Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit

Olivier Muller

N° d'ordre : 106

Thèse

présentée à

l'Université de Bretagne-Sud

en habilitation conjointe avec l'École Nationale Supérieure des Télécommunications de Bretagne

pour obtenir le grade de

Docteur de l'UBS

Mention : *Sciences pour l'Ingénieur*

par

Olivier Muller

Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit

---

soutenue le 13 Décembre 2007 devant la commission d'examen :

Composition du Jury :

Président :	E. BOUTILLON, Professeur à l'Université de Bretagne Sud
Directeur :	M. JEZEQUEL, Directeur d'études à l'ENST Bretagne
Encadrant :	A. BAGHDADI, Maître de conférences à l'ENST Bretagne
Rapporteurs :	A. JERRAYA, Directeur des programmes de conception au CEA-LETI M. FOSSORIER, Professeur à l'Université de Hawaï
Examinateurs :	B. CANDAELE, Directeur du département SOC, IC et EDA à Thalès G. MASERA, Professeur à l'École Polytechnique de Turin

Dédicace

À ma future femme, Julie.

Remerciements

Avant tout, je tiens à exprimer ma profonde reconnaissance à monsieur Amer Baghdadi, maître de conférences à Telecom Bretagne, pour le temps qu'il a consacré à mon encadrement, pour son soutien sans faille au cours de ces trois années de thèse et pour la qualité de son expertise qui a éclairé nombre de nos discussions.

J'adresse également mes remerciements à monsieur Michel Jézéquel, directeur d'études à Telecom Bretagne et responsable du département électronique, pour m'avoir fait l'honneur de diriger cette thèse et pour son soutien exemplaire, notamment lors de la rédaction de ce manuscrit.

J'exprime aussi mes remerciements aux membres du jury : monsieur Emmanuel Boutillon, professeur à l'UBS et directeur du laboratoire LESTER, pour m'avoir fait l'honneur de présider le jury de cette thèse ; messieurs Ahmed Amine Jerraya, directeur des programmes de conception au CEA-LETI, et Marc Fossorier, professeur à l'Université de Hawaï, pour avoir accepté d'être rapporteurs de ce travail ; messieurs Guido Masera, professeur à l'école Polytechnique de Turin, et Bernard Candaele, directeur du département SOC, IC et EDA à Thalès communications, pour leur participation à mon jury de thèse.

Je remercie l'ensemble des membres du département électronique pour leur accueil chaleureux et pour la bonne humeur ambiante qui a rendu ces trois années de travail très agréables. J'exprime également ma gratitude à tous ceux qui m'ont accordé de leur temps pour partager avec moi leur expertise scientifique, technique ou administrative. Je ne me risquerai pas à essayer de tous vous citer. Toutefois, je tiens à remercier particulièrement messieurs Patrick Adde pour m'avoir formé et aiguillé vers le monde de la recherche, et Christophe Jégo pour avoir soutenu ma candidature de financement de thèse. Je remercie également Michel Appéré pour son soutien moral dans le cadre du monitorat des cadets saoudiens.

De plus, je tiens à saluer les doctorants du département que j'ai eu le plaisir de côtoyer durant cette thèse : Jérôme, Matthieu, Emeric, Irène, Makram, Yi, Haisheng, Jorge, Daoud, Roua, Nicolas et Atif. Pour les quatre téméraires qui ont eu l'occasion de partager mon bureau, je n'ai qu'à me rappeler des grands moments de camaraderies et de complicités qui ont égayé le quotidien pour leur exprimer ma reconnaissance et mon amitié. Merci ainsi à Hazem, Charbel, Camille et Erwan.

Enfin, je remercie mes amis pour leur compréhension et leur soutien, mes parents pour m'avoir permis d'arriver jusque là et surtout Julie de m'avoir encouragé et supporté corps et âme durant cette épreuve.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Contexte

L'ESSOR fulgurant de l'électronique depuis l'invention du transistor dans les laboratoires Bell en 1947 a libéré de multiples facettes de l'appétence humaine. Par cette domestication de la physique, l'homme s'avère alors en mesure de déléguer la satisfaction d'un grand nombre de besoins à des machines au travers d'applications. Insatiable en besoins de connaissance, d'information, de questionnement (...), l'homme s'entoure d'un univers applicatif de plus en plus complexe et diversifié. Le recours aux applications s'affiche dans tous les secteurs d'activité (santé, connaissance, transport, communication, recherche...) et dans la vie au quotidien (téléphone, ordinateur, électroménager, télévision...) pour gérer des tâches sans cesse plus exigeantes. Pour s'y accomoder, les supports ou architectures d'exécution se doivent d'être performants et flexibles, c'est-à-dire être capables de répondre au plus vite et de s'adapter à des tâches à la fois complexes et différentes. Par ailleurs, cette adaptation ne précise ni son propre mandant (le concepteur, l'utilisateur, l'environnement, la technologie...), ni ses causes (réduire le temps de mise sur le marché, offrir de la qualité de service c'est-à-dire débit/taux d'erreur/latence, économiser de l'énergie, gérer des standards/modes...), ni les moyens de sa mise en oeuvre (adaptivité, reconfiguration, modularité, extensibilité, dynamisme, automatisation...). Bien souvent, les causes et les mandants de la flexibilité sont très liés. Dans un récepteur mobile, par exemple, un utilisateur peut exiger de celui-ci la multimodalité et, par le biais d'applications, la qualité de service nécessaire à leurs fonctionnements. L'environnement, toujours pour les besoins de l'utilisateur, peut également poser des contraintes comme un mode de fonctionnement économique lors d'une utilisation autonome. Finalement, le fabriquant du récepteur sera soucieux d'une part de respecter les exigences de son client et d'autre part de répondre au mieux aux contraintes économiques par une conception rapide et adaptée lui permettant d'être réactif aux évolutions de son marché. Ces différents couples cause-mandants influent directement sur les choix opérés au niveau des applications (décisions algorithmiques) et sur la manière dont ces dernières vont être mises en oeuvre (décisions architecturales).

Parlons encore du domaine des communications numériques qui n'échappe pas à l'appétence applicative. En témoigne l'apparition des turbo-communications qui représentent la généralisation du principe de processus itératif introduit par les turbocodes. Les applications de turbo-communications s'imposent depuis une dizaine d'années grâce à la véritable révolution qu'elles apportent sur la qualité de transmission. La recherche algorithmique en constante ébullition sur le sujet étoffe si rapidement un vivier d'applications que leur mise en œuvre ne suit pas la cadence. Pour cause, la complexité des systèmes de turbo-communications, communément appelés turbo-récepteurs, impose le recours à des architec-

tures matérielles performantes, ne serait-ce que pour atteindre des débits de communication toujours plus élevés. Certaines, principalement des architectures dédiées (i.e. ASIC), ont déjà vu le jour dans plusieurs équipes de recherches académiques et industrielles, mais elles ne sont pas assez flexibles et/ou performantes pour suivre l'effervescence algorithmique.

Problématique et Objectifs

Ce contexte soulève diverses interrogations : comment mettre en oeuvre des systèmes de turbo-communications capables d'ingérer les évolutions algorithmiques ? Comment permettre aux architectures de répondre aux exigences de qualité de transmission, de flexibilité et de haut-débit de communication ? Une voie royale a déjà été ouverte aux architectures multiprocesseurs pour répondre à ces exigences. Ces architectures offrent de nombreux avantages : modularité pour maîtriser la complexité, programmabilité pour s'adapter au contexte d'utilisation, extensibilité afin de s'adapter aux besoins de performance... Reste encore à montrer comment intégrer et adapter de manière adéquate ces actrices incontournables de l'électronique moderne aux turbo-communications.

Ainsi, l'objectif de cette thèse est de proposer une plate-forme architecturale multiprocesseur générique adaptée aux turbo-récepteurs. Son accomplissement nécessite une étude approfondie des algorithmes, qui exhibe à la fois leurs parallélismes et leurs complexités, mais aussi les compromis matériel/logiciel (i .e. performance/flexibilité) tant au niveau du calcul qu'au niveau des communications.

Contributions

Pour répondre à ces objectifs, nos travaux apportent sur les contributions algorithmiques et architecturales énumérées ci-dessous :

Contributions algorithmiques :

Etudes sur le parallélisme dans les architectures de turbo-décodeur :

- Classification des différents parallélismes existants.
- Analyse des différentes méthodes d'initialisation pour le parallélisme de sous-blocs.
- Analyse du parallélisme de décodeur composant avec la technique du décodage combiné ou "shuffled decoding".
- Optimisation de l'efficacité des architectures utilisant le décodage combiné par une méthodologie de conception conjointe entre entrelaceur et architecture.

Proposition d'un critère de caractérisation de l'importance des échanges d'informations extrinsèques.

Contributions architecturales :

Conception d'un processeur dédié aux turbocodes :

- Analyse des besoins en flexibilité autour de la fonction turbo-décodeur convolutif.
- Proposition d'une architecture de processeur à jeu d'instruction dédié au décodage des codes convolutifs.

Conception d'une plateforme multiprocesseur dédiée aux turbocodes :

- Proposition de plate-forme multiprocesseur associant ASIPs et réseaux multi-étages pour décoder les turbocodes de manière flexible et en haut-débit.
- Prototypage de la plate-forme multi-ASIP proposée.

Plan du mémoire

Ce mémoire est composé de six chapitres.

Le premier chapitre a pour objectif de dresser un état de l'art autour de la conception des systèmes multiprocesseurs afin de faciliter la compréhension du contexte architectural des travaux présentés dans le mémoire. Ainsi cette partie aborde les problématiques de la conception des principaux composants d'une architecture multiprocesseur (unités de traitements et réseaux d'interconnexions) ainsi que les différentes méthodologies de conception des systèmes sur puce. Ce chapitre s'achève en explicitant notre positionnement quant à la conception de systèmes multiprocesseur dédiés à une application.

Le deuxième chapitre donne, dans un premier temps, une large vision du domaine applicatif abordé dans ces travaux, i.e. la turbo-réception. Dans ce contexte applicatif d'une richesse inépuisable, nous nous focaliserons ensuite sur les turbocodes convolutifs pour expliquer leur structure, leur fonctionnement et leur performance. Le chapitre fournit les connaissances minimales pour aborder avec sérénité les autres chapitres du manuscrit.

Les turbocodes convolutifs sont ensuite étudiés en profondeur dans le chapitre trois autour des techniques permettant d'en accélérer l'exécution. Nous proposons d'abord une classification des parallélismes existant dans la littérature en tenant compte des aspects matériels et algorithmiques. En prenant appui sur l'analyse de cette classification, nous approfondissons nos investigations sur le niveau intermédiaire de notre classification, qui regroupe le parallélisme de sous-bloc et le parallélisme de décodeur composant. Les recherches présentées permettent d'évaluer et de comparer ces parallélismes. Elles permettent de prendre des décisions quant au choix des parallélismes dans des cas réels d'utilisation et explorent les moyens d'améliorer l'efficacité du turbo-décodeur au sens du compromis débit-surface.

Sur la base des résultats obtenus au chapitre trois et à l'aide d'une étude sur les besoins en flexibilité dans une application de turbo-décodage, le chapitre quatre propose un processeur dédié au décodage des codes convolutifs. Le processeur est présenté en détail sous différents angles : son architecture, ses unités dédiées de calcul, de communication et de mémorisation, son jeu d'instructions, ses performances sur différentes cibles technologiques en termes de fréquence de fonctionnement, débit et surface. Des variantes de ce processeur sont ensuite évaluées, répondant à des contraintes différentes.

Le chapitre cinq présente une plate-forme multiprocesseur utilisant le processeur dédié présenté au chapitre précédent. Dans un premier temps, une description approfondie de la plate-forme explique l'organisation de ses différentes ressources et plus particulièrement des structures de communication. Le débit de la plate-forme et sa surface sont ensuite évalués grâce aux résultats de synthèse logique. La fin du chapitre est consacrée au prototypage de la plate-forme sur une carte à base de circuits FPGA.

Le sixième et dernier chapitre présente des résultats autour d'une nouvelle idée d'adéquation algorithme-architecture pour un turbo-décodeur. L'idée proposée est de caractériser les échanges d'informations extrinsèques (i.e. le plus gros flux de communication dans un turbo-décodeur) pour attribuer aux communications des priorités différentes qui

peuvent être exploitées par l'architecture afin de réduire ses besoins en bande passante et/ou d'améliorer la qualité de service de ses réseaux de communication.

Enfin, le mémoire se termine en récapitulant les différentes contributions de nos travaux tout en mettant en perspective l'ensemble des voies ouvertes durant cette thèse et qui seront poursuivies à court et moyen termes voire à long terme.

CHAPITRE

1 Conception de systèmes multiprocesseurs monopuces

AU-DELÀ de l'ère naissante de la convergence des applications vers un même support, qui voit par exemple un assistant portatif embarqué un panel d'applications allant de la bureautique légère au géopositionnement en passant par une multitude de standards multimédias et télécoms, se cache un défi d'intégration gigantesque pour les industriels. Ces derniers sont confrontés à la fois à un délai de mise sur le marché très court, à cause de la pression concurrentielle et également au besoin de répondre à la demande croissante en innovation, qui impliquent l'intégration de complexité algorithmique toujours plus importante. Pour répondre à ce double défi, la tendance depuis près d'une dizaine d'années dans le monde de la conception de circuit sur puce (SoC pour System on Chip) est au multiprocesseur.

En reposant sur un modèle de programmation distribuée, les architectures multiprocesseurs sont naturellement plus performantes qu'un processeur seul pour traiter un ensemble de tâches, puisqu'elles autorisent une gestion en parallèle de ces tâches, ce que l'on appelle parallélisme de tâche. De plus, en adoptant une structure hétérogène, les systèmes multiprocesseurs peuvent traiter les tâches plus efficacement aussi bien en termes de surface que d'énergie, deux notions primordiales des systèmes embarqués. C'est pourquoi les SoCs modernes recourent à des unités de traitement spécialisées pour les tâches dont la complexité calculatoire dépasse largement les performances fournies par un processeur généraliste. Si les structures multiprocesseurs hétérogènes répondent à l'enjeu de l'intégration des systèmes, elles n'en demeurent pas pour autant des solutions miracles pour répondre au challenge du time-to-market. De nature hétérogène, ils se composent de multiples éléments que l'on peut classer dans l'une des quatre catégories suivantes : unité de traitement, unité de mémorisation, unité d'interconnexion ou unité d'entrée-sortie. En outre, chacun de ces éléments peut être affublé d'une certaine flexibilité (programmabilité, modularité, extensibilité) pour répondre aux contraintes de performance. En conséquence, plusieurs modèles de conception se cachent derrière la structure hétérogène et contrôler la conception d'un système reposant sur des flots éclatés est une tâche complexe.

La suite de ce chapitre présentera une vision des principaux composants d'une architecture multiprocesseur, c'est-à-dire des unités de traitements puis des réseaux d'interconnexions. La fin du chapitre sera consacrée aux méthodologies de conception des systèmes sur puce et à notre positionnement autour de cette problématique.

1.1 Unités de traitement

Une unité de traitement a pour but d'exécuter les tâches requises avec les ressources à sa disposition. Dans la suite de ce document, nous désignerons par architecture d'une unité de traitement un ensemble de ressources caractérisé par un agencement matériel et par un ordonnancement de l'exécution des tâches. Ainsi les variations de ressources, d'agencements ou d'ordonnancements autorisent un très grand nombre d'architectures capable de traiter l'ensemble des chemins de données caractérisant les tâches (chaque tâche peut être représentée par un graphe en flots de données spécifiant les dépendances entre les données). Dans cet immense espace de conception, le concepteur doit trouver une architecture satisfaisant aux mieux les différentes contraintes de conception : temps de conception, flexibilité, complexité, performance, consommation... L'exploration de ce panel de solutions nécessite donc une méthodologie efficace et une bonne connaissance du contexte d'implantation des tâches. L'aspect méthodologique ne sera abordé qu'indirectement dans cette section, au travers de différentes catégories de conception. Ces dernières seront décrites par un balayage de l'espace de conception suivant le contexte d'implantation. Nous nous intéresserons ensuite plus particulièrement aux méthodologies de conception de processeur dédié à une application (ASIP).

1.1.1 Compromis de conception

Avant de décrire l'espace de conception suivant les besoins et contraintes en performance, flexibilité, consommation, il convient de poser des définitions claires sur ces concepts.

1.1.1.1 Définitions

Performance : c'est la puissance de calcul d'une architecture pour réaliser une tâche donnée. La performance s'exprime généralement en fonction du nombre de milliards d'opérations nécessaires pour exécuter la tâche, le GOP[1]. Beaucoup de concepts gravitent autour de cette fonction de performance grâce aux définitions multiples de l'efficacité d'une architecture.

L'efficacité : elle est généralement définie par la performance d'une architecture sur une fonction de coût. Selon le contexte, la fonction de coût peut désigner la surface matérielle de l'architecture (exprimée soit une unité de surface pour une technologie cible, soit en équivalent portes logiques ou en transistor pour s'abstraire de la technologie), ou la consommation de l'architecture (exprimée généralement en mW) ou encore le temps de développement de l'architecture (en homme.mois).

Flexibilité : c'est la capacité d'adaptation d'une architecture à des tâches différentes. La flexibilité d'une architecture, qui pourrait se mesurer à la variété de tâche supportée, n'est en pratique pas mesurée, car elle est souvent posée comme une contrainte de conception. Par ailleurs, l'imprécision du terme flexibilité oblige à définir des concepts plus en rapport avec les architectures d'unités de traitement. On dira alors qu'une architecture est :

– *programmable*, si l'architecture sélectionne les chemins de données à chaque cycle d'horloge par l'intermédiaire d'une instruction. Dans ce cas, le concepteur contrôle l'unité de

[1]Acronyme de Giga OPérations

traitement, également nommé processeur, par l'intermédiaire d'un langage de programmation, qui est transcrit ensuite en langage machine, i.e. les instructions du processeur, par l'intermédiaire d'un compilateur.
- *reconfigurable*, si l'architecture a la capacité de changer la structure des chemins de données (c'est-à-dire les opérations qui y sont réalisées). Le concepteur peut alors contrôler l'architecture grâce à un jeu de configurations. Contrairement à une architecture programmable, le contrôle ne s'exerce pas à chaque cycle d'horloge.
- *adaptative*, si l'architecture a la capacité de sélectionner des chemins de données parmi un ensemble de chemin de données. Le contrôle est dans ce cas réalisé grâce à un jeu de paramètres dédié à l'application.

1.1.1.2 Performance et parallélisme

Outre un changement de cible technologique ou de conditions de fonctionnement (température, tension), le seul moyen d'augmenter les performances d'une architecture est de la modifier en prenant en compte le parallélisme inhérent aux tâches qu'elle doit exécuter. Le parallélisme désigne le fait de pouvoir traiter à un instant donné lors de l'exécution de la tâche plusieurs chemins de données en parallèle. Même si naturellement le parallélisme permet d'améliorer les performances, un concepteur d'architecture se doit de garder à l'esprit que l'accélération apportée par un parallélisme n'est valide que sur une portion du temps d'exécution de la tâche. Dès lors plus on cherche à accélérer cette portion, moins l'accélération globale sera conséquente. Cette loi, dite d'Amdahl du nom de son inventeur, sera particulièrement mise en lumière au cours du chapitre 3.

La figure 1.1 illustre les deux types de parallélisme existants : le parallélisme spatial et le parallélisme temporel. Le parallélisme spatial consiste à exécuter en parallèle plusieurs chemins de données indépendants. Deux chemins de données sont dits dépendants lorsque l'un de ces chemins produit directement ou indirectement une donnée utilisée par l'autre chemin. Le parallélisme temporel, ou pipeline, consiste à exécuter en parallèle plusieurs sections (ou étages) d'un même chemin de données. Ce parallélisme, qui permet, certes, d'accélérer le débit sur le chemin, présente quelques limitations. D'abord, il n'améliore pas la latence associée au chemin, c'est-à-dire le temps nécessaire pour qu'une donnée traverse le chemin. Au contraire, l'ajout de registre sur le chemin impose un léger surcoût au niveau de la latence. Par ailleurs, la structure pipelinée est très sensible aux déséquilibres entre les différents étages. En effet, comme l'horloge du système est partagée par tous les étages, le débit du chemin est dicté par l'étage ayant le temps de traversée le plus long. Ainsi l'écart de performances entre un pipeline idéal et un pipeline réel est caractérisé par la différence entre le temps de traversée de l'étage le plus long et le temps de traversée moyen des étages du chemin.

Dans le domaine des architectures de processeur [1], d'autres concepts évoluent autour du parallélisme. Premièrement, comme dans un processeur les chemins de données sont contrôlés par des instructions, le parallélisme est souvent appelé parallélisme d'instructions (Instruction Level Parallelism ou ILP). Cependant il faut garder à l'esprit que le parallélisme d'instructions n'est pas une transcription du parallélisme dans le monde des processeurs, puisqu'il est possible d'exécuter un vecteur de plusieurs chemins de données en parallèle au sein d'une même instruction. Selon la classification introduite par Flynn [2], cette exécution vectorielle s'appelle SIMD (Single Instruction Multiple Data), c'est-à-dire à un flot d'instruction est associé plusieurs flots de données. La classification de Flynn introduit également les architectures SISD (Single Instruction Single Data) correspondant à l'architecture classique de Von Neuman, les architectures MISD (Multiple Instruction Single Data) affectant un flot de

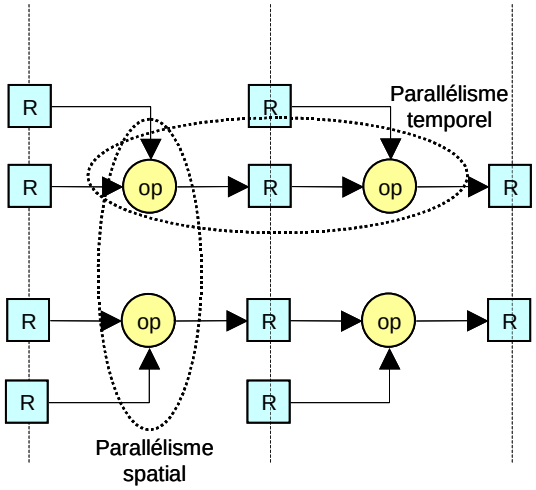

Figure 1.1 — Parallélisme spatial et temporel

données sur plusieurs instructions en parallèle (il s'agit en fait de la définition d'un pipeline), et les architectures MIMD (Multiple Instruction Multiple Data) qui traitent plusieurs flots de données à partir de plusieurs flots d'instructions. Pour caractériser les architectures de processeurs MIMD récents, cette classification n'est pas assez précise notamment dans l'expression des multiplicités M. Autant le M de MD est toujours l'expression d'un parallélisme spatial, autant dans MI il peut être interprété à la fois par la présence d'un pipeline dans l'architecture (parallélisme temporel) et par le lancement en parallèle de plusieurs instructions dans le pipeline (parallélisme spatial). Dans ce deuxième cas, puisqu'il s'agit d'un parallélisme spatial, le fonctionnement du processeur n'est correct que si l'indépendance des données traitées par le groupe d'instruction est garantie. Selon les moyens de garantir cette indépendance, on distingue deux familles utilisant du parallélisme spatial d'instruction :

- les processeurs superscalaires, qui assurent eux-mêmes l'indépendance des données en bloquant les instructions conflictuelles ou au moyen de techniques d'exécution dans le désordre, de spéculation et de renommage.
- les processeurs EPIC (Explicit Parallel Instruction Computing) et VLIW (Very Long Instruction Word) supposant que le compilateur a au préalable assuré l'indépendance des données dans chaque groupe d'instruction. Ainsi, en s'appuyant sur des processus de compilation plus complexes, ces processeurs peuvent gérer les instructions statiquement et sans bloquage, ce qui représente un gain substantiel en complexité matérielle.

1.1.1.3 Compromis et architecture

Outre la performance obtenue par parallélisme, le choix d'une architecture nécessite de prendre en compte bien d'autres contraintes. Or ces contraintes peuvent interagir entre elles de manière conflictuelle. On sait par exemple que chercher à rendre flexible une architecture se fait toujours au détriment de sa performance et vice-versa. Cela s'explique parce qu'une architecture flexible sous-utilise d'une part forcément ses chemins de données et nécessite d'autre part une structure de contrôle plus importante pour sélectionner et ordonnancer les ressources. Pour les mêmes raisons, une architecture flexible est également plus consommatrice d'énergie. Il est donc primordial de déterminer le plus précisément possible la flexibilité requise pour une architecture afin de conserver simplement la flexibilité nécessaire et donc maximiser

les performances. Cependant la flexibilité d'une architecture n'est pas uniquement entachée de défauts, bien au contraire, car elle facilite la réutilisation qui est souvent associée à une réduction des coûts de développement[2].

Ces différents compromis sont illustrés par la figure 1.2, qui positionne les principales catégories d'architecture selon les critères de conception. Dans la suite de cette section, nous aborderons ces catégories par ordre décroissant de flexibilité. Ainsi nous allons d'abord traiter des architectures programmables (processeurs généralistes, processeurs dédiés à un domaine, processeurs dédiés à une application) puis des architectures adaptatives au travers des circuits dédiés. De part la nature transversale de la flexibilité par reconfiguration, le cas des architectures reconfigurables ne sera abordé qu'en dernier.

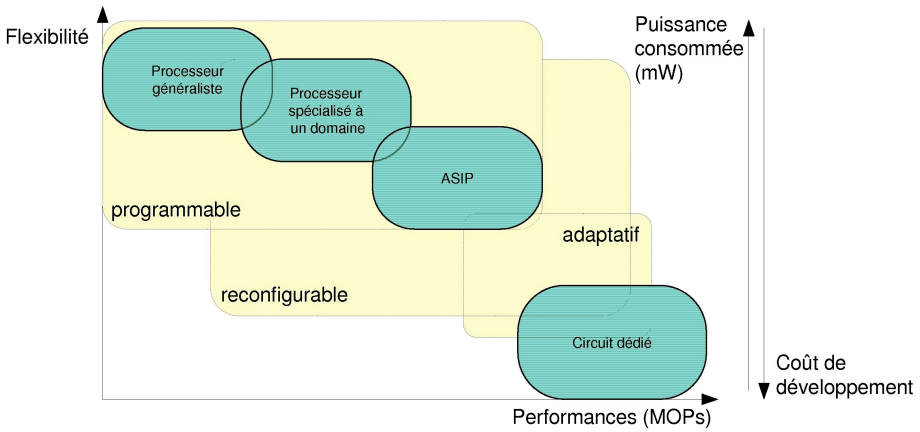

Figure 1.2 — Liens entre architectures et critères de conception pour une application

Processeur Généraliste (GPP[3])

Les processeurs à usage universel ou processeurs généralistes sont des architectures programmables dont le jeu d'instructions permet de traiter n'importe quel algorithme grâce à un support à la fois de l'algorithmie flottante et de l'algorithmie entière. Parmi les architectures les plus connues, on peut citer les architectures ARM dominant le marché des applications embarquées, les architectures MIPS reconnues pour leur efficacité (en surface et en consommation) pour les calculs scientifiques, et les architectures x86 monopolisant le marché de l'équipement informatique grand public. Pour améliorer leur performance, les processeurs généralistes reposent sur une architecture pipeline. La figure 1.3 représente un exemple de pipeline élémentaire à 4 étages avec un étage pour le chargement de l'instruction depuis la mémoire (C), un étage pour le décodage de l'instruction (D), un étage pour exécuter les chemins de données sélectionnés lors du décodage de l'instruction (E) et un étage dédié aux transferts entre les registres de données et la mémoire (M).

Une telle architecture permet de traiter jusqu'à 4 instructions en même temps à des étages différents. En autorisant le recouvrement des instructions, cette architecture ne nécessite que $n + 3$ cycles pour exécuter n instructions au lieu de $4.n$ cycles sans pipeline. En fonctionnement idéal, cette architecture peut atteindre pour un grand nombre d'instructions un gain de performances de 4. Le fonctionnement réel doit tenir compte de différents aléas de fonctionnement :

[2]Le développement logiciel est nettement plus rapide que le développement matériel.
[3]General Purpose Processor

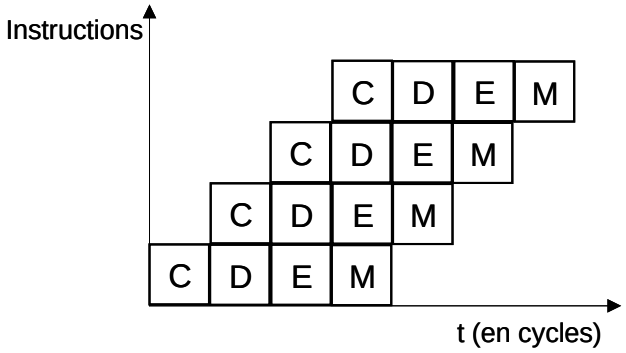

Figure 1.3 — Pipeline élémentaire d'un processeur

- des aléas de données, qui peuvent intervenir lorsqu'il y a des dépendances de données entre les instructions traitées dans le pipeline.
- des aléas de contrôle, qui interviennent lorsqu'une instruction modifie le chargement des instructions.
- des aléas structurels survenant à chaque conflit de ressources (unités d'exécution, registres, ports d'accès mémoires).

Ces aléas impliquent généralement l'arrêt partiel du pipeline pour débloquer l'architecture d'aléas de données et/ou de structure, ou l'obligation d'évacuer une partie du pipeline pour annuler des instructions improprement lancées à cause d'une instruction de contrôle. Dans tous les cas, ces aléas réduisent les performances d'un processeur. Par ailleurs, le risque de voir apparaître des aléas de données augmente avec la longueur du pipeline, car il y a plus de données potentiellement dépendantes à l'intérieur du pipeline. En conséquence, l'augmentation des performances d'un processeur par l'accroissement de la profondeur du pipeline devient rapidement limitée.

Pour améliorer leurs performances, les processeurs généralistes peuvent intégrer des mécanismes de prédiction ou d'anticipation pour minimiser les aléas du pipeline ou avoir recours à des unités d'exécution multiples (VLIW, superscalaire).

Processeur dédié à un domaine

L'objectif des processeurs dédiés à un domaine est de fournir pour un domaine d'application des performances supérieures aux GPPs tout en conservant la flexibilité nécessaire pour ce domaine d'application. Dans la pratique, le jeu d'instruction est allégé des instructions superflues ou trop générales et est complété par des instructions dédiées pour accélérer les traitements. Il existe pléthore de processeurs dédiés à un domaine : des processeurs graphiques (GPU) pour gérer le rendu graphique, des processeurs physiques (PPU) pour prendre en charge les calculs physiques (dynamique des fluides, des corps rigides, ...) principalement utilisé dans les jeux vidéos, des processeurs audios pour prendre en charge le traitement audio, des processeurs multimédias pour applications embarquées comme le Trimédia de NXP Semiconductors [3], ou encore des processeurs de traitement du signal (DSP) dédié au calcul numérique [4][5]. Pour prendre l'exemple des DSP, ces processeurs ne disposent dans leur grande majorité que de l'arithmétique flottante (jeu d'instruction allégé) et sont complétés par des instructions spécialisées comme l'instruction MAC (Multiplication Accumulation).

Processeur dédié à une application

Pour faire face à la diversité des algorithmes encore présents dans un domaine d'application, un processeur dédié à un domaine ne peut pas exploiter un degré de parallélisme optimal pour chaque application, ce qui se traduit dans les performances. Ainsi pour encore améliorer les performances tout en conservant la programmabilité, les processeurs dédiés à une application disposent de chemins de données exploitant au mieux le parallélisme de l'application tout en conservant un minimum d'opérations élémentaires pour permettre un minimum de flexibilité.

Les processeurs dédiés à une application sont le plus souvent désignés par l'appelation ASIP (Application Specific Instruction-set Processor) [6], qui fait référence au jeu d'instruction dédié du processeur. Mais la notion de processeur dédié à une application n'est pas limitée aux ASIPs. Par exemple, de récents travaux [7] présentent un processeur sans jeu d'instruction NISC (No Instruction Set Computer) pouvant exploiter à la fois le parallélisme temporel et spatial d'une application. Il est intéressant de noter que la compilation d'un code sur la structure NISC est assimilable à une synthèse architecturale. Cette structure pousse le concept de processeur dédié à une application à la limite entre la programmabilité et l'adaptativité, puisque les "instructions" correspondent au jeu de paramètres adaptatifs de l'architecture.

Circuit dédié

Les circuits dédiés, couramment appelés ASIC (Application Specific Integrated Circuit) lorsque la cible de conception est un masque de fonderie, désignent un ensemble d'architecture se limitant à l'exécution exclusive d'une application. Comme la conception de ce genre de circuit n'est affectée d'aucune contrainte architecturale pour exploiter le parallélisme, ces architectures permettent d'exprimer les performances optimales d'un algorithme. En revanche, à cause d'un temps de conception rédhibitoire, la flexibilité limitée de ce genre d'architecture est principalement cantonnée à l'expression d'une adaptativité sur un jeu de paramètres réduits ou à l'exploitation de cible de conception différente. Par exemple, l'utilisation de circuits logiques programmables (par exemple, les FPGAs pour Field-Programmable Gate-Array) comme cible de conception offre au prix d'une perte raisonnable de performance un peu de flexibilité aux circuits dédiés, puisque la cible peut ensuite être reconfigurée pour une autre application. Cette flexibilité de la cible, initialement introduite pour permettre un prototypage rapide de l'architecture en évitant les longues et coûteuses étapes de fabrication de masques, est de nos jours exploitée par les architectures sous le vocable architecture reconfigurable.

Architecture reconfigurable

Le domaine des architectures reconfigurables est vaste puisqu'il permet de manière transversale au choix de l'architecture d'ajouter de la flexibilité à une architecture. Grâce aux mécanismes de reconfiguration comme des FPGAs embarqués, on peut par exemple reconfigurer des chemins de données dans un processeur [8][9] ou un structure hétérogène [10] ou reconfigurer un co-processeur traitant une tâche complète [11][12]. Ainsi on distingue généralement différents niveaux de granularité dans la reconfiguration, ce qui explique pourquoi la reconfigurabilité s'applique aussi bien à des opérateurs dans un processeur (grain fin) qu'à des co-processeurs complets dans une architecture (gros gain).

Les architectures reconfigurables peuvent alors améliorer les performances d'une architecture en autorisant par reconfiguration l'exploitation d'une ressource non exploitée dans une architecture non reconfigurable. Ce gain de performance suppose au préalable une perte de performance initiale due à la cible reconfigurable et pose également de nombreux problèmes

quant à la gestion des configurations. Premièrement, l'architecture doit assurer un délai de reconfiguration faible devant le délai de calcul pour ne nuire ni aux performances ni à la consommation. Il faut également ensuite tenir compte du stockage des reconfigurations et du surcoût engendré en termes de surface pour l'architecture.

