détermination des lois de communication

Latence de communication

La latence de communication d'un octet depuis ou vers le système mémoire γ^i_{com} est la somme des latences dues aux communications vers les mémoires internes et externes. En appliquant l'équation 2-12, la latence de communication d'un octet depuis ou vers la mémoire interne du TMS320C6713 s'écrit :

$$\gamma^i_{com_{int}} = (1 - \mu^i)(1 + \frac{50}{4})ns \tag{4-7}$$

μ^i est le taux d'accès en mémoire externe pour une tâche τ_i.
En effet, la latence de transfert d'un octet sur le bus système (de largeur 32 bits) du TMS320C6713 est de 1 ns et la latence d'accès à la mémoire interne du TMS320C6713 est de 50 ns. Comme la largeur du mot mémorisable est de 32 bits (4 octets), la latence d'accès d'un octet à la mémoire interne est de $50/4 = 12.5ns$.

L'application de l'équation 2-13 au TMS320C6713 nous permet d'exprimer la latence

d'acheminement d'un octet depuis ou vers la mémoire externe comme suit :

$$\gamma^i_{com_{ext}} = \mu^i(\frac{3.75\beta}{ch_{I/O}} + 0.2(1-\beta) + \sigma + 10\Omega)ns \quad (4\text{-}8)$$

Le premier terme $\mu^i(\frac{3.75\beta}{ch_{I/O}})$ de cette équation représente la latence de transfert sur le bus périphérique si le transfert des données utilise ce médium et non un bus spécifique (ou $\beta = 1$). Le bus périphérique du TMS320C6713 est un bus PCI (32 bits) à 266 Mo/s. Ce qui fait que la latence de transfert d'un octet est de 3.75 ns. $ch_{I/O}$ est le taux de charge du bus périphérique.
Le deuxième terme $\mu^i(0.2(1-\beta))$ représente la latence de transfert dans le cas d'un couplage DART/TMS320C6713 en point à point c-à-d que le transfert des données se fait à travers le bus spécifique ($\beta = 0$). Dans ce cas, une connexion point à point à travers l'EMIF (*External Memory Interface*) assure un accès simultané du DSP et de l'ARD à une mémoire SDRAM 32 Mo avec une bande passante de 5 Go/s. Ce qui fait que la latence de transfert d'un octet est de 0.2 ns.
Les deux derniers termes représentent les latences d'accès d'un octet aux mémoires externes au processeur à savoir la mémoire principale (de 16 Mo et connectée à travers le bus périphérique) et celle spécifique rajoutée à travers l'EMIF dans le cas d'un couplage point à point. L'accès en mémoire principale est entièrement en DMA, ce qui fait que le taux d'accès en DMA α^i est de 100% et la latence d'accès d'un octet est de 1 ns. La latence d'accès d'un octet à la mémoire spécifique est de 10 ns. σ vaut 1 si le cluster DART utilise la mémoire externe principale du TMS320C6713 pour le chargement des données et des contextes de configuration et vaut 0 sinon. Ω vaut 1 si le cluster DART utilise la mémoire spécifique et 0 sinon.

Dissipation de puissance due aux communications

Nous appliquons l'équation 2-18 au TMS320C6713 afin de calculer pour chaque tâche τ_i, la puissance P^i_{Com} dissipée lors d'un seul transfert depuis ou vers le système mémoire. Cette équation s'écrit dans ce cas :

$$P^i_{Com} = 0.37(1-\mu^i) + 0.53\mu^i(mW) \quad (4\text{-}9)$$

μ^i est le taux d'accès mémoire externe. La puissance dissipée lors d'un transfert vers/-depuis la mémoire interne du TMS320C6713 est $P_{int} = 0.37$ mW. Celle dissipée lors d'un transfert vers/depuis la mémoire externe du TMS320C6713 est $P_{ext} = 0.53$ mW. P_{int} et P_{ext} sont des paramètres architecturaux.

Nous étudions par la suite les différents couplages entre le cluster DART et le DSP TMS320C6713 en appliquant les modèles d'accélération de calculs et de réduction d'énergie définis au chapitre 2 afin de déterminer le couplage optimal du point de vue des performances pour le démodulateur DVB-T/H.

B.2 Détermination de l'accélération et du taux de réduction d'énergie

La première métrique d'évaluation de performances que nous avons adoptée est l'accélération apportée par l'exécution de l'application sur l'architecture hybride par rapport à l'exécution sur le CPU seul. Au niveau du chapitre 2, nous avons montré que cette accélération est égale à : $gain \times (1 - Taux_{cycles\_perdus})$.

Le gain défini par $\frac{T^i_{execCPU}}{T^i_{exec_{hybride}}}$, représente le gain en accélération apporté par l'exécution de l'application sur l'ARD par rapport à l'exécution sur le CPU sans prise en compte des cycles perdus dans le contrôle, la reconfiguration et le transfert des données et des contextes de reconfiguration.

Le taux de cycles perdus par l'architecture hybride est le rapport entre la latence perdue (contrôle, reconfiguration, transfert des données et des contextes de configuration) et la latence totale écoulée dans le traitement (T^i_{total}). Ce taux de cycles perdus est calculé en appliquant l'équation 2-7.
La reconfiguration dynamique du cluster DART est partielle, ainsi la latence de reconfiguration $T^i_{confARD}$ d'une tâche τ_i n'est pas aperçue par le cluster DART puisque ce dernier exécute le contexte précédent au moment de la reconfiguration du contexte en question.
La deuxième métrique est le taux de réduction d'énergie $Taux_{Red\_E}$, étant le gain potentiel en performances énergétiques par rapport à celles du CPU seul et est calculé en appliquant l'équation 2-10.

Pour les différentes tâches de la chaîne de réception DVB-T/H et pour les deux phases "acquisition" et "tracking", nous avons déterminé le taux de cycles perdus et la puissance dissipée par l'architecture hybride reconfigurable DART/TMS320C6713 ainsi que l'accélération et le taux de réduction de l'énergie dissipée par rapport au performances du DSP seul.

Dans cet objectif, l'application est portée en premier lieu sur le TMS320C6713. Chaque

tâche du démodulateur DVB-T/H sus-définie, est écrite à l'aide d'un code C.

Le code complet est compilé, débogué et implémenté sur le TMS320C6713 à l'aide de l'outil Code Composer Studio. Ce dernier, nous permet d'évaluer les performances temporelles du DSP quand il implémente seul l'application. L'estimation de l'énergie P^i_{CPU} consommée par le DSP pour l'exécution de chaque tâche τ_i a été estimée par l'outil Softexplorer [54].

A ce même code C, nous avons appliqué le flot de compilation classique de DART afin de générer les différents contextes de reconfiguration. En effet, le démodulateur DVB-T/H a été implémenté en temps réel sur le cluster DART en reconfigurant dynamiquement les différentes tâches définies dans la section antérieure.
Après un "*Reset*", un basculement d'un mode DVB à un autre ou une perte de synchronisation, le démodulateur multi-mode DVB-T/H reconfigure ses 6 DPRs (mode reconfiguration Hw) afin d'exécuter la phase acquisition pendant 16 cycles. Dans le cas d'un basculement d'un mode à un autre, cette latence de reconfiguration n'aura pas d'impact sur la latence globale de traitement puisque la reconfiguration est partielle et se fait en parallèle avec l'exécution du mode DVB-T/H précédent.
Comme la démodulation OFDM et l'estimation de canal sont communes entre les phases "acquisition" et "tracking", la configuration de 3 DPRs est maintenue. La reconfiguration du contexte de *la tâche 1* demande alors 7 cycles. L'implémentation de *la tâche 2* demande seulement 5 UFs (2 DPRs) et donc 7 cycles de reconfiguration. Comme 4 DPRs sont libres, nous avons porté *la tâche 3* en même temps que la tâche 2. Ces 4 DPRs sont reconfigurés en mode Sw (à chaque cycle) puisque les traitements relatifs à cette tâche sont irréguliers (Figure 4-8).

Selon le couplage appliqué entre le CPU et DART, nous avons généré des instructions de contrôle (code C) traduisant les opérations que doit réaliser le CPU afin de gérer les traitements réalisés par DART. Ce code est compilé, débogué et implémenté sur le TMS320C6713 à l'aide de l'outil Code Composer Studio afin d'évaluer les performances temporelles du DSP lors du contrôle du traitement sur le cluster DART. $P^i_{controlCPU}$ est estimée par l'outil Softexplorer [54].

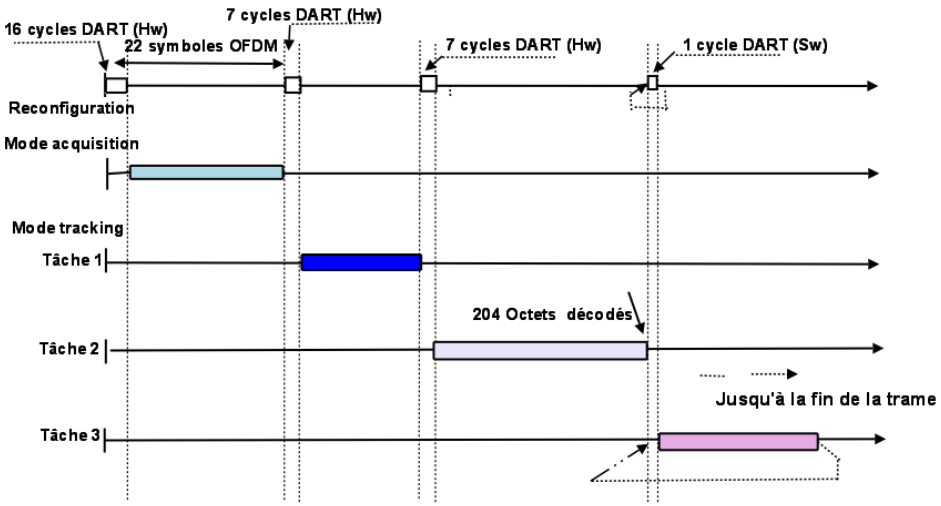

Figure 4-8 – **Diagramme de Gantt de l'implémentation du démodulateur DVB-T/H sur le cluster DART pour chaque super-trame reçue (204 octets).** *Le démodulateur multi-mode DVB-T/H reconfigure ses 6 DPRs (mode reconfiguration Hw) afin d'exécuter la phase acquisition pendant 16 cycles DART. Comme la démodulation OFDM et l'estimation de canal sont communs entre les phases "acquisition" et "tracking", la configuration de 3 DPRs est maintenue. L'implémentation de* la tâche 2 *demande seulement 5 UFs. Comme 4 DPRs sont libres, nous avons porté* la tâche 3 *en même temps que la tâche 2. Ces 4 DPRS sont reconfigurés en mode Sw (à chaque cycle). Les temps d'exécution des tâches de la figure ne reflète pas l'échelle du temps d'exécution réel sur le cluster DART.*

B.2-1 APPLICATION DES MODÈLES DE COUPLAGE À LA PHASE "ACQUISITION"

Bien que ne s'appliquant que sur 22 symboles OFDM, cette phase présente une grande complexité calculatoire (voir tableau 4-2) et opère sur des données complexes.

Les paramètres applicatifs de la tâche "Acquisition" sont :

- le volume de données en octet $\rho^{acquisition}$, à charger depuis les ressources de mémorisation varie entre 80 Ko et 150 Ko suivant le mode d'exécution DVB-T/H dont la complexité calculatoire est respectivement la moins et la plus importante. La distribution en volume de données des différentes sous-tâches de la phase acquisition, pour le mode de fonctionnement considéré (le plus compliqué), est représentée par la figure 4-9 ;
- le nombre d'itérations des différentes sous-tâches de l'acquisition diffèrent d'une sous-tâche à une autre et d'un mode de fonctionnement à un autre. En effet, la correction fréquentielle étant une multiplication complexe. Celle-ci s'exécute 8192

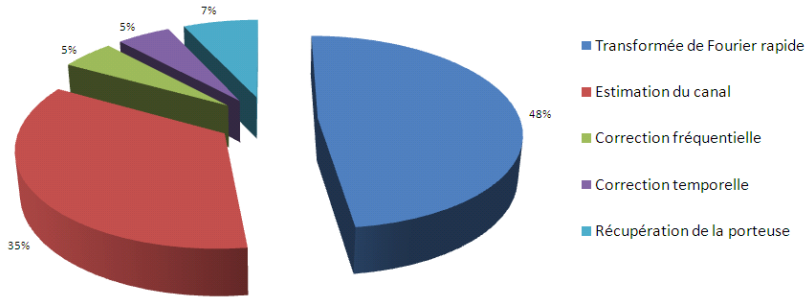

Figure 4-9 – **Distribution en volume de données lors du traitement en phase acquisition (total 150 Ko) pour le mode de fonctionnement : 64-QAM, intervalle de garde $\frac{1}{32}$, rendement du code convolutif $\frac{7}{8}$.** *L'essentiel du volume de données est consommé par la FFT et l'estimation de canal (83%)*

fois sur un symbole OFDM pour le mode de fonctionnement le plus complexe contre 2048 fois pour celle du mode le moins compliqué. Le papillon implémenté sur le cluster DART afin de calculer la FFT s'exécute en 53284 itérations pour le mode de fonctionnement le plus complexe alors que, pour celui le moins compliqué, le calcul de la FFT demande 11264 itérations ;

- le format des données de traitement de la phase acquisition $F_{data}^{acquisition} = 8$ bits ; les données sont complexes et les flots I et Q sont démultiplexés ;
- la complexité algorithmique en MOPS est $C^{acquisition} = 720MOPS$ pour le mode de traitement le plus complexe (voir le tableau 4-2).

Nous allons utiliser ces paramètres applicatifs, afin de déterminer la variation de l'accélération de calcul et du taux de réduction d'énergie en fonction du couplage entre le cluster DART et le TMS320C6713.
Le nombre de cycles perdus a été réduit par le chevauchement pour chaque tâche entre le chargement des données de l'échantillon $n+1$ en même temps que l'échantillon n est en train d'être exécuté. Ce chevauchement est initié et synchronisé par le CPU d'autant plus que le pipeline logiciel permet l'exécution en parallèle des différentes itérations. Par exemple, lors de l'exécution de la correction temporelle pour l'échantillon n, la récupération de la porteuse est en train d'être exécutée pour l'échantillon $n-1$. Ceci n'est pas applicable pour la FFT qui s'exécute pour un symbole OFDM entier.

DART couplé en UFRD au TMS320C6713

Le principe de l'architecture d'un CPU VLIW, est que plusieurs unités fonctionnelles UF sont inter-connectés à travers une file de registres multi-port. Cet aspect présente

une opportunité pour coupler DART en UFRD au TMS320C6713, en connectant le cluster DART comme unité fonctionnelle au fichier de registre multi-port déjà existant. Pour des besoins de reconfiguration dynamique, il faut que le cluster DART ait accès à la mémoire directement pour les transferts des contextes de configuration en même temps que l'exécution. Mais ceci implique des instructions de type mémoire-mémoire ce qui ne correspond pas au modèle registre-mémoire qu'utilise le CPU. Pour remédier à cela, nous avons utilisé un mode d'adressage indirect : les adresses mémoire source et destination sont stockées dans la file de registre du processeur. Une instruction UFRD fait référence aux registres source et destination là ou les adresses mémoire sont stockées ainsi qu'à la taille des blocs mémoire source qui seront lus à partir de l'adresse source et des blocs mémoires destination qui seront écrits à partir de l'adresse destination.

Comme la communication entre le cluster DART et le VLIW est réalisée à travers la file de registres et la hiérarchie mémoire, le code C écrit auparavant pour le VLIW n'a pas besoin de ré-écriture et est pris en main par le flot de conception de l'architecture hybride automatiquement. En effet, en langage de haut niveau, les variables locales sont placés dans le fichier de registre par contre les variables statiques sont placées dans l'espace mémoire qui sont partagés avec l'UFRD. Aucune ré-écriture du code n'est nécessaire puisque le CPU n'a pas besoin de communiquer ces variables à l'UFRD.
Le tableau 4-3 montre des comparaisons des performances par rapport à ceux du VLIW seul.

Tâche	Nbre. Cycles perdus	Taux Cycles perdus(%)	Accélération versus CPU seul	Puissance dissipée (mW)
Correction fréquentielle de la porteuse	4905	37.5	5.0	10.0
FFT	56460	51.5	2.2	31.4
Correction temporelle	4088	20	2.8	3.90
Récupération de la porteuse	539	6.2	10.5	5.46
Estimation du canal	1968	7.4	6.8	27.3
Total (Acquisition)	67960	38.2	3.13	78.06

Tableau 4-3 – *Performances pour un symbole OFDM de l'implémentation de l'acquisition sur l'architecture hybride DART/TMS320C6713 avec un couplage UFRD.*

La première sous-tâche "correction fréquentielle de la porteuse", présente un large nombre de cycles perdus (37.5 % du total des cycles d'exécution) ainsi qu'une puissance dissipée non négligeable bien qu'elle ne présente pas un large volume de données à transférer. Ceci s'explique par le fait que cette première tâche supporte tous les cycles perdus dus à la latence de reconfiguration.

Comme les ressources mémoires propres au cluster DART ne suffisent pas pour supporter les échanges mémoire importants dûs à la FFT, l'exécution des papillons de la FFT est accompagnée par un nombre de cycles perdus important. Cependant, comme l'exécution de la FFT pour un symbole OFDM signifie que le papillon mappé sur un DPR de DART s'exécute en 53284 itérations, ceci amortit le coût des cycles perdus lors du transfert des données depuis ou vers la mémoire interne entre les différentes exécutions.

DART couplé en CRD au TMS320C6713

En couplage CRD, le cluster DART est connecté sur le bus système du DSP. Ainsi l'ARD a accès à toute la hiérarchie mémoire du processeur. Afin d'assurer l'aspect de reconfiguration dynamique qui impose un chargement simultané des données et des contextes de configuration, nous avons connecté le cluster DART à un cache de 4 Ko à partir duquel il charge les instructions de reconfiguration.
En outre, nous avons utilisé des queues (FIFOs) mémoires afin d'augmenter significativement la cadence (voir fig.4-10). Quand une tâche accède en un cycle plusieurs fois à la mémoire interne, les files d'attente mémoire permettent de ne pas avoir un "goulot d'étranglement" sur le bus interne et de réduire le taux de défaut de cache. Par exemple, la "correction fréquentielle de la porteuse" fait 4 accès mémoire par itération. Sans les queues mémoire, la congestion sur le bus mémoire limitera la cadence maximale à une itération tous les 4 cycles. Avec l'utilisation des queues mémoires, l'intervalle des itérations peut être réduit à 2 et même à 1 cycle.

Les résultats issus de l'expérimentation du couplage en CRD entre le cluster DART et le TMS320C6713 ainsi que l'application des lois de communication pour l'exécution des tâches de la phase "acquisition" sur un symbole OFDM, sont présentés dans le tableau 4-4. Le nombre de cycles total représente le nombre de cycles nécessaires à l'exécution sur le cluster DART incluant les cycles perdus pendant le chargement du contexte de reconfiguration et le transfert des données entre les tâches. L'accélération totale est de 4.1 versus le TMS320C6713 seul.

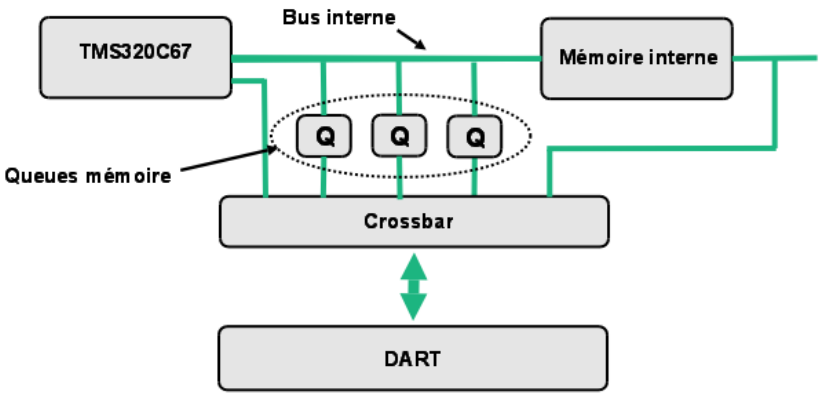

Figure 4-10 – **Positionnement des queues mémoire entre la mémoire interne et le cluster DART**. *Quand une tâche accède en un cycle plusieurs fois à la mémoire interne, les files d'attente mémoire permettent de ne pas avoir un "goulot d'étranglement" sur le bus interne et de réduire le taux de défaut de cache.*

	Nbre. cycles perdus	Taux Cycles perdus	Taux d'accès en mémoire externe (%)	Accélération versus CPU seul	Puissance dissipée (mW)
Correction fréquentielle de la porteuse	16384	66.7	7.3	2.7	8.33
FFT	50133	48.5	20.0	2.3	27.01
Correction temporelle	8388	34.0	4.6	3.33	2.87
Récupération de la porteuse	2356	22.4	2	8.5	4.66
Estimation du canal	8192	25.0	19.4	5.56	24.67
Total (Acquisition)	85453	43.6	15.7	3.0	67.54

Tableau 4-4 – *Performances pour un symbole OFDM de l'implémentation de l'acquisition sur l'architecture hybride DART/TMS320C6713 en modèle CRD.*

Nous remarquons une augmentation significative du nombre et du taux de cycles perdus par rapport au couplage UFRD sauf pour la FFT. Cette augmentation est expliquée par le fait que la latence due aux transferts des données est plus importante pour ce couplage. Cependant, cette augmentation des latences de transferts s'accompagne d'une diminution de la latence de contrôle CPU. Cette diminution est d'autant plus significative que le volume de données augmente (et par voie de fait que le nombre

d'accès mémoires augmente). Ce qui explique que, pour la FFT, le nombre de cycles perdus a diminué.