1.1.2 Méthodologies de conception ASIP

En offrant un compromis idéal entre performance et flexibilité autour d'une application, les ASIPs ont fait l'objet de toute notre attention au cours de cette thèse. C'est pourquoi cette section présente les méthodologies de conception existantes pour des processeurs dédiés à une application. Dans le monde de la conception électronique assistée par ordinateur (CAO électronique), il existe deux approches différentes pour concevoir ces systèmes : une approche par extension et une approche par description. Dans un cas comme dans l'autre, les outils de CAO doivent fournir le coeur de processeur matériel ainsi que tous les outils de développement qui lui sont associés. Cette suite d'outils doit, entre autres, comprendre un simulateur de fonctionnement du jeu d'instructions pouvant être précis jusqu'au cycle près, des descriptions du processeur dans plusieurs langages (HDL, SystemC) afin de faciliter la co-conception du processeur dans son environnement, un compilateur pour transformer le code du langage de programmation vers le langage machine incluant les instructions spécifiques du processeur et des débogueurs pour gérer les erreurs de programmations logicielles et matérielles.

1.1.2.1 ASIP par extension

Avec l'approche par extension, le concepteur d'ASIP dispose d'un environnement lui permettant de sélectionner et de configurer ses propres blocs matériels. Ces blocs sont ensuite intégrés à un coeur de processeur prédéfini et sélectionné en fonction des besoins du concepteur. Généralement ce type de conception utilise comme base des processeurs à jeu d'instruction réduit et l'ajout des blocs matériels implique alors une extension du jeu d'instruction. C'est notamment le cas avec les plateformes *Xtensa* de Tensilica, *OptimoDE* de ARM, et *ARChitect Processor Configurator* de ARC.

Un avantage principal de cette méthodologie de conception par extension réside dans la rapidité de conception qu'elle autorise grâce à la réutilisation des coeurs de processeurs de base. Cependant ce qui fait sa force fait également sa faiblesse puisque les coeurs de processeurs de base imposent des limitations à l'architecture, notamment sur la longueur du pipeline et sur le dimensionnement des chemins de données la plupart du temps figé à 16 ou 32 bits. Pour remédier à ces limitations, une description complète de l'architecture devient nécessaire.

1.1.2.2 ASIP par description

La méthodologie de conception par description offre une plus grande liberté lors de la conception de l'ASIP en utilisant un langage de description d'architecture qui permet au concepteur de spécifier conjointement l'architecture du processeur et son jeu d'instruction. Parmi les langages de description d'architecture existants, on peut citer les langages ISDL [13], NML [14], archC [15] ou encore LISA [16]. Nous décrirons par la suite le langage LISA et la suite d'outils *processor designer* de CoWare, qui ont été utilisés dans le cadre de cette thèse pour la conception d'ASIP.

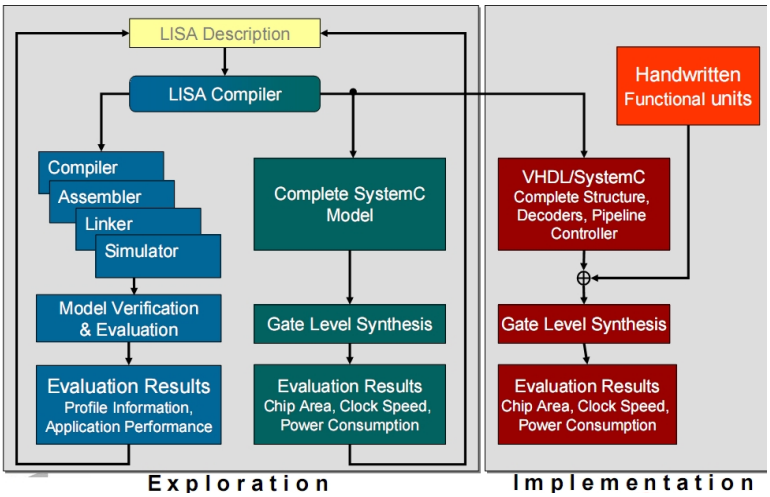

Figure 1.4 — Flot de conception de l'outil *Processor Designer* de *CoWare*

Le langage LISA permet la description de l'architecture d'un processeur au travers de deux aspects différents : les instructions et l'architecture. Pour chaque instruction, le langage permet de définir le codage, la syntaxe et le comportement associés. Pour l'architecture, LISA offre une grande liberté dans le dimensionnement des ressources (registres, mémoires, pipeline...) et autorise par ailleurs une modélisation précise du comportement des ressources. Ainsi, on peut modéliser les aléas de fonctionnement du pipeline (par blocage ou vidange), les spécificités des mémoires (temps d'accès, cache, nombre de ports d'accès ...), les mécanismes d'interruptions ou le comportement des interfaces (bus, broches d'entrée). L'ensemble des possibilités du langage permet au concepteur de travailler à plusieurs niveaux d'abstraction selon la précision souhaitée pour le modèle et cela jusqu'à une précision au niveau bit.

En interprétant le modèle LISA, l'outil *processor designer* peut ensuite générer automatiquement une suite d'outils facilitant à la fois l'exploration et l'implantation (figure 1.4). L'exploration de l'architecture du processeur peut alors être réalisée au moyen d'outils de compilation (complilateur C, assembleur, éditeur de liens...) assurant le lien logiciel-matériel et de simulation permettant à la fois le débogage logiciel-matériel et le profilage de la simulation. Par ailleurs, l'exploration peut être étendue à l'environnement du processeur par exportation d'un modèle systemC du simulateur compatible avec la plupart des environnements de simulation supportant le SystemC. L'outil permet également l'implantation de l'architecture en fournissant une description RTL synthétisable (VHDL ou Verilog) et des scripts de synthèse pour les outils commerciaux de synthèse logique. Dès lors, il devient possible d'évaluer une architecture en termes de surface matérielle, consommation et fréquence d'horloge, donc indirectement ses performances.

1.2 Réseaux d'interconnexions

Cette section décrit les évolutions actuelles des réseaux d'interconnexions au travers des réseaux sur puce.

1.2.1 L'émergence des réseaux sur puce

L'amélioration constante de la densité d'intégration sur les circuits intégrés permet d'ajouter de plus en plus d'unités de traitement à un système sur puce. De facto, cela implique globalement plus de communication pour permettre les échanges entre les unités de traitement et en conséquence plus de surface associée aux communications.

Les solutions classiques d'intégration d'interconnexions ne peuvent plus répondre à cette double contrainte concernant à la fois le volume de communication et la surface engendrée. D'un côté, les liaisons point-à-point qui relient directement émetteur et récepteur offrent des performances de communication optimales avec un coût quadratique en surface en fonction du nombre de noeuds. De l'autre, les bus, qui partagent le canal de transmission entre plusieurs émetteurs-récepteurs, permettent de maîtriser cette complexité aux dépens des performances puisque les différents utilisateurs du bus se partagent également sa bande passante limitée. L'utilisation du bus a également introduit de la flexibilité dans la gestion des communications. Cette flexibilité s'exprime par la réutilisation des ressources sous le contrôle d'un mécanisme d'arbitrage. Parallèlement, la flexibilité d'un bus s'exprime aussi avec l'apparition des interfaces de communication (OCP[4], VCI[5]) qui a permis de dissocier comportement des modules et communications entre les modules. Cette flexibilité, nommée modularité, permet l'interchangeabilité des modules en entrée du canal de communication.

Comme pour l'intégration des unités de traitement, l'intégration des communications impose aussi un compromis entre performance, flexibilité et coût. Le paradigme des réseaux sur puce ou NoC[6] [17][18] est né du besoin impérieux d'avoir des ressources de communication flexibles tout en préservant des performances acceptables. Avant de détailler les concepts relatifs aux NoC, définissons les éléments constitutifs d'un réseau sur puce. Pour permettre l'échange de données entre les unités de traitement, un NoC repose sur une structure de noeuds (aussi nommés routeurs ou switchs), qui sont interconnectés par des liens (ou canaux) de communication point à point. L'organisation physique des noeuds et des liens est appelée topologie du réseau. L'accès d'une unité de traitement au réseau est réalisé par l'intermédiaire d'une interface réseau (NI), qui est reliée à un noeud du réseau. L'utilisation d'interfaces réseaux assure entre autres la modularité du réseau.

1.2.2 Topologies

La topologie définit le positionnement des liens par rapport aux noeuds. Elle est caractérisée par des critères physiques comme le diamètre (i.e. le nombre maximum de liens entre deux interfaces réseaux), la connectivité (i.e. le nombre de liens par noeud) et la distance moyenne (i.e. le nombre de liens en moyenne entre deux interfaces réseaux). Les grandes familles de topologies sont illustrées dans la figure 1.5 [19].

Les topologies de type grille[7] sont les plus fréquentes dans la littérature, principalement pour les multiples avantages apportés par leur régularité (routage géométrique et connectivité limité) [20]. Selon les besoins en distance moyenne et en connectivité, la topologie grille peut se déployer sous plusieurs formes (grille classique comme sur la figure 1.5.a, cylindre, tore...) et dans plusieurs dimensions (ligne, plan, cube...). Dans la pratique, les topologies de dimension supérieure à 2 ne sont pas utilisées à cause de la nature plane des circuits intégrés. Grâce à

[4]Open Core Protocol
[5]Virtual Component Interface
[6]Network on Chip
[7]Mesh

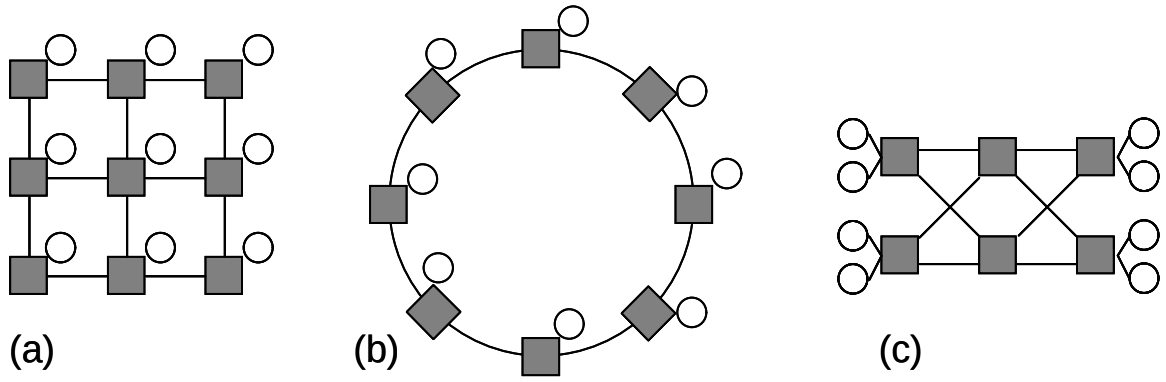

Figure 1.5 — Exemples de topologie : (a) grille 2D, (b) anneau, (c) indirecte (Beneš)

un maillage uniquement avec les voisins physiques, les topologies grilles sont par ailleurs bien adaptées pour les applications nécessitant principalement des communications locales. Pour des scénarii de communication aléatoire, la latence du réseau dépend plutôt de la distance moyenne de la topologie, qui évolue en racine carrée du nombre de noeuds pour une topologie grille 2D. Les topologies en anneau (figure 1.5.b) et les topologies indirectes (figure 1.5.c), bien qu'un peu moins régulières, permettent des distances moyennes plus petites. En partant d'une topologie en anneau, il est possible de réduire la distance moyenne en ajoutant des liens traversant l'anneau. On parle alors de topologies en anneau avec cordes. En contrepartie, ces changements topologiques impliquent une plus grande connectivité des noeuds et donc un coût de routage supérieur. Les réseaux indirects ou réseaux multi-étages permettent de conserver une connectivité limitée, tout en affichant une distance moyenne faible. En revanche, ces topologies ne permettent pas des communications locales rapides puisque le trafic doit transiter par des étages intermédiaires qui ne disposent pas de ressources connectées. Ce genre de topologie englobe les topologies en arbre [21] et les topologies du type Beneš, butterfly, omega [19].

Globalement, le choix d'une topologie est dicté par des contraintes de performances (débit, latence) pour véhiculer le trafic de l'application, par des contraintes de complexité (connectivité, nombre de noeuds et complexité des noeuds), mais également par les contraintes physiques inhérentes à l'intégration sur silicium.

1.2.3 Mécanisme de commutation

La commutation désigne la manière utilisée pour véhiculer l'information au travers du réseau. On distingue deux types de commutation : la commutation de circuit qui alloue un lien physique à la communication sur toute sa durée et la commutation de paquets, qui sépare l'information en paquets acheminés jusqu'au destinataire par des chemins éventuellement différents. Par la suite, le propos se réduira à la commutation par paquets, qui permet un meilleur partage des ressources. L'inconvénient potentiel de la commutation paquet est de ne pas pouvoir garantir les délais de communication, puisque les noeuds du réseau peuvent être congestionnés.

La communication entre noeuds doit suivre un protocole qui définit à la fois la liaison logique entre les noeuds avec des méthodes comme le store-and-forward, le virtual cut-through et le wormhole, et aussi les mécanismes physiques de la transmission comme les mécanismes de

synchronisation. La synchronisation physique est de très grande importance, car la consommation de l'arbre d'horloge devient prépondérante pour assurer la même horloge sur l'ensemble d'un SoC. Ce problème est largement soulagé si l'on limite l'arbre d'horloge à une utilisation locale (dans une unité de traitement) et que le réseau se charge de synchroniser les différentes communications. Ce concept Globalement Asynchrone Localement Synchrone (GALS) est notamment rendu possible avec des NoCs utilisant des liaisons asynchrones [22].

1.2.4 Routage

Les routeurs doivent assurer le transport des paquets au sein du réseau en gérant au mieux les flux et les congestions sur le réseau. Ils utilisent pour cela des mécanismes de routage, d'arbitrage, de contrôle de flux, ainsi que les informations de contrôle fournies avec chaque paquet. Ces informations de contrôle sont regroupées généralement dans l'*en-tête* du paquet, qui contient par la suite la *charge utile* c'est-à-dire le message à transmettre.

1.2.4.1 Algorithmes de routage

L'algorithme de routage décide du chemin à emprunter pour envoyer le message de sa source à sa destination. Si le chemin choisi ne dépend que de la source et de la destination, l'algorithme de routage est déterministe, autrement il est dit adaptatif s'il tient compte de la congestion des liens. Comme exemple de routage déterministe, on peut utiliser dans un réseau de topologie grille un routage de type XY qui consiste à avancer d'abord sur la direction X puis sur la direction Y. Ce type d'algorithme de routage très simple permet de distribuer le contrôle au sein du réseau. Remarquons qu'il est également possible de laisser à la source le choix du routage ; on parle alors de routage source.

1.2.4.2 Ressources de routage

Outre la connaissance de la prochaine étape de routage, le routeur doit également être en mesure de contrôler le flux de paquets en cas de congestion et d'arbitrer les conflits pouvant intervenir sur ces ports de sortie. En cas de contention, le routeur effectue un contrôle de flux soit par temporisation, à l'aide de file d'attente (FIFO[8]) mémorisant les paquets, soit par demande de retransmission du paquet, considéré perdu par le routeur. La temporisation, solution la plus couramment utilisée, peut se faire selon plusieurs méthodes : FIFOs en entrée (une FIFO par entrée), FIFOs en sortie (chaque sortie dispose d'un nombre de FIFOs égal au nombre d'entrée), canaux virtuels (plusieurs FIFOs par entrée alimentées selon la destination et la priorité des paquets). Comparativement aux FIFOs en entrée, les canaux virtuels permettent aux routeurs une meilleure utilisation des liens. De même, comparativement aux FIFOs en sortie, ils nécessitent moins de complexité mémoire [23].

Après le mécanisme de contrôle de flux, le routeur doit arbitrer chaque sortie pour déterminer quelle file d'attente va prendre le lien. L'arbitrage classique Round-Robin consiste à alterner les décisions d'arbitrage sur chacune des files. Si les paquets apportent des priorités dans leur en-tête, ces informations peuvent être exploitées pour l'arbitrage.

[8] First In First Out

1.2.5 Exemples de NoC et d'outils de génération de NoC

Historiquement, le réseau SPIN introduit par le laboratoire LIP6 est le premier réseau sur puce à commutation paquets [24]. Aujourd'hui l'engouement des chercheurs pour la thématique NoC a fait éclore un grand nombre de réseaux. Sans en détailler les spécificités, on peut néanmoins citer les réseaux NOSTRUM (introduction d'un algorithme de routage par déviations[9] et d'une technique de contrôle de congestion) [25], ÆTHEREAL (introduction du trafic garanti dans la conception de NoC) [26] et le réseau ANoC du CEA-LETI (introduction de la logique asynchrone dans la conception du réseau) [22]. Outre la conception de réseau à vocation généraliste, il existe également des flots de conception permettant de générer des réseaux dédiés à une application. Les solutions générées offrent alors de meilleures performances aux dépens de la flexibilité. Le flot de conception Xpipes permet par exemple d'explorer les besoins de l'application avec l'outil SUNMAP et, après exploration, de générer des modèles SystemC et RTL du réseau [27]. La société Arteris propose une solution commerciale similaire avec ses outils NoCexplorer et NoCcompiler [28]. Citons également l'outil μspider développé à l'UBS, qui permet, lui aussi, la génération automatique de réseaux [29].

1.3 Méthodologies et outils d'intégration de systèmes multiprocesseurs

Dans les deux sections précédentes, nous avons abordé deux composants primordiaux des architectures multiprocesseurs ainsi que les méthodologies associées à leurs conceptions. Cependant la conception d'un système multiprocesseur exige plus qu'une simple juxtaposition de méthodologie pour répondre aux contraintes de temps et de coûts de conception du marché et pour soutenir la complexité croissante des applications. Cette section présente la conception au niveau système (SLD[10]) au travers de ses concepts, de son flot de conception et de stratégies permettant de le mettre en oeuvre, puis donne notre positionnement dans le cadre de cette thèse.

1.3.1 Flot de conception au niveau système

Pour faire face au plus vite à la complexité croissante des systèmes, la conception au niveau système repose sur deux concepts : la réutilisation et l'abstraction. Dans le principe, l'apport de la réutilisation est évident : une partie du travail de conception est récupérée soit d'un autre système soit d'un fournisseur tierce. Le modèle de conception récupéré est nommé composant virtuel[11] ou IP[12]. Un composant peut représenter aussi bien du matériel que du logiciel, des traitements ou de la communication, une description précise ou abstraite. L'ensemble des composants constituant un système est appelé plateforme. Pour tirer un maximum de bénéfices de la réutilisation, un composant doit d'une part assurer sa pérennité dans le temps en présentant suffisamment de flexibilité pour éviter une reconception prématurée et, d'autre part, permettre une adaptation rapide dans le système. L'adaptation d'un composant est assurée par une interface, qui d'un point de vue architectural fait le lien entre deux tâches

[9] Patate chaude
[10] System Level Design
[11] Nommé simplement composant par la suite.
[12] Intellectual Proporty

logicielles[13], ou entre une tâche logicielle et une matérielle[13], ou encore, entre des tâches de traitement, communication ou mémorisation.

Outre la réutilisation, l'autre grand concept pour accélérer la conception est l'abstraction, qui consiste à s'intéresser plus au comportement du système (ce qu'il doit faire) qu'à la manière dont le système doit être implanté. Avec l'abstraction, le gain en temps s'obtient soit grâce à des opérations de synthèse qui permettent un raffinement automatique vers des niveaux de conception de plus en plus détaillés, soit en autorisant pour l'implantation du système des prises de décision fiable et plus précoce dans le cycle de développement. Pour une conception de circuit intégré, les différentes étapes d'abstraction ont permis de cacher la réalité physique des circuits intégrés grâce à l'utilisation de portes logiques, puis de cacher la logique au moyen d'architectures matérielles décrivant avec un langage RTL (Register Transfert Level) les opérations entre les registres. L'étape finale consistant à cacher la notion d'architecture n'est pas acquise. Certes, la synthèse d'architecture est d'ors et déjà possible avec de nombreux outils à partir de spécifications fonctionnelles sans indications temporelles (untimed) écrites dans des langages haut-niveau (C, Matlab, SpecC [30] ou systemC [31]), mais les outils de synthèse sont généralement conçus pour traiter un aspect de la conception (unité de traitement [32][33], unité de communication [34], interface logiciel/matériel [35]) et ne sont pas adaptés pour traiter globalement un système complexe. Pour pallier ses limitations, la tendance est à l'orthogonalisation de la conception [36] en cherchant d'une part à séparer la spécification comportementale du système et l'implémentation d'architecture et d'autre part à séparer les traitements calculatoires et les communications.

Ces séparations amènent à une phase d'exploration de l'espace de conception durant laquelle le concepteur cherche à aboutir à une plateforme virtuelle (ou abstraite) de référence à partir d'une spécification initiale respectant la fonctionnalité et les contraintes du système. Les plateformes virtuelles explorées utilisent pour leur exécution des modèles haut-niveau de composants. Par exemple, les communications peuvent être représentées à l'aide de modèles au niveau transaction[14] [37][38] et les processeurs à l'aide de simulateurs de jeu d'instructions (ISS). La plateforme virtuelle de référence est sélectionnée parmi les plateformes virtuelles explorées de manière à répondre au mieux aux contraintes de la spécification en termes de performances, flexibilité, consommation. Cette méthodologie permet ainsi d'anticiper les décisions architecturales pour évaluer les performances d'une plateforme.

Après la phase d'exploration, la seconde étape consiste à mettre en oeuvre une architecture en accord avec la plateforme virtuelle de référence. Avec les hypothèses de séparations et en utilisant des interfaces adéquates entre les composants [39], la coexistence de divers flots de conception est envisageable à cette étape : pour les communications, pour les composants matériels, pour les composants logiciels (génération de système d'exploitation, d'API [35]). Cette étape de co-conception [40] aboutit généralement à l'architecture RTL du système. Certaines étapes n'étant pas réalisées automatiquement, il peut s'avérer nécessaire de procéder à des vérifications de l'architecture ou des composants obtenus lors d'une co-simulation dudit composant dans la plateforme de référence.

1.3.2 Stratégies de conception de systèmes sur puce

Pour réaliser le flot de conception au niveau système précédemment décrit, les outils de CAO pour SoC, aussi bien académiques qu'industriels, suivent des stratégies différentes selon

[13] On parle dans ces deux cas d'API (Application Programming Interface).
[14] TLM (Transaction Level Modeling)

qu'ils traversent le flot de conception au niveau système dans le sens descendant (top-down) ou ascendant (bottom-up).

Avec la synthèse de système, l'approche est descendante puisque le point d'entrée est une spécification fonctionnelle sans modèle de temps [34], ensuite calculs et communications de la spécification sont raffinés vers des modèles de temps approximés dans une plateforme abstraite [41]. Finalement, l'architecture du système incluant des modèles RTL des composants de communication et de traitement peut être implantée automatiquement à partir d'outils de synthèse d'architecture (comportementale), de synthèse de communication, et de synthèse d'interfaces ou éventuellement à partir de bibliothèques [42]. Par sa nature descendante, cette stratégie de conception n'est pas très adaptée pour la réutilisation de composants pouvant nécessiter des définitions manuelles d'interfaces.

En considérant le flot de conception dans le sens ascendant, la conception fondée sur le composant[15][43] est plus à même d'exploiter le concept de réutilisation. Cette stratégie de conception consiste à utiliser un jeu de composants et d'interfaces standards prédéfinis et à les adapter de manière à construire le système global. La force de cet environnement de conception est donc de fournir, à partir d'une plateforme abstraite constituée par le concepteur dans une bibliothèque de composants (matériel, logiciel, communication,...), des encapsulations (wrapper) pour les composants matériels (incluant les communications) et logiciels, et également l'environnement système associé à la plateforme multiprocesseur (système d'exploitation, pilotes). Cette stratégie permet donc un raffinement des différents composants indépendamment de leur assemblage. Malgré son intérêt indéniable dans la phase d'implantation, cette stratégie ne contribue pas à faciliter la tâche d'exploration.

En prenant le juste milieu des deux stratégies précédentes, la stratégie de conception orientée plateforme (platform-based design) [44][36] a pour point d'entrée une spécification fonctionnelle du système et une plateforme virtuelle prédéfinie. Sur la base d'affectation entre des modules fonctionnels de la spécification et des composants de la plateforme, les performances du système peuvent être analysées pour mettre en avant les lacunes de la plateforme et les améliorations pouvant y remédier. La majorité des outils de conception est maintenant tournée vers cette stratégie de conception.

1.4 Conclusion et positionnement

Ce chapitre offre un rapide tour d'horizon de la conception des systèmes multiprocesseurs monopuces notamment à travers la conception de composants de traitement et de communication, mais en balayant également les méthodologies de conception associées aux systèmes sur puce. Dans le cadre de cette thèse, la spécificité de la famille d'algorithmes visée, qui sera décrite dans le prochain chapitre, nous a dès le début amené à choisir des composants très performants mais néanmoins flexibles. La cible était dès lors définie comme une plateforme multiprocesseur haute performance spécifique à une famille d'applications.

Dans cette optique, le choix des unités de traitement s'est naturellement orienté vers les processeurs ASIPs et notamment vers le flot de conception *processor designer* de CoWare, qui grâce au langage haut-niveau LISA offre une grande liberté de conception dans la manière d'exploiter et de programmer le parallélisme de l'application visée. L'outil permet également la génération de descriptions RTL du processeur et celle de modèles SystemC à plusieurs niveaux d'abstraction. L'exploitation des modèles SystemC, accompagnée des outils CoWare

[15]Component-based design

utilisant la méthodologie de conception fondée sur les plateformes, devait viser la vérification de la plateforme et l'exploration des besoins en communication. Finalement la décision de développer un composant de communication de type réseau sur puce également dédié à l'application fût prise et fait l'objet d'une thèse orthogonale à la présente thèse. Une partie du ferment de cette première sera discutée dans le chapitre 5.

CHAPITRE

2 Turbo-réception : les codes correcteurs d'erreurs

APRÈS avoir détaillé les aspects de l'intégration des systèmes sur puce dans le chapitre précédent, nous allons présenter l'un des contextes qui contribue de plus en plus fortement à l'accroissement de la complexité algorithmique et à la diversification des tâches, à savoir, la turbo-communication.

Dans un premier temps, ce chapitre présentera les différentes étapes constituantes d'une chaîne de transmission afin d'introduire le contexte algorithmique de ces travaux sans toutefois avoir la prétention d'être exhaustif. Après cette présentation, nous nous intéresserons à la partie réception de la chaîne ou turbo-récepteur qui concentre l'essentiel de la complexité entre autres à cause de la palette d'algorithmes potentiellement utilisés et le volume d'échange entre ces algorithmes. La fin du chapitre fournira les bases algorithmiques nécessaires à la compréhension des turbocodes convolutifs, principale application du présent manuscrit.

2.1 Chaîne de transmission

Une chaîne de transmission modélise les différentes étapes autorisant le transfert d'une information d'une source vers un destinataire. Une chaîne de transmission peut différer grandement selon la nature du système de transmission, qui peut aussi bien être un système de stockage de données qu'un système de télécommunication. Néanmoins certaines étapes sont applicables dans tous les contextes. Ainsi, le couple codeur-décodeur de source peut être employé pour réduire la taille du message à transmettre par une opération de compression-décompression réalisée avec ou sans perte d'information. À l'inverse, le couple codeur-décodeur de canal augmente la taille du message à transmettre afin d'accroître la qualité de la transmission sur le canal. Le canal, justement, comprend toutes les étapes entre la sortie du codeur et l'entrée du décodeur. Cela intègre le lieu physique de la transmission, dans lequel le signal transmis est bruité d'une façon aléatoire, et le couple modulateur-démodulateur, qui est utilisé dans la plupart des systèmes pour transformer l'information numérique en un signal continu compatible avec le milieu de transmission et vice-versa.

La suite de cette section illustre un peu plus en détail chacune des étapes d'une chaîne de transmission en se basant sur l'exemple de chaîne de transmission dans le cadre d'une communication sans fil de la figure 2.1. Ainsi les prochaines sections décriront le codage de canal, puis la modulation en général ainsi que dans les cas de transmission multi-utilisateurs ou multi-antennes (connu sous le nom de MIMO[1]), et finalement les aspects liés à un milieu de transmission.

2.1.1 Codage de canal

Le codage de canal est utilisé pour transmettre l'information avec le maximum de fiabilité en palliant les perturbations survenues lors de la manipulation physique de l'information sur le canal. Le principe consiste à introduire à l'émission de la redondance dans le message pour permettre à la réception de détecter ou corriger les erreurs de transmission. La théorie du codage, introduite par Shannon en 1948 [45], associe à chaque canal une capacité représentant le maximum d'information transmissible sur ce canal (exprimée en bits par seconde). La théorie repose sur le théorème suivant également énoncé par Shannon : *Si le débit d'information à l'entrée du canal est inférieur à la capacité, alors il est possible de transmettre un message numérique avec une probabilité d'erreur arbitrairement petite.* Dans sa démonstration, ce théorème assure de l'existence d'un code (le code aléatoire) permettant une transmission fiable, mais en pratique ce code est trop complexe à décoder. Depuis, la communauté scientifique s'efforce de trouver des codes correcteurs d'erreurs de longueur finie ayant une complexité raisonnable et s'approchant le plus près possible de la capacité.

Pour des codes de longueur finie, on définit le code $C(n, k)$, qui transforme un message d de k symboles issues d'un alphabet fini de taille q (dans la pratique, un corps de Galois) en un mot de code c de longueur n dans le même alphabet ($n > k$), par l'ensemble des q^kmots de code retenus parmi les q^n mots de code possibles. On définit également la distance de Hamming entre deux mots de code comme le nombre de symboles différents entre les deux mots de code. La distance minimale $d_{\min}$ d'un code, i.e. la distance minimum entre deux mots de code de $C(n, k)$, représente le pouvoir de discrimination du code. C'est une caractéristique essentielle du code qui, avec en autres le rendement du code ($R = \frac{k}{n}$), permet d'évaluer la performance d'un code, c'est-à-dire la proximité entre le rapport signal à bruit permettant

[1] Multiple Input Multiple Output

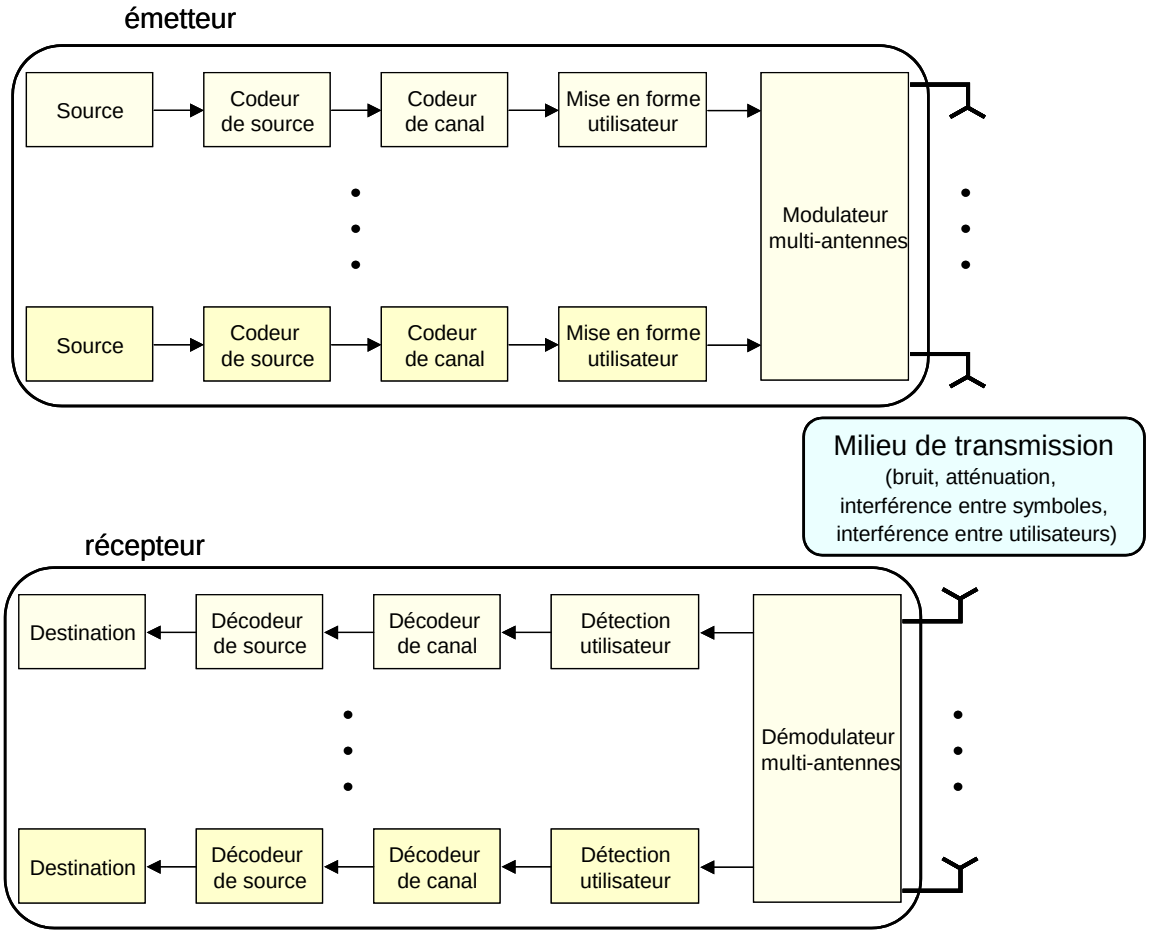

Figure 2.1 — Modélisation d'une chaîne de transmission numérique pour des communications sans fil

au code d'atteindre un taux d'erreur bit (TEB) ou taux d'erreur paquet (TEP) satisfaisant à l'application et la limite théorique correspondant à la capacité du canal. À fort rapport signal à bruit $\frac{E_b}{N_0}$, le TEP peut être estimé avec la borne de l'union :

$$TEP \approx \frac{1}{2} N(d_{\min}) erfc\left(\sqrt{R d_{\min} \frac{E_b}{N_0}}\right) \tag{2.1}$$

dans laquelle la multiplicité $N(d_{\min})$ représente le nombre de mots de code à la distance minimale.

Les codes sont classiquement classés en deux grandes familles :

- les codes algébriques (communément appelés codes en bloc), qui assurent une indépendance du codage à chaque bloc.
- les codes convolutifs [46], qui, dans leur version originale, codent l'information sortante en flot continu en utilisant à la fois le symbole entrant et un effet mémoire sur les entrées précédentes.