Bien que le nombre de cycles perdus augmente pour ce type de couplage, la puissance dissipée diminue par rapport au couplage UFRD. En effet, comme le taux de contrôle par le CPU est plus important en couplage UFRD qu'en couplage CRD ceci cause une énergie dissipée par le TMS320C6713 plus importante.

Bien que l'estimation du canal ne présente pas un taux de cycle perdus important, la puissance dissipée est importante. Ceci est dû à son taux d'accès mémoire externe important puisqu'un accès en mémoire externe dissipe une énergie plus importante qu'un accès en mémoire interne.

DART couplé en PRD au TMS320C6713

Pour ce mode de couplage, le cluster DART est couplé au bus PCI du TMS320C6713 comme un périphérique. Le tableau 4-5 présente les résultats de l'implémentation de la phase acquisition du démodulateur DVB-T/H sur le cluster DART couplé dans ce mode au DSP.

Tâche	Nbre. Cycles perdus	Taux Cycles perdus(%)	Accélération versus CPU seul	Puissance dissipée (mW)
Correction fréquentielle de la porteuse	28672	77.8	1.77	11.20
FFT	39936	42.8	2.5	30.4
Correction temporelle	12745	40.9	1.87	4.27
Récupération de la porteuse	3578	30.4	7.64	5.98
Estimation du canal	15565	38.8	4.38	31.63
Total (Acquisition)	100496	47.1	2.59	83.48

Tableau 4-5 – *Performances pour un symbole OFDM de l'implémentation de l'acquisition sur l'architecture hybride DART/TMS320C6713 en modèle PRD*

Nous remarquons que le couplage PRD augmente la latence de transfert de données. En effet, le taux de cycles perdus dans l'exécution des sous-tâches de "l'acquisition"

a augmenté par rapport à ceux dans le cas des couplages CRD et UFRD. Le nombre de cycles perdus de même a augmenté sauf dans le cas de la FFT. Ceci revient au fait que pour la FFT, le nombre d'itérations d'exécution est important et par conséquent le nombre d'accès mémoire l'est aussi. Comme pour le modèle CRD, le CPU initie et contrôle continuellement tous les transferts ce qui cause une latence supplémentaire assez importante. Par contre, pour le modèle PRD, l'unité reconfigurable a plus d'autonomie et n'a pas besoin de la supervision continuelle du CPU. Puisque tous les accès mémoires sont off-chip, la puissance dissipée par l'architecture hybride est plus importante que pour les 2 couplages précédents sauf pour la FFT, où la dissipation d'énergie (30.4 mW) est moins importante que celle pour le couplage UFRD (31.4 mW). En effet, nous avons montré au chapitre 3, que ce type de couplage trouve son intérêt par rapport aux autres couplages au niveau de la dissipation d'énergie pour les tâches s'exécutant en un nombre d'itérations $> 10^4$ car dans ce cas la dissipation due au contrôle par le CPU des différents accès devient prépondérante pour les couplages CRD et UFRD. Ce qui est confirmé dans le cas de la FFT qui s'exécute en 53284 itérations.

DART couplé en PPRD au TMS320C6713

Dans ce cas, le cluster DART et le DSP sont couplés en point-à-point avec un bus haute performances assurant une bande passante de 5 Go/s. Le DSP accède à une SDRAM, de 32 Mo partagée avec l'ARD, à travers l'EMIF. Un contrôleur mémoire assure le contrôle des accès à cette mémoire.
Le VLIW a accès aux fonctionnalités du cluster DART par l'intermédiaire des fonctions API (*Application Programming Interface*) qui permettent une utilisation transparente des ressources mémoires. Le CPU demande le chargement du contexte de configuration correspondant en appelant la fonction *loadconf(src,T)*, avec src l'adresse de début et T la taille du bloc mémoire à transférer. Le contrôleur mémoire comporte un contrôleur DMA.
Les résultats de l'implémentation de la phase "acquisition" sur notre architecture hybride en modèle PPRD sont présentés par le tableau 4-6.

Nous remarquons que ce mode de couplage présente l'avantage de réduire la latence due aux cycles perdus. En effet, de même qu'un couplage en périphérique, le contrôle du CPU est minime ce qui procure un gain considérable. Ce dernier n'est pas suffisant dans le cas du couplage PRD, quand le volume de données (en octets) à transférer entre les tâches n'est pas assez important, pour palier à la latence causée par les accès en mémoire externe. Dans le cas PPRD, un avantage supplémentaire est rajouté par la bande passante importante et non partagée dont bénéficie l'ARD. Ce qui permet de

Tâche	Nbre. Cycles perdus	Taux Cycles perdus(%)	Accélération versus CPU seul	Puissance dissipée (mW)
Correction fréquentielle de la porteuse	11878	59.2	3.26	10.21
FFT	34102	39.0	2.7	30.77
Correction temporelle	9830	37.5	2.2	4.01
Récupération de la porteuse	1804	18	9.2	5.61
Estimation du canal	6553	21	5.85	29.16
Total (Acquisition)	64167	36.6	3.21	79.76

Tableau 4-6 – *Performances pour un symbole OFDM de l'implémentation de l'acquisition sur l'architecture hybride DART/TMS320C6713 en modèle PPRD*

réduire considérablement le taux d'accès mémoire et par conséquent accélérer le calcul. Du point de vue de la puissance dissipée, ce mode présente les mêmes caractéristiques que le couplage en PRD dans la mesure où les différents accès mémoire se font off-chip.

Performances de l'architecture hybride

La figure 4-11 résume les performances (en taux de cycles perdus, accélération et taux de réduction de l'énergie dissipée) pour le traitement de la phase acquisition (mode de fonctionnement DVB-T/H le plus complexe).
Avec un taux de cycles perdus de 36.7 %, le couplage en point-à-point permet d'avoir la meilleure accélération de calcul par rapport aux autres couplages (3.11 par rapport au temps de traitement sur le TMS320C6713) sachant que la valeur maximale de l'accélération correspondant à un taux de cycles perdus de 0 % est de 4.6. Le couplage qui accélère le moins le traitement de la "phase acquisition" est en PRD avec une accélération de 2.5.
Nous remarquons que le taux de réduction d'énergie dissipée par notre architecture hybride par rapport la dissipation du DSP (s'il traite seul la phase acquisition) est important (environ 80%, soit une réduction de l'énergie par un facteur 5). Le couplage CRD (avec 85.2 %) présente le meilleur taux de réduction de dissipation d'énergie.

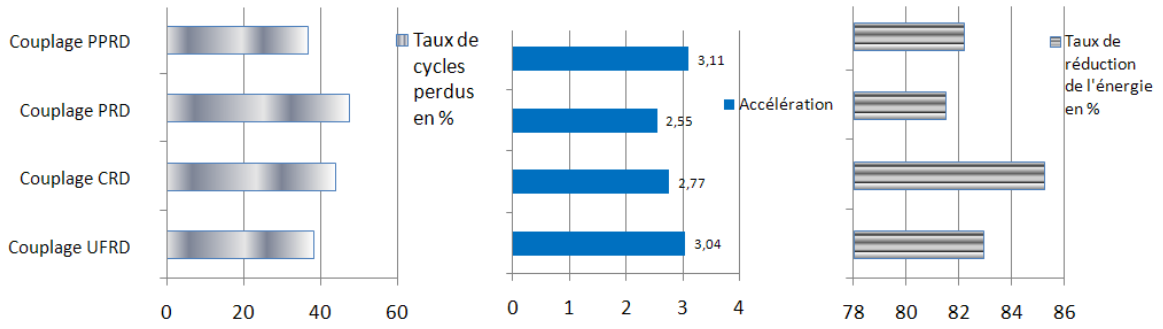

Figure 4-11 – **Performances de l'architecture hybride DART/TMS320C6713 pour les différents couplages sur la phase acquisition**. *Avec un taux de cycles perdus de 36.7 %, le couplage en point-à-point permet d'avoir la meilleure accélération de calcul par rapport aux autres couplages. Le taux de réduction d'énergie dissipée par rapport à la dissipation du DSP (s'il traite seul la phase acquisition) est intéressant (environ 80%). Le couplage CRD (avec 85.2 %) présente le meilleur taux de réduction de dissipation d'énergie.*

B.2-2 APPLICATION DES MODÈLES DE COUPLAGE À LA PHASE "*tracking*"

De la même manière que pour la phase acquisition, nous avons appliqué les différents modes de couplage aux différentes tâches de la phase "*tracking*".
La figure 4-12 présente les résultats d'exécution de la tâche 1 pour les différents couplages. Nous remarquons que le couplage PPRD présente les meilleures performances temporelles : une accélération de 3.5 correspondant à un taux de cycles perdus de 37% (56483 cycles). Le seuil d'accélération maximale pour cette tâche (correspondant à un taux de cycles perdus de 0%) est de 5.92.
Le couplage CRD présente les meilleures performances énergétiques (une puissance dissipée de 66.93 mW avec un taux de réduction de l'énergie dissipée de plus de 80%). En effet, la tâche 1 présente presque les mêmes paramètres applicatifs que la phase acquisition du démodulateur ce qui justifie que les résultats sont très similaires.

Les résultats de l'exécution de la tâche 2 pour les différents modes de couplage, sont présentés par la figure 4-13. Pour cette tâche très calculatoire (contenant le décodeur de Viterbi) avec une complexité algorithmique de plus de 3 GOPS, le couplage UFRD présente les meilleures performances temporelles (une accélération de calcul de 12.6 fois avec un taux de cycles perdus de seulement 13%). Ceci vient du fait que l'algorithme de Viterbi est très calculatoire mais les motifs de calculs sont répétitifs. Par conséquent, pour le couplage UFRD, ces paramètres sont enregistrées dans les registres du CPU et accédés directement par l'ARD sans besoins de trop de contrôle du CPU. L'accélération

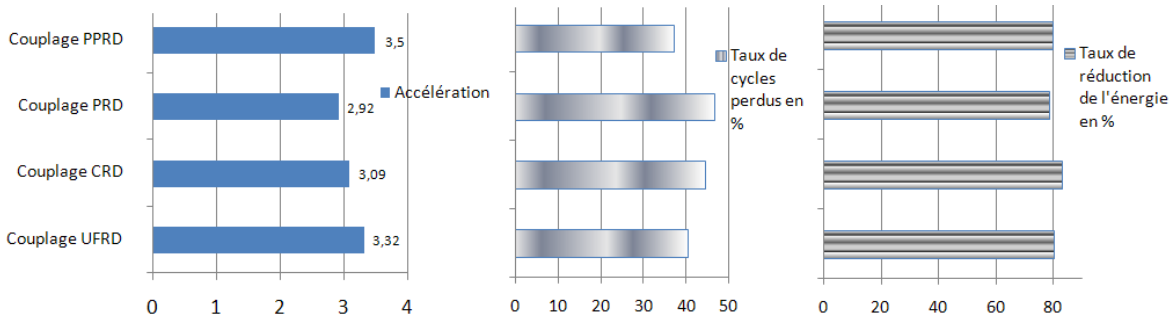

Figure 4-12 – **Performances de l'architecture hybride DART/TMS320C6713 pour les différents couplages pour la tâche 1 de la phase "tracking"**. *Le couplage PPRD présente les meilleures performances temporelles : une accélération de 3.5 correspondant à un taux de cycles perdus de 37.04% (56483 cycles DART). Alors que le couplage CRD présente les meilleures performances énergétiques (une puissance dissipée de 66.93 mW avec un taux de réduction de l'énergie dissipée de 83%).*

maximale pouvant être atteinte pour cette tâche est de 13.85.
Pour les performances énergétiques, le modèle CRD est le plus efficace avec un taux de réduction d'énergie de 92%.