Le décodage est dit optimal s'il trouve le mot de code le plus probable en ayant la connaissance du code et de la sortie du canal. On parle alors de décodeur MAP (Maximum A Posteriori) paquet, puisqu'il minimise le TEP. Le seul moyen de réaliser ce décodage est de tester

tous les mots de code, ce qui explique pourquoi dans la pratique le décodeur utilise des algorithmes sous-optimaux. Parmi ces algorithmes, nous n'évoquerons que l'algorithme MAP symbole afin de bien différencier cet algorithme du MAP paquet. Le but du MAP symbole est de minimiser la probabilité d'erreur sur les symboles (et donc le TEB), ce qui ne garantit pas pour autant que le résultat obtenu soit un mot de code.

2.1.2 Modulation

Afin de transmettre le message au travers du milieu de transmission, le modulateur génère un signal porteur, dont la forme d'onde peut-être soit une suite d'impulsions soit une onde sinusoïdale. Dans le cas de la modulation numérique, le modulateur transpose chaque ensemble de m bits du message entrant dans le modulateur au débit binaire D_b en un signal physique de durée $T = \frac{mR}{D_b}$. Les 2^m signaux physiques possibles forment ce que l'on appelle la constellation de la modulation. Dans le cas d'une onde sinusoïdale, le message peut être porté par la phase, l'amplitude ou/et la fréquence (OOK, PSK, FSK, CP FSK, QAM, ...).

Les caractéristiques principales d'une modulation sont : sa taille m, sa constellation et sa largeur de bande, qui d'après le critère de Nyquist est supérieure à $\frac{1}{2T}$. Par ailleurs, on mesure les performances d'une modulation grâce à l'efficacité spectrale η, qui fournit le nombre de bits d'information transmis par unité de temps et par unité de bande, soit:

$$\eta = \frac{D_b}{B} = \frac{mR}{BT} \text{ bit/Hz/s} \tag{2.2}$$

Ainsi à un rapport signal à bruit donné, une modulation sera d'autant plus efficace que l'efficacité spectrale sera proche de la limite imposée par le théorème de Shannon.

Le démodulateur effectue le travail inverse du modulateur, c'est-à-dire qu'il convertit le signal reçu du milieu dans le format d'entrée du décodeur de canal. En pratique, le démodulateur doit détecter le signal entrant et fournir une mesure de fiabilité pour chaque point de la constellation. De ces mesures et de la détection dépend la probabilité d'erreur à un rapport signal à bruit donné.

L'approche de la modulation, comme interface entre le code canal et le milieu de propagation, peut dans certains cas rendre l'opération de modulation très complexe par une augmentation conséquente du nombre d'états de la constellation. Par exemple, la modulation peut devoir offrir un accès au canal à plusieurs utilisateurs ou supporter un canal ayant plusieurs entrées et plusieurs sorties.

2.1.2.1 Multi-utilisateurs

Utiliser un canal à accès multiple permet de partager les ressources du canal, c'est-à-dire d'exploiter au mieux la bande passante mise à disposition par le canal. L'enjeu d'une telle opération est à la mesure des coûts de ces ressources. Malheureusement comme dans tous partages, cela implique des interactions entre les protagonistes (ici, les signaux des utilisateurs). Pour limiter les interactions, plusieurs méthodes existent :

- le multiplexage temporel (TDMA en anglais), qui répartit la transmission des signaux des utilisateurs sur des intervalles de temps distincts.
- le multiplexage fréquentiel (FDMA en anglais), qui répartit la transmission des signaux des utilisateurs sur des bandes de fréquences distinctes.

– le multiplexage par code (CDMA en anglais), qui répartit la transmission des signaux des utilisateurs grâce à des codes d'étalement orthogonaux entre eux permettant à l'ensemble des utilisateurs d'accéder au même espace temps-fréquence.
– le multiplexage par entrelaceur (IDMA en anglais), qui répartit la transmission des signaux des utilisateurs en affectant à chacun un entrelacement différent derrière le code de canal.

Sans insister sur les avantages et inconvénients de ces techniques, toutes imposent de changer la détection au niveau du démodulateur. Cette étape, qui peut devenir très complexe, doit garantir l'élimination des interférences entre utilisateurs pour assurer de bonnes performances.

2.1.2.2 MIMO

Un système MIMO est basé sur un milieu de propagation ayant plusieurs entrées et plusieurs sorties. Il peut s'agir par exemple des très à la mode systèmes sans fil multi antennes ou encore des systèmes filaires haut-débits soumis à la diaphonie (croostalk). Malgré les interférences des signaux transmis entre eux, un système MIMO peut servir deux objectifs antagonistes : augmenter la capacité du canal ou augmenter la diversité de codage. Le premier cas aurait pu être développé dans la section précédente, car il s'agit en fait en d'un multiplexage spatial, i.e. que chaque flux (utilisateur) est envoyé sur une entrée du milieu de propagation. A l'inverse, un système MIMO cherchant à augmenter la diversité de codage ne transmet qu'un seul flux séparé sur les multiples entrées du milieu par l'intermédiaire d'un code. Le choix du code (espace-temps) et le nombre d'antennes permettent d'améliorer la probabilité d'erreur de la transmission.

2.1.3 Milieu de transmission

Pour transiter de l'émetteur au récepteur, l'information utilise un média que l'on nomme milieu de transmission. Qu'il s'agisse de l'air, d'une fibre optique, d'un support de stockage, d'une paire de câbles torsadés, etc., le milieu de transmission est toujours soumis à des perturbations pouvant déformer le message transmis. Ces perturbations peuvent s'exprimer sous la forme :

– d'un bruit thermique, que l'on trouve dans la plupart des milieux de transmission. Cette perturbation pouvant être modélisée par un processus aléatoire gaussien sera abordée pus tard.
– d'effacement, lorsque la donnée est perdue par le canal, ce qui arrive principalement dans les canaux à entrée et sortie binaire.
– d'atténuation. On parle de canaux à évanouissements lorsque l'atténuation évolue au cours du temps (par exemple le canal de Rayleigh) et de canal sélectif en fréquence, lorsque l'atténuation n'est pas uniforme dans la bande de fréquence utilisée.
– d'interférences, qui peuvent provenir d'autres utilisateurs (canal multi-utilisateur), ou d'autres signaux (canal multi-utilisateur multi entrée), ou d'autres symboles du même signal. L'interférence entre symboles caractérise les canaux multi trajets, qui, par définition, génèrent en superposition au signal transmis un ou plusieurs échos de ce signal.

De toutes ces perturbations, seules les interférences entre utilisateurs sont prévisibles et peuvent donc être annulées lors de la détection du signal. Les autres interférences et les

atténuations lentes du canal peuvent également être annulées en procédant à une estimation du canal. En revanche, la nature aléatoire des autres perturbations rend impossible leur annulation. Pour les contrecarrer, il faut alors réaliser la transmission en exploitant les diversités (i.e. protection de l'information par divers moyens : temps, fréquence, espace, codage) adaptées pour ce canal. Par exemple, la diversité en temps peut s'avérer très utile sur les canaux à évanouissement, car elle permet d'affecter à une même information plusieurs instants de codage distincts. Or comme chaque instant de codage est associé à des atténuations différentes, l'information bénéficie en moyenne d'une meilleure protection. De même, la diversité en fréquence présente un intérêt sur les canaux sélectifs en fréquence comme c'est le cas dans l'OFDM.

2.1.3.1 Canal à Bruit Blanc Additif Gaussien (BBAG)

Le canal BBAG est un canal à entrée binaire et sortie analogique [47] tel que chaque symbole sortant du canal Y est la somme du symbole émis X et d'un bruit gaussien centré de variance σ^2 tel que : $\sigma^2 = N_0/2$, où N_0 représente la densité spectrale de puissance de bruit. Dans ces conditions, la probabilité conditionnelle que la sortie Y corresponde au symbole x_i est :

$$P(Y/X = x_i) = \frac{1}{\sigma\sqrt{2\Pi}} e^{-\frac{(x_i - Y)^2}{2\sigma^2}} \tag{2.3}$$

Les performances en taux d'erreur (TEB et TEP) en découlent ensuite et sont généralement exprimées en fonction du rapport signal à bruit $\frac{E_b}{N_0}$ où E_b est l'énergie par bit d'information.

2.1.3.2 Capacité d'un canal de transmission

La performance d'un code sur ce canal est alors évaluée en fonction de sa proximité avec la limite de Shannon, i.e. le rapport $\frac{E_b}{N_0}$ atteignant la capacité du canal. Pour le canal gaussien, la capacité s'exprime :

$$C = B.\lg(1 + SNR) \text{ bit/s} \tag{2.4}$$

avec le rapport signal à bruit :

$$SNR = \frac{D_b.E_b}{B.N_0} \tag{2.5}$$

d'où d'après 2.2 :

$$C = B.\lg\left(1 + \eta\frac{E_b}{N_0}\right) \tag{2.6}$$

Par définition, on peut en déduire que pour l'efficacité spectrale $\eta_{\max}$ atteignant la capacité, on aura :

$$\eta_{\max} = \lg\left(1 + \eta_{\max}\frac{E_b}{N_0}\right) \tag{2.7}$$

On conséquence la limite de Shannon pour cette efficacité spectrale sera de :

$$\left.\frac{E_b}{N_0}\right|_{limite} = \frac{2^{\eta_{\max}} - 1}{\eta_{\max}} \tag{2.8}$$

2.2 Turbo-réception

Dans une chaîne de transmission, la partie d'émission suit des règles prédéterminées de sorte qu'elles soient connues du récepteur. Ainsi les principales difficultés algorithmiques sont reportées à la réception. Outre la complexité algorithmique du récepteur, ce dernier, de par l'enchaînement séquentiel des étapes élémentaires du récepteur, répercute les erreurs intervenant lors d'une étape élémentaire sur l'ensemble des étapes élémentaires suivantes dans la chaîne de transmission. En introduisant le décodage itératif, l'invention des turbocodes [48] a révolutionné la manière de concevoir le récepteur d'une chaîne de transmission. Nous introduirons ce concept de turbo-récepteur en présentant le principe turbo et ces applications au sein du récepteur.

2.2.1 Principe turbo ou traitement itératif

Le principe turbo, aussi nommé traitement itératif [49], consiste à casser l'enchaînement séquentiel des étapes élémentaires de la réception par l'introduction d'une boucle de retour. Grâce à ce principe, la réception est globalement améliorée, car le principe autorise qu'une étape élémentaire bénéficie d'informations obtenues par des étapes élémentaires en aval, ce qui s'avère impossible avec l'enchaînement séquentiel des étapes. Cet effet de retour, qui dans la pratique est réalisé de manière itérative, permet donc de traquer les anomalies de transmission non plus localement dans le récepteur mais de manière collective en son sein. Même s'il faut garder à l'esprit que l'obtention d'un optimum global ne réside pas forcément dans une somme d'optima locaux, l'avantage de ce principe est indéniable. Néanmoins, il implique bon nombre d'interrogations : comment et quand échanger l'information, faut-il tout échanger. Ces questions demeurent d'actualité, même si la littérature dans le domaine permet de répondre partiellement à ces questions.

On sait par exemple que les échanges d'informations doivent être de nature probabiliste. Ainsi chaque étape élémentaire doit être en mesure d'émettre et recevoir des informations pondérées. On parle alors de bloc à entrées et sorties pondérées (désigné par la suite par l'acronyme SISO provenant de l'anglais Soft Input Soft Output). L'information échangée doit également respecter la règle suivante : "ne pas contenir d'information déjà connue du bloc SISO de destination". On dit alors de l'information échangée qu'elle est extrinsèque.

La question du moment de l'échange ne trouve pas vraiment d'écho dans la littérature. Bien souvent, les échanges d'informations sont réalisés entre deux blocs élémentaires sous la forme d'un dialogue : d'abord le bloc en amont, puis le bloc en aval, amont, aval, etc. Seulement à plus de deux, les règles du dialogue doivent évoluer. Autant il est clair que les premières répliques doivent partir du bloc le plus en amont pour arriver au bloc le plus en aval, autant l'enchaînement des répliques suivantes entre les blocs est déterminé de manière plus ou moins empirique pour aboutir à la réplique finale du bloc le plus en aval, concluant le dialogue. Certes, il est possible d'établir pour chaque bloc élémentaire des critères d'arrêt afin de déclencher leur mutisme dans le dialogue [50][51]. Il existe également des moyens d'évaluer la convergence d'un processus itératif (diagramme EXIT [52][53], diagramme d'évolution de densité [54], analyse des systémes dynamiques non-linéaires [55]). Mais ce sont des moyens

qui permettent d'orchestrer le dialogue sur sa durée et pas forcément de le séquencer. La question du séquencement a d'ailleurs été remise en question avec l'apparition du décodage combiné [56] (ou shuffled decoding), puisque dans ce décodage les blocs élémentaires discutent en même temps. Pour rester dans l'analogie avec la communication orale, cette méthode de décodage transpose la communication des blocs d'un dialogue à un chœur. Trop peu d'éléments permettent de dire s'il en résulte de la cacophonie ou, au contraire, de l'euphonie.

2.2.2 Du turbo dans la chaîne de transmission

Le décodage de canal en tant que berceau du principe turbo a largement profité de ses bienfaits. Il existe aujourd'hui de nombreuses familles de code de canal : turbocodes convolutifs, turbocodes produits, codes LDPC, codes repeat-accumulate,... autant de codes, qui approchent la limite de Shannon, en recourant à des échanges itératifs d'information, soit entre des décodeurs élémentaires, soit à l'intérieur même de l'algorithme de décodage. Le principe s'est ensuite propagé à d'autres blocs de la chaîne de réception. La figure 2.2 est un exemple de turbo-récepteur pour la chaîne de transmission présentée sur la figure 2.1.

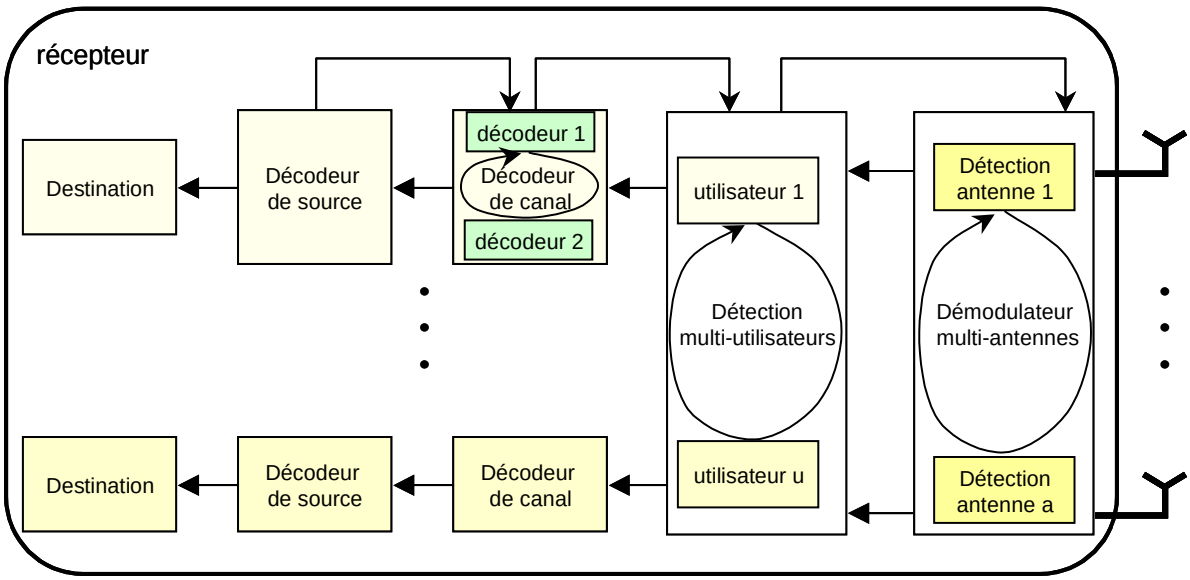

Figure 2.2 — Exemple de traitement itératif dans un récepteur

Par exemple, Hagenauer propose dans [57] un décodage itératif entre les codeurs de source et de canal afin d'exploiter la redondance résiduelle à l'issue du codage de source. De nombreux retours d'informations sont également possibles en détection : la turbo-démodulation entre le démodulateur et le décodeur de canal [58], la turbo-détection multi utilisateur [59], la turbo-détection MIMO [60], la turbo-égalisation entre un égaliseur (nécessaire pour les canaux multi-trajet) et le décodeur de canal [61], la turbo-synchronisation [62].

2.3 Les turbocodes convolutifs

Cette section présente la famille des turbocodes convolutifs à l'origine de l'effet turbo. C'est pourquoi nous présenterons d'abord les codes élémentaires d'un turbocode convolutif,

puis les structures permettant de concaténer plusieurs codes élémentaires de manière à former un turbocode seront introduites. La fin de la section détaillera le décodage itératif utilisé pour ces codes.

2.3.1 Codes convolutifs

2.3.1.1 Définition

Un code convolutif de rendement $\frac{m}{n}$ est une fonction linéaire qui transforme à chaque instant i un vecteur d'entrée d_i de m bits en un vecteur de sortie c_i de n bits ($n > m$) en tenant compte de v bits mémorisés dans un registre à décalage s_i lors des transformations précédentes. La valeur du registre s_i définit un état du codeur parmi les 2^v états possibles. Les valeurs des sorties, ainsi que les futurs états de mémorisations, sont obtenues par combinaison linéaire (avec l'opérateur "ou exclusif" comme additionneur) des bits d'entrée $d_{i,k}$ et de mémorisation $s_{i,k}$.

Un code est dit systématique si le message d_i est entièrement inclus dans la séquence codée c_i. On dit également d'un code qu'il est récursif s'il existe une boucle de retour au sein du registre à décalage.

2.3.1.2 Structure du codeur

D'après la définition, le code peut être représentée de façon matricielle avec la matrice de code G :

$$\begin{pmatrix} c_{i,1} \\ \dots \\ c_{i,n} \\ s_{i+1,1} \\ \dots \\ s_{i+1,v} \end{pmatrix} = \begin{pmatrix} g_{1,1} & & g_{1,m} & g_{1,m+1} & & g_{1,m+v} \\ & \dots & & & \dots & \\ g_{n,1} & & g_{n,m} & g_{n,m+1} & & g_{n,m+v} \\ g_{n+1,1} & & g_{n+1,m} & g_{n+1,m+1} & & g_{n+1,m+v} \\ & \dots & & & \dots & \\ g_{n+v,1} & & g_{n+v,m} & g_{n+v,m+1} & & g_{n+v,m+v} \end{pmatrix} \begin{pmatrix} d_{i,1} \\ \dots \\ d_{i,m} \\ s_{i,1} \\ \dots \\ s_{i,v} \end{pmatrix} \quad (2.9)$$

La sous-matrice de G intégrant seulement les n premières lignes est généralement appelée matrice des polynômes générateurs. Dans le cas d'un code systématique, la sous-matrice kxk dans le coin supérieur gauche de G est une matrice diagonale et le complément de matrice mxv à sa droite est une matrice nulle.

La sous-matrice vxv dans le coin inférieur droit de G est la matrice de mémorisation. Classiquement il s'agit d'une diagonale de 1 sur les coefficients $(g_{n+1+k,m+k})_{0<k<v}$ représentant le registre à décalage. Avec cette représentation, la récursivité d'un code s'exprime par la présence d'un 1 au dessus de cette diagonale. Dans la pratique, les codes récursifs utilisés n'appliquent une boucle de retour que sur le premier registre du registre à décalage, i.e. des 1 dans le triangle de récursivité uniquement sur la première ligne. Cela permet de définir un polynôme de récursivité pour le code.

La figure 2.3 représente plusieurs codeurs convolutif couramment utilisés ayant comme matrice :

$$G_a = \begin{pmatrix} \boxed{\begin{matrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{matrix}} \\ 1 \quad 0 \quad 0 \quad 0 \\ 0 \quad 1 \quad 0 \quad 0 \\ 0 \quad 0 \quad 1 \quad 0 \end{pmatrix}, G_b = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & \boxed{\begin{matrix} 1 & 0 & 1 \end{matrix}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \text{ et } G_c = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

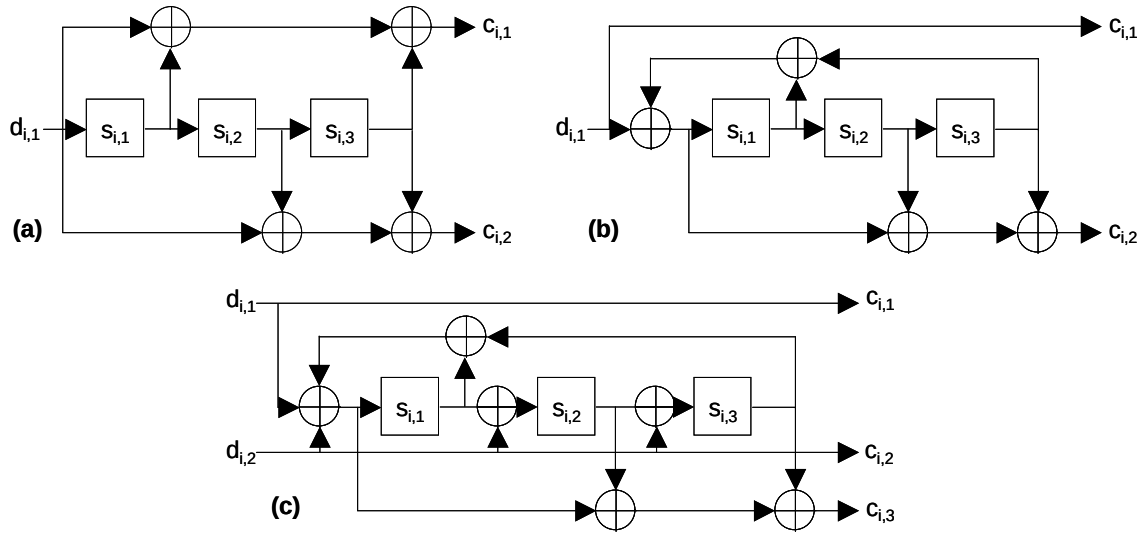

Figure 2.3 — Exemples de codeurs : (a) code convolutif (15,13), (b) code convolutif systématique récursif simple binaire, (c) code convolutif systématique récursif double binaire

Le codeur (a) est un code convolutif non-récursif non-systématique. Il est donc entièrement caractérisable par sa matrice génératrice qui est souvent donnée en octal (15,13). Le codeur (b), notamment utilisé dans le standard UMTS, est la version récursive du code (a). En conséquence, on peut retrouver la matrice génératrice de (a) en sommant le polynôme de récursivité de (b) ('101' encadré dans la matrice) et sa matrice génératrice. Le codeur (c) est le codeur double binaire systématique récursif présent dans les derniers standards.

Il existe beaucoup d'autres manières de représenter un code convolutif, et notamment la représentation en treillis, qui est très adaptée pour illustrer le décodage. Dans un treillis, on représente à chaque instant de codage les 2^v états possibles. De chacun de ces états partent 2^m transitions. Ainsi une transition constitue un vecteur d'entrée de la matrice G du code et on peut y associer un état de sortie et une étiquette portant le mot codé associé à cette transition. Par exemple, la figure 2.4 représente le treillis associé au codeur (b).

Outre la structure génératrice du codeur, le codeur doit également intégrer d'une part un moyen de rendre flexible le rendement pour le besoin des applications et d'autre part un moyen de gérer des tailles de bloc finie.

Pour les codes utilisés en pratique, le rendement naturel d'un code convolutif est assez faible et généralement inférieur à $\frac{2}{3}$. Pour autoriser des rendements plus élevés, on utilise la technique du poinçonnage qui consiste à ne pas transmettre l'ensemble des bits codés. Chaque bit du symbole codé est alors affecté à un motif de poinçonnage cyclique où un '0' signifie une donnée poinçonnée et un '1' une donnée transmise. Ainsi un code de rendement $\frac{1}{2}$ poinçonné avec le motif $\begin{matrix} c_{i,1} \\ c_{i,2} \end{matrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ aboutit à un rendement de $\frac{3}{4}$. L'avantage principal de cette technique se situe au niveau du décodeur dont la structure est inchangée. La seule contrainte

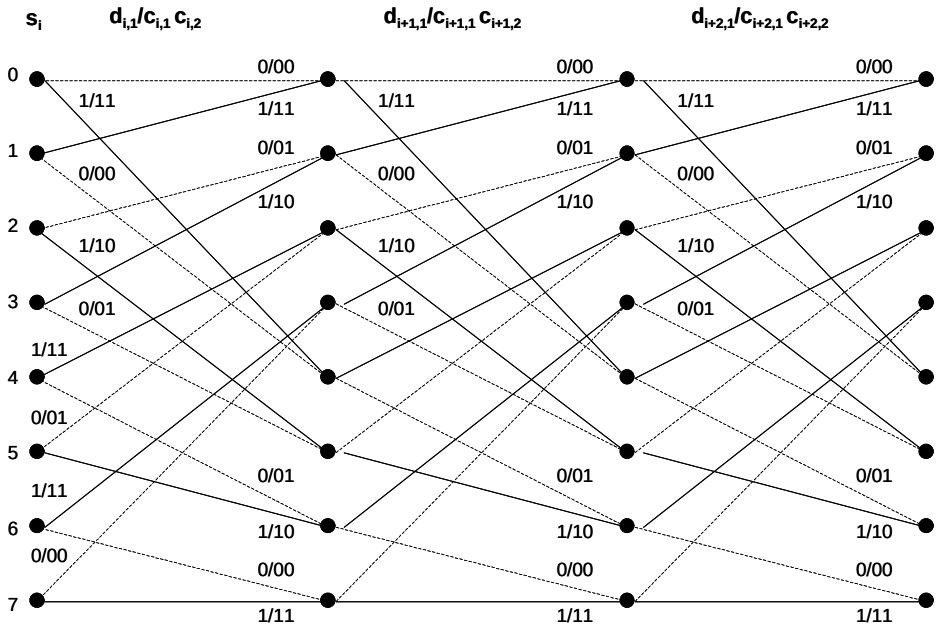

Figure 2.4 — Treillis associé au codeur de la figure 2.3.b

consiste à dépoinçonner le code en amont du décodeur en remplacant les données poinçonnées par des valeurs neutres pour le décodeur.

Pour décoder correctement un code convolutif, le décodeur doit être en mesure de connaître chacun des états du codeur. Or, pour un bloc de taille finie et un codeur initialisé à l'état 0, on ne dispose d'aucune information sur l'état final du codage. Ce problème dit de fermeture du treillis peut être résolu de deux manières. La première technique consiste à forcer le codeur à retourner dans un état final connu en ajoutant à la fin du bloc transmis des symboles supplémentaires. Cette méthode a l'inconvénient d'impliquer une diminution du rendement. La deuxième méthode contourne ce problème en rendant le code circulaire, i.e. un code dont l'état d'initialisation est identique à l'état final. L'état d'initialisation du décodeur peut être obtenu à partir de la séquence d'entrée et des paramètres du code comme proposé dans [63]. Comme cet état reste inconnu du décodeur, la structure du treillis est circulaire, ce qui assure une égale protection de tous les symboles.

2.3.1.3 Algorithmes de décodage à entrées et sorties pondérées

Historiquement de nombreux algorithmes existent pour décoder un code convolutif. Les premiers algorithmes de Fano [64] puis Viterbi [65] étaient à entrées et sorties binaires. L'algorithme de Virtebi, le plus efficace des deux, a d'abord été modifié pour accepter des entrées souples [66] avec à la clé une amélioration du décodage. La concaténation des codes a ensuite poussé l'apparition d'algorithmes de décodage à entrées et sorties pondérées (SISO en anglais) tels que l'algorithme SOVA [67]. Parmi les algorithmes SISO, aujourd'hui indispensable à la turbo-réception, l'algorithme Bahl-Cock-Jelinek-Raviv (BCJR) [68] s'est imposé grâce à son optimalité au sens symbole, car cet algorithme, aussi appelé MAP (Maximum A Posteriori) ou algorithme aller-retour, calcule la probabilité de chaque symbole à partir des probabilités de tous les chemins possibles dans le treillis entre l'état initial et l'état final. En pratique, pour

des raisons de complexité, l'algorithme BCJR n'est pas implanté sous sa forme probabilité, mais est dérivé dans le domaine logarithmique de manière à transformer les multiplications en additions. On parle alors de l'algorithme log-MAP ou, dans une version sous-optimale, de l'algorithme max-log-MAP.

La suite de cette section décrit les algorithmes MAP, log-MAP et max-log-MAP pour décoder les codes convolutifs m-binaires.

Décodage avec l'algorithme MAP

Un décodeur MAP fournit, pour chaque symbole codé d_i, 2^m probabilités *a posteriori* (i.e. avec l'entière connaissance de la séquence y reçue par le décodeur), soit une par décision possible sur le symbole. La décision dure correspondante est la valeur j qui maximise la probabilité *a posteriori*. Ces probabilités peuvent s'exprimer en fonction des vraisemblances conjointes $p(d_i \equiv j, y)$:

$$\Pr\left(d_i \equiv j | y\right) = \frac{p(d_i \equiv j, y)}{\sum_{k=0}^{2^m-1} p(d_i \equiv k, y)} \tag{2.10}$$

Or la structure en treillis du code permet de décomposer le calcul des vraisemblances conjointes en séparant les observations sur le passé des observations sur le futur. Cette décomposition utilise ainsi une métrique récurrente aller $\alpha_i(s)$ (forward) pour déterminer la vraisemblance d'un état du treillis à l'instant i selon son passé, une métrique récurrente retour $\beta_i(s)$ (backward) pour déterminer la vraisemblance d'un état du treillis à l'instant i selon son futur, et une métrique mesurant la vraisemblance d'une branche entre deux états du treillis $\gamma_i(s', s)$.

$$p(d_i \equiv j, y) = \sum_{(s',s)/d_i \equiv j} \beta_{i+1}(s)\, \alpha_i\left(s'\right) \gamma_i\left(s', s\right) \tag{2.11}$$

Les métriques récurrentes aller et retour sont calculées de la manière suivante :

$$\alpha_{i+1}(s) = \sum_{s'=0}^{2^v-1} \alpha_i\left(s'\right) \gamma_i\left(s', s\right) \quad \text{pour } i = 0...N-1 \tag{2.12}$$

$$\beta_i(s) = \sum_{s'=0}^{2^v-1} \beta_{i+1}\left(s'\right) \gamma_i\left(s, s'\right) \quad \text{pour } i = N-1...0 \tag{2.13}$$

L'initialisation de ces métriques dépend respectivement de la connaissance de l'état de départ et de l'état final du treillis. Par exemple, la connaissance de l'état de départ S_0 du codeur permet d'initialiser $\alpha_0(S_0)$ à 1 et les autres $\alpha_0(s)$ à 0. Si l'état est inconnu, on initialise toutes les métriques à la même valeur.

Par ailleurs, la métrique de vraisemblance d'une branche s'exprime :

$$\gamma_i\left(s', s\right) = p(y_i|x_i).\Pr{}^a\left(d_i = d_i(s', s)\right) \tag{2.14}$$

avec $p(y_i|x_i)$ la probabilité de transition du canal, qui s'exprime dans le cas d'un canal gaussien comme :

$$p(y_i|x_i) = \prod_{k=1}^{n} \left(\frac{1}{\sigma\sqrt{2\Pi}} . e^{-\frac{\left(y_{i,k}-x_{i,k}\right)^2}{2\sigma^2}} \right) = K.e^{\frac{\sum_{k=1}^{n} y_{i,k}.x_{i,k}}{\sigma^2}} \tag{2.15}$$

où x_i est le vecteur de symboles modulés.

La probabilité *a priori* d'émettre le m-uplet d'information correspondant à la transition de s' vers s $\Pr^a(d_i = d_i(s', s))$ vaut 0 si la transition n'existe pas dans le treillis. Sinon sa valeur est dépendante de la statistique de la source. Pour une source équiprobable, on a $\Pr^a(d_i = j) = \frac{1}{2^m}$. Dans le cadre d'un décodage itératif, l'information *a priori* tient également compte de l'information extrinsèque entrante.

L'information extrinsèque générée par le décodeur est simplement l'information a posteriori (2.10) avec une métrique de branche modifiée :

$$\Pr^{ex}(d_i \equiv j|y) = \frac{\sum_{(s',s)/d_i \equiv j} \beta_{i+1}(s)\,\alpha_i(s')\,\gamma_i^{ex}(s', s)}{\sum_{(s',s)} \beta_{i+1}(s)\,\alpha_i(s')\,\gamma_i^{ex}(s', s)} \tag{2.16}$$

La métrique de branche ne tient alors plus compte des informations déjà disponible dans le décodeur auquel l'information extrinsèque est destinée. Pour un turbocode convolutif concaténé en parallèle, il s'agit de retirer la partie systématique et la nouvelle métrique de branche s'exprime :

$$\gamma_i^{ex}\left(s', s\right) = K.e^{\frac{\sum_{k=m+1}^{n} y_{i,k}.x_{i,k}}{\sigma^2}} \tag{2.17}$$

Décodage avec l'algorithme log-MAP ou max-log-MAP

L'algorithme log-MAP, introduit par [69], est la transposition directe de l'algorithme MAP dans le domaine logarithmique.Ainsi toute métrique M de l'algorithme MAP est remplacé par une métrique $\sigma^2 \ln M$ dans l'algorithme log-MAP. Cela permet de passer l'algorithme du semi-anneau somme-produit $(R^+, +, \times, 0, 1)$ au semi-anneau $(R, max^*, +, -\infty, 0)$ avec l'opérateur max^* défini comme :

$$max^*(x, y) = \sigma^2 \ln\left(e^{\frac{x}{\sigma^2}} + e^{\frac{y}{\sigma^2}}\right) = \max(x, y) + \sigma^2 \ln\left(1 + e^{-\frac{|x-y|}{\sigma^2}}\right) \approx \max(x, y) \tag{2.18}$$

Cet opérateur peut être simplifié par un opérateur max, ce qui donne lieu à l'algorithme max-log-MAP. Le terme eclipsé est connu sous le nom de logarithme Jacobien. La suite de la section détaille les métriques de l'algorithme log-MAP. Les métriques de branches (2.17) et (2.14) peuvent alors s'écrire :

$$c_i^{ex}\left(s', s\right) = \sigma^2 \ln \gamma_i^{ex}\left(s', s\right) = K' + \sum_{k=m+1}^{n} y_{i,k}.x_{i,k} \tag{2.19}$$

et

$$c_i\left(s',s\right)=\sigma^2\ln\gamma_i\left(s',s\right)=K'+L_i^a(j)+\sum_{k=1}^{n}y_{i,k}.x_{i,k}=c_i^{ex}\left(s',s\right)+L_i^a(j)+L_i^{sys}(j) \quad (2.20)$$

avec $L_i^a(j)$ la métrique d'information *a priori* et $L_i^{sys}(j)$ la métrique correspondant à l'information provenant des données systématiques. Notons que la constante K' n'est pas nécessaire en pratique puisqu'elle s'annule dans les calculs d'informations (2.23) et (2.24).

De la même manière, les métriques récurrentes aller et retour peut être réécrites :

$$a_{i+1}\left(s\right)=\sigma^2\ln\alpha_{i+1}\left(s\right)=\underset{s'=0}{\overset{2^v-1}{max^*}}\left(a_i\left(s'\right)+c_i\left(s',s\right)\right) \quad \text{pour } i=0...N-1 \quad (2.21)$$

$$b_i\left(s\right)=\sigma^2\ln\beta_i\left(s\right)=\underset{s'=0}{\overset{2^v-1}{max^*}}\left(b_{i+1}\left(s'\right)+c_i\left(s,s'\right)\right) \quad \text{pour } i=N-1...0 \quad (2.22)$$

Dans ce cas, les métriques sont initialisées à l'état initial ou final :

- en cas d'état S connu, par $a(S)=0$ pour l'état connu et $a(s\neq S)=-\infty$.
- en cas d'état inconnu, par $a(s)=0$ pour tous les états.

Les métriques d'information *a posteriori* (2.10) et d'information extrinsèque (2.16) sont transformées en :

$$\begin{aligned}L_i(j)&=\sigma^2\ln\Pr\left(d_i\equiv j|y\right) \quad (2.23)\\&=\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)-\underset{(s',s)}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)\end{aligned}$$

$$L_i^{ex}(j)=\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i^{ex}\left(s',s\right)+b_{i+1}\left(s\right)\right)-\underset{(s',s)}{max^*}\left(a_i\left(s'\right)+c_i^{ex}\left(s',s\right)+b_{i+1}\left(s\right)\right) \quad (2.24)$$

En simplifiant le terme de droite de ces métriques en utilisant le symbole le plus probable $\widehat{j}$ de la manière suivante :

$$\begin{aligned}\underset{(s',s)}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)&=\underset{j=0}{\overset{m}{max^*}}\left(\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)\right) \quad (2.25)\\&\approx\underset{j=0}{\overset{m}{max}}\left(\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)\right)\\&=\underset{(s',s)/d_i\equiv\widehat{j}}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)\end{aligned}$$

et comme par distributivité :

$$\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i\left(s',s\right)+b_{i+1}\left(s\right)\right)= \quad (2.26)$$
$$L_i^a(j)+L_i^{sys}(j)+\underset{(s',s)/d_i\equiv j}{max^*}\left(a_i\left(s'\right)+c_i^{ex}\left(s',s\right)+b_{i+1}\left(s\right)\right)$$

On peut lier dans le cadre de cette simplification les métriques d'information par la relation suivante :

$$L_i(j) = L_i^a(j) + L_i^{sys}(j) + L_i^{ex}(j) - L_i^a\left(\widehat{j}\right) - L_i^{sys}\left(\widehat{j}\right) \tag{2.27}$$

Cette simplification est souvent entendue comme faisant partie de l'algorithme log-MAP et les différences de performance introduites par rapport au MAP sont minimes (moins de 0,05dB). Bien que un peu plus sous-optimal que l'algorithme log-MAP, l'algorithme max-log-MAP présente de nombreux avantages. Premièrement la dégradation en performance entre les deux est relativement faible, inférieure à 0,1 dB pour un code double binaire [70]. Deuxièmement, en éliminant le logarithme jacobien de l'algorithme log-MAP, l'algorithme max-log-MAP d'une part économise sur la complexité (car l'implantation du logarithme Jacobien est réalisée avec des tables) et d'autre part permet d'augmenter la fréquence de fonctionnement (car le chemin critique traverse dans la plupart des circuits l'opérateur max^*). En outre, il n'est pas nécessaire de connaître la valeur de σ^2 pour utiliser l'algorithme max-log-MAP. Il n'y a donc pas besoin d'estimer le canal pour obtenir ce paramètre.

L'algorithme présenté jusqu'ici était considéré dans $\mathbb{R}$. De nombreux travaux existent quant à la quantification de cet algorithme, qui peut être séparée en 3 étapes. Premièrement, il faut déterminer la quantification nécessaire en entrée du décodeur, ce qui inclut à la fois le nombre de bits et le pas entre deux échantillons [71] [72] [73]. Cette étape nécessite un compromis entre les performances de décodage désirés, qui augmentent avec le nombre de bits de quantification jusqu'à se rapprocher du cas réel, et la complexité du décodeur, qui augmente dans les mêmes conditions. Selon les codes et les conditions de fonctionnement du décodeurs, entre 3 et 6 bits de quantification suffisent généralement à avoir de bonnes performances de décodage. La deuxième étape consiste à propager la dynamique du signal entrant sur l'ensemble des métriques du décodeur afin de conserver les performances de correction du décodeur [74] [75]. La dernière étape consiste à choisir le format de représentation des métriques. Traditionnellement, pour éviter les dépassements de capacité (overflow) sur les métriques récurrentes, ces métriques dites d'états sont normalisées à chaque instant de codage par rapport à l'une d'entre elle, souvent la plus faible. De cette manière, l'ensemble des métriques conserve une dynamique restreinte au cours du décodage. Cette méthode de normalisation a l'inconvénient d'imposer des étapes de calcul supplémentaire. Une autre méthode de normalisation permet de les éviter en utilisant non plus l'arithmétique classique, mais l'arithmétique modulo [76]. Cette méthode, que nous avons utilisée dans ces travaux, nécessite simplement un format de représentation deux fois plus grand que celui nécessaire avec l'arithmétique classique, ce qui est illustré par la figure 2.5. La sous-figure (a) y représente le format de représentation nécessaire en arithmétique classique pour couvrir au plus juste la dynamique des métriques. De plus, grâce à des soustractions effectuées sur les métriques par rapport à la plus petite de ces métriques, il est possible par exemple d'aligner la dynamique des métriques sur 0. En gardant le même format de représentation mais transposé en arithmétique modulo (sous-figure b), la dynamique des métriques n'est plus alignée sur une valeur fixe et il devient alors impossible de préserver l'ordonnancement correct des métriques. En ajoutant un bit de quantification au format de représentation (sous-figure c), on sait que la dynamique des métriques utilise moins de la moitié du format de représentation. On peut donc utiliser le bit de poid fort de la différence de deux métriques pour déterminer l'ordre de ces deux métriques.

2.3.2 Concaténation de codes

On sait depuis longtemps créer des codes ayant un pouvoir de correction garantissant une transmission fiable pour la plupart des applications. Néanmoins cette connaissance ne peut être mise en oeuvre à cause de la complexité prohibitive des décodeurs qui seraient associés à de tels codes. Or quand un problème est trop complexe, l'approche "diviser pour régner" peut être un bon moyen de le simplifier. La concaténation de codes est une conséquence de ce problème. L'idée, introduite par Forney [77], consiste à bâtir un code possédant un pouvoir de correction suffisant à partir de plusieurs codes simples. Cette section présente brièvement des différentes structures de concaténation.

2.3.2.1 Structure de concaténation

La proposition originale de Forney concatène un code intérieur à un code extérieur comme illustré sur la figure 2.6.a. On parle alors dans ce cas d'une concaténation série, puisque la sortie du code extérieur sert d'entrée au code intérieur. Par la suite, il a été observé que l'ajout d'une fonction d'entrelacement entre les deux codes permet d'augmenter significativement la robustesse du code concaténé. C'est pourquoi ce qu'on appelle de nos jours un code concaténé série ressemble plutôt à la représentation 2.6.b. Avec l'apparition des turbocodes, une nouvelle structure a été introduite : la concaténation parallèle (figure 2.6.c). Cette structure associée à des codeurs systématiques permet de générer plusieurs sources de redondance différentes pour coder le même message.

Dans une structure de concaténation, le nombre de codes élémentaires est appelé dimension du code concaténé. Dans la pratique, les codes concaténés se contentent de deux dimensions, car il faut garder à l'esprit que la concaténation implique une diminution du rendement de codage du code concaténé. Par exemple, un code série constitué de deux codes élémentaires de rendements R_1 et R_2 a un rendement global de codage de $R = R_1 R_2$ tandis qu'un code parallèle utilisant des codes élémentaires systématiques a un rendement globale de $R = \frac{R_1 R_2}{1-(1-R_1)(1-R_2)}$. Outre le fait qu'un code parallèle systématique a un rendement meilleur que celui d'un code série, on vérifie bien que le rendement globale chute lors d'une concaténation. L'ajout d'une dimension supplémentaire ne peut donc être envisagé qu'avec des codes composants ayant un rendement proche de 1.

Parmi les différences que comptent les structures série et parallèle, il est important de noter qu'une concaténation série présente de meilleures performances asymptotiques ($d_{\min}$ plus grande), puisque le décodeur extérieur bénéficie de la correction du décodeur intérieur

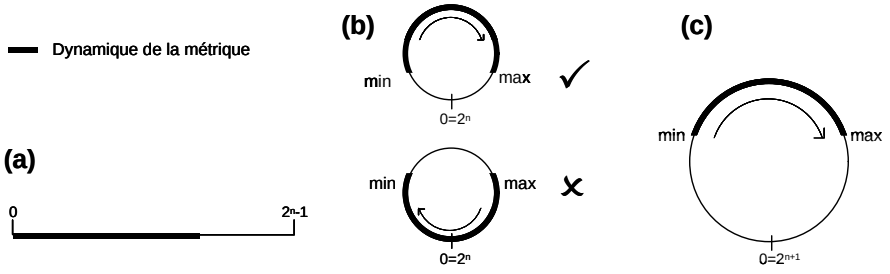

Figure 2.5 — Format des métriques : (a) arithmétique classique n bits, (b) arithmétique modulo n bits précision insuffisante, (c) arithmétique modulo n+1 bits précision suffisante

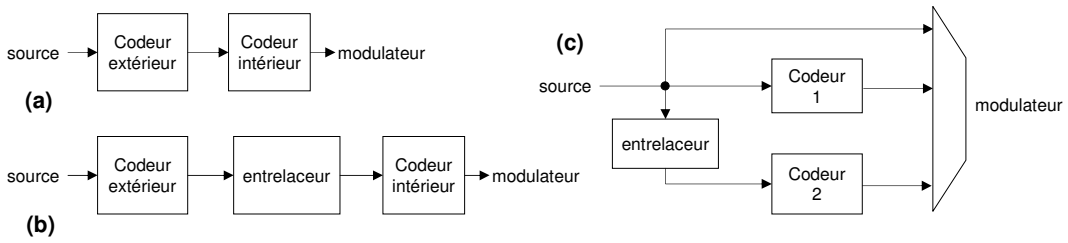

Figure 2.6 — Structure de concaténation : (a) série original, (b) série, (c) parallèle

[78]. En revanche, les performances dans la zone de convergence, i.e. proche de la limite théorique, sont meilleures avec la concaténation parallèle qu'avec la concaténation série.

En marge des structures classiques série et parallèle, la littérature fleurit de structures hybrides entre série et parallèle dont l'idée est de profiter des avantages des deux structures [79]. La structure proposée dans [80] permet, par exemple, avec des deux codes convolutifs très simples (4 états) d'offrir à la fois de bonnes performances asymptotiques et une convergence assez proche de la limite de Shannon.

Les codes concaténés sont largement utilisés dans les standards de communication et le choix de leurs codes constituants ne se limite pas seulement aux codes convolutifs [81]. On trouve par exemple des codes algébriques (BCH, RS,...) dans des turbocodes produits qui peuvent être interprétés comme une concaténation doublement série[2], ou les codes repeat-accumulate qui concatène en série un code à répétition et un accumulateur. Néanmoins la plupart des standards repose sur une concaténation parallèle de code convolutif. Par exemple, les standards WiMAX (IEEE 802.16), Eutelsat (skyplex), DVB-RCT (canal de retour pour la diffusion télévisuelle terrestre) et DVB-RCS (canal de retour pour la diffusion télévisuelle satellitaire) utilisent le codeur double binaire huit états de la figure 2.3.c pour les deux codeurs composants de la structure parallèle, tandis que le codeur binaire huit états de la figure 2.3.b est intégré dans la structure parallèle des standards de téléphonie mobile de $3^{ème}$ génération UMTS et CDMA2000. Des standards comme CCSDS pour l'espace lointain ou Inmarsat pour les communications satellitaires ont recours à un codeur binaire seize états.

2.3.2.2 Entrelaceur

L'utilisation en communication numérique d'un entrelaceur résulte d'un besoin de dispersion temporelle des données. L'intérêt premier de son utilisation dans des codes concaténés est donc d'éclater les regroupements d'erreurs sortant d'un décodeur composant afin de présenter des erreurs isolées au décodeur suivant. On peut vérifier qu'un entrelaceur Π respecte cette propriété en étudiant son facteur de dispersion S donné par la distance minimale entre deux symboles dans l'espace ordre naturel/ordre entrelacé :

$$S = \min_{i,j} \left(|i - j| + |\Pi(i) - \Pi(j)| \right) \tag{2.28}$$

La conception d'entrelaceurs respectant un facteur de dispersion raisonnable peut être réalisée grâce à l'algorithme S-random proposé dans [82]. Cependant, même si ce genre d'entrelaceur convient pour valider les performances dans la zone de convergence d'un code, il ne

[2] C'est-à-dire la redondance de la redondance est interprétable par les deux décodeurs composants.

permet pas d'obtenir une distance de Hamming suffisante pour assurer au code de bonnes performances asymptotiques. Dès lors pour améliorer ces dernières, la conception de l'entrelaceur doit également tenir compte de la nature des codeurs composants. Ainsi, en connaissant les motifs d'erreur des décodeurs composants, l'entrelaceur doit associer des motifs d'erreur ayant un poids de Hamming faible dans un décodeur à des motifs de poids fort dans l'autre décodeur. Dans le cas contraire, l'association de motifs de poids faible ensemble implique une faible distance du code concaténé. La conception d'entrelaceurs respectant cette règle a été abondamment traitée dans la littérature. Nous nous limiterons au cas des entrelaceurs structurés que l'on peut générer à partir d'un petit nombre de paramètres et qui n'imposent pas de fait la coûteuse mémorisation de la table de permutation. Parmi les entrelaceurs appartenant à cette famille, on peut citer les entrelaceurs ARP [83] (Almost Regular Permutation) qui sont utilisés dans de nombreux standards, les entrelaceurs DRP [84] (Dithered Relative Prime) qui atteignent le gain d'entrelacement optimal et également les entrelaceurs QPP [85] (Quadratic Permutation Polynomial) qui, en ayant des performances très proches du gain d'entrelacement optimal, assurent également un entrelacement sans collision, ce qui est primordial dans les implantations matérielles.

2.3.3 Turbo-décodage de codes convolutifs

Cette section illustre le principe du turbo-décodage pour une concaténation parallèle de codes convolutifs (voir figure 2.6.c) et montre les performances de ces codes.

2.3.3.1 Décodage itératif d'un turbocode

Pour contourner la complexité exponentielle inhérente au décodage optimal (pour rappel : il nécessite l'algorithme MAP paquet comparant le mot reçu à l'ensemble des mots de code existants), le décodage d'un turbocode a recours au décodage itératif. Ainsi chaque code composant de la structure de concaténation est décodé séparément en utilisant un algorithme du type MAP symbole (voir section 2.3.1.3). En optimisant la probabilité d'erreur sur les symboles, l'algorithme MAP symbole permet à un décodeur composant de fournir l'information optimale[3] à l'autre décodeur composant. Comme le montre la figure 2.7, le décodeur composant 0 envoie pour chaque symbole ordonnancé localement à l'indice i une information extrinsèque $L_i^{ex}|_0$ au décodeur composant 1, qui intègre cette information dans son information *a priori* sur le symbole $L^{a'}_{\Pi(i)}\Big|_1$ ordonnancé ici à l'indice $\Pi(i)$. Réciproquement, le décodeur composant 0 intègre l'information extrinsèque du décodeur composant 1. D'où :

$$\begin{cases} L^{a'}_{\Pi(i)}\Big|_1 = L^{a}_{\Pi(i)}\Big|_1 + L_i^{ex}|_0 \\ L^{a'}_{i}\Big|_0 = L_i^{a}|_0 + L^{ex}_{\Pi(i)}\Big|_1 \end{cases} \tag{2.29}$$

Le décodage itératif est sous-optimal, car le processus itératif converge vers un optimum local à proximité des optima locaux des décodeurs composants et cet optimum local n'est pas forcément l'optimum global. Pour pallier cette sous-optimalité, la plupart des algorithmes itératifs pondèrent les informations échangées de manière à limiter l'influence des décodeurs l'un sur l'autre. Dans [55], les auteurs utilisent la théorie du chaos et la théorie de la bifurcation pour analyser l'influence réciproque des processus itératifs et réduire la sous-optimalité d'un

[3] L'information échangée est optimum pour le décodeur composant qui l'émet, mais pas forcément pour celui qui la reçoit.

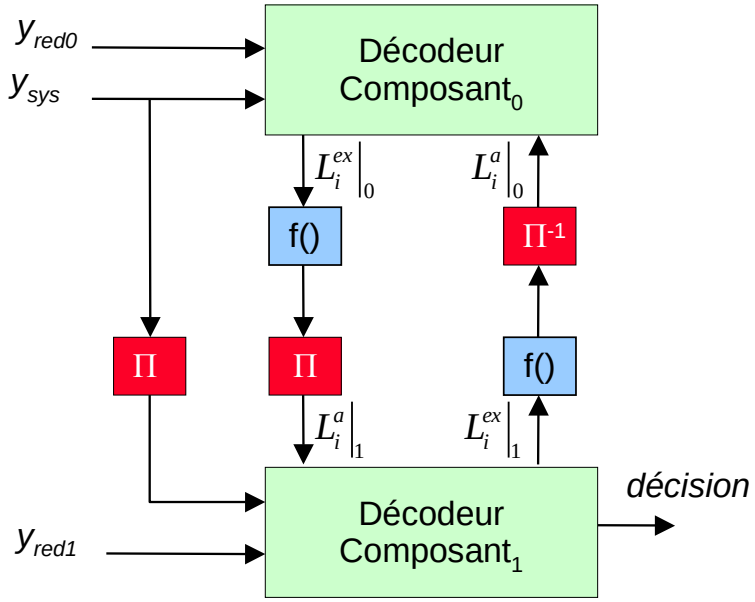

Figure 2.7 — Schéma d'un turbo-décodeur avec fonctions de correction

processus itératif. Il ressort de cette étude que corriger l'information extrinsèque avec des fonctions f correctement choisies (par exemple, dans [55], $f(x) = \alpha x e^{-\beta|x|}$ avec $\alpha \in [0,8;1]$ et $\beta \in \left[10^{-3};10^{-2}\right]$) permet une amélioration des performances de décodage. Les équations d'échanges deviennent alors :

$$\begin{cases} L^{a'}_{\Pi(i)}\Big|_1 = L^{a}_{\Pi(i)}\Big|_1 + f\left(L_i^{ex}|_0\right) \\ L_i^{a'}\Big|_0 = L_i^a|_0 + f\left(L^{ex}_{\Pi(i)}\Big|_1\right) \end{cases} \tag{2.30}$$

Dans la pratique, des corrections linéaires sont préférables car moins complexe à mettre en oeuvre. Selon [86] et [87], les meilleurs facteurs de correction sont de 0,75 pour l'algorithme max-log-MAP et 0,875 pour l'algorithme log-MAP. Il est également possible de faire varier ces facteurs au cours des itérations [88].

2.3.3.2 Performances des turbocodes

La figure 2.8 illustre les performances du turbocode double binaire huit états utilisé dans les normes pour un rendement de codage $\frac{2}{3}$ et une modulation BPSK avec les algorithmes de décodage log-MAP, max-log-MAP corrigé d'un facteur 0,75 et log-MAP corrigé d'un facteur 0,875. La courbe intègre également la limite de Shannon associée à cette modulation et compensée pour la taille de donnée de 1504 bits, taille correspondant aux paquets MPEG utilisés pour ce jeu de courbes. En outre, la borne de l'union du code (équation 2.1) présente les performances asymptotiques qui sont associées à ce code.

Comme on a pu le dire précédemment, la limite de Shannon constitue un seuil en dessous duquel il est impossible de garantir une communication sans erreur. Plus les courbes de décodage sont proches de cette limite, plus le couple codeur-décodeur sera considéré comme performant. Sur cette figure, on peut donc dire que l'algorithme log-MAP corrigé d'un facteur 0,875 est le plus performant surclassant l'algorithme max-log-MAP corrigé d'un facteur 0,75

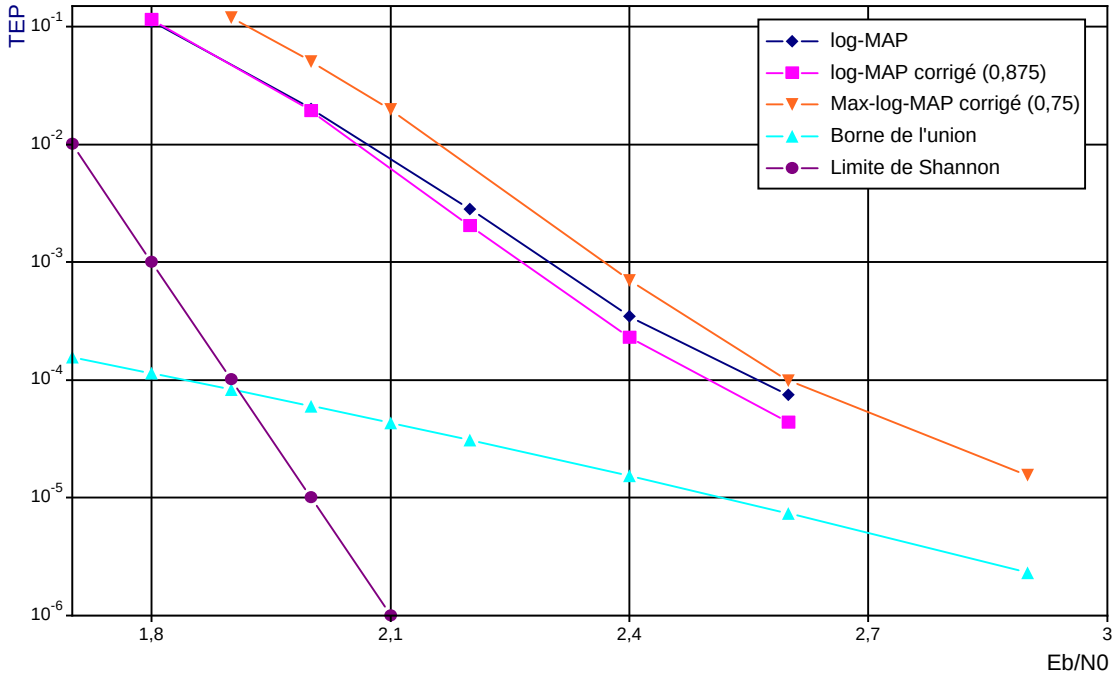

Figure 2.8 — Performances des algorithmes de turbo-décodage pour une trame de 188 octects, R=2/3, 8 itérations, BPSK

d'environ 0,1 dB. Selon la nature des communications, cette différence n'aura pas du tout la même valeur. Par exemple pour des communications en espace lointain, un dixième de dB vaut beaucoup plus cher qu'en téléphonie mobile où le bilan de liaison est moins critique. La nature de l'application fournit aussi des taux d'erreur cibles afin d'évaluer le code correcteur d'erreur autour de son point de fonctionnement. Par exemple, les applications utilisant des mécanismes de renvoi[4] ciblent couramment des taux d'erreur paquet autour de 10^{-2} tandis que des applications de type diffusion (télévision) nécessitent des taux beaucoup plus faible. Dans notre exemple, l'écart entre la courbe de décodage du log-MAP corrigé et la limite de Shannon est de 0,35 dB pour un taux d'erreur paquet de 10^{-2} et 0,6 dB pour un taux d'erreur de 10^{-4}. Le code est donc plus performant à faible taux d'erreur qu'à très faible taux d'erreur.

2.4 Conclusion

Dans ce chapitre, le domaine d'application visé par la plateforme multiprocesseur a été présenté avec un intérêt tout particulier pour la turbo-réception. Ainsi le lecteur peut trouver des repères pour situer les différentes étapes sur la chaîne de communication numérique et comprendre l'engouement que suscitent les applications de turbo-réception pour un architecte système. En reposant sur un processus itératif, ces applications nécessitent à la fois une

[4] ARQ : Automatic Request Query

complexité calculatoire imposante au sein des modules mais également un grand volume de communication entre les tâches.

Parmi les applications de turbo-réception, l'accent a été mis au niveau du décodage de canal sur le cas passionnant du turbo-décodage convolutif, qui sera traité tout au long de ce manuscrit. Ce chapitre a introduit le turbo-décodage, de manière à fournir une visibilité globale sur cette application. Ainsi, en balayant les fondamentaux sur les codes convolutifs et leur décodage au sein des décodeurs SISO, on a pu ouvrir la voie à une explication plus fine du décodage itératif à partir d'une concaténation de codes. Les notions introduites dans ce chapitre seront mises à profit tant dans le prochain chapitre qui traite des moyens de paralléliser le turbo-décodage, que dans le chapitre 4 qui évalue la flexibilité nécessaire pour aboutir à une architecture de processeur dédié au turbocode.

CHAPITRE

3 Parallélisme et turbocodes convolutifs

DANS la littérature, beaucoup de parallélismes ont été mis en avant pour le décodage des codes convolutifs. Ils ont été analysés avec leurs problèmes respectifs mais peu comparé les uns aux autres. Or, sans éléments de comparaison, l'effet des interactions entre les parallélismes au sein d'un décodeur de code convolutif est difficilement évaluable sur l'architecture globale du décodeur et les décisions même de sélection de parallélisme peuvent paraître un peu approximatives. Ce chapitre se fixe pour objectif d'éclaircir les relations entre les parallélismes et d'estimer leur efficacité respective au vu de la métrique débit-complexité.

Dans la première section de ce chapitre, nous définirons l'ensemble des métriques utilisées. Dans un deuxième temps, nous allons recenser les différents parallélismes existants dans l'état de l'art, les classer à différents niveaux puis les évaluer au sens du critère d'efficacité préalablement défini. Le second niveau de cette classification, qui présente un bon compromis entre potentiel d'accélération et coût en surface, fait l'objet d'une étude approfondie dans la section 3 avec le parallélisme de sous-bloc ainsi que dans la section 4 avec le parallélisme de décodeur composant.

3.1 Parallélisme et définitions

Le parallélisme se définit comme l'exécution simultanée de plusieurs traitements dans le but d'améliorer les performances d'un système. Néanmoins un parallélisme ne peut être mis en oeuvre que dans la mesure où les traitements exécutés en parallèle sont indépendants. Ainsi l'exploitation du parallélisme dans un système est naturellement limitée du fait des dépendances dudit système. Dans la pratique, les algorithmes sont toujours constitués d'une partie séquentielle et incompressible (comme la synchronisation des traitements, par exemple) et d'une partie parallélisable. Il est donc impossible de paralléliser intégralement un algorithme et l'accélération maximale A que l'on peut obtenir par le parallélisme suit la loi découverte par Amdahl [89].

$$A = 1/(r_s + r_p/d) \tag{3.1}$$

avec r_s le ratio de la partie séquentiel, r_p le ratio de la partie parallélisable et d le degré de parallélisme.

L'étude de l'accélération n'est pas suffisante pour implanter un algorithme sur une puce. Outre la rapidité d'exécution, la conception matérielle peut être évaluée par sa fonctionnalité ou sa précision par rapport au modèle idéal, par sa flexibilité, par son adaptivité, par sa latence, par sa consommation énergétique et dans la plupart des cas par la complexité de l'architecture, i.e. l'aire sur la puce

Dans ce chapitre, nous évaluerons l'efficacité d'une architecture de turbo-décodeur convolutif au vu des critères de complexité et de rapidité d'exécution en assurant des fonctionnalités équivalentes en termes de taux d'erreur. L'efficacité d'une architecture sera donc définie au sens de la métrique débit-complexité :

$$M_e = \frac{1}{t.C} \tag{3.2}$$

où t représente le temps de décodage de l'architecture et C sa complexité surfacique. Nous parlerons également de l'efficacité d'un parallélisme comme étant la métrique d'efficacité du système avec un degré de parallélisme d pour le parallélisme étudié relativement à la métrique d'efficacité du même système n'utilisant pas ce parallélisme :

$$E(d) = \frac{M_e(d)}{M_e(1)} = \frac{t_1.C_1}{t_d.C_d} = \frac{A}{S_C} \tag{3.3}$$

Cette métrique peut également être définie comme étant le rapport de l'accélération du décodage A sur le surcoût matériel du parallélisme S_C :

$$A = \frac{t_1}{t_d} \tag{3.4}$$

$$S_C = \frac{C_d}{C_1} \tag{3.5}$$

En considérant qu'une partie de la complexité surfacique est duplicable et l'autre non (i.e. $C_1 = C^{dup} + C^{non}$), le surcoût matériel peut se réécrire :

$$S_C = \frac{d.C^{dup} + C^{non}}{C^{dup} + C^{non}} = 1 + \%C^{dup}(d-1) \tag{3.6}$$

avec $\%C^{dup}$ le pourcentage de la partie duplicable. Dans les cas où $\%C^{dup}$ est inconnu, on peut majorer le surcoût matériel par d et redéfinir l'efffficacité d'un parallélisme par sa minoration :

$$E'(d) = \frac{A}{d} \tag{3.7}$$

L'efficacité d'un parallélisme est alors réduite à son accélération normalisée.

3.2 Classification en niveaux de parallélisme

Le processus de décodage d'un turbocode avec l'algorithme BCJR fait intervenir du parallélisme à trois différents niveaux de granularité. A un niveau de granularité fin, on peut extraire du parallélisme sur les métriques utilisées au sein de l'algorithme BCJR. Un niveau de granularité intermédiaire permet d'extraire du parallélisme par une exécution simultanée de plusieurs décodeurs BCJR SISO travaillant sur une même trame. Par ailleurs, il est également possible de paralléliser des turbo-décodeurs complets pour les faire travailler sur des trames différentes.

3.2.1 Parallélisme des métriques BCJR

Le niveau de parallélisme des métriques BCJR concerne les calculs de toutes les métriques impliquées dans le décodage de chaque symbole reçu à l'intérieur d'un décodeur BCJR SISO. Ce niveau de parallélisme exploite à la fois le parallélisme inhérent à la structure du treillis [90][74], mais également le parallélisme des calculs du BCJR [90][91][74].

3.2.1.1 Parallélisme de transition du treillis

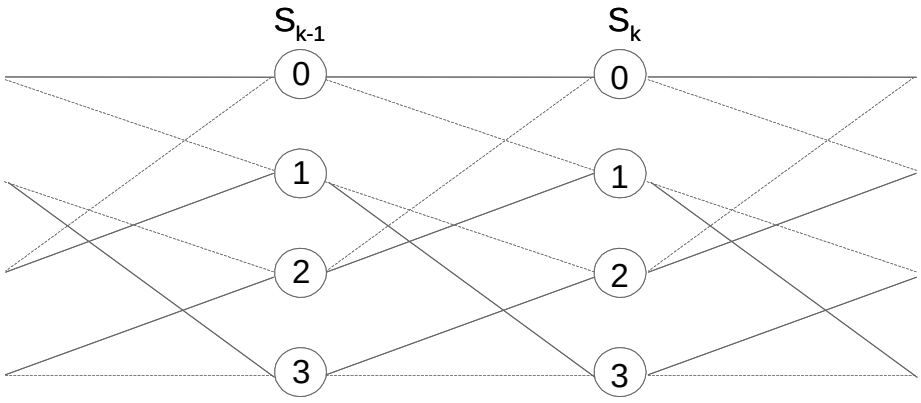

Figure 3.1 — Exemple de treillis

Dans une section de treillis, les calculs de l'algorithme BCJR répètent des opérations identiques et indépendantes entre elles pour toutes les transitions. On peut ainsi extraire de

cette répétition d'opérations un parallélisme dit des transitions de treillis ayant un degré de parallélisme pouvant aller jusqu'au nombre de transitions.

Avant d'entrer dans les détails afférents à chacun des calculs, il convient de préciser que ces opérations dépendent du semi-anneau retenu pour appliquer le BCJR, en notant le semi-anneau (ensemble, opération 1, opération 2, élément neutre de l'opération 1, élément neutre de l'opération 2). Ainsi l'algorithme BCJR original, aussi appellé MAP, utilise le semi-anneau somme-produit $(R^+, +, \times, 0, 1)$. Du fait de la complexité matérielle des multiplications, l'algorithme a été transposé dans le domaine logarithmique donnant naissance à l'algorithme log-MAP basé sur le semi-anneau $(R, \max^*, +, -\infty, 0)$ et à sa version simplifiée l'algorithme max-log-MAP basé sur le semi-anneau max-somme $(R, \max, +, -\infty, 0)$ [69]. Dans la littérature, les opérateurs dérivent directement de ces deux derniers semi-anneaux. On utilise soit des opérateurs ACS (Addition Comparaison Sélection) pour l'algorithme max-log-MAP où la comparaison-sélection réalise la fonction maximum, soit des opérateurs ACSO (ACS avec un offset) pour l'algorithme log-MAP et dans ce cas, l'offset correspond au logarithme jacobien (voir chapitre 2).

Observons maintenant les opérations nécessaires à l'exécution du BCJR sur une section de treillis. En premier lieu, il faut calculer les métriques de branches γ associées aux transitions. Ce calcul est complètement parallélisable avec un degré de parallélisme naturellement borné au nombre de transitions dans le treillis. Mais dans la pratique, le degré de parallélisme du calcul des métriques de branches est borné par le nombre de codage possible pour une transition (i.e. pour le couple (d_i, c_i) défini dans la section 2.3.1). Ainsi, plusieurs transitions peuvent avoir la même vraisemblance au sein d'un treillis.

Concernant les autres calculs de l'algorithme BCJR, on peut observer des regroupements entre certaines transitions. D'une part, les calculs de récursions aller (équation 2.21) et retour (2.22), aussi appelés métriques d'état, regroupent les transitions selon leurs états d'arrivée et d'autre part les calculs d'information *a posteriori* (2.23) et extrinsèques (2.24) font des regroupements sur les transitions ayant les mêmes décisions dures. Dans les deux cas, l'opération 2 du semi-anneau est d'abord appliquée à chaque transition pour composer soit une métrique d'état avec une métrique de branche (e.g. $\alpha + \gamma$) soit deux métriques d'état et une métrique de branche (e.g. $\alpha + \beta + \gamma$). Ensuite, les résultats obtenus pour les transitions à l'intérieur d'un même regroupement sont composés avec l'opération 1 par l'intermédiaire d'un arbre de calcul (par exemple la figure 3.2).