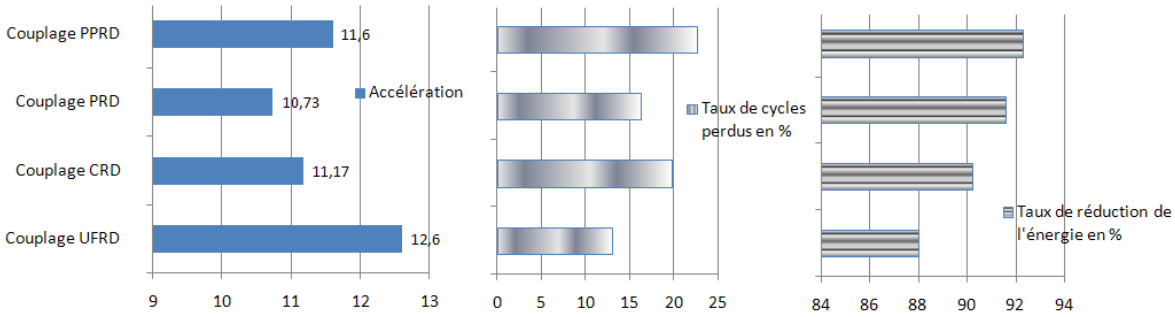

Figure 4-13 – **Performances de l'architecture hybride DART/TMS320C6713 pour les différents couplages pour la tâche 2 de la phase "tracking"**. *Le couplage UFRD présente les meilleures performances temporelles (une accélération de calcul de 12.6 fois avec un taux de cycles perdus de seulement 13% soit 407230 cycles DART). Pour les performances énergétiques, le modèle CRD est le plus efficace avec un taux de réduction d'énergie de 92%.*

La tâche 3 n'est pas trop complexe mais présente des motifs de calculs irréguliers, beaucoup d'accès mémoire et un grand volume de données à transférer aussi bien pour l'entrelacement externe que pour le décodage RS et le débrassage. Ce qui explique que le couplage PPRD présente la meilleure accélération de calcul : 5.01 fois contre 1.95 fois pour le couplage PRD. Le taux de cycles perdus est de 59% correspondant à 5193

cycles. L'accélération maximale pouvant être atteinte pour cette tâche est de 12.19. Comme pour les cas précédent le couplage CRD présente un meilleur taux de réduction de la dissipation d'énergie (78.9%) puisque les couplages PPRD et PRD soufrent de la dissipation d'énergie due aux accès "*off-chip*" et le couplage UFRD souffre du contrôle intensif des calculs par le CPU ce qui cause une dissipation non négligeable (figure 4-14).

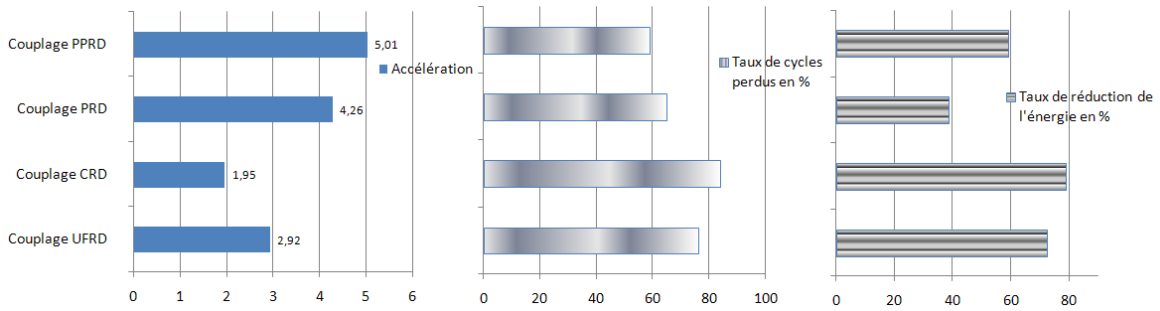

Figure 4-14 – **Performances de l'architecture hybride DART/TMS320C6713 pour les différents couplages pour la tâche 3 de la phase "*tracking*"**. *Le couplage PPRD présente la meilleure accélération de calcul : 5 fois contre 1.95 fois pour le couple PRD. Le couplage CRD présente un meilleur taux de réduction de la dissipation d'énergie (78.9%).*

Synthèse des résultats

Nous pouvons conclure que, pour le démodulateur DVB-T/H, le couplage en point-à-point entre le cluster DART et le TMS320C6713 est le plus adéquat. En effet, avec ce type de couplage, notre architecture hybride accélère de 6.4 fois l'exécution du démodulateur par rapport à l'exécution sur le DSP seul.

De même, ce type de couplage présente des performances énergétiques intéressantes (presque 80% de taux de réduction d'énergie dissipée), avec certes le couplage CRD qui présente les meilleures performances énergétiques. Cependant, le couplage PPRD présente le meilleur compromis de performances temporelles/énergetiques et a l'avantage d'être facile à mettre en œuvre par rapport aux autres couplages.

Dans la section suivante, nous allons optimiser l'implémentation du démodulateur DVB-T/H multimode en exploitant au mieux la reconfiguration dynamique, pour le couplage PPRD choisi.

C Optimisation de l'implémentation

Dans l'étude pour le choix du couplage adéquat entre le cluster DART et le TMS320C6713, nous avons partitionné notre application en contextes afin de la porter sur le cluster DART. Ce partitionnement est basé sur l'utilisation d'un nombre maximum de ressources du cluster DART (généralement la totalité) pour chaque contexte.
Nous nous proposons dans cette section d'appliquer la méthodologie définie au chapitre 3 afin d'optimiser le découpage de l'application en contextes de configuration. Nous avons adopté le couplage PPRD à partir de l'étude menée dans la section précédente.

Dans cet objectif, nous avons tout d'abord adapté le flot de conception de DART présenté dans la section B.1-2 du chapitre III pour les besoins de la nouvelle architecture hybride.
Comme pour le flot classique de DART, le flot commence par une description en langage C des tâches de l'application (modélisée par un graphe de tâches). En premier lieu, le code complet est compilé, débogué et implémenté sur le TMS320C6713 à l'aide de l'outil Code Composer Studio. Ce dernier, nous permet d'évaluer les performances temporelles du DSP quand il implémente seul l'application en question. L'estimation de l'énergie consommée par le DSP a été estimée par l'outil Soft Explorer [54].
Ce code source est manipulé par le front-end SUIF. Par contre, nous avons ajouté une première étape de profilage/partitionnement qui identifie les boucles candidates à être accélérées par l'ARD DART : parties calculatoires consommant du temps de l'application. Ces portions de code seront "mappées" sur le cluster DART et sont généralement des traitements réguliers.

Selon le couplage appliqué entre le CPU et DART, des instructions de contrôle traduisant les opérations que doit réaliser le CPU afin de gérer les traitements réalisés par DART seront exécutées par le DSP.

Ensuite des passes de déroulage partiel de boucle permettent de pipeliner le code et d'extraire les cœurs de boucles. Pour le code qui sera exécuté sur DART, un graphe DFG est généré et ensuite le flot classique de DART lui sera appliqué. Alors que pour le code à exécuter sur le TMS320C6713, les techniques d'ordonnancement ILP utilisées classiquement par le flot de compilation de ce DSP lui sont appliquées.
Ensuite, nous obtenons les performances temporelles et énergétiques du DSP à l'aide des outils : Code Composer Studio et Soft Explorer. Pour DART, les codes binaires exécutables sur DART générés sont simulés au moyen de l'outil SCDART permettant d'estimer les latences de traitement et l'énergie consommée lors de l'exécution de l'application.

A ce stade, une analyse de performances est réalisée en appliquant les modèles d'accélération et de réduction d'énergie dissipées que nous avons définis.

La méthodologie de découpage des contextes de configuration que nous avons définie, est basée sur un algorithme de détermination du facteur de déroulage de boucle pour lequel les performances sont optimales. En effet, nous appliquons pour chaque tâche de l'application ce flot de conception plusieurs fois en faisant varier le facteur de déroulage de boucle et nous analysons à chaque reprise les performances. Pour le facteur de déroulage qui maximise les performances de cette tâche c'est à dire l'accélération et le taux de réduction de l'énergie dissipée, la simulation sur SCDART du code binaire généré par les outils liés à l'architecture DART nous donne le nombre d'unités fonctionnelles pour chaque configuration. Le plus petit de ces valeurs d'UFs correspond à la surface associée à ce facteur de déroulage de boucle.
A ce stade, nous faisons le partitionnement de l'application en tenant compte de cette contrainte de surface.

Tout en tenant compte du couplage choisi, nous avons appliqué cette méthodologie au démodulateur multi-mode DVB-T/H. Le tableau 4-7 représente les résultats de l'analyse de performances de l'architecture hybride DART/TMS320C6713 (couplage en PPRD) lors de la variation du facteur de déroulage de boucles α pour les différentes tâches du démodulateur. Les performances pour $\alpha = 0$ sont celles obtenues sans déroulage de boucle. Nous avons défini le facteur de performances m^i pour une tâche τ_i au chapitre 3 comme étant le produit de l'accélération de traitement de cette tâche par son taux de réduction d'énergie.

Pour l'acquisition, le facteur de déroulage de boucle optimal correspond à $\alpha = 1$ avec un facteur de performances de 3.1. Au niveau du mode "Tracking", le facteur de déroulage de boucle optimal est $\alpha = 2$ pour les tâches 1 et 2 et est $\alpha = 0$ pour la tâche 3. Le fait que le code de la tâche 3 est irrégulier et donc n'admet pas beaucoup de parallélisme explique qu'un déroulage de boucle n'améliore pas les performances.

Pour chaque tâche et pour le facteur de déroulage de boucle optimal correspondant, nous avons déterminé les contextes de configuration DART. Ce découpage en contextes exploite les possibilités d'exploitation du parallélisme des données ce qui améliore la fréquence d'exécution des traitements.