Il existe donc un motif commun entre ces différents calculs qui combine en arbre un étage de l'opérateur 2 avec un nombre d'étages $e_{\min}$ de l'opérateur 1 défini par le logarithme en base deux du minimum entre le nombre d'opérations nécessaires pour aboutir à une métrique d'information (normalement égal au nombre d'états du treillis) et le nombre d'opérations nécessaires pour aboutir à une métrique de récursion (égal au nombre de décisions possibles pour une section de treillis). Pour le semi-anneau max-somme, ce motif s'écrit $A\,(CS)^{e_{\min}}$. Dans le cas des codes simple binaire, le motif se simplifie pour donner le fameux opérateur ACS.

De la même manière, on peut définir un motif maximum recouvrant l'ensemble des calculs du BCJR où cette fois $e_{\max}$ désigne le logarithme en base deux du maximum entre le nombre d'états du treillis et le nombre de décisions possibles pour une section de treillis. Les deux opérations 2 intervenant dans le calcul des métriques d'information conduisent à un motif du type $AA\,(CS)^{e_{\max}}$. Ce motif peut être simplifié en $A\,(CS)^{e_{\max}}$ dans la mesure où les calculs d'information peuvent être anticipés dans les calculs de récursions. Par exemple, lors de la récursion aller on peut stocker les résultats intermédiaires (i.e. pour chaque transition $\alpha + \gamma$)

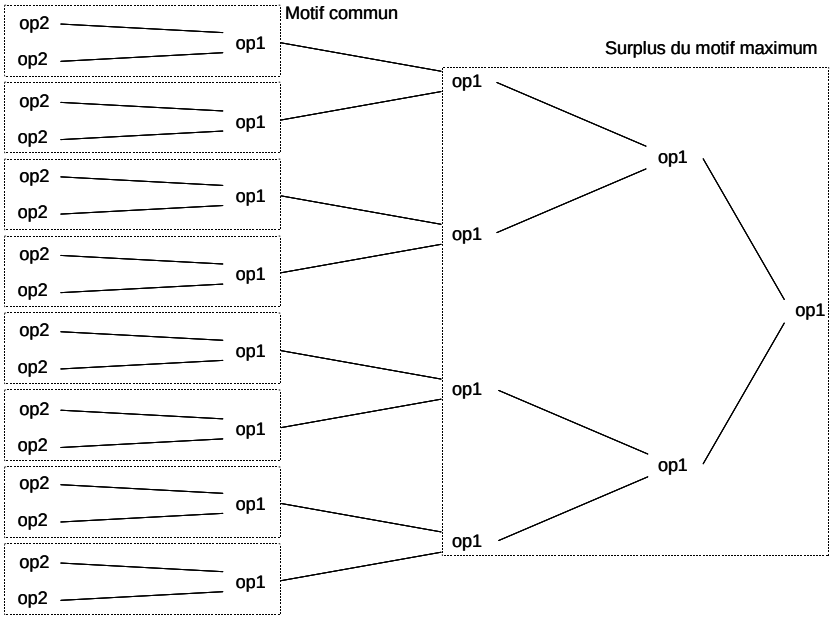

Figure 3.2 — Motif de calcul d'une métrique d'information d'un code convolutif 16 états simple binaire

et les réutiliser juste après dans le calcul de l'information *a posteriori* (i.e. $(\alpha + \gamma) + \beta$).

Il convient de noter que le motif maximal simplifié est du même ordre de complexité que le motif commun si on suppose une complexité égale pour les opérations 1 et 2. En effet, un motif avec e étages pour l'opération 1 nécessite 2^e opérations 2 et $2^e - 1$ opérations 1 pour traiter 2^e transitions, soit un nombre d'opérations par transition s'établissant à $2 - 2^{-e}$. Ainsi la différence relative de complexité entre le motif commun et le motif maximum est de :

$$\frac{2^{-e_{\min}} - 2^{-e_{\max}}}{2 - 2^{-e_{\min}}} \tag{3.8}$$

Pour les codes existant dans les standards, cela représente une différence variant de 7% pour un code 8 états double binaire à 29% pour un code 16 états simple binaire. Le surplus de complexité des motifs maximum peut être traité soit à part dans un arbre de calcul prenant en entrée les sorties de $2^{e_{\max} - e_{\min}}$ motifs communs, soit en réutilisant les opérateurs 1 des motifs communs.

La faible différence de complexité entre motif commun et motif maximal nous permet de considérer le motif commun comme la brique de base des calculs de l'algorithme BCJR et donc comme unité de mesure pour le parallélisme de transition de treillis. Ainsi le degré de parallélisme de transition du treillis $d_{treillis}$ est égal au maximum au nombre de transitions dans une section divisée par $2^{e_{\min}}$. En considérant la complexité du motif commun C_{MC}, la complexité d'un calcul BCJR peut être approximée par :

$$C_{calcul} = d_{treillis}.C_{MC} \tag{3.9}$$

On peut noter que ce parallélisme est peu coûteux en ressources matérielles, car il ne duplique que les motifs communs. En particulier, les mémoires, très coûteuses en surface, sont épargnées, puisque toutes les opérations sont exécutées sur une même section de treillis et donc

avec les mêmes données. Comme le nombre de transition dans une section de treillis limite naturellement le potentiel de parallélisme, il a été proposé dans [92] de compacter plusieurs sections successives du treillis afin d'augmenter le parallélisme exploitable de transition du treillis.

3.2.1.2 Parallélisme des calculs du BCJR

Orthogonalement au parallélisme de transition de treillis basé sur les opérations, on peut extraire du parallélisme en réalisant en parallèle les trois calculs de l'algorithme BCJR (i.e. les récursions aller et retour, plus le calcul de l'information extrinsèque) [93][74]. Dans cette optique, il convient d'être prudent puisqu'il existe des dépendances de données entre ces calculs. D'une part, les calculs des récursions (aller et retour) sont récursifs et doivent de ce fait être calculés de manière séquentielle sur le bloc à traiter. D'autre part, les calculs d'information extrinsèque ne peuvent être réalisés sans la connaissance des deux métriques de récursion, ce qui nécessite la mémorisation d'au moins une métrique de récursion. Toutes ces contraintes imposent le déroulement de l'algorithme selon un schéma (figure 3.3). Sur ces schémas, la zone grisée représente l'utilisation des mémoires contenant les métriques d'état au fil du décodage BCJR. La complexité associée à ces mémoires, notée M_{BCJR}, dépend linéairement de la profondeur des mémoires, qui est dictée par le pic temporel d'utilisation de la mémoire. Il est possible de réduire l'influence de M_{BCJR} sur la complexité en utilisant la méthode des fenêtres glissantes [94][95][96]. Bien que complémentaire, cette méthode ne sera pas considérée par la suite puisqu'elle n'est pas liée au parallélisme.

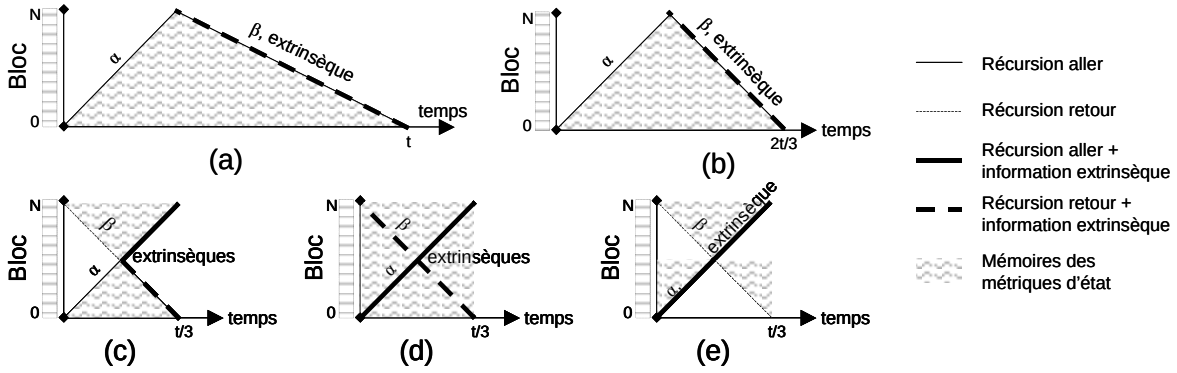

Figure 3.3 — Schémas de calcul BCJR : (a) schéma aller-retour sans parallélisme, (b) schéma aller-retour original, (c) schéma papillon, (d) schéma papillon replica, (e) schéma papillon aller

Le schéma original de décodage est appelé aller-retour (figure 3.3.b) [68]. On y observe une exécution parallèle du calcul de la récursion retour et de l'information extrinsèque, soit un degré de parallélisme des calculs BCJR d_{calcul} de un dans la partie aller et de deux dans la partie retour. Ce schéma permet une accélération moyenne de $1,5$ par rapport à une exécution séquentielle des calculs du BCJR (figure 3.3.a). Par ailleurs, la profondeur mémoire est égale à celle du bloc à traiter. D'autres schémas ont été proposés pour améliorer l'accélération. Parmi eux, le schéma papillon (figure 3.3.c) [97] double le degré de parallélisme du schéma original en parallélisant les récursions aller et retour (degré 2 dans la première moitié du papillon et degré 4 dans la seconde moitié) pour atteindre une accélération moyenne de 3. Des variantes du schéma papillon existent pour équilibrer le degré de parallélisme sur l'ensemble du calcul. Par exemple, le schéma papillon replica (figure 3.3.d) étend les calculs

d'informations extrinsèques à la première partie du papillon. Cette extension fut initialement proposée dans [98] pour améliorer la convergence du décodage combiné (cf. 3.2.2.2). Elle a pour conséquence de nécessiter quatre unités de calcul BCJR. Du fait des dépendances entre information extrinsèque et métriques d'état, le schéma papillon replica impose la conservation des métriques d'état entre deux itérations. Ainsi l'utilisation de la mémoire est plus conséquente mais également plus uniforme. Ceci permet de conserver la complexité mémoire M_{BCJR} au même niveau que pour les schémas papillons ou aller-retour. Autre variante du schéma papillon, le schéma papillon aller (figure 3.3.e) ne calcule l'information extrinsèque que dans le sens aller. En conséquence, son degré de parallélisme des calculs BCJR est de 3 sur l'ensemble du papillon. On peut également noter qu'il nécessite deux fois moins de mémoire que les autres schémas pour stocker les métriques d'état.

$$M_{BCJR\ papillon\ aller} = \frac{M_{BCJR\ aller\ retour}}{2} \tag{3.10}$$

Comme pour le parallélisme de transition de treillis, le parallélisme des calculs du BCJR duplique uniquement des unités de calculs BCJR. La complexité d'un décodeur BCJR s'écrit :

$$C_{BCJR} = \max(d_{calcul}).C_{calcul} + M_{BCJR} \tag{3.11}$$

L'impact de ce parallélisme sur la complexité mémoire est nul pour la plupart des schémas, mais bénéfique pour le schéma papillon aller. Au maximum de parallélisme de calcul, ce dernier schéma présente une complexité minimale de :

$$C_{BCJR\ papillon\ aller} = 3.C_{calcul} + \frac{M_{BCJR\ aller\ retour}}{2} \tag{3.12}$$

contre, pour les autres schémas papillons :

$$C_{BCJR\ papillon} = 4.C_{calcul} + M_{BCJR\ aller\ retour} \tag{3.13}$$

d'où

$$\frac{1}{2} < \frac{C_{BCJR\ papillon\ aller}}{C_{BCJR\ papillon}} < \frac{3}{4} \tag{3.14}$$

Sa complexité est donc au moins de 25% plus faible que celle des autres, mais cet avantage se voit contrebalancer par une convergence un peu plus longue (voir section 3.4.3). Plus généralement, on peut évaluer l'efficacité du parallélisme de calcul BCJR (au sens de l'équation 3.3 appliqué sur un décodeur BCJR seul) par :

$$E_{BCJR} = A.\frac{C_{BCJR\ sans\ parallélisme}}{C_{BCJR}} = moyenne\,(d_{calcul})\,.\frac{C_{calcul} + M_{BCJR\ aller\ retour}}{\max(d_{calcul}).C_{calcul} + M_{BCJR}} \tag{3.15}$$

Malgré un facteur d'accélération A limité à trois pour l'ensemble de ces schémas, on peut dire que le parallélisme des calculs BCJR est très efficace en surface, puisque comme le parallélisme de transition du treillis il ne duplique que des unités de calcul et pas de mémoire.

Pour conclure, le niveau de parallélisme des métriques BCJR est très économique en surface, mais il souffre d'une limitation inhérente à l'algorithme BCJR. Ainsi un concepteur désireux d'augmenter le parallélisme devra chercher à un niveau de granularité supérieur.

3.2.2 Parallélisme de décodeur SISO BCJR

Le parallélisme de décodeur BCJR-SISO consiste à utiliser plusieurs décodeurs SISO exécutant chacun l'algorithme BCJR sur un sous-bloc de la trame décodée dans un des ordres d'entrelacement. Le parallélisme peut donc être appliqué soit sur les sous-blocs, soit sur les décodeurs composants.

3.2.2.1 Parallélisme de sous-blocs

L'idée sous-jacente au parallélisme de sous-bloc consiste à diviser chaque trame en M sous-blocs et d'associer chaque sous-bloc à un décodeur SISO. Cependant les métriques de récursion aller et retour constituent des dépendances de données au sein d'un bloc. Chaque décodeur SISO doit donc être initialisé de manière adéquate afin de pouvoir casser cette dépendance de données [99][96]. Ceci peut être réalisé de deux manières. La première méthode consiste à estimer l'initialisation du sous-bloc grâce à un précalcul sur le bloc voisin. C'est l'initialisation par acquisition. La deuxième méthode, nommée initialisation par passage de message ou initialisation à la prochaine itération (NII en anglais pour Next Iteration Initialization), profite du processus itératif du décodage pour réutiliser les métriques d'état à l'issue d'une itération comme initialisations dans l'itération suivante. Les deux méthodes existantes seront analysées par la suite.

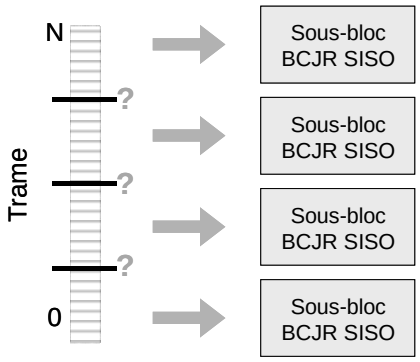

Figure 3.4 — Parallélisme de sous-bloc

Outre la duplication des décodeurs SISO et la gestion des initialisations, ce parallélisme pose également un problème de communication au niveau de l'entrelacement. En effet, le parallélisme de sous-bloc implique l'introduction de parallélismes dans l'entrelacement pour offrir une bande passante permettant l'échange des informations extrinsèques. Ainsi on peut voir survenir des conflits lors des accès aux mémoires contenant les informations extrinsèques. Ces conflits peuvent être résolus lors de la conception de l'architecture grâce en modifiant les accès mémoires pour chaque entrelaceur de manière à éviter les collisions [100], ou au moyen d'une architecture de communication gérant les conflits [101][102]. Cette dernière approche, offrant plus de flexibilité, sera traitée dans le chapitre 5 par la description de structures de communication adaptées.

3.2.2.2 Parallélisme de décodeur composant

L'idée de faire travailler en parallèle les décodeurs composants n'est pas nouvelle, car elle permettrait de diviser par deux la durée d'une itération (figure 3.5.b). Cependant, dans la

pratique, cette technique de décodage s'avère très inefficace. En regardant de plus près son mode de fonctionnement, on s'aperçoit que deux décodages sont réalisés en parallèle (D1-D2-D1... et D2-D1-D2...) et qu'ils ne travaillent ensemble que pour produire les décisions dures, soit après le processus itératif. Ce manque de collaboration entre les décodeurs composants impose à cette technique près de deux fois plus d'itérations que le décodage itératif série (figure 3.5.a) pour converger.

En introduisant des échanges d'informations extrinsèques aussitôt après leur création pour offrir aux décodeurs composants des informations *a priori* plus fiables, la technique du décodage combiné (figure 3.5.c), introduite sous le nom de shuffled decoding [56], permet d'accélérer le processus itératif du décodage parallèle et redonne ainsi un vif intérêt au parallélisme de décodeur composant.

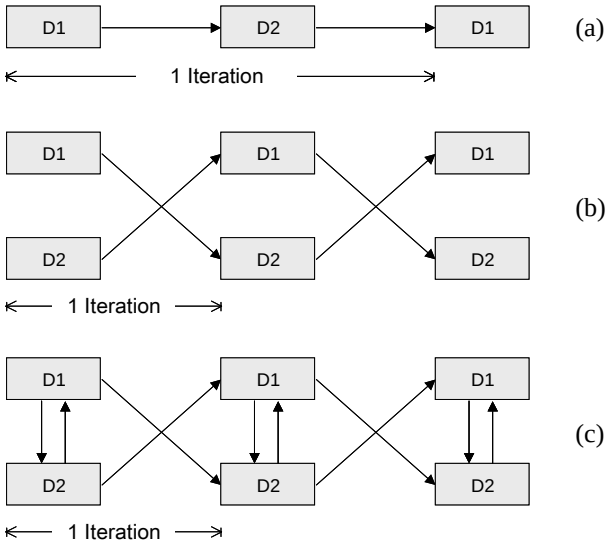

Figure 3.5 — Décodage (a) série, (b) parallèle, (c) combiné

Ainsi on a observé que conserver des performances en taux erreur équivalentes au décodage série nécessite entre 5 et 50 % d'itérations supplémentaires avec le décodage combiné selon les conditions d'utilisation. L'intérêt de cette méthode est donc très relatif aux conditions d'utilisation et nous verrons par la suite dans quelles conditions on peut l'exploiter au mieux.

Par définition, le décodage combiné dispose d'une propriété très intéressante : les tâches de calcul (décodage) et de communication (entrelacement) sont réalisées simultanément. Cela simplifie fortement la réalisation de l'entrelacement parallèle qui est un des goulets d'étranglement des implantations de turbo-décodeur haut-débit.

Pour conclure ce paragraphe, on ne peut que constater que le parallélisme de décodeur BCJR-SISO permet d'atteindre un degré de parallélisme conséquent (bien que limité par la taille de la trame et le nombre de décodeur composant). Par ailleurs, il est relativement efficace d'un point de vue surface, car il ne duplique en plus des parties calculs qu'une partie des mémoires du turbo-décodeur. La complexité d'un turbo-décodeur s'écrit :

$$C_{turbodécodeur} = M_{extrinsèque} + M_{entrée\ canal} + d_{SB}.d_{DC}.C_{BCJR} \tag{3.16}$$

en tenant compte des mémoires contenant l'information extrinsèque $M_{extrinsèque}$ et l'information provenant du canal de transmission $M_{entrée\ canal}$, et des degrées de parallélisme de

sous-bloc d_{SB} et de décodeur composant d_{DC}.

3.2.3 Parallélisme de turbo-décodeur

On peut également dupliquer l'ensemble du turbo-décodeur afin de traiter en parallèle les itérations et/ou les trames. Le parallélisme d'itérations s'opère de manière pipelinée sur une profondeur maximum égale à deux fois le nombre d'itérations [103]. Le parallélisme de trame ne présente aucune limitation de parallélisme.

Bien que très facile à mettre en oeuvre, le parallélisme de turbo-décodeur a deux inconvénients majeurs. D'une part, son coût en surface est élevé puisque ce niveau de parallélisme duplique toutes les ressources aussi bien de calcul que de mémorisation. La complexité du module de turbodécodage s'exprime alors linéairement en fonction des degrés de parallélisme de trame d_{trame} et d'itérations $d_{itération}$:

$$C = d_{trame}.d_{itération}.C_{turbodécodeur} \tag{3.17}$$

Comme l'accélération apportée par ces parallélismes est également linéaire en fonction des degrés de parallélisme, ce niveau de parallélisme est sans influence sur l'efficacité globale de l'architecture.

D'autre part, en travaillant sur des trames différentes, ce niveau de parallélisme ne peut offrir aucun gain sur la latence de décodage d'une trame, qui est un paramètre important dans certains systèmes de communications.

Niveau	Parallélisme
Métriques BCJR	Transition du treillis
	Calculs du BCJR
Décodeur BCJR-SISO	Sous-blocs
	Décodeur composant
turbo-décodeur	Itérations
	Trames

Tableau 3.1 — Niveaux de parallélisme d'un turbo décodeur

Le tableau 3.1 récapitule l'ensemble des différents niveaux de parallélisme existants en classant les parallélismes du moins coûteux en surface (duplication des ressources de calcul uniquement) au plus coûteux en surface (duplication de toutes les ressources). A partir des équations 3.9, 3.11, 3.16 et 3.17, on peut estimer la complexité surfacique globale à :

$$C = d_{trame}.d_{itération}\left(M_{extrinsèque} + M_{entrée\ canal} + d_{SB}.d_{DC}\left(d_{calcul}.d_{treillis}.C_{MC} + M_{BCJR}\right)\right) \tag{3.18}$$

On peut également observer avec ce classement que la limitation sur le degré de parallélisme s'assouplit au fil des niveaux. En offrant un bon compromis entre le degré de parallélisme atteignable et le coût en surface, le niveau de parallélisme de décodeur BCJR-SISO est une étape incontournable pour réaliser des turbo-décodeurs haut débit. Ce niveau sera détaillé dans les sections suivantes.

3.3 Parallélisme de sous-blocs

Comme décrit dans la section 3.1, le parallélisme de sous-bloc travaille sur une trame de N symboles décomposée en d_{SB} sous-blocs et nécessite l'initialisation des métriques de récursion, puisque seules les extrémités de trame disposent d'information sur les métriques de récursions et non les extrémités de sous-blocs. Il existe deux méthodes pour faire une estimation des informations indéterminées : l'acquisition ou le passage de message entre des sous-blocs voisins. Le choix de la méthode d'initialisation influe principalement sur le temps de décodage et donc sur l'efficacité du parallélisme. L'utilisation de ces métriques nous permettra de comparer ces méthodes dans cette section.

3.3.1 Initialisation par acquisition

La technique d'initialisation par acquisition permet d'estimer les métriques de récursion grâce à une fenêtre d'acquisition (ou prologue) chevauchant le sous-bloc voisin. Ainsi on va commencer le calcul de la récursion à la distance AL en amont du sous-bloc avec une initialisation équiprobable des états dans cette section de treillis. Cette distance AL ou longueur de la fenêtre d'acquisition est déterminée à la conception de façon à obtenir une métrique de récursion fiable au début d'un sous-bloc et donc de minimiser les dégradations du décodeur en termes de taux erreur. Empiriquement elle est fixée entre 3 et 5 fois la longueur de contrainte du code [99] ou de manière à fixer un certain nombre de redondances dans le prologue, typiquement 6 [104].

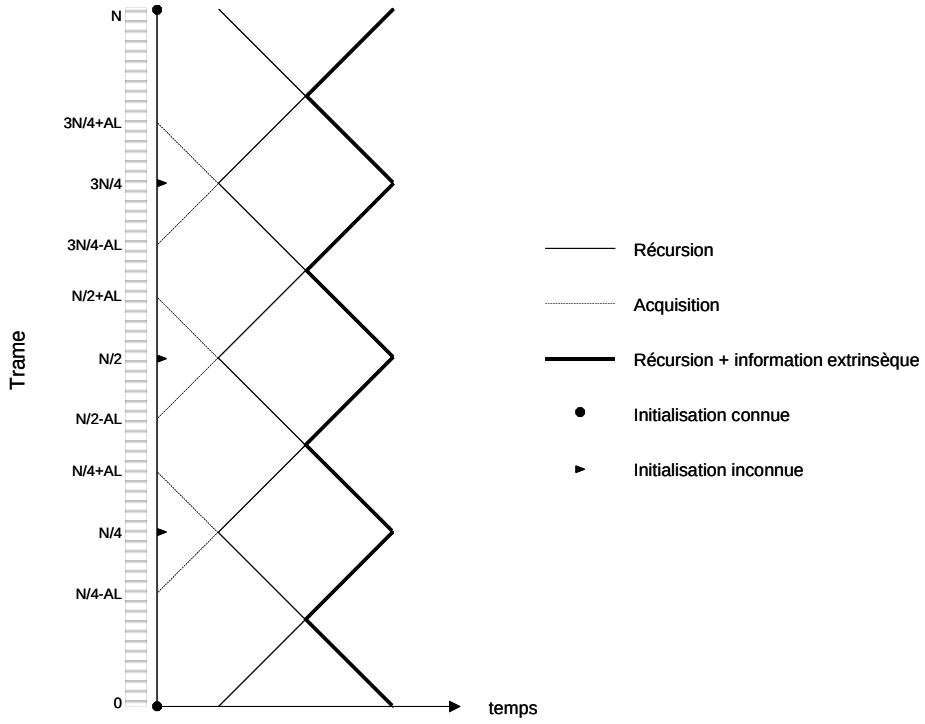

Figure 3.6 — Parallélisme de sous-bloc avec initialisation par acquisition

Pour évaluer cette méthode d'initialisation, nous allons supposer que chaque sous-bloc est traité avec le schéma papillon et que l'on dispose d'informations sur les états de début et fin de trame (soit par treillis circulaire soit par tailbitting). Ainsi, quand tous les sous-blocs

sont initialisés avec cette méthode, le temps de décodage t_d, l'accélération A et l'efficacité du parallélisme $E(d)$ peuvent être exprimés :

$$t_d \propto (\frac{N}{d_{SB}} + AL).it \qquad \text{pour } d_{SB} > 1 \qquad \text{et} \qquad t_1 \propto N.it \tag{3.19}$$

$$A = \frac{t_1}{t_d} = \frac{d_{SB}}{\left(1 + \frac{AL \cdot d_{SB}}{N}\right)} \tag{3.20}$$

$$E(d) = \frac{A}{d_{SB}} = \frac{1}{\left(1 + \frac{AL \cdot d_{SB}}{N}\right)} \tag{3.21}$$

avec d_{SB} le degré de parallélisme de sous-bloc et it le nombre d'itérations.

L'équation 3.19 montre que le temps de décodage tend vers une constante quand le degré de parallélisme croît. D'une certaine manière, on peut considérer qu'il suit une loi d'Amdahl où les périodes d'acquisition seraient des périodes séquentielles et incompressibles nécessaires pour fiabiliser le calcul. En conséquence, le parallélisme de sous-blocs avec l'initialisation par acquisition est plafonné en débit par une accélération maximum de $\frac{N}{AL+1}$. L'efficacité du parallélisme (selon l'équation 3.7) tendrait dans ce cas vers $\frac{1}{AL+1}$. En revanche, le surplus de calcul, qui affecte le calcul des récursions et les accès aux mémoires de données entrantes, s'accroît de manière linéaire avec le degré de parallélisme.

3.3.2 Initialisation par passage de message

Une seconde méthode consiste à initialiser un décodeur SISO dynamiquement avec les métriques de récursions calculées par les décodeurs SISO voisins durant la dernière itération. De fait, cette technique ne nécessite presque aucun ajout matériel, si ce n'est des ressources de communications entre les unités BCJR SISO. Pour évaluer cette technique, la dégradation des taux d'erreur a été étudiée et compensée par l'ajout d'itérations. La figure 3.7 représente pour cette méthode la convergence du taux d'erreur trame (TEP) au fil des itérations pour différents degrés de parallélisme (allant de 1 à 100 décodeurs SISO). On peut y observer d'une part que la convergence du processus itératif est ralentie avec l'augmentation du degré de parallélisme de sous-bloc et d'autre part que le taux d'erreur asymptotique, i.e. atteint pour un nombre infini d'itérations, est le même quelque soit le degré de parallélisme. En conséquence, même si le recours à des itérations supplémentaires peut s'avérer nécessaire pour maintenir les exigences de performances de correction, on est certain de pouvoir compenser de ce fait une dégradation éventuelle des performances de correction.

Le temps de décodage et l'accélération s'expriment en fonction du nombre d'itérations nécessaire (it_{PM}) avec cette technique :

$$t_d \propto \frac{N}{d_{SB}}.it_{PM} \tag{3.22}$$

$$A = \frac{t_1}{t_d} = d_{SB}.\frac{it}{it_{PM}} \tag{3.23}$$

$$E(d) = \frac{A}{d_{SB}} = \frac{it}{it_{PM}} \tag{3.24}$$

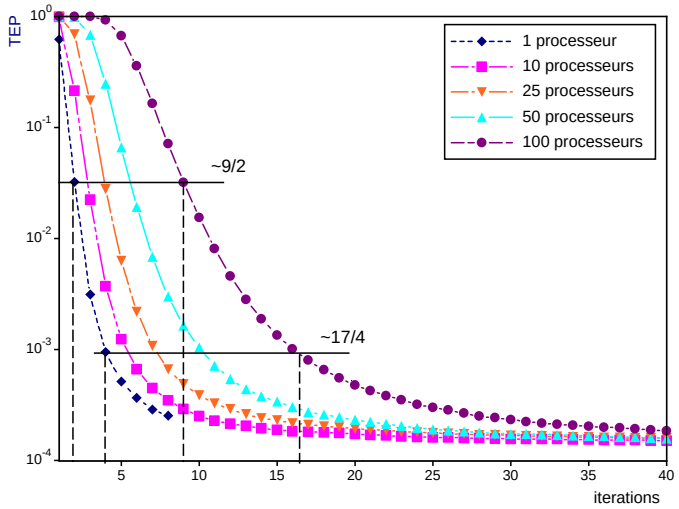

Figure 3.7 — Convergence de la technique par passage de message, DVB-RCS, R=6/7, trame de 188 octets, $\frac{E_b}{N_0}$=4.2 dB, 5 bits de quantification, algorithme Log-MAP

Une estimation de it_{PM} peut être obtenue par simulation à taux d'erreur trame fixé. Par exemple, la figure 3.8 montre l'évolution de cette estimation en fonction du degré de parallélisme. La figure représente également les accélération et efficacité induites dans ce contexte. On peut remarquer sur cette figure que le nombre d'itérations est constant pour un faible degré de parallélisme puis croît linéairement passé un seuil T (autour de 8 dans ce cas).

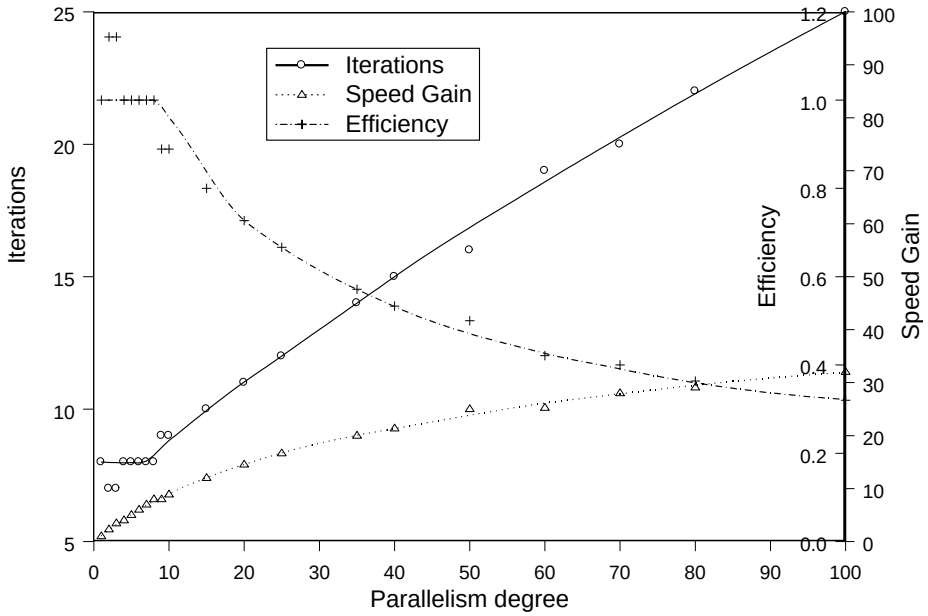

Figure 3.8 — Iterations, accélération et efficacité avec la technique par passage de message, DVB-RCS, R=6/7, trame de 188 octets, SNR=4.2 dB, 5 bits de quantification, algorithme Log-MAP, référence 8 itérations sans parallélisme

On peut donc écrire :

$$it_{PM} = it + \lfloor d_{SB}/T \rfloor \tag{3.25}$$

où it le nombre d'itérations sans parallélisme est une constante entière quand le taux d'erreur paquet et $\frac{E_b}{N_0}$ sont fixés, et où le seuil T est une constante à rendement de codage et taille de paquet fixée.

Le tableau 3.2 contient les valeurs de seuils obtenus pour un code DVB-RCS. On observe que les seuils dépendent fortement de la taille de bloc et du rendement de codage. Pour un rendement donné, on peut observer que le seuil croît linéairement avec la taille de bloc. Donc, pour chaque rendement de codage, on peut exhiber une taille de sous-bloc clé en deçà de laquelle le parallélisme de sous-bloc nécessite des itérations supplémentaires. Cette taille est par exemple de 100 bits pour un rendement 1/2. A taille de bloc constante, on s'aperçoit également que le seuil décroît avec le rendement de codage. Ceci s'explique par un rendement de codage plus faible qui permet d'obtenir plus rapidement des métriques de récursions fiables et en conséquence des informations extrinsèques fiables permettant la convergence du processus itératif. C'est pourquoi on peut interpréter la taille de sous-bloc clé comme la taille de sous-bloc minimum fournissant des métriques de récursions fiables au moment de la première transmission d'information extrinsèque. Sous cette taille de bloc, les métriques de récursion doivent être affinées par l'intermédiaire d'itérations supplémentaires.

Rendement du codage / **Taille de bloc (bits)**	**6/7**	**3/4**	**1/2**	**1/3**
424	3	3	4	5
848	4	5	8	9
1504	8	11	16	20
taille de sous-bloc clé	180	135	100	85

Tableau 3.2 — Taille de sous-bloc clé et seuils obtenus pour différent rendement de codage et différente taille de bloc avec le code DVB-RCS

Avec la caractérisation de it_{PM} (équation 3.25) et l'équation 3.24, l'efficacité du parallélisme de sous-bloc avec initialisation par passage de message peut se réécrire :

$$E(d) = \frac{1}{1 + \frac{\lfloor d_{SB}/T \rfloor}{it}} \tag{3.26}$$

On observe alors, comme pour la méthode d'initialisation par acquisition, que le parallélisme de sous-bloc avec l'initialisation par passage de message suit une loi d'Amdahl, qui le plafonne en débit. L'accélération maximum se situe autour de $it.T$ et l'efficacité de parallélisme correspondant autour de $\frac{it.T}{N}$.