	$\alpha = 0$	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Mode Acquisition				
Accélération	3.11	3.7	3.26	2.9
Puissance (mW)	79.76	73.4	71.13	75.3
Facteur de performance	2.56	3.10	2.74	2.41
Mode Tracking				
Tâche1				
Accélération	3.5	3.8	4.2	2.9
Puissance (mW)	67.8	65.4	59.13	66.3
Facteur de performance	2.79	3.06	3.46	2.33
Tâche2				
Accélération	11.6	12.06	13.13	11.8
Puissance (mW)	176	161.2	160.13	175.2
Facteur de performance	10.46	10.97	11.96	10.65
Tâche3				
Accélération	5.01	4.88	4.20	3.4
Puissance (mW)	11.3	15.40	18.23	20.5
Facteur de performance	2.98	2.19	1.46	0.91

Tableau 4-7 – *Impact de la variation du déroulage de boucles sur les performances de l'architecture hybride DART/TMS320C6713 pour le démodulateur multi-mode DVB-T/H.*

Le tableau 4-8 montre que l'application de cette méthode d'optimisation pour le démodulateur DVB-T/H donne six contextes de configuration. En effet, la tâche acquisition avec un déroulage de boucle d'une itération 1 auquel correspond une surface optimale de 20 UFs, est découpée en deux contextes de configuration. La latence de traitement totale de ces deux contextes est de 643 μs contre 874 μs dans le cas du portage de l'acquisition en un seul contexte et sans déroulage de boucles. De même, la puissance dissipée a diminué de 79.76 mW à 66.45 mW.
Pour le mode "*tracking*", la tâche 2 a aussi été optimisée en deux contextes de configuration diminuant ainsi la latence de traitement (par rapport au portage en un seul contexte) de 24% et la dissipation de puissance de 15%.
Sans déroulage de boucles, les tâches 2 et 3 s'exécutaient dans le même contexte de reconfiguration DART. Avec le déroulage de boucle, chacune s'exécute dans un contexte de configuration unique. Le parallélisme de données a augmenté considérablement les performances du décodeur de Viterbi passant d'une fréquence d'exécution de 66 MHz

à 130 MHz. La latence de traitement de la tâche 2 a diminué de 28% et la dissipation de puissance de 32.2%.
La tâche 3 est implémentée en un seul contexte de configuration mais sans déroulage de boucles. Elle ne voit donc pas ses performances s'améliorer.

Contextes	Surface N_{UFs}	Latence de traitement (μs)	Puissance dissipée (mW)
Mode Acquisition			
1	20	375	35.36
2	18	268	31.09
Mode Tracking			
Tâche 1			
3	22	362	31.02
4	20	220	26.2
Tâche 2			
5	15	1180	119.3
Tâche 3			
6	16	46.54	11.3

Tableau 4-8 – *Performances de l'architecture hybride PPRD DART/TMS320C6713 des différentes partitions pour le démodulateur multi-mode DVB-T/H*

D Résultats de l'implémentation d'un démodulateur DVB-T/H multi-mode

Dans cette section, nous présentons les résultats de l'implémentation du démodulateur DVB-T/H sur l'architecture hybride cluster DART couplé en point-à-point avec le DSP TMS320C6713. Nous mettrons en exergue l'aspect multimode de ce démodulateur que nous avons pu mettre en œuvre grâce à la reconfiguration dynamique de notre architecture.

Comme le montre la figure 4-15, la consommation d'énergie de l'architecture hybride varie en fonction du mode de configuration DVB-T/H implémenté. Nous remarquons que l'augmentation de la complexité du démodulateur (augmentation du rendement du code convolutif, réduction de l'intervalle de garde et élévation du nombre de bits par

porteuse pour la modulation) induit une augmentation de la consommation d'énergie.

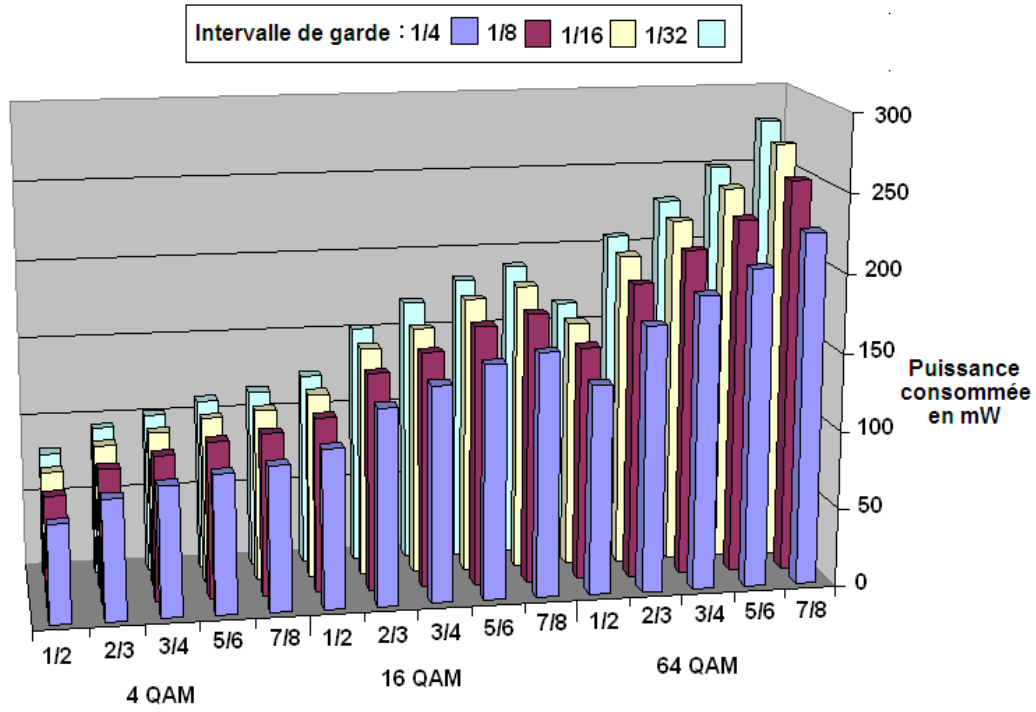

Figure 4-15 – **Consommation d'énergie de l'architecture hybride DART/TMS320C6713 en fonction des différentes configurations du système DVB-T/H**. *L'augmentation de la complexité du démodulateur, l'augmentation du rendement du code convolutif, la réduction de l'intervalle de garde et l'élévation du nombre de bits par porteuse (pour la modulation de la porteuse) induit une augmentation de la consommation d'énergie.*

Comme le débit utile augmente avec la complexité du modulateur, une gestion et une optimisation de l'énergie consommée peut être mise en place à travers un compromis entre débit et consommation d'énergie.

En effet, le basculement de la configuration DVB-T/H correspondant à la transmission avec un débit minimal (4.98 Mbps) correspondant à la transmission la mieux protégée (intervalle de garde = 1/4 et un rendement de code convolutionnel de 1/2, modulation 4 QAM) à la configuration DVB-T/H avec un débit maximal (31.67 Mbps) correspondant à la transmission la moins bien protégée (intervalle de garde = 1/32 et un rendement de code convolutionnel de 7/8, modulation 64 QAM), permet d'avoir un gain en débit net de 84% au coût de 77 % d'augmentation de consommation d'énergie. Ceci montre le besoin de gérer efficacement le débit net d'information et la protection de la transmission afin d'optimiser la durée de vie de la batterie.

L'objectif de l'implémentation d'un récepteur DVB-T/H reconfigurable est de commuter d'un mode à un autre selon la qualité instantanée du canal et l'état de la batterie. Dans [57], les auteurs présentent le C/N (Carrier-to-Noise ratio) nécessaire pour obtenir une transmission QEF (Quasi Error Free) ce qui veut dire une transmission avec un taux d'erreur binaire $TEB = 10^{-11}$ à l'entrée du décodeur MPEG-2. Pour tous les modes de DVB-T/H, la figure 4-16 montre la consommation d'énergie de l'architecture hybride en fonction du C/N nécessaire pour une réception QEF.

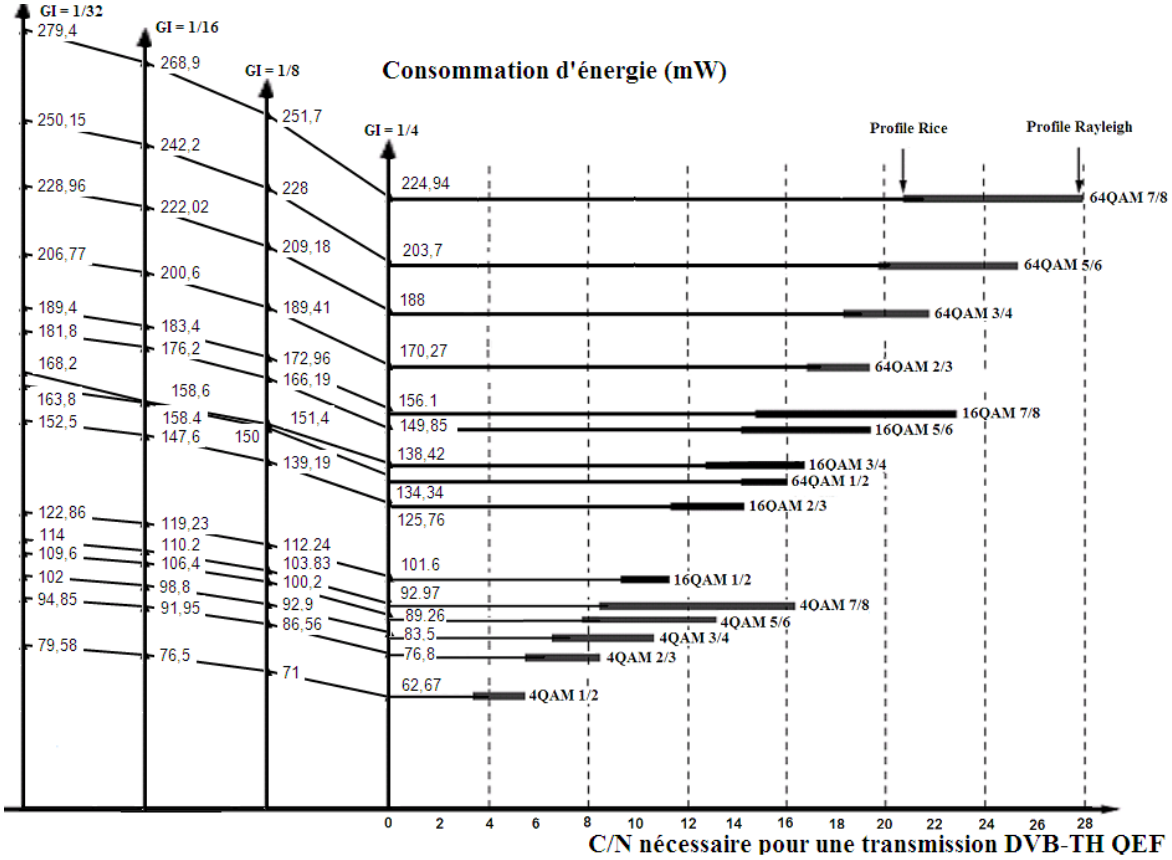

Figure 4-16 – **Consommation d'énergie du démodulateur DVB-T/H en fonction du C/N nécessaire pour une réception QEF**. *Pour un mode DVB-T/H donné, la consommation d'énergie peut être optimisée en modifiant le niveau de fiabilité avec un débit net plus bas.*

Pour un mode DVB-T/H donné, la consommation d'énergie peut être optimisée en modifiant le niveau de fiabilité avec un débit net plus bas.
Dans ce cas, nous avons trois niveaux de flexibilité :

– si la mesure du C/N est inférieure au C/N nécessaire pour une réception QEF, la diminution du rendement du code convolutif permet de réduire la consommation

d'énergie et augmenter la protection contre les erreurs ;
- si le C/N actuel répond au C/N nécessaire pour une réception QEF, nous pouvons adapter l'intervalle de garde afin d'optimiser la consommation d'énergie sans affecter la fiabilité ni le débit net ;
- si la mesure du C/N est supérieure par rapport au C/N nécessaire pour une réception QEF, nous pouvons basculer à un autre mode avec une modulation de porteuse donnant un nombre de bits par porteuse moins important.

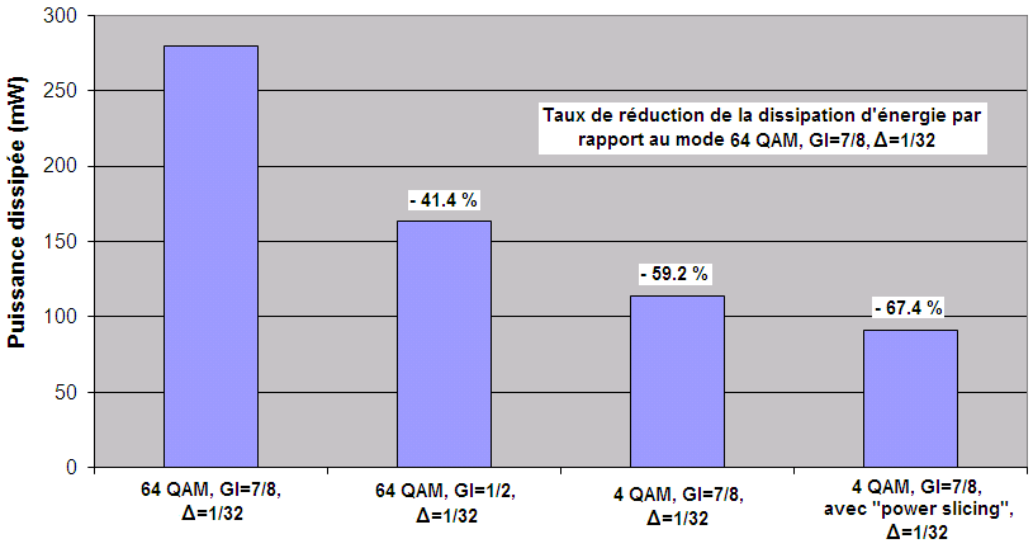

Figure 4-17 – **Exemple d'optimisation de la consommation d'énergie**. *Pour le mode DVB-T/H 64 QAM, GI* $= \frac{7}{8}$, $\Delta = \frac{1}{32}$, *le décodeur dissipe 279.4 mW. Cette dissipation d'énergie est réduite de 41.4 % en passant d'un GI* $= \frac{7}{8}$ *à un GI* $= \frac{1}{2}$. *Une réduction suppélementaire d'environ 18% en passant d'une modulation 64 QAM à 4 QAM. Pour ce dernier mode en appliquant le "power slicing", il est possible d'exécuter un ACS en un cycle DART et couper ainsi l'alimentation pendant quatre cycles. Ce qui permet d'atteindre jusqu'à 67% de réduction de l'énergie dissipée.*

En diminuant le débit net d'informations, la contrainte temps réel des différentes tâches diminue, ainsi les DPR correspondants deviennent inactifs pendant quelques cycles. Il est donc possible de couper l'alimentation des DPR en question pour quelques cycles. Cette méthode s'appelle le "*power slicing*".
En appliquant les optimisations présentées en haut (adaptation du rendement du code convolutif, le changement de la modulation de porteuse et *power slicing*), jusqu'à 67% de réduction de l'énergie dissipée peut être atteinte (figure 4-17).

E Synthèse

Dans ce chapitre, nous avons présenté les résultats d'implémentation d'un démodulateur DVB-T/H multi-mode, c'est à dire basculant d'une configuration à une autre suivant les caractéristiques du canal et l'état de la charge de la batterie, sur une architecture hybride couplant un cluster reconfigurable dynamiquement DART et un processeur de traitement de signal, le TMS320C6713.

Nous avons étudié l'interaction entre DART et le DSP afin d'implémenter en temps réel le mode de configuration avec la complexité algorithmique maximale du système DVB-T/H. Cette étude nous a permis de choisir le mode de couplage point à point comme étant le mode donnant des performances optimales pour l'implémentation du démodulateur (une accélération de 6.4 et un taux de réduction de la consommation d'énergie d'environ 80%).

Nous avons ensuite optimisé cette implémentation en exploitant au mieux l'aspect reconfiguration dynamique. Pour cela, nous avons appliqué la méthodologie de partitionnement temporel que nous avons définie permettant de découper l'application en partitions sous contrainte de surface. L'application de cette méthode nous a permis d'avoir six partitions et d'améliorer les performances.

Notre architecture implémente en temps réel le mode avec la complexité algorithmique la plus importante du système DVB-T/H (correspondant au mode donnant le maximum de débit net (31.67 Mb/s) en dissipant une puissance de 279.4 mW. Ce résultat correspond donc au cas où l'architecture consomme le maximum de puissance et une large gamme de performances peut être obtenue. Par exemple, pour le mode DVB-T/H le moins complexe la consommation obtenue est de 62.67 mW.
Dans [65], le démodulateur DVB-T/H est implémenté sur un FPGA. Ce dernier peut implémenter plusieurs modes DVB-T/H mais pas dynamiquement et il dissipe environ 439 mW ce qui est nettement supérieur à la dissipation de l'architecture hybride DART/TMS320C6713.
Dans [66], un démodulateur DVB-T/H en bande de base implémenté sur un ASIC en 0.18 μm consomme 250 mW pour le mode le plus complexe algorithmiquement ; ce qui est légèrement meilleur que les performances de notre architecture. Cependant l'implémentation du DVB-T/H sur l'architecture DART/TMS320C6713 a l'avantage de pouvoir s'adapter grâce à la reconfiguration dynamique aux conditions de transmission chose impossible à réaliser avec un ASIC.

Conclusions et perspectives

Synthèse des travaux

Les travaux présentés dans ce mémoire portent sur la définition de solutions d'aide à la conception ciblant les plate-formes hybrides reconfigurables constituées d'un processeur connecté à une unité reconfigurable dynamiquement. Le rôle de cette dernière est d'accélérer le traitement des applications en respectant des contraintes de consommation, de flexibilité et de coût.
L'accélération de calcul et la basse consommation occupent les premiers rangs parmi les nombreux défis à relever dans les projets de conception des architectures hybrides reconfigurables. L'optimisation de ces deux métriques de performances souffre désormais du manque de modèles et d'outils permettant de les quantifier. Le premier objectif de cette thèse était donc de développer une méthodologie d'estimation des performances des systèmes hybrides reconfigurables simple et précise.

En analysant l'état de l'art de ces architectures, nous avons abouti à la définition des paramètres ayant un impact sur les performances de ces systèmes. Parmi ceux-ci, nous trouvons ceux caractéristiques de l'application (applicatifs), ceux dépendants de l'architecture (architecturaux) et ceux fonctions de l'exécution du code (algorithmiques).

Aborder le problème de l'optimisation des performances d'une architecture hybride, impose de définir des modèles permettant de représenter des comportements réalistes de l'échange entre le processeur et l'architecture reconfigurable dynamiquement. Nous

avons dans ce sens défini des lois de communication qui permettent de quantifier la latence et la dissipation d'énergie dues aux transferts de données et de contextes de configuration entre le CPU et l'ARD pour une application donnée. Ces lois de communication sont des fonctions mathématiques utilisant les différents paramètres applicatifs, architecturaux et algorithmiques. Ensuite, nous avons appliqué ces différentes lois aux modèles de couplages CPU/ARD.

A partir de ces lois de communications et de manière à exploiter les caractéristiques de la reconfiguration dynamique, nous avons par ailleurs développé un modèle d'estimation de performances des architectures hybrides reconfigurables. L'application de ce modèle aux architectures hybrides GARP et XIRISC pour l'application *filtrage d'image median*, nous a permis de valider notre modèle en comparant nos résultats à ceux fournis par les publications relatives à ces architectures. Nos résultats montrent que l'erreur maximale de prédiction par rapport aux mesures est de 8% pour cette application test.

Nous avons également montré dans cette thèse que notre méthode d'estimation de performances peut servir de métrique pour les processus d'adéquation algorithme/architecture. En effet, nous avons développé une méthodologie d'Adéquation Algorithme Architecture Hybride **AAAH** qui permet à l'utilisateur possédant une librairie de cibles architecturales de déterminer quel est le couplage le mieux adapté entre un CPU et une ARD, d'un point de vue consommation et accélération du calcul, à l'exécution de son application.
Nous avons par ailleurs étudié l'impact des paramètres applicatifs et algorithmiques sur les performances d'une architecture hybride. Pour cela, nous avons porté un récepteur WCDMA avec un comportement dynamique sur un modèle générique d'architecture hybride en appliquant notre méthodologie.

Les contraintes de haute performance et de faible consommation se traduisent nécessairement par la nécessité d'optimiser au mieux l'architecture, pour la rendre en adéquation avec le traitement à réaliser. Pour cela, nous avons présenté une méthodologie d'optimisation de l'implémentation sur l'architecture hybride, visant à exploiter de façon optimale la reconfiguration dynamique. Cette méthodologie est basée sur la détermination de la surface (en nombre d'unités fonctionnelles) de l'ARD nécessaire pour avoir des performances optimales, et ce, en trouvant le facteur de déroulage de boucle qui assure le maximum de performances pour l'architecture hybride.
En intégrant cette méthodologie, nous avons adapté le flot de compilation classique de l'architecture reconfigurable dynamiquement DART pour les besoins d'une implémentation sur une architecture hybride.

Nous avons validé nos méthodologies par l'implémentation d'une application de calcul intensif : un démodulateur DVB-T/H multi-mode (c'est à dire basculant d'une configuration à une autre suivant les caractéristiques du canal et l'état de la charge de la batterie) sur une architecture hybride couplant un cluster reconfigurable dynamiquement DART et un processeur de traitement de signal, le TMS320C6713.