La méthode de calcul présentée pour l'efficacité permet d'exposer facilement la loi d'Amdahl sur le parallélisme de sous-bloc avec cette initialisation. Or dans la pratique, elle peut s'avérer assez imprécise car elle nécessite de trouver un nombre d'itérations avec parallélisme de sous-bloc it_{PM} fournissant des performances (taux d'erreur) identiques au décodage sans parallélisme à l'itération it. La nature entière du nombre d'itérations empêche en pratique d'avoir des performances identiques. Pour pallier ces imprécisions, nous utilisons en réalité non pas le nombre maximum d'itérations nécessaire pour atteindre un taux d'erreur, mais le nombre moyen d'itérations assurant la convergence (si possible) du processus itératif. Le

nombre moyen d'itérations est obtenu selon le critère d'arrêt du génie[1]. Notez que cette méthode rend le résultat indépendant du TEP et qu'elle n'est possible justement parce que l'efficacité n'en dépend pas (la constance de l'efficacité au cours de la convergence du processus itératif est d'ailleurs visible sur la figure 3.7). En prenant un nombre maximal d'itérations suffisamment grand pour faire converger les processus itératifs vers les mêmes performances, cette méthode de calcul garantit des résultats d'efficacité beaucoup plus précis que la méthode précédemment présentée.

3.3.3 Efficacité de parallélisme et méthodes d'initialisations

Pour comparer les deux méthodes d'initialisation qui viennent d'être présentées, nous allons considérer leurs efficacités respectives qui tiennent compte de la dégradation du nombre moyen d'itérations nécessaire pour converger et également du surplus de calcul due à l'acquisition Les figures 3.9 et 3.10 comparent l'efficacité du parallélisme de sous-blocs pour différentes méthodes d'initialisation : passage de message, acquisition sur 24 et 32 symboles pour le rendement $\frac{6}{7}$ et sur 16 et 24 symboles pour le rendement $\frac{1}{2}$, et passage de message en utilisant un prologue (i.e. une acquisition juste pour la première itération) de 24 symboles pour le rendement $\frac{6}{7}$ et sur 16 symboles pour le rendement $\frac{1}{2}$.

Sans équivoque, ces figures montrent clairement que l'initialisation par passage de message est plus efficace que l'initialisation par acquisition qu'importe la taille choisie pour la période d'acquisition. De plus, la différence d'efficacité s'accroît entre ces deux méthodes lorsque l'on considère de forts degrés de parallélisme. Si grâce à ces comparaisons le choix de la méthode d'initialisation entre deux itérations devient évident, il reste encore à se poser la question de l'initialisation de la première itération qui n'est pas traité par la méthode par passage de message. L'ajout d'un prologue permet de fournir une initialisation non uniforme des métriques d'état lors de la première itération. Cela assure au décodeur un départ sur de bonnes bases et permet donc une diminution du nombre moyen d'itérations. Ainsi l'apport d'un prologue sur l'efficacité d'une initialisation par passage de message est positif lorsque la diminution du nombre moyen d'itérations est plus importante que le surcoût de complexité apporté par le prologue. Les figures 3.9 et 3.10 montrent que l'utilisation d'un prologue peut améliorer l'efficacité de parallélisme pour de faibles degrés de parallélisme (surtout pour des rendements élevés), mais à l'inverse elle s'avère pénalisante pour l'efficacité à fort degré de parallélisme.

Les résultats qui viennent d'être énnoncés sur l'efficacité de parallélisme de sous-bloc selon la méthode d'initialisation sont d'autant plus importants que la comparaison des efficacités présentées jouent en défaveur de l'initialisation par passage de message. En effet, pour être comparé les turbo-décodeurs doivent normalement présenter des performances égales si on s'en tient strictement à notre définition (voir équation 3.2). Or comme le révèle la figure 3.11 pour un rendement $\frac{6}{7}$, c'est loin d'être le cas. Par rapport à la courbe de référence sans parallélisme, la courbe d'un décodeur ayant un degré de parallélisme de sous-bloc de 47 initialisé par acquisition accuse une dégradation de 0,1 dB de performance avec une fenêtre d'acquisition de 32 symboles et la dégradation s'empire si on diminue la taille de la fenêtres d'acquisition (0,7 dB pour une fenêtre de 24 symboles). A l'inverse, en initialisant ce même système par passage de message, les performances obtenues sont légèrement meilleures (0,1 dB de mieux) que celles de la référence. D'une manière générale, nous avons constaté par

[1]Le décodage s'arrête automatiquement à la première itération fournissant le bon mot de code et ne commence pas si la trame ne peut être décodée.

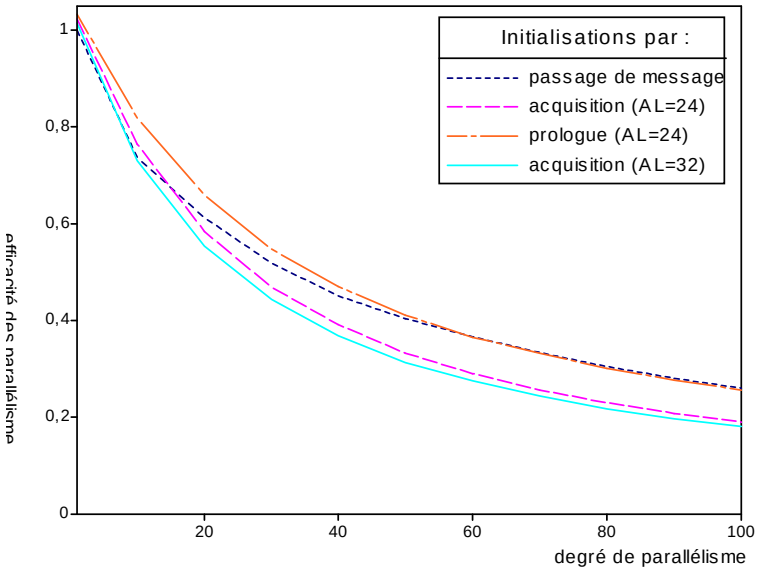

Figure 3.9 — Efficacité simulée pour plusieurs méthodes d'initialisation pour un rendement R=$\frac{6}{7}$, code DVB-RCS, trame de 188 octets, $\frac{E_b}{N_0}$=4.2 dB, 5 bits de quantification, algorithme max-log-MAP

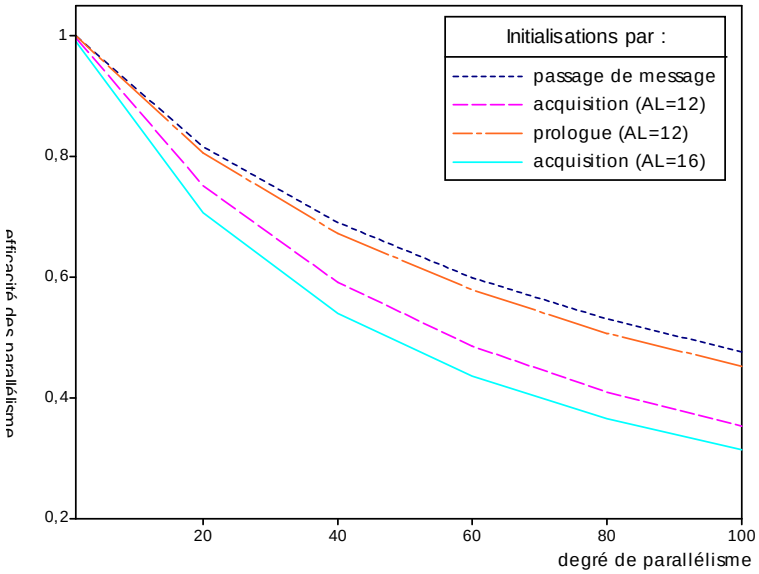

Figure 3.10 — Efficacité simulée pour plusieurs méthodes d'initialisation pour un rendement R=$\frac{1}{2}$, code DVB-RCS, trame de 188 octets, $\frac{E_b}{N_0}$=1,4 dB, 5 bits de quantification, algorithme max-log-MAP

simulation que plus les turbo-décodeurs sont parallélisés, mieux ils convergent. Ce résultat a également été observé dans le cadre d'implantation de turbo-décodeur analogique [105].

La comparaison entre les deux méthodes d'initialisation tend donc clairement en faveur de l'initialisation par passage de message, qui se révèle à la fois plus efficace et plus performante. Malgré ces avantages, il faut garder à l'esprit que le parallélisme de sous-bloc devient peu

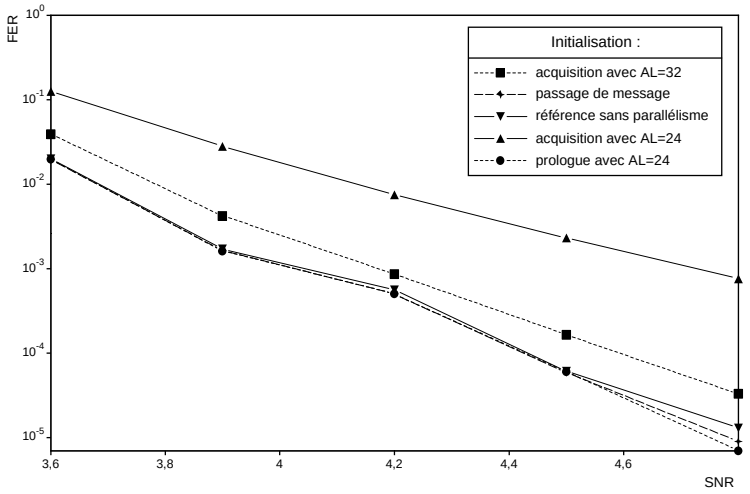

Figure 3.11 — Performances en taux d'erreur paquet et méthodes d'initialisation pour un fort degré de parallélisme (47), DVB-RCS, R=6/7, trame de 188 octets, 5 bits de quantification, algorithme Log-MAP

efficace à fort degré de parallélisme (figure 3.9), si bien que l'architecture dans sa globalité peut s'avérer moins efficace à de tel degré de parallélisme. En repartant de la définition globale de l'efficacité d'un parallélismeselon l'équation 3.3 et des équations 3.25 et 3.23, l'efficacité du parallélisme de sous-bloc s'écrit :

$$E = \frac{d_{SB}}{(1 + \lfloor d_{SB}/T \rfloor / it)\,(1 + \%C^{dup}(d_{SB} - 1))} \tag{3.27}$$

Cette métrique sera maximale pour l'une des deux valeurs suivantes :

$$d_{SB\max 1} = T.\left\lfloor \sqrt{\left(\frac{1}{\%C^{dup}} - 1\right).\frac{it}{T}} \right\rfloor - 1 \tag{3.28}$$

$$d_{SB\max 2} = T.\left\lceil \sqrt{\left(\frac{1}{\%C^{dup}} - 1\right).\frac{it}{T}} \right\rceil - 1 \tag{3.29}$$

Ainsi il est certain qu'au-dessus d'un degré de parallélisme de sous-bloc $d_{SB\max 2}$, l'efficacité de l'architecture sera dégradée. Il sera donc préférable d'envisager un autre type de parallélisme. Une option consiste à passer au niveau de parallélisme de turbo-décodeur. En effet, comme la composante de l'efficacité d'architecture apportée par les parallélismes de turbo-décodeur avoisine 1 (débit et surface doublés), l'efficacité de l'architecture ne sera pas dégradée. L'autre option, que nous allons aborder dans la prochaine section, consiste à utiliser le parallélisme de décodeur composant.

3.4 Parallélisme de décodeur composant

Comme décrit précédemment, le parallélisme de décodeur composant ne peut s'employer efficacement qu'avec la technique du décodage combiné, ou shuffled decoding [56]. Néanmoins les échanges immédiats d'informations extrinsèques entre les décodeurs peuvent affecter grandement l'efficacité du parallélisme selon les autres choix de parallélismes réalisés et les règles d'entrelacement.

3.4.1 Efficacité du décodage combiné

Comme défini par l'équation 3.7, le calcul de l'efficacité d'un parallélisme nécessite la connaissance de l'accélération qu'il présente. Appliqué à l'efficacité du décodeur composant, il nous faut évaluer les temps de décodage avec un décodage simple (pour atteindre un taux d'erreur fixé) et avec un décodage combiné (pour atteindre ce même taux d'erreur).

Dans le cas d'un décodage simple, le temps de décodage s'exprime :

$$t_{d\ serie} \propto 2.it_{serie}.\left(\frac{N}{d_{SB}} + t_p\right) \tag{3.30}$$

où t_p est le temps de propagation nécessaire pour que l'information sur un symbole obtenu dans un décodeur puisse être utilisée dans l'autre décodeur.

Comme cette contrainte de temps s'applique à tous les symboles, il faut attendre un temps t_p avant de pouvoir commencer la prochaine demi-itération. Sans cette attente le système s'expose au risque de conflits de consistance (la figure 3.15 illustre un conflit de consistance). c'est-à-dire qu'un décodeur composant utilise comme information a priori l'information extrinsèque qu'il a créée à l'itération précédente, au lieu de l'information extrinsèque créée par l'autre décodeur. Ces conflits de consistance ont un impact très négatif sur la convergence du processus itératif, puisqu'il n'y a plus d'échange d'information entre les décodeurs composants sur le symbole souffrant d'un conflit de consistance. Dans [106], plusieurs méthodes sont proposées pour anticiper le décodage sans attendre t_p. Il s'agit soit de gérer à l'exécution les conflits en dupliquant la mémoire où les conflits peuvent survenir, soit de construire l'entrelaceur de manière à faire disparaître ces conflits de consistance.

Dans le cas d'un décodage combiné, le temps de décodage ne dépendra plus du temps de propagation, vue que les deux décodeurs composants sont exécutés dans le même temps. Il s'exprimera simplement :

$$t_{d\ combiné} \propto it_{combiné}\frac{N}{d_{SB}} \tag{3.31}$$

D'où l'efficacité du décodage combiné

$$E_{combiné} = \frac{t_{d\ serie}}{d_{DC}.t_{d\ combiné}} = \frac{it_{serie}}{it_{combiné}}.\left(1 + \frac{t_p.d_{SB}}{N}\right) \tag{3.32}$$

Par souci de précision, l'efficacité du parallélisme est calculée en utilisant le nombre moyen d'itérations nécessaires pour converger vers la bonne solution à un SNR donné. Ainsi, on observe des variations importantes sur l'efficacité du parallélisme de décodeur composant, l'efficacité variant de 0,6 à 0,95. Ces variations s'expliquent d'une part par le choix des autres

techniques de parallélisme employées (sections 3.4.2 et 3.4.3), d'autre part par l'effet du temps de propagation et son exploitation (section 3.4.4 et 3.4.5).

L'influence des autres techniques de parallélisme sur l'efficacité du décodage combiné sera étudié en considérant un temps de propagation nul, de sorte que l'efficacité du décodage combiné se confond avec la rapidité de convergence :

$$V_C = \frac{it_{serie}}{it_{combiné}} \tag{3.33}$$

3.4.2 Décodage combiné et parallélisme des calculs du BCJR

L'efficacité du parallélisme de décodeur composant est très variable selon le schéma de décodage retenu pour l'algorithme BCJR (section 3.2.1.2). Par exemple comme nous le verrons dans la section suivante, l'efficacité du parallélisme de décodeur composant vaut 0,6 pour le décodage avec le schéma papillon aller, 0,7 pour le schéma papillon, et 0,78 pour le schéma papillon replica en supposant dans tous les cas un décodage combiné sans utiliser de parallélisme de sous-bloc sur une trame de 188 octets. Plus généralement, l'efficacité du décodage combiné sera toujours la plus grande avec le schéma papillon replica et la plus petite avec le schéma papillon aller.

La suprématie du décodage papillon replica est naturelle : rappelons que ce schéma a été introduit dans [98] et [107] justement pour accélérer la convergence du processus itératif du décodage combiné. L'idée originale du décodage combiné replica est de faire travailler en parallèle deux unités BCJR pour chaque décodeur composant : une travaillant dans le sens aller et l'autre dans le sens retour. Ceci est équivalent à émettre au sein d'une même unité BCJR des informations sur les deux récursions (aller et retour) du papillon. Ainsi le schéma papillon replica génère deux informations extrinsèques par symbole et par itération au lieu d'une information extrinsèque par symbole et par itération pour les autres schémas. En conséquence, les décodeurs composants jouissent d'une information *a priori* mise à jour plus rapidement et donc convergent plus vite. Cependant le schéma papillon replica implique une augmentation de la complexité calculatoire (besoin de 4 unités de calcul BCJR au lieu de 3 en moyenne pour les autres) d'une part et requiert deux fois plus de communication pour échanger les informations extrinsèques d'autre part.

On peut obtenir une idée plus juste du parallélisme de décodeur composant en exprimant son efficacité en vertu de l'équation 3.3 qui s'exprimera ici :

$$E = \frac{2.V_C}{1 + \%C^{dup}} \tag{3.34}$$

avec $\%C^{dup}$ qui représente le ratio de complexité d'un décodeur BCJR-SISO C_{BCJR} sur l'ensemble du turbo-décodeur $C_{turbodécodeur}$. Ce ratio peut être évalué avec l'équation 3.16 sans utiliser de parallélismes, i.e. :

$$\%C^{dup} = \frac{C_{BCJR}}{C_{turbodécodeur}} = \frac{C_{BCJR}}{M_{extrinsèque} + M_{entrée\ canal} + C_{BCJR}} \tag{3.35}$$

Remarquons que le parallélisme de décodeur composant s'avère efficace (i.e. $E > 1$) pour tous les schémas de parallélisme des calculs du BCJR à condition que :

$$\%C^{dup} < 2V_C - 1 \tag{3.36}$$

Ainsi le décodage combiné est efficace même pour le schéma papillon aller si ce ratio est inférieur à 20%, ce qui est généralement le cas pour les circuits implantant un décodeur de code convolutif.

Par ailleurs, la complexité d'un décodeur BCJR-SISO diffère selon le schéma de calcul retenu comme on l'a constaté dans la section 3.2.1.2, notamment pour le schéma papillon aller. Avec l'inégalité 3.14, on peut montrer que le pourcentage de complexité $\%C^{dup}$ est plus faible pour le schéma papillon aller que pour les autres schémas. Ainsi la complexité confère un avantage au schéma papillon aller, mais cet avantage ne suffit pas dans les conditions présentées à contrebalancer la mauvaise rapidité de convergence du schéma papillon aller. Sur cet exemple, on peut montrer que le papillon replica est toujours le schéma le plus efficace, suivi du schéma papillon aller et ensuite du schéma papillon.

Il est important de garder à l'esprit que cette métrique d'efficacité ne tient compte ni du taux de calcul moyen ni des accès mémoires. Ces deux métriques affectent notamment la consommation du circuit et jouent en la défaveur du schéma papillon replica pour un contexte basse consommation. Le choix d'un des schémas sera donc très dépendant des contraintes architecturales imposées.

3.4.3 Décodage combiné et parallélisme de sous-blocs

Dans cette section, nous allons voir que l'utilisation conjointe de parallélisme de décodeur composant et de parallélisme de sous-bloc permet d'améliorer la rapidité de convergence du décodage combiné. Pour rappel, la rapidité de convergence désigne le rapport entre le nombre d'itérations nécessaire pour une exécution séquentielle des décodeurs composants (i.e. dans ce cas, d'abord les d_{SB} décodeurs BCJR-SISO du premier décodeur composant, puis ceux du deuxième décodeur composant) et entre le nombre d'itérations nécessaire pour un décodage combiné (i.e. avec $d_{SB}.d_{DC}$ décodeurs BCJR-SISO en parallèle). La figure 3.12 représente la rapidité de convergence associée au décodage combiné en fonction du degré de parallélisme de sous-bloc. Incontestablement, peu importe le parallélisme des calculs du BCJR retenu, la rapidité de convergence du décodage combiné croît avec le parallélisme de sous-bloc. On peut également remarquer qu'à fort degré de parallélisme de sous-bloc, les rapidités de convergence des schémas papillon et papillon aller tendent vers celle du schéma papillon replica. On notera que cette dernière est quasi constante pour les forts degré de parallélisme à une valeur de rapidité de convergence de 0,95.

D'après les résultats obtenus dans la section 3.3, on sait que le parallélisme de sous-bloc devient de moins en moins efficace quand on augmente son degré de parallélisme. Donc passé un certain seuil, le parallélisme de décodeur composant sera préférable au parallélisme de sous-bloc. Pour déterminer ce seuil, il convient de considérer le degré de parallélisme de décodeur BCJR-SISO d (i.e. $d = d_{SB}.d_{DC}$). La figure 3.13 représente la rapidité de convergence associée au degré de parallélisme de décodeur BCJR-SISO (i.e. le ratio des itérations nécessaires sans parallélisme de décodeur BCJR-SISO sur les itérations avec) pour quatre configurations initialisant les sous-blocs par passage de message.

La première configuration n'utilise que le parallélisme de sous-bloc ($d = d_{SB}$) avec un schéma papillon comme parallélisme des calculs du BCJR. Comme à faible degré de parallélisme le parallélisme de sous-bloc est le plus efficace que le parallélisme de décodeur composant, on observe logiquement que cette configuration est la plus rapide à converger.

Les autres configurations utilisent, en plus du parallélisme de sous-bloc, le parallélisme de décodeur composant avec les schémas de décodage papillon, papillon aller et papillon replica.

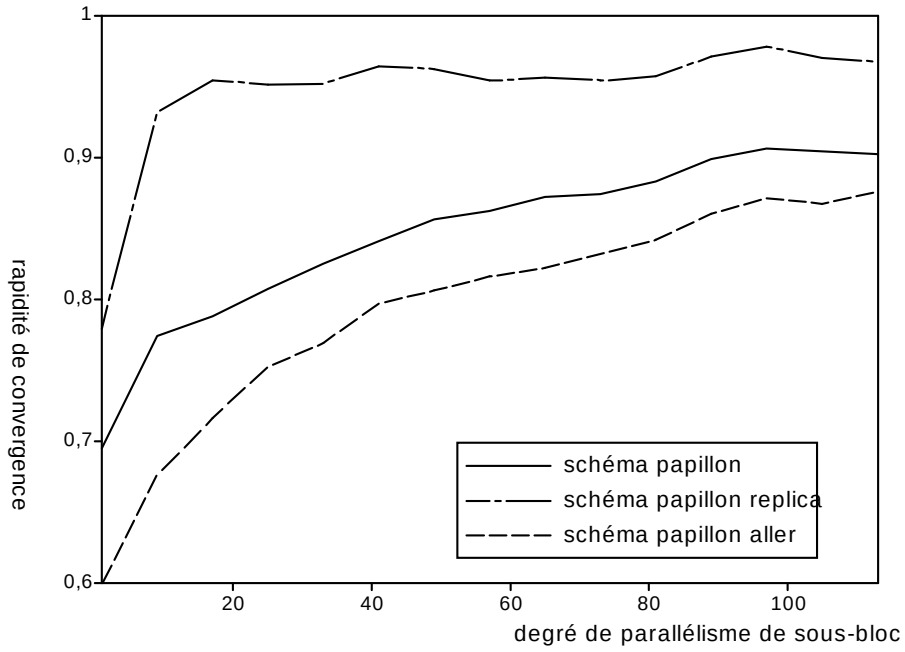

Figure 3.12 — Rapidité de convergence du décodage combiné en fonction du degré de parallélisme et du schéma de décodage BCJR, initialisation par passage de message, divers entrelaceurs, R=6/7, trame de 188 octets

Comme annoncé dans la section 3.4.2, l'ordre de ces trois configurations selon la rapidité de convergence reste toujours : schéma papillon replica pour le plus rapide, schéma papillon et schéma papillon aller pour le plus lent. A fort degré de parallélisme, ces trois configurations convergent plus rapidement que la configuration sans décodage combinée à cause de la perte d'efficacité du parallélisme de sous-bloc. Notons que les écarts entre ces trois configurations diminuent à mesure que le degré de parallélisme de décodeur BCJR-SISO augmente. Dans ce cas, le choix de la configuration dépendra essentiellement des contraintes matérielles et des métriques de décisions retenues.

Pour notre part, nous continuerons l'analyse de ces configurations avec la métrique débit-surface. En reprenant les métriques d'efficacité du parallélisme de sous-bloc seul (équation 3.27) et du parallélisme de décodeur composant seul (équation 3.34), on obtient l'efficacité associée au niveau de parallélisme de décodeur BCJR-SISO :

$$E = \frac{d_{SB}.d_{DC}.V_C}{(1 + \lfloor d_{SB}/T \rfloor / it)\,(1 + \%C^{dup}(d_{SB}.d_{DC} - 1))} \tag{3.37}$$

Dans cette métrique, seul les facteurs V_C et $\%C^{dup}$ différencient les configurations avec décodage combiné.

La figure 3.14 illustre cette métrique dans un contexte d'implantation classique où un décodeur BCJR-SISO occupe 25% de la complexité totale du circuit

En tirant profit de sa complexité inférieure ($\%C^{dup}$ de 20%), la configuration avec le schéma papillon aller devient la plus efficace à fort degré de parallélisme. Seulement l'important n'est pas de trouver la configuration ayant la meilleure efficacité possible à fort

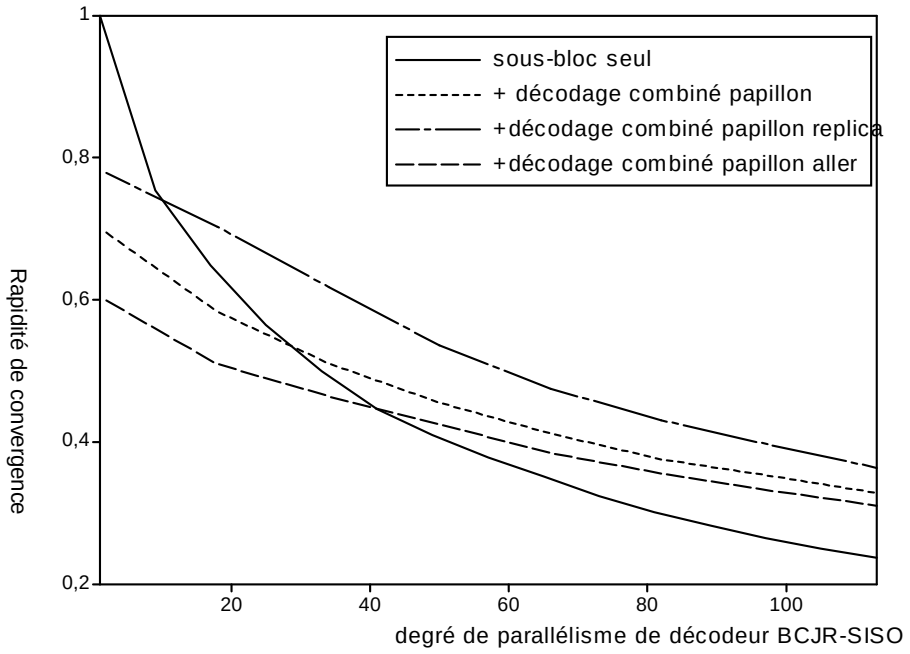

Figure 3.13 — Rapidité de convergence associée au degré de parallélisme de décodeur BCJR-SISO, initialisation par passage de message, divers entrelaceurs, R=6/7, trame de 188 octets

degré de parallélisme, mais plutôt celle qui maximise l'efficacité du niveau de parallélisme de BCJR-SISO. Dans l'exemple présenté, la configuration maximisant l'efficacité du niveau de parallélisme de BCJR-SISO est celle utilisant le décodage combiné et le schéma papillon replica. En se positionnant sur le maximum associé au niveau de parallélisme de BCJR-SISO, on est certain que le moyen le plus efficace pour augmenter les performances consiste à passer au niveau de parallélisme supérieur, c'est-à-dire celui de parallélisme de turbodécodeur.

Lorsque la conception n'exige pas de telles exigences sur les débits (donc des degrés de parallélisme plus faibles), il est intéressant de connaître les limites en deçà desquels il est plus efficace d'utiliser le parallélisme de sous-bloc que celui de décodeur composant. Les valeurs de d de ces limites peuvent être obtenues lorsque :

$$\frac{1}{it + \left\lfloor \frac{d}{T} \right\rfloor} = \frac{V_C}{it + \left\lfloor \frac{d}{2.T} \right\rfloor} \tag{3.38}$$

Dans la plupart des cas, ces limites sont atteintes pour des valeurs de d entre une et deux fois la valeur du seuil T.

3.4.4 Effet du temps de propagation

Nous allons maintenant considérer une implantation réelle en prenant en compte le temps de propagation nécessaire pour que l'information sur un symbole obtenu dans un décodeur puisse être utilisée dans l'autre décodeur. Ce temps de propagation inclut donc le temps d'accès en lecture à la mémoire contenant l'information a priori, le temps de calcul pour l'information extrinsèque et le temps d'écriture dans la mémoire de destination. Ce temps peut

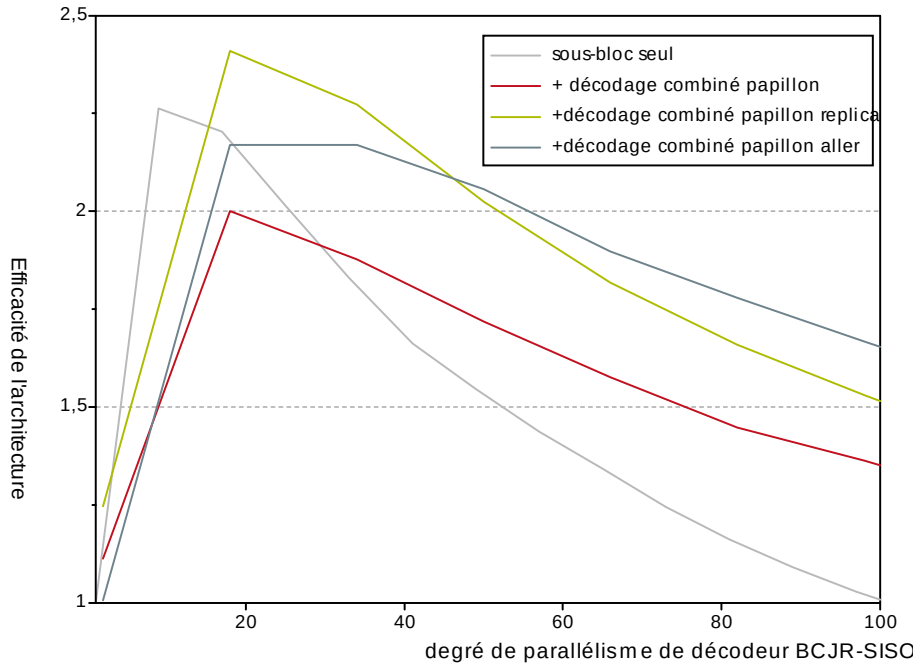

Figure 3.14 — Efficacité du niveau de parallélisme de décodeur BCJR-SISO pour différentes configurations, initialisation par passage de message, divers entrelaceurs, $t_p = 0$, R=6/7, trame de 188 octets, $\%C^{dup}_{papillon} = 0,25$, $\%C^{dup}_{aller} = 0,2$

représenter plusieurs cycles de calculs notamment lorsque l'on a recours à des structures de communications complexes nécessaires pour réaliser l'entrelacement sans générer de conflits.

Comme nous l'avons vu avec l'équation 3.31, le temps de propagation n'influe pas directement sur le temps de décodage lorsque l'on réalise un décodage combiné. Cependant une augmentation du temps de propagation, i.e. le temps entre une lecture et une écriture d'information extrinsèque, accroît le nombre conflits de consistance durant le décodage combiné. On parle de conflit de consistance dans une mémoire lorsqu'une adresse mémoire est lue avant que la donnée désirée ne soit écrite à cette adresse. La figure 3.15.a représente un conflit de consistance dans le cas où les deux décodeurs composants partagent le même plan mémoire. Dans cet exemple, le décodeur composant 1 est en conflit de consistance puisque son accès en lecture (L_1) précède l'accès en écriture du décodeur composant 0 (E_0). En conséquence, les deux décodeurs composants sont condamnés à n'utiliser que l'information extrinsèque fourni par le décodeur composant 1 puisque l'autre information est écrasée à chaque itération par l'écriture E_1. L'impossibilité d'échanger ces informations extrinsèques pour les symboles en conflit de consistance empêche une convergence correcte du processus itératif et dégrade les performances en taux d'erreur.

Il est possible de résoudre ce problème au prix d'un surcoût de complexité en intégrant un deuxième plan mémoire pour les symboles en conflit (figure 3.15.b). Ainsi, l'échange itératif entre les décodeurs composants est maintenu d'une itération à l'autre en cas de conflit de consistance. Ce mécanisme permet d'assurer la convergence du processus itératif, mais cette dernière est ralentie à cause des symboles en conflit. Dans le décodage combiné normal, un des deux décodeurs composants profite dans la même itération de l'information extrinsèque de l'autre et cela pour tous les symboles. Or pour les symboles en conflit, les échanges sont

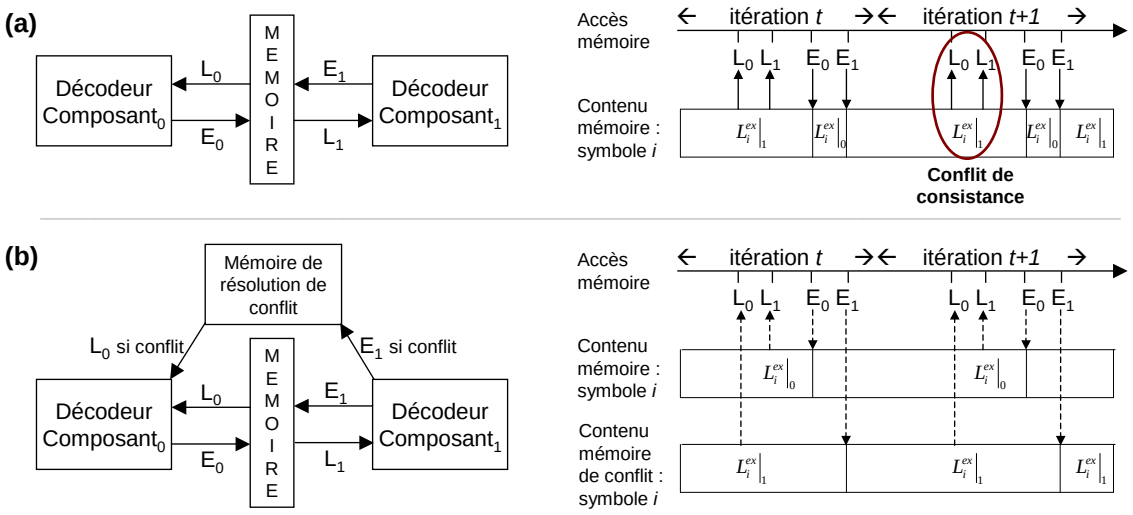

Figure 3.15 — Conflit de consistance dans un turbo-décodeur : (a) avec un plan mémoire ; (b) avec gestion par dédoublement du plan mémoire

repoussés à l'itération d'après. En conséquence, ces symboles itèrent deux fois moins vite que les symboles sans conflit. Du coup, la rapidité de convergence du décodage combiné est dégradée.