Nous avons étudié l'interaction entre DART et le DSP en appliquant notre méthodologie AAAH afin d'implémenter en temps réel le mode de configuration avec la complexité algorithmique maximale du système DVB-T/H. Cette étude nous a permis de choisir le mode de couplage point à point comme étant le mode donnant des performances optimales pour l'implémentation du démodulateur (l'exécution du démodulateur sur l'architecture hybride par rapport à l'exécution sur le DSP seul est accélérée de 6.4 avec un taux de réduction de la consommation d'énergie d'environ 80%).

Nous avons ensuite optimisé cette implémentation en exploitant au mieux l'aspect reconfiguration dynamique. Pour cela, nous avons appliqué notre méthodologie d'optimisation de la reconfiguration dynamique dans une architecture hybride afin de découper l'application en partitions sous contrainte de surface. L'application de cette méthode nous a permis d'avoir six partitions et d'améliorer les performances.

Perspectives

Dans le cadre des travaux présentés dans ce document, plusieurs perspectives sont envisageables.

Dans le cas général, les applications pouvant être exécutées sur les architectures hybrides ne sont pas uniquement des applications requérant du calcul intensif mais aussi du contrôle intensif ou mixant les deux. Nous pourrons donc tester si nos méthodologies sont capables d'estimer les performances d'autres applications que celles de type calculs intensifs.

Une autre extension à considérer pour nos travaux est de pouvoir étendre notre modèle pour prendre en compte une panoplie plus large d'architectures. Ceci peut être rendu possible par un travail de caractérisation préalable qui aurait pour but d'uniformiser

les modèles en mettant l'accent sur les caractéristiques communes des différentes architectures. Cela nécessitera de rajouter des paramètres à notre modèle initial puisque typiquement la consommation dépendra de la technologie employée.

Nous pourrions étudier comment notre méthodologie AAAH pourrait être appliquée sur des SOCs. En effet, les systèmes intégrés sur une même puce (SoC) sont de plus en plus hétérogènes. Il faudra donc étudier comment notre méthode pourrait être généralisée à tous types d'architectures.
A court terme, nous devrons automatiser notre méthodologie AAAH afin qu'elle puisse devenir un atout dans l'estimation de performances des systèmes sur puce.

Une autre perspective serait de développer notre méthodologie d'optimisation de la reconfiguration dynamique dans une architecture hybride en une méthodologie de partitionnement temporel et de l'intégrer dans l'étape de partitionnement du flot de conception des SOCs. Concrètement, nous pourrions envisager d'intégrer notre méthodologie dans des outils de Co-Design.

Bibliographie

[1] ITRS. International technology roadmap for semiconductors overview. 2008 update. Technical report, 2008.

[2] K. Compton and S. Hauck. Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34(2) :171–210, 2002.

[3] T. Fujii and al. A dynamically reconfigurable logic engine with a multi-context/ multi-mode unified-cell architecture. In *IEEE International Solid State Circuits Conference (ISSCC'99)*, pages 15–17, 1999.

[4] J.R. Hauser and J. Wawrzybek. Garp : A mips processor with a reconfigurable coprocessor. In *5th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97)*, pages 12–21, Los Alamitos, California, 1997.

[5] http ://www.chameleonsystems.com/.

[6] http ://www.xilinx.com.

[7] G. Sassatelli, L. Torres, J. Galey, G. Gambon, and C. Diou. The systolic ring : A dynamically reconfigurable architecture for embedded systems. In *International Workshop on field Programmable Logic and Applications (FPL'01)*, volume 2147/2001, pages 409–419, Belfast, Irlande du nord, Août 2001.

[8] S. Hauck, Z. Li, and E. Schwabe. Configuration compression for the xilinx xc6200 fpga. In *6th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98)*, pages 138–146, USA, April 1998.

[9] At40k fpgas with freeram. Technical report, Atmel, 1999.

[10] Atmel coprocessor field programmable gate arrays. Technical report, AT6000 datasheet, 1999.

[11] H.Singh, G.Lu, M.Lee, E. Filho, and R. Maestre. Design and implementation of the morphosys reconfigurable computing processor. *VLSI signal Processing systems for signal, image and video Technology*, 24 :147–164, March 2000.

[12] Z. Li, K. Compton, and S. Hauck. Configuration caching management techniques for reconfigurable computing. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pages 22–33, Washington, DC, USA, 2000.

[13] http ://www.semiconductors.philips.com.

[14] R. Subrammanian, U. Jha, J. Medlock, C. Woodthorpe, and K. Rieken. Novel application- specific signal processing architectures for wideband cdma and tdma applications. In *IEEE Vehicular Technology Conference (VTC'02)*, volume 2, pages 1311–1317, May 2002.

[15] J. L. Hennessy and D. A. Patterson. *Computer Architecture : A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[16] S. Pillement and R. David. Architectures reconfigurable et faible consommation : Réalité ou prospective ? *Technique et science informatiques*, 26/5 :595 – 621, 2007.

[17] S. Sirowy, Y. Wu, S. Lonardi, and F. Vahid. Two-level microprocessor-accelerator partitioning. In *The Design, Automation, and Test in Europe conference (DATE'07)*, pages 313–318, 2007.

[18] M. D. Galanis, A. Milidonis, G. Theodoridis, D. Soudris, and C. E. Goutis. Automated framework for partitioning dsp applications in hybrid reconfigurable platforms. *Microprocessors and Microsystems*, 31(1) :1–14, 2007.

[19] Dapdna-ims, a dynamically reconfigurable processor for image processing applications. Technical report, IPFLEX, http ://www.ipflex.com/en/E5-corporate_profile/pr_060905_en.html, 2007.

[20] M. Mishra and S.C. Goldstein. Virtualization on the tartan reconfigurable architecture. In *International Conference on Field Programmable Logic and Applications (FPL'07)*, pages 323–330, August 2007.

[21] K. BenChehida. *Méthodologie de Partitionnement Logiciel/Matériel pour Plateformes Reconfigurables Dynamiquement*. PhD thesis, Université de Nice-Sophia Antipolis, November 2004.

[22] http ://www.arm.com/products/solutions/amba_spec.html.

[23] http ://www-01.ibm.com/chips/techlib/techlib.nsf/productfamilies/coreconnect_bus_architecture.

[24] M. Borgatti, F. Lertora, B. Foret, and L. Cali. A reconfigurable system featuring dynamically extensible embedded microprocessor, fpga, and customizable i/o. In *IEEE Journal of Solid-State Circuits*, volume 38, Issue 3, pages 521–529, Mars 2003.

[25] http ://www.st.com.

[26] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The garp architecture and c compiler. *Computer*, 33(4) :62–69, 2000.

[27] http ://suif.stanford.edu/suif/suif2/.

[28] Christian Plessl and Marco Platzner. Zippy - a coarse-grained reconfigurable array with support for hardware virtualization. In *ASAP '05 : Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors*, pages 213–218, Washington, DC, USA, 2005. IEEE Computer Society.

[29] J. Jacob. Memory interfacing for the onechip reconfigurable processor. Master's thesis, University of Toronto, 1998.

[30] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri. A vliw processor with reconfigurable instructionn set for embedded applications. *IEEE Journal of Solid-State Circuits*, vol. 38(11) :pp. 1876–1886, 2003.

[31] C. Mucci, C. Chiesa, A. Lodi, M. Toma, and F. Campi. A c-based algorithm development flow for a reconfigurable processor architecture. In *IEEE International Symposium on System on Chip*, pages 69– 73, 2003.

[32] C. Rupp, M. landguth, T.Graverick, E.Gomersall, and H.Holt. The napa adaptative processing architecture. In *6th IEEE Symp. on FPGAs for Custom Computing Machines (FCCM'98)*, pages 28–36, 1998.

[33] D. C. Cronquist, P. Franklin, S. G. Berg, and C. Ebeling. Specifying and compiling applications for rapid. In *FCCM '98 : Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, page 116, Washington, DC, USA, 1998. IEEE Computer Society.

[34] K. Morris. Catapult c : Mentor announces architectural synthesis. Technical report, http ://wwww.fpgajournal.com, 2004.

[35] Using handel-c with dk. Technical report, Celoxica, Ltd., 2005.

[36] Systemc 2.0.1 language reference manual. Technical report, OSCI, 2003.

[37] M. Lòpez-Vallejo and J. C. López. On the hardware-software partitioning problem : System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems*, 08(03) :269 – 297, July 2003.

[38] B. Arafeh, K. Day, and A. Touzene. A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems. *J. Syst. Archit.*, 54(5) :pp. 530–548, 2008.

[39] Y. C. Jiang and J. F. Wang. Temporal partitioning data flow graphs for dynamically reconfigurable computing. *IEEE Transctions on Very Large Scale Integration (VLSI)*, 15(12) :1351–1361, December 2007.

[40] C. Tanougast, Y. Berviller, P. Brunet, S. Weber, and H. Rabah. Temporal partitioning methodology optimizing fpga resources for dynamically reconfigurable embedded real-time system. *Microprocessors and Microsystems, Elsevier*, 27(3) :115–130, April 2003.

[41] J. Noguera and R. M. Badia. Dynamic run-time hw/sw scheduling techniques for reconfigurable architectures. In *Tenth International Workshop on Hardware/Software Codesign (CODES'02)*, pages 205–210, Estes Park, Colorado, May 2002.

[42] R. Maestre, F. J. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, and M. Fernandez. Kernel scheduling in reconfigurable computing. In *the conference on Design, Automation and Test in Europe (DATE'09)*, pages 90–96, Munich, Germany, 1999.

[43] T. Ntakpé. Algorithmes d'ordonnancement de graphes de tâches parallèles sur plates-formes hétérogènes en deux étapes. In *Rencontres francophones du Parallélisme (RenPar'17)*, Perpignan, october 2006.

[44] Imène Benkermi. *Modèle et algorithme d'ordonnacement pour architecture reconfigurables dynamiquement.* PhD thesis, Université de Rennes 1, 2007.

[45] Y. Le Moullec. *Aide à la conception de systèmes sur puce hétérogènes par l'exploration paramétrable des solutions au niveau système.* PhD thesis, Université De Bretagne Sud, Avril 2003.

[46] C.Takahashi, M. Sato, D. Takahashi, T. Boku, H. Nakamura, M. Kondo2, and Motonobu Fujita. Empirical study for optimization of power-performance with on-chip memory. *High-Performance Computing*, 4759 :466–479, 2008.

[47] D.Saillé, D.Chillet, and O.Sentieys. Modélisation de la consommation pour les mémoires sram. In *Faible Tension Faible Consommation FTFC'01*, 2001.

[48] D. Menard, M. Guitton, P. Quemerais, and O. Sentieys. Efficient implementation of a rake receiver on the tms320c64x. In *37th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 2161 – 2165, Monterey, USA, november 2003.

[49] P. Benoit, G. Sassatelli, L. Torres, D. Demigny, M. Robert, and G. Cambon. Metrics for reconfigurable architectures characterization : Remanence and scalability. In *IPDPS '03 : Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 176.1, Washington, DC, USA, 2003. IEEE Computer Society.

[50] R.David. *Architecture reconfigurable dynamiquement pour applications mobiles.* PhD thesis, Université de Rennes I, july 2003.

[51] V.Messé. *Production de compilateurs flexibles pour la conception de processeurs programmables spécialisés.* PhD thesis, Université de Rennes 1, 1999.

[52] E. Martin, O. Sentieys, H. Dubois, and J. Philippe. Gaut : An architectural synthesis tool for dedicated signal processors. In *Design Automation Conference (EURO-DAC' 93)*, pages 14–19, 1993. Lasti-ENSSAT et GDR 134, convention CENT.

[53] D-J.Wang and Y. H. HU. Rate optimal scheduling of recursive dsp algorithms by unfolding. *IEEE Transactions Circuits and Systems I : Fundamental Theory and Applications*, 41 :672–675, October 1994.

[54] Eric Senn, Johann Laurent, Nathalie Julien, and Eric Martin. Softexplorer : estimating and optimizing the power and energy consumption of a c program for dsp applications. *EURASIP J. Appl. Signal Process.*, 2005(1) :2641–2654, 2005.

[55] Digital video broadcasting (dvb) ; framing structure, channel coding, and modulation for digital terrestrial television. Technical report, ETSI EN 300 744 v1.5.1, June 2004.

[56] Digital video broadcasting (dvb) ; transmission system for handheld terminals (dvb-h). Technical report, ETSI EN 302 304 v1.1.1, Jun.2004.

[57] Implementation guideline for dvb-t transmission aspects, dvb document a037. Technical report, DVB Project Office, 1998.

[58] M. Speth, S. Fechtel, G. Fock, and H. Meyr. Optimum receiver design for wireless broad-band systems using ofdm : Part ii. *IEEE Transactions on Communications*, pages 571–578, April 2001.

[59] A. Filippi and S. Serbetli. Ofdm symbol synchronization using frequency domain pilots in time domain. *IEEE Vehicular Technology Conference (VTC' 07)*, pages 1376–1380, 2007.

[60] H. Benoit. *La télévision numérique Satellite, câble, terrestre (principes et applications du système DVB).* 3 ème edition, Paris, 2002.

[61] Y. Shen and E. F. Martinez. Channel estimation on ofdm systems. Technical report, Freescale Semiconductor Inc, Application Notes, AN3059, Janvier 2006.

[62] L. Howath, I. Ben Dhaou, H. Tenhunen, and J. Isoaho. A novel, high-speed, reconfigurable demapper-symbol deinterleaver architecture for dvb-t. In *International Symposium on Circuits and Systems (ISCAS'99)*, volume 4, pages 382–385, 99.

[63] S. Lin and D. J. Costello. Error control coding. *IEEE Transactions on Information Theory*, 51 :1616–1617, 2004.

[64] Tms320c6713 floating-point digital signal processor, sprs186. Technical report, Texas Instruments, 2005.

[65] Ruei-Dar Fang and Hsi-Pin Ma. A dvb-t/h baseband receiver for mobile environments. In *Proceedings of the 2007 WSEAS Int. Conference on Circuits, Systems, Signal and Telecommunications*, Gold Coast, Australia,, 2007.

[66] L. F. Chen and C. Y. Lee. Design of a dvb-t/h cofdm receiver for portable video applications. *IEEE Communications Magazine*, 45(8) :112–120, August 2007.

Table des figures

Liste des tableaux

Glossaire

AAAH : Adéquation Algorithme Architecture Hybride reconfigurable, 70

ACSU : *Add-Compare-Select Unit*, 110

APE : *Associative Processing Element*, 16

API : *Application Programming Interface*, 25, 125

ARC : *Advanced RISC Compuing* , 16

ARD : *Architecture Reconfigurable Dynamiquement*, 4

BMU : *Branch Metric Unit*, 110

BSS : *Breizh Synthesis System*, 87

C/N : Carrier-to-Noise ratio, 135

CFB : *Configurable Fonctionnel Bloc*, 15

CLB : *Configurable Logic Block*, 11

CPU : *CPU : Central Processing Unit*, 10

CRD : Coprocesseur Reconfigurable Dynamiquement, 60

CTR : *Compile-Time Reconfiguration*, 9

CU : *Computing Unit*, 29

DAP : *Digital Application Processor*, 24

DLP : *Data Level Parallelism*, 45

DMA : *Direct Memory Access*, 40

DNA : *Distributed Network Architecture*, 24

DPE : *Data Processing Engine*, 2

DPGA : *Dynamically Programmable Gate Array*, 32

DPR : *Data Path Reconfigurable*, 83

DSP : *Digital Signal Processor*, 7

DVB : *Digital Video Broadcasting*, 98

DVB-H : *DVB-Handled*, 98

EMIF : *External Memory Interface*, 116

FEQ : *Frequency-domain EQualizer*, 109

FFT : *Fast Fourier Transform*, 105

FGB : *Fine Grain Bloc*, 13

FIR : *Finite Impulse Response*, 109

FPGA : *Field Programmable Gate Array*, 11

GA : Générateur d'Adresses, 85

GFD : Graphe de Flot de Données, 88

GI : *Guard Interval*, 100

HCDFG : *Hierarchical and Control Data Flow Graph*, 37

ILP : *Instruction Level Parallelism*, 31

IP : *Internet Protocol*, 98

IPC : *Instruction Per Cycle*, 46

ITRS : *International Technology Roadmap for Semiconductors*, 2

LPI : *Low-Pass Interpolation*, 108

LUT : *Look-Up Table*, 15

MAC : *Multiplication-Accumulation*, 109

MOPS : Mega Operations Per Second, 46

NOC : *Network On Chip*, 26

OFDM : *Orthogonal Frequency Division Multi-plexing*, 99

PiCoGA : *Pipelined Configurable Gate Array*, 34

PLP : *Process Level Parallelism*, 45

PRBS : Pseudo Random Binary Sequence, 113

PRD : Périphérique Reconfigurable Dynamiquement, 62

QEF : Quasi Error Free, 135

QoS : *Quality Of Service*, 99

RC : *Reconfigur-able Cell*, 13

RF : *Reconfigurable Fabric*, 25

RFU : *Reconfigurable Functional Unit*, 30

RISC : *Reduced Instruction Set Computer*, 13

RPF : *Reconfigurable Processing Fabric*, 16

RS : *Reed Solomon*, 100

RSB : Rapport Signal sur Bruit, 73

RSoC : *Reconfigurable Systems on Chip*, 9

RTL : *Register Transfer Level*, 34

RTR : *Run-Time Reconfiguration*, 9

RU : *Reconfigurable array Unit*, 29

SCMD : *Single Configuration Multiple Data*, 86

SFN : *Single Frequency Network*, 100

SIMD : *Single Instruction Multiple Data*, 16

SMU : *Survivor Management Unit*, 111

SOC : *System On Chip*, 1

SWP : *Sub Word Parallelism*, 84

TLP : *Task Level Parallelism*, 45

TPS : *Transmission Parameter Signalling*, 107

UAL : Unité Arithmétique et Logique, 8

UCR : Unité de Calcul Reconfigurable, 26

UFRD : Unité Fonctionnelle Reconfigurable Dynamiquement, 58

VLIW : *Very Long Instruction Word*, 31

WCDMA : *Wideband Code Division Multiple Access*, 71

ZNF : *Zippy Netlist Format*, 34

Résumé

Les applications de télécommunications mobiles et de multimédia, notamment dans le domaine de l'embarqué, deviennent de plus en plus complexes au niveau calculatoire et consomment de plus en plus d'énergie. Afin de palier aux besoins calculatoires et énergétiques de ces applications, les concepteurs se sont orientés vers les architectures hybrides, associant des systèmes de nature et paradigme différents. Ces architectures ont retenu l'attention des concepteurs parce qu'elles présentent un bon compromis coût/performances calculatoires d'autant plus qu'elles possèdent des propriétés énergétiques intéressantes.
En outre, l'émergence dans la dernière décade des architectures reconfigurables dynamiquement associant haute performance et encore plus de flexibilité, a fait que les dernières générations des architectures hybrides associent un ou plusieurs processeurs à une ou plusieurs architectures reconfigurables dynamiquement (ARD).

Cette thèse s'inscrit dans cette thématique et a ainsi pour objectif d'apporter une modélisation précise de ces architectures ainsi que des méthodologies permettant d'exploiter leurs potentiels de performances.
Une modélisation des mécanismes d'échange d'informations entre un processeur couplé à une ressource reconfigurable est d'abord proposée ce qui a permis une identification précise de modèles de performances. En utilisant ces modèles de performances, une méthodologie d'adéquation algorithme architecture permettant suivant les paramètres de l'application de déterminer le couplage CPU/ARD adéquat est présentée. Nous introduisons ces modèles de performances dans le flot de développement logiciel de ces architectures afin de permettre un partitionnement temporel automatique basé sur la détermination de la surface (en nombre d'unités fonctionnelles) de l'ARD nécessaire pour avoir des performances optimales et ce en trouvant le facteur de déroulage de boucle qui assure le maximum de performances pour l'architecture hybride.

Le dernier aspect de ce travail concerne la validation de ces méthodologies et leur mise en oeuvre. Nous présentons pour cela les mécanismes d'implémentation d'un démodulateur multimode DVB-T/H et d'un récepteur WCDMA dynamique sur une architecture hybride reconfigurable dynamiquement.

Mots-clés : architectures hybrides reconfigurables, reconfiguration dynamique, adéquation algorithme-architecture, partitionnement logiciel/matériel, démodulateur DVB-T/H, récepteur WCDMA, télécommunications

Abstract

Telecommunication applications, especially in embedded systems, have become more complicated and so require more resources to accelerate calculation and reduce power consumption. To satisfy these requirements, system designers are looking to hybrid architectures which associate different systems with different paradigms. These new architectures deserve more attention because they permit interesting calculation cost/performance compromise and attractive power consumption properties.
Furthermore, the Dynamic Reconfigurable Architectures, giving high performances and more flexibility during the last decade, have been associated to one or several processors to create the new hybrid-architecture generations.

This thesis treats this area and presents a new and precise modelisation for these architectures. The document gives also methodologies permitting their high performance exploit.
First, this document details a modelisation of information exchange process between a processor and a reconfigurable unit. This modelisation has permitted a precise identification of different performance criteria.
Using these latter, we present an algorithm/architecture adequacy. It allows the determination of the CPU/ARD coupling kind depending on the application parameters.

In the second part, we introduce these performance criteria in the hybrid-architecture software development flow to permit an automatic timing partitioning. This partitioning is based on the ARD surface determination (in terms of functional-unit number) required to obtain optimum level performances. It is feasible by the loop unrolling factor calculation which guarantees a high level performance for the hybrid architecture.

The last part of this document concerns the validation of these proposed methodologies. For that, we present the exploration and the implementation process of a DVB-T/H demodulator and a WCDMA dynamic receiver on an dynamically reconfigurable hybrid architecture.

Keywords :Hybrid reconfigurable architectures, dynamic reconfiguration, algorithm/architecture adequacy, software/hardware partitioning, systems on chip.