Nous avons simulé la dégradation moyenne sur la rapidité de convergence pour plusieurs entrelaceurs. Elle est représentée pour différents temps de propagation (l'unité de temps étant le temps nécessaire pour traiter un symbole) en fonction du degré de parallélisme de sous-bloc aussi bien pour le schéma papillon (figure 3.17) que pour le schéma papillon replica (figure 3.16).

Sur les deux figures, on observe une diminution de la vitesse de convergence à mesure que le temps de propagation augmente. On peut constater que le schéma papillon replica est plus robuste à ces dégradations que le schéma papillon. De plus, on peut remarquer que les dégradations deviennent considérables lorsque le temps de calcul d'un sous-bloc devient comparable à t_p. Cette dégradation brutale s'explique puisque le pourcentage des informations extrinséques non mises à jour dans l'itération courante devient très important comme nous le verrons dans la section suivante. Par exemple, sur la figure 3.17, on peut observer pour des temps de propagation de 6 et 10 la limite où aucune information extrinsèque n'est échangée durant l'itération. Dans ce cas, la rapidité de convergence du décodage combiné papillon tend vers 0,5. Dans ce cas, le décodage combiné nécessite deux fois plus d'itérations qu'un décodage séquentiel pour converger. Malgré l'absence de gain sur le temps de calcul, le décodage combiné reste plus rapide que le décodage séquentiel. L'accélération est alors réalisée non plus sur les calculs mais sur les communications, puisque le décodage combiné n'a pas à attendre après chaque itération que les informations extrinsèques soient propagées (voir équation 3.32).

Donc, la préservation d'une bonne rapidité de convergence du décodage combiné passe par la préservation d'un pourcentage raisonnable d'informations extrinsèques échangeant au cours d'une itération. Or comme nous allons le voir, ce pourcentage dépend de la valeur du temps de propagation et des règles d'entrelacement, c'est pourquoi une conception soigneuse de l'entrelaceur permet de maximiser la rapidité de convergence du décodage combiné et en conséquence de l'efficacité de celui-ci.

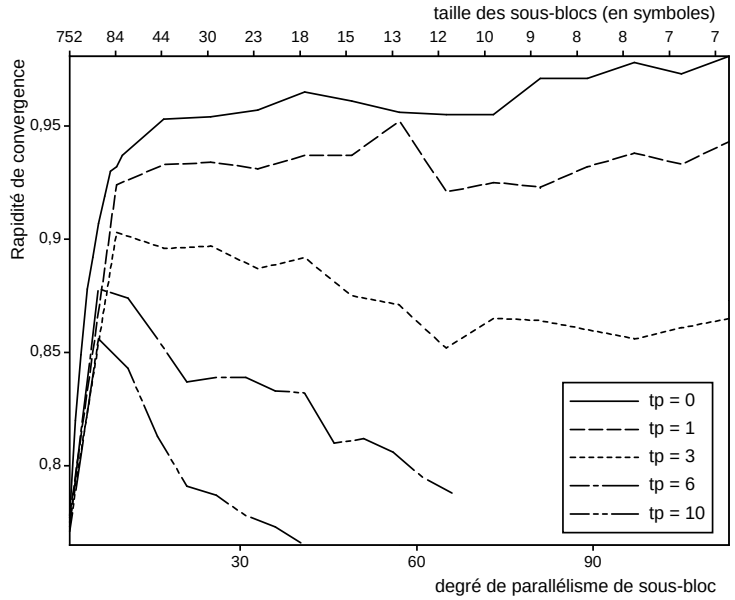

Figure 3.16 — Rapidité de convergence du décodage combiné avec le schéma papillon réplica pour plusieurs temps de propagation, initialisation par passage de message, divers entrelaceurs, R=6/7, trame de 188 octets

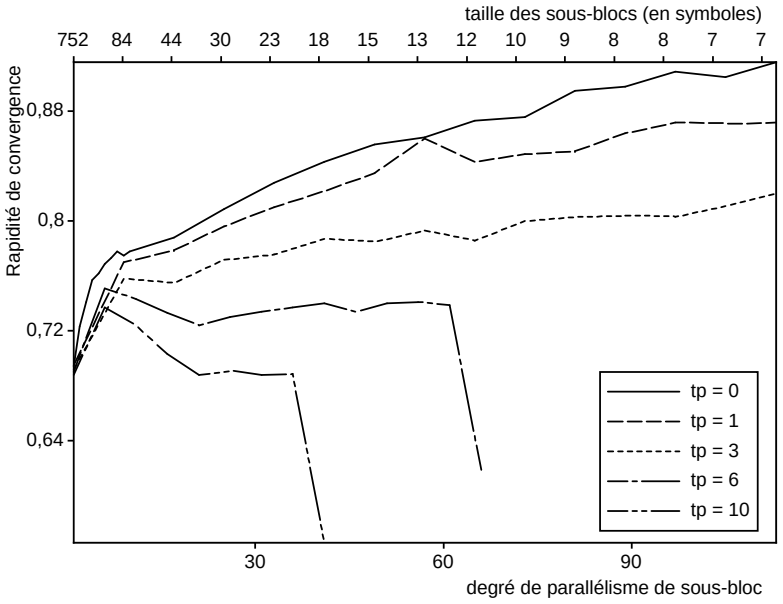

Figure 3.17 — Rapidité de convergence du décodage combiné avec le schéma papillon pour plusieurs temps de propagation, initialisation par passage de message, divers entrelaceurs, R=6/7, trame de 188 octets

3.4.5 Contraintes sur la conception d'entrelaceurs

3.4.5.1 Règles de conception pour l'entrelaceur pour optimiser le décodage combiné

Soit S_j^k le décodeur SISO traitant le k$^{\text{ème}}$ sous-bloc de la trame sur le j$^{\text{ème}}$ décodeur composant (par exemple j=0 ou 1 pour une concaténation paralléle de deux codes convolutifs).

Soit t_p le temps de propagation nécessaire pour mettre à jour l'information extrinsèque dans l'autre décodeur.

Soit N la taille de la trame, M la taille de sous-bloc et d_{SB} le degré de parallélisme de sous-bloc.

Selon le schéma des calculs BCJR considéré, S_j^k peut délivrer à la i$^{\text{ème}}$ itération des informations extrinsèques pour le symbole estimé $\hat{u}_n$ dans la direction aller, noté $\overrightarrow{L}^{(i)}_{eS_j^k}(\hat{u}_n)$, et/ou dans la direction retour, noté $\overleftarrow{L}^{(i)}_{eS_j^k}(\hat{u}_n)$. Le temps d'émission de cette information extrinsèque sera noté $\overrightarrow{t_j}(n)$ pour le sens aller et $\overleftarrow{t_j}(n)$ pour le sens retour.

Pour annuler l'effet du temps de propagation constaté précédemment, l'information a priori doit, pour chaque décodeur BCJR-SISO et pour chaque symbole, bénéficier d'au moins une mise à jour avant d'être retraitée. Puisque la création d'information extrinsèque peut se faire dans le sens aller et retour et dans les deux décodeurs, il existe 4 possibilités pour transférer l'information entre les deux décodeurs au cours d'une itération et une seule suffit au bon fonctionnement du processus itératif. On peut donc interpréter cette contrainte comme une règle de conception sur l'entrelaceur Π s'exprimant :

$$\forall n \in \{1..N\}, \quad \begin{cases} \left|\overrightarrow{t_0}(n) - \overrightarrow{t_1}(\Pi(n))\right| > t_p \\ ou \\ \left|\overrightarrow{t_0}(n) - \overleftarrow{t_1}(\Pi(n))\right| > t_p \\ ou \\ \left|\overleftarrow{t_0}(n) - \overrightarrow{t_1}(\Pi(n))\right| > t_p \\ ou \\ \left|\overleftarrow{t_0}(n) - \overleftarrow{t_1}(\Pi(n))\right| > t_p \end{cases} \tag{3.39}$$

Pour faciliter l'utilisation de ces règles, nous allons utiliser une représentation bidimentionnelle de l'entrelaceur comme introduit dans [108]. Dans cette représentation, l'ordre naturel (resp. entrelaçé) est indexé sur l'axe horizontal (resp. vertical) et le symbole u_n sera représenté par le point $(n, \Pi(n))$. Ainsi une contrainte de conception sur l'entrelaceur sera transformée en une région bannie [109] et les points d'un entrelaceur respectant cette contrainte ne pourront pas appartenir à la zone bannie. La représentation graphique des contraintes de conception sur l'entrelaceur est dénommée masque de l'entrelaceur.

Pour notre contrainte (3.39), un point d'index n est banni si et seulement si aucune des inégalités est respectée. c'est-à-dire, pour cette index n, l'intersection des régions violant une inégalité. Le masque de l'entrelaceur représentant la règle complète est ensuite obtenu par union des intersections associés aux différents index.

Les fonctions de temps étant dépendantes des différents parallélismes choisis, nous allons maintenant regarder comment on peut obtenir les masques d'entrelaceur associés aux méthodes de parallélismes choisies.

3.4.5.2 Masque d'entrelaceur et parallélisme des calculs BCJR

Dans cette section, nous montrerons comment le choix du parallélisme des calculs BCJR affecte le masque de l'entrelaceur. Les seuls parallélismes considérés seront les parallélismes de décodeur composant et des calculs BCJR et nous supposerons que les itérations se succèdent sans temps d'attente, de sorte que le premier et le dernier symbole d'un sous-bloc sont voisins.

Commençons par l'exemple le plus simple : utilisation du schéma papillon aller par les deux décodeurs. Comme le schéma papillon aller n'émet que dans la direction aller, la première inégalité est la seule à être valide. Ainsi la région bannie définie par cette inégalité est centrée sur la diagonale directe avec une largeur de t_p autour de la diagonale comme le montre la figure 3.18.a. L'hypothèse d'itération continue sans attente prolonge la diagonale aux coins inférieur droit et supérieur gauche.

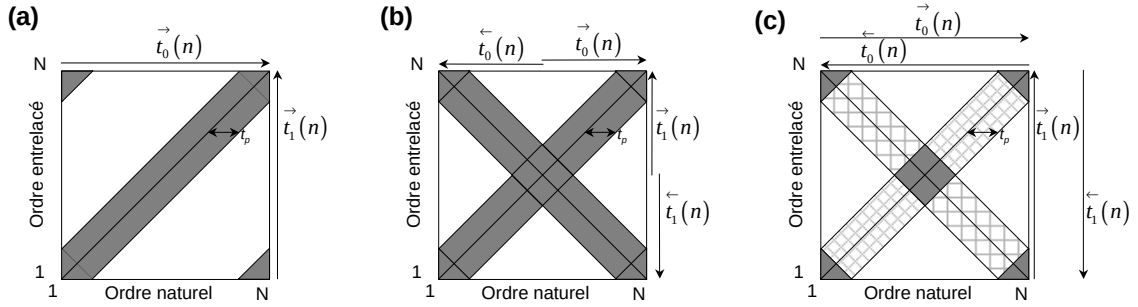

Figure 3.18 — Masque d'entrelaceur et parallélisme de calcul BCJR

Considérons maintenant l'utilisation du schéma papillon sur les deux décodeurs composants. Pour ce schéma (figure 3.3.b), $\overrightarrow{t_j}$ et $\overleftarrow{t_j}$ sont définies de manière exclusive et n'existent que dans la deuxième partie du papillon. En conséquence, un symbole ne sera affecté que par une seule inégalité. Chacune pouvant être valide, on obtient donc quatre régions bannies ayant la forme d'une demie-diagonale (figure 3.18.b). La combinaison de ces régions nous donne le masque de l'entrelaceur associé à cette combinaison de parallélisme. On peut constater que ce masque en forme de croix recouvre intégralement le plan dès que $t_p = \frac{N}{2}$. Dans ces conditions, aucune information extrinsèque n'est échangée au cours d'une itération. On comprend alors pourquoi la vitesse de convergence de la figure 3.17 tombe à 0,5 lorsque la taille des sous-blocs s'approche de deux fois le temps de propagation.

Le cas d'un schéma papillon replica est plus compliqué, puisque $\overrightarrow{t_j}$ et $\overleftarrow{t_j}$ sont définies pour tous les symboles. Ainsi pour chaque symbole, les quatres inégalités existent et chacune définit une diagonale de largeur t_p accompagnée de coins. Contrairement à ce qui a été vu précédemment, les règles ne s'appliquent pas de manière exclusive sur un symbole ; le masque sera donc obtenu en faisant l'intersection des différentes diagonales (figure 3.18.c). Avec seulement un carré au centre du plan et un autre sur les bords[2], ce masque est nettement moins contraignant que celui du schéma papillon, ce qui explique pourquoi le décodage combiné converge plus rapidement avec ce schéma.

Lorsque t_p tend vers $\frac{N}{2}$, ce masque tend à recouvrir l'ensemble du plan. Pourtant contrairement à ce qui se passe avec le schéma papillon, la vitesse de convergence ne chute pas brutalement à 0,5 (voir figure 3.16), car il existe encore des échanges d'informations extrinsèques dans une même itération. Ces échanges secondaires ne sont pas pris en compte dans l'équation

[2]Le carré est éclaté visuellement en 4 triangles

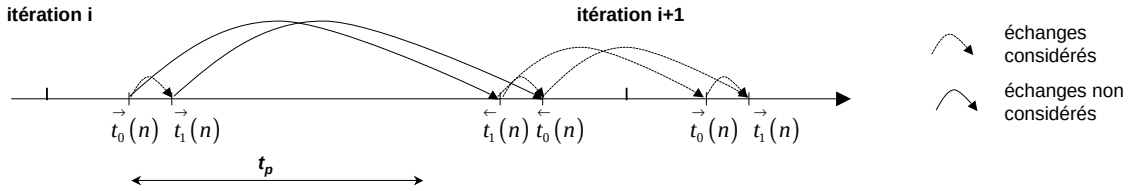

Figure 3.19 — Echanges primaires et secondaires pour un schéma replica

3.39 qui utilise uniquement les échanges primaires (voir figure 3.19). Or quand t_p deviens supérieur à $\frac{N}{2}$, les échanges primaires n'existent plus. En revanche, les échanges secondaires restent possibles tant que t_p est inférieur à N. Plus généralement, il est possible de définir une nouvelle classe d'échange (tertiaire, quaternaire,...) pour chaque intervalle $\left[\frac{N}{2}i, \frac{N}{2}i + \frac{N}{2}\right[$ de temps de propagation. Par ailleurs, la classe secondaire[3] n'existe que pour le schéma replica, car ce type d'échange nécessite la production d'une même information à deux endroits différents dans la même itération.

3.4.5.3 Masque d'entrelaceur et parallélisme de sous-bloc

Nous allons maintenant considérer l'ajout de parallélisme de sous-bloc de degré M. On supposera que chaque décodeur composant délivre les informations extrinsèques issues de ces M décodeurs BCJR-SISO de manière synchrone.

Ainsi, d'après la règle 3.39, la région bannie pour un symbole d'index n dans l'ordre naturel (resp. entrelacé) est identique sur tous les sous-blocs dans l'autre entrelacé (resp. naturel). Donc, le masque de l'entrelaceur combine simplement M^2 sous-masques associés à chacun des couples de sous-blocs entre l'ordre naturel et l'ordre entrelacé. Chaque sous-masque est défini en fonction du parallélisme des calculs BCJR utilisé par le couple de décodeur BCJR-SISO.

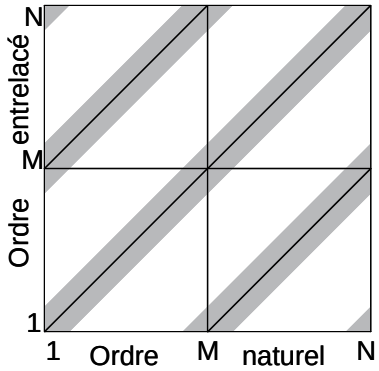

Figure 3.20 — Masque d'entrelaceur pour un degré de parallélisme de sous-bloc 2 et avec un schéma papillon aller

Par exemple, la figure 3.20 représente le masque d'entrelaceur associé à l'équation 3.39 pour un turbo-décodeur ayant un degré de parallélisme de sous-bloc égal à 2 et dans lequel chaque décodeur BCJR SISO suit un schéma papillon aller.

[3]Et par extension toutes les autres classes paires

Evidement la région bannie sur le masque s'agrandit avec le degré de parallélisme de sous-bloc rendant du même coup le choix d'un bon entrelaceur plus difficile. Il en va de même lorsque le temps de propagation devient trop grand. Dans un cas comme dans l'autre, le masque devient trop contraint et il devient impossible de trouver un entrelaceur respectant le masque et de bonnes propriétés d'étalement pour garder des performances de décodage correctes.

Evidemment, le masque peut être utilisé à titre uniquement indicatif pour détecter les symboles n'ayant plus d'échanges primaires. Pour passer outre le masque, il faut accepter le ralentissement de la convergence impliqué par ces symboles et dédoubler la mémoire pour permettre à ces symboles de suivre le processus itératif (voir section 3.4.4).

3.4.5.4 Exemple de conception

Ce paragraphe est consacré à la conception d'un entrelaceur efficace pour le décodage combiné d'un paquet MPEG avec un code double binaire (188 octets, soit 752 symboles à entrelacer). Afin de présenter un masque d'entrelaceur très contraint, le degré de parallélisme de sous-bloc P et la taille de sous-bloc M sont respectivement de 47 et 16. Le temps de propagation (t_p) est égal au temps requis pour traiter 3 symboles. Dans ces conditions et avec un entrelaceur quelconque, la rapidité de convergence du décodage combiné vaut 0,78 pour le schéma papillon et 0,87 pour le schéma papillon replica (voir les figures de la section 3.4.4).

Pour respecter le masque et limiter les conflits de communication inhérents au parallélisme de sous-bloc, nous utiliserons un entrelaceur hiérarchique tel que ceux présentés dans [106]. La fonction d'entrelacement est alors décomposée en une permutation spatiale P-periodique et une permutation temporelle, où $n = M.r + t$ est l'index du symbole, r l'index du sous-bloc et t l'index temporel dans le sous-bloc.

$$\Pi(n) = \Pi(t, r) = \Pi_S(t, r).M + \Pi_T(t) \tag{3.40}$$

De cette manière, le symbole situé à l'index n dans l'ordre naturel est traité à l'instant t par le décodeur SISO S_1^r et dans l'ordre entrelacé à l'instant $\Pi_T(t)$ par le décodeur SISO $S_2^{\Pi_S(t,r)}$. Ainsi cette famille d'entrelaceur assurera l'absence de collisions grâce à la permutation spatiale et le respect du masque grâce à la permutation temporelle. La permutation spatiale choisie est définie par l'équation 3.41 tandis que la permutation temporelle est représentée par la figure 3.21.

$$\Pi_S(t, r) = (2.t + r) \bmod 47 \tag{3.41}$$

La permutation temporelle respecte au mieux le masque du schéma papillon puisque 4 des 16 symboles ne respectent pas le masque qui occupe 69% de l'espace de conception. Par ailleurs, la permutation respecte parfaitement les masques des schémas papillon aller et papillon replica. Selon les résultats de simulation, la rapidité de convergence du décodage combiné associée à cette entrelacement est de 0,84 pour le schéma papillon et 0,94 pour le schéma replica (soit respectivement une efficacité de 1 et 1,12). Ces rapidités de convergence correspondent à celles obtenues sur les figures 3.17 et 3.16 avec un temps de propagation nul. Ainsi, on s'aperçoit que l'effet du temps de propagation s'estompe grâce aux règles d'entrelacement proposées. Même si la conception de l'entrelaceur doit forcément tenir compte d'autres contraintes pour conserver de bonnes propriétés asymptotiques, cet exemple montre

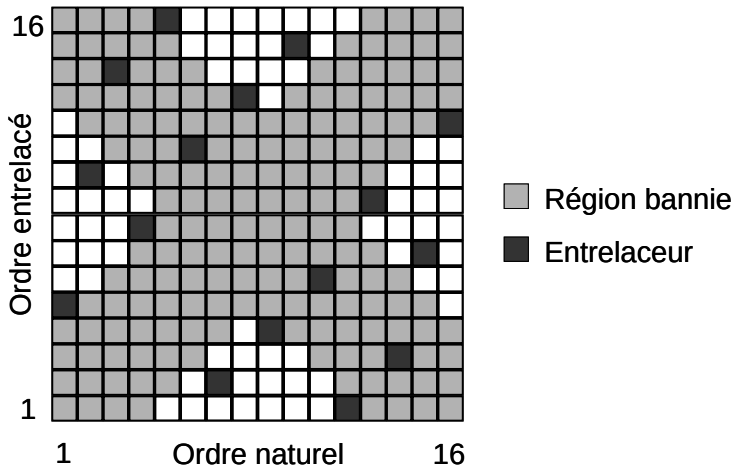

Figure 3.21 — Permutation temporelle et masque du schéma papillon

que la conception conjointe de l'entrelaceur et du parallélisme permet d'augmenter significativement l'efficacité de parallélisme de décodeur BCJR-SISO grâce à la technique du décodage combiné.

3.5 Conclusion

Ce chapitre présente une étude détaillée du parallélisme exploitable dans une application de turbo-réception utilisant l'algorithme BCJR, comme, par exemple, un turbo-décodeur convolutif. Le chapitre expose une classification de ces parallélismes à trois niveaux de granularité : parallélisme des métriques BCJR, parallélisme de décodeur SISO BCJR et parallélisme de turbo-décodeur. Nous avons constaté que le niveau ayant la granularité de parallélisme la plus fine offrait logiquement un surcoût en surface faible, mais ne disposait que d'un potentiel d'accélération plutôt modeste. A l'inverse, la granularité à l'échelle du turbo-décodeur permet une accélération illimitée au prix d'un surcoût matériel maximal. En présentant un compromis intéressant entre potentiel d'accélération et complexité, le second niveau de cette classification a été analysé en détail. Nous avons ainsi montré que le parallélisme de sous-bloc largement utilisé dans la littérature était plus efficace en initialisant les sous-blocs par passage de message que par initialisation, mais également que quelque soit la méthode d'initialisation le parallélisme de sous-bloc subit une loi d'Amdahl. Ainsi, à fort degré de parallélisme, ce parallélisme peut s'avérer moins intéressant que ceux du troisième niveau de la classification. Dans ce chapitre, nous avons également montré que le parallélisme de décodeur composant, qui jusqu'à ces travaux était délaissé par les concepteurs de turbo-décodeur, permet d'améliorer l'efficacité des architectures déjà fortement accélérées. En outre, nous avons évalué l'impact de sa mise en oeuvre dans des circuits réels (i.e. avec un temps de propagation non nul) et proposé des règles de conception d'entrelaceur pour optimiser l'efficacité de ce parallélisme.

Les connaissances sur le parallélisme des turbocodes convolutifs, qui sont regroupées ou produites dans ce chapitre, seront mises à profit dans le prochain chapitre qui leur donne corps sous la forme de parallélisme de tâche ou bien sous la forme de parallélisme d'instruction.

CHAPITRE

4 Réalisation d'un processeur dédié au décodage des codes convolutifs

CE chapitre présente un processeur dédié au décodage des codes convolutifs dans une application de turbo-décodage. L'objectif est d'avoir un processeur flexible et performant pour supporter un grand nombre de codes, mais il doit également être modulaire pour faciliter son intégration dans une plate-forme multiprocesseur très haut débit. Le chapitre s'organise comme suit. Dans un premier temps, une section état de l'art décrit le spectre des mises en oeuvre de turbo-décodeurs et leurs limitations autour du compromis flexibilité-performance. Une analyse est ensuite menée pour aboutir aux décisions sur la flexibilité et sur les besoins en parallélisme. Sachant cela, nous aborderons en détail l'architecture du processeur proposé ainsi que ses unités de calculs dédiés, son jeu d'instruction, des exemples d'utilisation et les performances obtenues. Pour finir, nous évaluerons rapidement quelques architectures de processeurs envisageables en prenant des décisions différentes.

4.1 Etat de l'art sur les turbo-décodeurs de code convolutif

De part la complexité de son algorithme, le turbo-décodeur est un sujet d'implantation très prisé par la communauté scientifique. Il est possible de trouver dans la littérature beaucoup de mises en oeuvre pouvant atteindre des très hauts débits grâce à des architectures dédiés à certains standards. Pour le standard 3GPP, par exemple, on peut citer des implantations ASIC [110] et FPGA [111]. Dans cette catégorie, on peut inclure les propositions de nouveaux codes plus à même d'atteindre le haut débit, comme par exemple les turbocodes à roulettes [106]. Cependant ces architectures "haut-débit" manquent cruellement de flexibilité et ne peuvent pas s'adapter à d'autres codes.

A l'inverse, d'autres implantations offrent cette flexibilité par programmabilité et/ou par reconfiguration. Par exemple, le processeur XiRISC [112] offre à la fois la programmabilité grâce à son architecture RISC et également la reconfigurabilité grâce à des FPGAs embarqués. Dans [113], cette flexibilité est obtenue grâce au processeur DSP TigerShark T201 d'Analog Device qui intègre quelques instructions dédiées aux turbocodes pour permettre un débit raisonnable. Bien d'autres implantations existent [114] [115] [116], mais globalement, malgré leur grande flexibilité, ces solutions n'offre pas la puissance de calcul nécessaire pour respecter les débits des normes les plus exigeantes ; par exemple les 150 Mbps de la norme HomeplugAV.

Pour répondre conjointement aux contraintes de performance et de flexibilité, la solution ASIP présentée à la section 1.1.2 a fait l'objet de plusieurs implantations. La première en date [117] propose la mise en oeuvre d'un turbo-décodeur sur une architecture multiprocesseur basée sur des ASIPs XTENSA. Fondé sur la méthodologie de conception d'ASIP par extension, le processeur obtenu souffre des limitations du flot de conception sur les choix architecturaux (pipeline du processeur, format des données). En conséquence, les performances en débit du processeur ne sont pas au rendez-vous.

Pour pallier les inconvénients de cette méthodologie, nous avons proposé dans [118] l'utilisation d'une méthodologie de conception ASIP par description. L'ASIP obtenu offre des résultats bien plus convaincants et surtout plus proches des implantations dédiées, malgré des limitations de jeunesse et l'absence d'exploitation du parallélisme d'instruction. Dans [119], le choix d'un très long pipeline a été fait pour exploiter au mieux le parallélisme au niveau des instructions (ILP en anglais) et de fait améliorer les performances. Cet ASIP supporte en plus les codes convolutifs présents dans les standards et intègre des interfaces pour une intégration multiprocesseur. Cependant la longueur de son pipeline présente une contrainte sévère pour s'étendre à de futures applications et une contrainte insurmontable pour exploiter le parallélisme de décodeur composant comme nous le constaterons dans la suite de ce chapitre

4.2 Vers un turbo-décodeur convolutif flexible

Comme nous l'avons vu au cours du chapitre 1, le concept de processeur dédié à une application (ASIP) permet d'atteindre un compromis idéal entre les performances ciblées par une architecture et la flexibilité désirée pour cette architecture. C'est donc naturellement au moyen d'un processeur dédié que nous avons choisi de mettre en oeuvre un turbo-décodeur flexible pour des codes convolutifs.

Cette section présente l'ensemble des décisions qui ont permis d'aboutir au processeur décrit dans la suite du chapitre. Dans un premier temps, les variants de l'application turbo-décodage sont analysés pour décider de la flexibilité recherchée et donner un cadre à la

structure programmable. Ensuite, nous décrivons les décisions de parallélisme, en accord avec les résultats du chapitre précédent, permettant à l'ASIP d'assurer de bonnes performances en terme de débit.

4.2.1 Assurer la flexibilité

4.2.1.1 Au niveau du codeur

Les spécifications d'une norme utilisant les turbocodes ne décrivent généralement que le turbocodeur, ses modes de fonctionnement et des contraintes comme la latence, car cela suffit pour présager de ces qualités de transmission. En analysant les codes existants et en dégageant les tendances futures, on peut donc être en mesure de caractériser les différents variants au sein du turbocodeur convolutif.

La concaténation

Actuellement les standards ont majoritairement recours à la concaténation parallèle de codes convolutifs, car elle offre une meilleure convergence que leur concaténation série tout en préservant des performances asymptotiques intéressantes. Cependant les très bonnes performances asymptotiques des concaténations séries et des concaténations hybrides [80] laissent présager d'un besoin de flexibilité dans cette direction. En étendant légèrement la flexibilité, le modèle générique pourrait supporter de nouveaux codes dérivés tel que le Flexicode [120], qui nécessite le support d'un code de parité pour faire varier le rendement de codage, ou tel que les turbocodes irréguliers [121], qui impliquent le support d'un accumulateur pour augmenter le degré de certain bits.

Les entrelaceurs, qui font la liaison entre les différents codes composants, sont d'autres variants de la concaténation, car les permutations associées changent avec les paramètres du code. Bien que la flexibilité sur l'entrelaceur s'avère indispensable au modèle générique, cette flexibilité a un coût conséquent car elle implique la réservation d'un espace de stockage pour la table de permutation, alors que les permutations des nouveaux codes peuvent être générées à la volée grâce à des jeux d'équations simples et beaucoup moins gourmand en silicium.

Les codes composants

Les briques de base du turbocode, que sont les codes composants, enrichissent également considérablement l'espace de flexibilité du turbocode convolutif. Ainsi un code convolutif est caractérisé par :

- Son nombre de bits d'entrée m, définissant alors un code m-binaire,
- Sa longueur de contrainte $\upsilon + 1$ qui fixe à υ la longueur du registre de codage,
- Son nombre de bits de redondance n,
- Ses n polynômes générateurs de redondances,
- Son éventuel polynôme générateur de la boucle de récursivité si le codeur est récursif,
- Son nombre de sortie compris entre n pour un code non-systématique et $n + m$ pour un code systématique, qui permet également de définir le rendement R

Tous ces paramètres caractéristiques peuvent être remplacés par la connaissance d'une section complète du treillis associé au code (voir figure 2.4). Ainsi le code composant est caractérisé par l'ensemble des $2^{\upsilon+m}$ transitions d'une section de treillis et on doit pouvoir associer à chacune des transitions un état de départ, d'arrivée, une entrée du codeur (symbole non codé) et une sortie de ce dernier (symbole codé). Du fait de la croissance exponentielle du nombre de transitions, une description exhaustive de tous les treillis peut s'avérer très

coûteuse surtout dans l'optique de solutions futures cherchant à accroître leur pouvoir de correction par la longueur du registre de codage et à améliorer la convergence par la taille du symbole d'entrée. Un maximum doit donc être envisagé pour ces deux valeurs. Dans les standards actuels et émergents, la complexité des codes composants est limitée à des codes double binaire 8 états ou des codes simple binaire 16 états, soit 32 transitions dans les deux cas.

Le code composant, tel qu'il vient d'être décrit, n'offre que peu de flexibilité sur le rendement. Or cette flexibilité est nécessaire pour gérer la qualité de service sur la couche physique et notamment le compromis entre débit et taux d'erreur. C'est pourquoi les standards ont recours à la technique du poinçonnage, qui consiste à ne transmettre qu'une partie des bits codés. Ainsi pour obtenir des rendements plus élevés que le code d'origine, un motif ou masque de poinçonnage est appliqué périodiquement sur la trame codée. C'est donc uniquement la période maximale de motifs, qui pourra limiter la flexibilité sur le rendement du turbocode.

Une autre caractéristique des codes convolutifs, inhérente à la taille finie des trames, est le choix du schéma de terminaison du treillis, aussi appelé fermeture du treillis [88]. Plusieurs méthodes peuvent être appelées à être supportées comme la fermeture classique (transmission de bits de fermeture pour forcer l'état final des codeurs convolutifs à 0 en fin de trame), la fermeture circulaire [122] (rend l'état de départ identique à l'état d'arrivée), la fermeture en roulette (fermeture circulaire appliquée aux sous-trames).

Finalement on peut aussi considérer la taille maximum de la trame comme un critère de flexibilité.

4.2.1.2 Au niveau du décodeur

L'espace de conception au niveau du décodeur, bien que très vaste dans la littérature, peut être cantonné à certains paramètres.

Premièrement, plusieurs algorithmes de décodage peuvent être utilisés : les algorithmes dérivés de l'algorithme de Viterbi comme le SOVA ou le bi-SOVA et les algorithmes dérivés de l'algorithme BCJR comme le MAP, le log-MAP ou le max-log-MAP. Dans la pratique, les algorithmes log-MAP et surtout max-log-MAP sont préférés pour leur complexité maîtrisée et leur bonne performance par rapport au décodage optimal. Le support de ces deux algorithmes n'implique qu'une légère flexibilité au niveau des opérateurs. Pour passer de l'algorithme max-log-MAP à l'algorithme log-MAP, il faut ajouter aux opérateurs maximum le terme de correction du logarithme Jacobien (voir équation 2.18) qui est stocké dans des tables addressées par la différence entre les opérandes [69].

Deuxièmement, le décodeur doit s'adapter aux tailles de bloc même les plus grandes sans pour autant gaspiller les ressources mémoires. Une gestion des tailles de bloc à la fois flexible et économique en ressource mémoire est possible avec la méthode du fenêtrage [94]. Cette méthode consiste à décoder séquentiellement des sous-blocs alors appelés fenêtres et a l'avantage de limiter la profondeur de la mémoire des métriques de récursions à la taille de la fenêtre et non à la taille de la trame entière. Etant l'expression séquentielle du parallélisme de sous-bloc étudié au chapitre 3, cette méthode souffre des mêmes contraintes quant à l'initialisation des fenêtres. Ainsi le choix de la taille de la fenêtre influe sur l'efficacité du décodage. En accord avec les résultats du chapitre 3 et notamment le tableau 3.2, nous avons retenu une taille de fenêtre de 128 bits pour assurer une efficacité de décodage convenable dans tous les contextes.

Troisièmement, le processus itératif peut nécessiter plusieurs paramètres de contrôle pour améliorer sa convergence. Par exemple, des fonctions de correction peuvent être appliquées aux informations extrinsèques ; l'arrêt du processus itératif peut être contrôlé soit de manière statique en fixant un nombre d'itérations maximum, soit de manière dynamique par un des nombreux critères d'arrêts existants ; les échanges d'informations extrinsèques peuvent être gérés plus efficacement avec un critère de limitation de la bande passante comme nous le verrons au chapitre 6.

Quatrièmement, pour réaliser les calculs de l'algorithme BCJR au niveau du décodeur, les transitions doivent être organisées de manière à faciliter différents regroupements de transitions. Ainsi les transitions doivent être regroupées par :

- état d'arrivée commun pour calculer une récursion aller,
- état de départ commun pour calculer une récursion retour,
- entrée commune du codeur pour calculer les informations sur les symboles non codés,
- sortie commune du codeur pour calculer les informations sur les symboles codés[1].

Réaliser ces regroupements de transitions qui dépendent du treillis impose un contrôle de la structure de communication véhiculant les données par multiplexage ou par réseau d'interconnexion générique (type réseau shuffle) vers leur opérateur. Ce contrôle est obtenu à partir d'un codage des transitions (au sens de leur représentation binaire associée). Le choix d'un codage approprié peut donc réduire le contrôle et en conséquence la complexité. Par définition du codeur, un codage est possible grâce au couple état de départ et entrée du codeur qui caractérise de manière unique une transition. Ainsi, ce codage des transitions facilite grandement les regroupements pour calculer les récursions retour et les informations sur les symboles non codés, mais en contrepartie les contrôles des regroupements pour les calculs des récursions aller et des symboles codés sont plus compliqués. Il est néanmoins possible d'alléger ce contrôle en restreignant la flexibilité avec certaines hypothèses. Par exemple, en supposant uniquement des concaténations parallèles, on peut s'abstraire des regroupements sur les symboles codés. De même, en supposant un code systématique récursif comme c'est le cas dans tous les standards, le treillis dispose de la propriété suivante : les transitions associées à un état de sortie ont forcément des entrées différentes. Il est alors possible de coder une transition grâce à l'état de sortie et l'entrée du codeur, ce qui facilite les calculs sur les récursions aller et sur les informations sur les entrées. Même si aucun codage n'est optimal pour l'ensemble des calculs, le choix d'un codage approprié peut permettre de minimiser le multiplexage lorsque les ressources sont partagées entre plusieurs calculs. Les regroupements sont alors réalisés plus rapidement que s'ils avaient été réalisés avec un réseau d'interconnexion générique. Pour ces raisons, nous avons fait le choix de nous limiter à des turbocodes concaténés en parallèle et dont les codes composants sont systématiques récursif.

4.2.1.3 Nos choix

Le tableau 4.1 résume l'ensemble des choix sur les flexibilités précédément abordées ainsi que les extensions possibles. Les extensions marquées (*) indiquent les extensions les plus exigeantes.

[1] Les informations extrinsèques sur les symboles codés ne sont pas calculées dans le cas des turbocodes concaténés en parallèle.

Flexibilité	Choix	Extension possible
concaténation	parallèle	série*, flexicode*, irrégulier*
entrelacement	tous	-
nb d'entrées du codeur	2	-
nb de sorties du codeur	4	-
nombre d'états	16	+ si besoin
polynôme	systématique récursif	non systématique ou non récursif*
Période de poinçonnage	16	+ si besoin
fermeture	toutes	-
taille de bloc	jusqu'à 65536 symboles	+ si besoin
algorithme	max-log-MAP	log-MAP
fenêtrage	jusqu'à 1024 fenêtres tout types d'initialisations	+ si besoin -
contrôle du processus itératif	itérations fixes correction linéaire des extrinsèques	critères d'arrêt
compaction de treillis	du simple au double binaire	-
architecture	ASIP	-

Tableau 4.1 — Paramètres de flexibilité et choix associés

4.2.2 Assurer les performances

Comme décrit dans le chapitre 3, le parallélisme au niveau des métriques BCJR est le seul niveau de parallélisme situé à l'intérieur d'un décodeur BCJR-SISO. Nous avons vu par ailleurs que ce niveau de parallélisme maximise le critère débit-complexité. Ainsi, un processeur voulant atteindre le haut débit se doit d'exploiter au mieux tout ce parallélisme.

D'après les décisions prises pour la flexibilité du codeur dans la section précédente, un degré de parallélisme égal à 32 permet d'exploiter totalement le parallélisme de transition de treillis (section 3.1.1.1) pour tous les standards. En outre, nous avons décidé d'organiser ce parallélisme de manière à optimiser les performances d'un code double binaire 8 états en veillant à conserver la flexibilité pour l'ensemble des autres codes.
Pour les turbocodes simple binaire disposant d'un degré de parallélisme de transition de treillis inférieur à 32, l'exploitation sous-optimale du parallélisme de transition de treillis peut être améliorée en recourant à la technique de compaction de treillis [92] [123]. Le principe de la compaction de treillis consiste à regrouper plusieurs sections consécutives du treillis initial pour ne former qu'une section du treillis compacté. Par exemple, il est possible de compacter le treillis d'un code convolutif simple binaire 8 états en un code convolutif double binaire 8 états, qui maximise l'utilisation des ressources mises en parallèle.
Pour les turbocodes disposant d'un degré de parallélisme de transition de treillis supérieur à 32, tels qu'ils pourraient apparaître dans de futurs standards, le treillis peut être décomposé en sous-sections ayant un degré de parallélisme 32. Ces sous-sections pourront ensuite être traités séquentiellement.

Au vue du parallélisme des calculs BCJR (section 3.1.1.2), nous avons choisi d'intégrer seulement deux unités alors que trois ou quatre unités auraient été nécessaires pour complètement paralléliser tous les schémas de décodage BCJR. De part ces bonnes performances, le schéma de décodage papillon a été retenu comme schéma de référence. Or, l'utilisation de quatre unités de calculs BCJR avec un schéma papillon conduit à une sous-

utilisation des unités, car les unités servant à produire les informations extrinsèques ne sont utilisées que la moitié du temps. En revanche, deux unités suffisent à maximiser l'activité des unités de calculs BCJR. Dans ce cas, les unités doivent traiter séquentiellement récursions et production d'informations extrinsèques dans la seconde moitié du schéma.

Les décisions qui viennent d'être décrites fixent les performances maximum de l'ASIP qui sont évaluées dans la suite de ce chapitre. Cependant ces performances risquent d'être insuffisantes pour l'ensemble des utilisations. Dans ces cas, une architecture multiprocesseur est indispensable et l'ASIP doit être conçu de manière à favoriser l'exploitation du parallélisme de décodeur SISO BCJR.

Concernant le parallélisme de sous-bloc, la principale contrainte réside dans la capacité de l'ASIP à initialiser les métriques de récursions. Or la gestion de multiples fenêtres impose déjà à l'ASIP des mécanismes d'initialisation. Ce dernier doit simplement intégrer en plus des interfaces lui permettant de recevoir des initialisations depuis l'extérieur, i.e. depuis d'autres ASIPs.

Pour implanter efficacement le parallélisme de décodeur composant au travers du décodage combiné, le temps de propagation t_p d'une information extrinsèque doit être le plus faible possible par rapport à sa période d'émission t_e. En se basant sur les simulations exposées par exemple sur la figure 3.17 (avec des trames de 752 symboles), on peut estimer l'accélération apportée par le décodage combiné à l'ASIP. Ainsi elle est représentée sur cette figure dans la zone où les degrés de parallélisme sont supérieurs à 12 ce qui correspond à des tailles de sous-bloc inférieure à 64 symboles (taille maximum de sous-bloc pour le processeur). Dans cette zone, l'accélération apportée par ce parallélisme s'affaiblit fortement lorsque le temps de propagation devient supérieur à trois temps d'émission. On tâchera donc de respecter l'inégalité suivante pour mettre en oeuvre efficacement le décodage combiné :

$$\frac{t_p}{t_e} \leq 3 \tag{4.1}$$

Dans le cas d'une mise en oeuvre sur un processeur, on peut écrire ce ratio en fonction du temps $t_{réseau}$ nécessaire pour mettre à jour une information extrinsèque entre deux processeurs au travers d'un réseau d'interconnexion, du nombre de cycles d'horloge $\#cycle\_ext$ du processeur pour exécuter le programme d'un calcul d'information extrinsèque, de la profondeur du pipeline c_{pipe_ext} entre l'étage d'entrée pour une information extrinsèque et son étage de sortie, et de la fréquence de fonctionnement du processeur f :

$$t_p = t_{réseau} + \frac{c_{pipe\_ext} - 1 + \#cycle\_ext}{f} \tag{4.2}$$

$$t_e = \frac{\#cycle\_ext}{f} \tag{4.3}$$

$$\frac{t_p}{t_e} = \frac{t_{réseau} \cdot f}{\#cycle\_ext} + \frac{c_{pipe\_ext} - 1}{\#cycle\_ext} + 1 \tag{4.4}$$

Réécrit ainsi l'équation de contrainte 4.1 impose :

$$\frac{c_{pipe\_ext} - 1}{\#cycle\_ext} < 2 \tag{4.5}$$

Vue comme une contrainte de conception, cela impose de concevoir un processeur avec un pipeline plutôt court (au moins pour les chemins de données des informations extrinsèques) tout en morcelant en plusieurs instructions le calcul d'une information extrinsèque. Dis autrement, cette décision réduit un peu les performances du processeur car elle impose une exploitation sous-optimale du parallélisme d'instruction (ou temporel) des calculs BCJR. Une exploitation optimale de ce parallélisme, comme c'est le cas dans l'ASIP proposé dans [119], conduirait à un ratio $\frac{t_p}{t_e}$ supérieur à 8 rendant peu efficace l'utilisation d'un parallélisme de décodeur composant. Outre un décodage combiné efficace, la contrainte de conception a aussi l'avantage d'imposer une architecture de processeur plus facilement extensible, puisqu'elle impose un pipeline court et donc moins contraignant sur le parallélisme temporel d'applications d'extension.

4.3 Architecture de l'ASIP

4.3.1 Vision globale

L'ASIP est principalement composé d'une partie opérative, d'une partie contrôle, d'interfaces de communication et d'une architecture mémoire recourant à des mémoires externes attachées (figure 4.1).

Pour schématiser le fonctionnement, on peut dire que sous les instructions de la partie contrôle, la partie opérative exécute des calculs BCJR au moyen de deux unités de calculs BCJR correspondant au sens aller et retour de traitement dans l'algorithme MAP. Chaque unité, dimensionnée pour traiter des fenêtres de 64 symboles, produit des métriques de récursion et les informations extrinsèques associés à un symbole. Les résultats (respectivement les opérandes) de la partie opérative sont distribués toujours sous la houlette de l'unité de contrôle vers (resp. depuis) les mémoires ou les entrée-sorties du processeur. La suite de cette section présentera une vision détaillée du rôle des différentes mémoires, des unités de traitement et de contrôle, puis finalement des mécanismes d'entrée-sorties.

4.3.2 Mémoires de l'ASIP

Le stockage des métriques de récursion, crées par une unité pour être utilisées par la suite dans l'autre unité, est réalisé dans huit mémoires d'échanges de 16 bits de large et de profondeur 32. C'est donc un total de 16 mémoires internes qui est alloué pour fournir la bande passante nécessaire au stockage des métriques de récursions. Une autre mémoire interne de 96 bits de large est utilisée pour stocker jusqu'à 256 decriptions de treillis différentes pour que le processeur se configure pour le turbocode correspondant. Evidemment, le processeur intègre également une mémoire contenant le programme qu'il doit éxecuter. Cette mémoire de 16 bits de large peut contenir jusqu'à 256 instructions.

D'autres mémoires externes au processeur sont nécessaires pour stocker les informations extrinsèques et les données entrantes issues du canal, i.e. aussi bien les informations systématiques que les redondances. La mémoire des données entrantes d'une largeur de 32 bits peut contenir jusqu'à quatre informations de canal (redondantes ou systématiques) de 8 bits chacune. De la même manière, la mémoire d'information extrinsèque a une largeur de 64 bits lui permettant de contenir jusqu'à quatre informations extrinsèques de 16 bits chacune, comme requis pour un code double binaire. Selon le nombre maximum de fenêtres choisi par le concepteur entre 1 et 1024, la profondeur de ces deux mémoires est dimensionnée entre 64

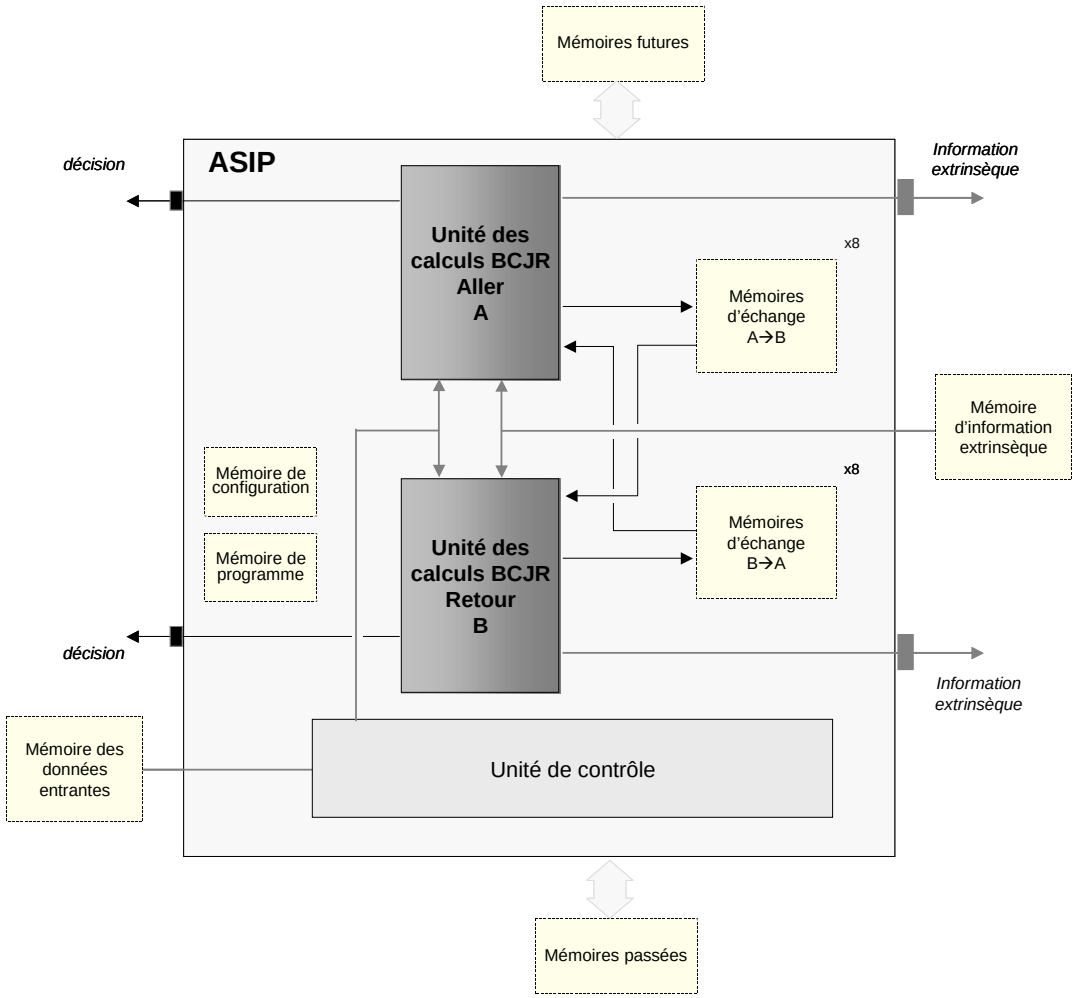

Figure 4.1 — Vision générale de l'ASIP

et 65536, ce qui permet au processeur de traiter jusqu'à 65536 symboles d'informations et donc de gérer à lui-seul tous les standards existants.

Les bancs de mémoires externes passé et futur sont quant à eux utilisés pour initialiser les valeurs des métriques de récursions au début et à la fin de chaque fenêtre d'un sous-bloc. Chaque banc est constitué de deux mémoires de 128 bits de large. L'une des mémoires sert à stocker les récursions aller tandis que la seconde est utilisée pour les récursions retour. La profondeur de ces mémoires est dimensionnée par le nombre maximum de fenêtre choisi, d'où un profondeur entre 1 et 1024. L'utilisation de ces mémoires pour mettre en oeuvre la méthode d'initialisation par passage de message sera décrite à la section 4.3.5.

4.3.3 Unité des calculs BCJR

Chaque unité de calcul BCJR repose sur une architecture SIMD (Single Instruction Multiple Data) afin d'exploiter au mieux le parallélisme de transition de treillis. Ainsi chacune incorpore 32 noeuds *addition* (un par transition) et 8 noeuds *maximum* (figure 4.2.a). Les 32 noeuds *addition* sont organisés en matrice de calcul 4x8. Dans cette matrice, un noeud *addition* correspond à une transition de la section traitée. La ligne du noeud désigne la décision

considérée sur le symbole tandis que la colonne du noeud désigne l'état final de la transition associée. La matrice est naturellement adaptée pour un code 8 états double binaire, qui présente 4 lignes (4 décisions possibles par symboles) et 8 colonnes (une par états).

Un code 16 états simple binaire peut également y être adapté en affectant les transitions se terminant dans les états allant de 0 à 7 sur les lignes 0 et 1 et les transitions se terminant dans les états allant de 8 à 15 sur les lignes 2 et 3.

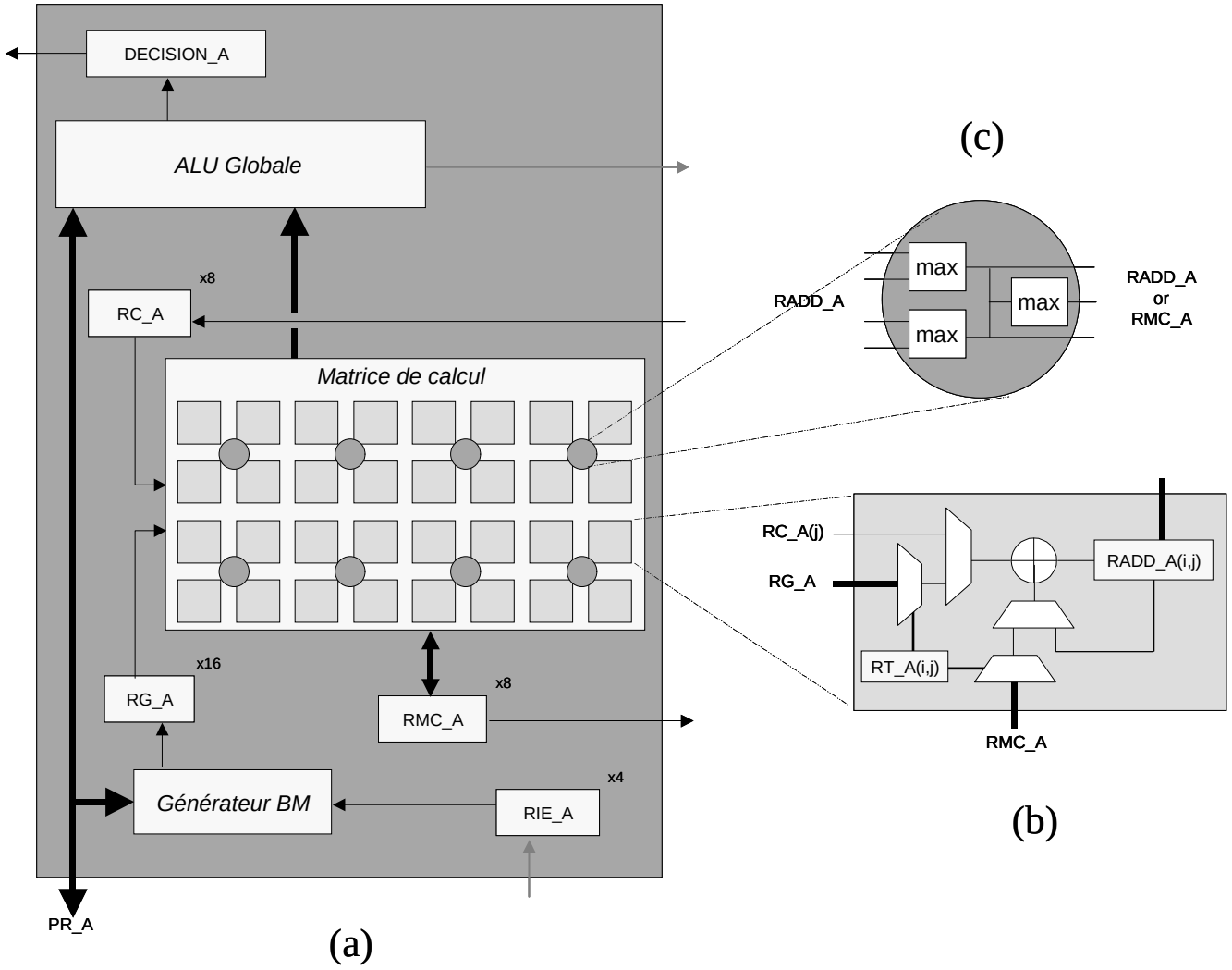

Figure 4.2 — Vision détaillée d'une unité de calcul BCJR

Un noeud *addition* (figure 4.2.b) contient un additionneur, un registre pour la configuration (RT) et un registre de sortie (RADD). Ce noeud peut réaliser l'addition requise dans une récursion entre une métrique d'état (provenant du banc de registre des métriques d'état RMC) et une métrique de branche (provenant du banc de registre des métriques de branche RG). Le noeud supporte également l'addition requise lors de la génération d'une information (extrinsèque ou décision) puisqu'il peut accumuler le résultat précédent avec la métrique d'état de l'autre récursion issue du banc de registre RC.

Les noeuds *maximum* (figure 4.2.c) sont partagés au sein de la matrice de calcul, de sorte que les opérations *max* s'exécutent en fonction des instructions de l'ASIP soit sur les

lignes soit sur les colonnes de la matrice de registres *RADD*. Un noeud *maximum* contient trois opérateurs *max* disposés en arbre. Cette disposition permet soit de trouver le maximum sur les quatres registres associés au noeud en utilisant les trois opérateurs, soit de trouver deux maximums, un maximum par paire de registres, en utilisant seulement le premier étage de l'arbre de comparaison. Les résultats de ces noeuds peuvent être stockés soit dans les premières lignes de la matrice *RADD*, soit dans les premières colonnes de la matrice *RADD*, ou dans le banc de registre *RMC* pour les métriques de récursion.

Une unité de calcul BCJR contient également :

- une unité arithmétique et logique globale (*GLOBAL ALU*), qui peut calculer entre autres les informations extrinsèques (envoyées vers une sortie du processeur) et les décisions dures du décodeur (stockées dans les registres *DECISION*).
- un générateur de métrique de branche (*BM*), qui calcule les métriques de branche à partir du banc de registre d'information extrinsèque (*RIE*) et des informations de canal disponible dans les registres du pipeline (*PR*). Le générateur *BM* peut supporter des motifs de poinçonnage cyclique ayant une longueur de cycle maximum de 16. La longueur du cycle est configurable grâce à un registre de 4 bits tandis que les motifs de poinçonnage associés à chacune des quatres informations du canal sont configurables par quatre registres de 16 bits. Lorsque l'un de ces registres de poinçonnage indique pour le symbole traité un poinçonnage (i.e. il y a un 0 à la position correspondant au symbole dans le cycle de poinçonnage), le générateur utilise la valeur neutre 0 au lieu de la valeur issue du registre de pipeline pour calculer les métriques de branche. Avec ce mécanisme, le générateur est en mesure de supporter des rendements allant de 1/2 à 1 pour un code double binaire et allant de 1/4 à 1 pour un code simple binaire. Par ailleurs, le générateur de métrique de branche applique également un facteur de correction aux informations extrinsèques *a priori* présentes dans le banc *RIE* grâce à un registre de 4 bits modélisant le facteur multiplicatif de la correction par pas de 0,125 de 0 à 1,875.

4.3.4 Unité de contrôle

Le contrôle de l'ASIP est réalisé conjointement par le pipeline et un ensemble de registres dédiés à différentes instructions de contrôle.

4.3.4.1 Stratégie de pipeline

La partie contrôle de l'ASIP est centrée autour d'un pipeline à six étages (Figure.4.3). Les deux premiers étages (FE, DC) servent classiquement à charger l'instruction depuis la mémoire de programme et à décoder cette instruction. Dans le troisième étage (OPF), les opérandes sont chargées :

- depuis la mémoire des données entrantes vers les registres du pipeline PR,
- et /ou depuis la mémoire d'information extrinsèque vers les registres RIE,
- et/ou depuis les mémoires passé et futur vers les registres RMC,
- et/ou depuis la mémoire de configuration vers les registres RT.

Ensuite, l'étage *BM* effectue les calculs de métriques de branches par l'intermédiaire des générateurs de métriques de branches. Après, l'étage EX exécute le reste des opérations BCJR à savoir les calculs de récursions ou d'informations. Finalement, l'étage *ST* réalise les stockages des résultats dans les mémoires ou sur des interfaces de sorties de l'ASIP.

La seule exception à cette stratégie de pipeline concerne les accès (en lecture et en écriture) aux mémoires d'échanges qui sont réalisés à l'étage d'exécution. Cette exception permet d'éviter des aléas de données lorsque les accès en écriture et en lecture se rapprochent (typiquement lorsque l'on arrive au milieu du papillon).

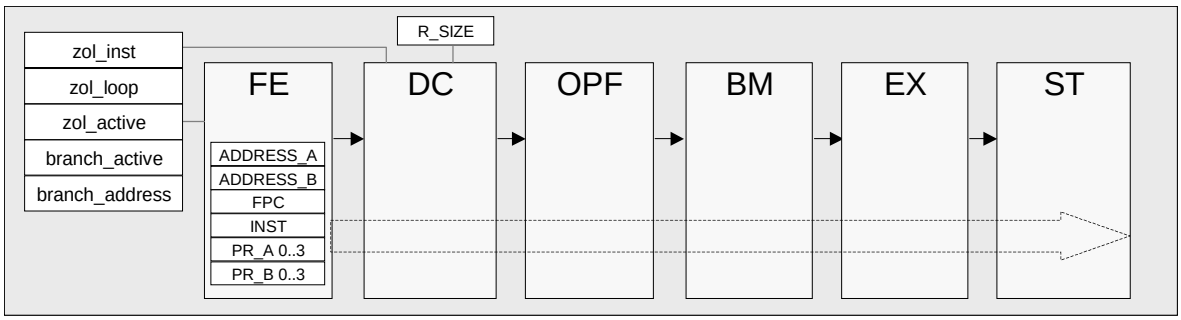

Figure 4.3 — Unité de contrôle de l'ASIP : pipeline et registres de contrôle

Notons finalement qu'avec ce pipeline, la décision d'un pipeline court (voir équation 4.5) est respectée. En effet, une information extrinsèque traverse le pipeline de OPF à ST donc en 4 étages ($c_{pipe\_ext} = 4$). Par ailleurs, comme le calcul d'une information nécessite au moins deux fois l'usage des unités de traitement (pour les calculs de récursions et puis d'informations), l'architecture nécessite au moins deux cycles pour créer une information extrinsèque ($\#cycle\_ext \geqslant 2$) ce qui suffit à respecter l'inégalité de contrainte.

4.3.4.2 Registres de contrôle

La partie contrôle du processeur est aussi constituée de registres de contrôle dédiés. Par exemple, la taille de la fenêtre est fixée dans le registre R_SIZE, et l'adresse de la section de treillis traitée dans l'unité de calcul BCJR A (ou B) est disponible dans le registre de pipeline ADDRESS_A (ou ADDRESS_B). Ces adresses, ainsi que la valeur du compteur de programme et le codage de l'instruction, sont pipelinées afin d'être accessible à tous les étages du pipeline. Comme le codage de ces adresses sur 6 bits limite l'espace d'adressage direct du processeur à 64 symboles, le processeur peut identifier la fenêtre calculée grâce à un registre de 10 bits ASIP_ID, ce qui permet à un processeur d'adresser jusqu'à 65536 symboles soit 1024 fenêtres de 64 symboles. Par ailleurs, pour respecter la politique d'initialisation des métriques de récursions pour chaque fenêtre, le processeur connaît également le nombre de fenêtres réellement traitées par le processeur grâce au registre de 10 bits R_SUB_BLOCK.

En plus de la gestion d'adresse, la partie contrôle fournit des mécanismes de branchement classique et un mécanisme de boucle sans pénalité ZOL (Zero Overhead Loop) complètement dédié au schéma papillon (section 3.1.1.2). Ce mécanisme ZOL est fortement lié à la génération d'adresses (figure 4.4), ce qui permet d'épurer le jeu d'instruction pour la partie contrôle et d'effectuer la génération d'adresses sans recourir à un étage de pipeline dédié.

Ce mécanisme marque trois champs d'instructions. Le premier champ d'instructions sera répété tant que l'adresse du symbole traitée dans l'unité A est plus petite que celle traitée dans l'unité B. Le deuxième champ d'instructions est facultatif car il permet de traiter le symbole au milieu des fenêtres de tailles impaires. Le dernier champ qui représente la deuxième boucle est répété tant que l'adresse de l'unité B est positive.

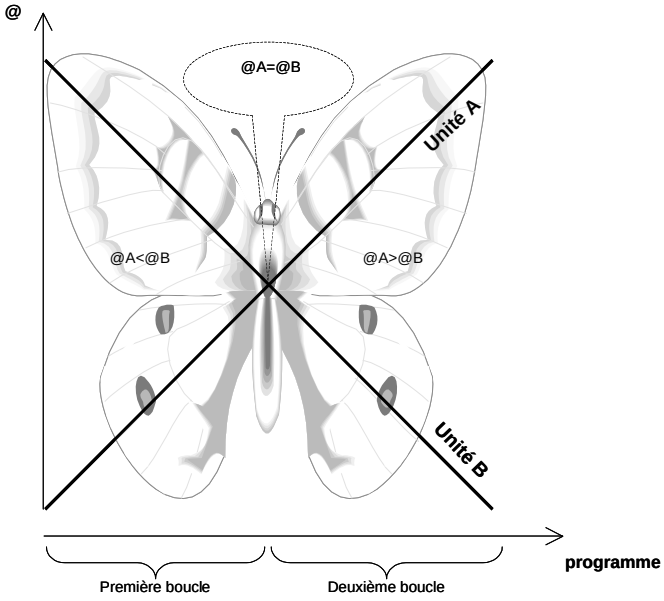

Figure 4.4 — Méchanisme ZOL dédié au schéma papillon

4.3.5 Entrées-sorties et initialisation du processeur

L'ASIP est muni d'interfaces permettant des échanges d'informations avec l'extérieur, i.e. un autre processeur ou une mémoire système.

Premièrement, il est doté d'une interface permettant d'exporter les décisions dures prises au fil de l'eau sur les symboles. Cette interface de 4 bits de large regroupe les décisions prises sur les symboles traités dans les deux unités de traitement.

Le processeur dispose également de deux interfaces pour échanger ses informations extrinsèques. Ces interfaces regroupent l'information sous la forme d'un paquet. Les paquets d'information extrinsèque peuvent contenir jusqu'à quatre informations de 16 bits sur le symbole courant et une en-tête. Cette en-tête contient l'adresse du symbole traité, c'est-à-dire la valeur du registre ASIP_IP et l'adresse locale. Au total, le paquet a une largeur de 80 bits.

Le processeur doit aussi pouvoir échanger les valeurs d'initialisation des métriques d'état avec d'autres processeurs ou avec lui-même. Le processus d'initialisation des métriques s'opère de la manière suivante :

- Au début de la première itération, les registres contenant les métriques d'états doivent être initialisés. Si aucune information sur la section de treillis n'est disponible, tous les registres sont remis à zéro pour avoir une distribution uniforme des probabilités des états sur la section de treillis. Si l'état de la section est connu, généralement l'état zéro, tous les registres sont remis à zéro sauf le registre correspondant à l'état qui est initialisé au quart de la dynamique du registre pour des raisons liées à l'arithmétique modulo utilisée dans le processeur.
- A l'issue de cette première étape, qu'elle est été réalisée pour une acquisition ou directement pour des récursions, les métriques d'état obtenues au début et à la fin de chaque fenêtre sont stockées dans un banc mémoire.
- Lors des itérations suivantes, les bancs mémoires contennant les métriques d'états des début et fin de fenêtre obtenues à l'itération précédente servent à initialiser les métriques

d'états et sont mis à jour à la fin de l'itération. De cette manière, les métriques d'état deviennent de plus en plus fiables au fil des itérations.

Pour la fenêtre i associée au processeur, les métriques d'initialisation seront lues pour les métriques aller à l'adresse i de la mémoire aller du banc passé et pour les métriques retour à l'adresse $i+1$ de la mémoire retour du banc passé. Les métriques raffinées par récursions sont ensuite stockées pour les métriques aller à l'adresse $i+1$ de la mémoire aller du banc passé et pour les métriques retour à l'adresse i de la mémoire retour du banc passé. Le banc mémoire futur est accédé uniquement pour les métriques d'états de la fin du sous-bloc, c'est-à-dire pour la dernière fenêtre associée au processeur. Pour cette section de treillis, les métriques retour seront lues à l'adresse 0 de la mémoire retour du banc futur et les métriques aller seront stockées à l'adresse 0 de la mémoire aller du banc futur.

4.4 Jeu d'instruction du processeur

Le jeu d'instruction de notre ASIP a été codé sur 16 bits. Dans une version basique, ce jeu d'instruction est composé d'une trentaine d'instructions, qui réalisent les différentes opérations de base de l'algorithme BCJR sans parallélisme d'instruction. Pour améliorer les performances, l'ASIP dispose également d'instructions compactant plusieurs opérations basiques, s'exécutant alors à des étages différents. Les sections suivantes ne détailleront que le jeu d'instruction de base, qui peut être décomposé en trois classes : contrôle, opérative et entrée-sortie.

4.4.1 Partie contrôle

Comme indiqué précédemment, l'instruction ZOL dédiée au schéma papillon fait répéter R_SIZE/2 fois chacune des deux boucles du schéma papillon. Trois marqueurs retiennent, relativement à l'adresse de l'instruction ZOL, l'adresse de fin de la première boucle, l'adresse de début de la seconde boucle et l'adresse de fin de la seconde boucle. Ils délimitent alors les trois champs d'instructions du mécanisme de contrôle ZOL (figure 4.4).

Les autres instructions de branchement (conditionnel ou inconditionnel) utilisent un adressage direct.

L'instruction SET_SIZE permet de fixer la taille de la fenêtre sur laquelle l'ASIP travaille à une maximum de 64 symboles. De même, les instructions SET_ASIP_ID et SET_SUBBLOCK_NB fixent le numéro de la fenêtre dans la trame traité (registre ASIP_ID) et le nombre de fenêtres traitées par cet ASIP (registre R_SUB_BLOCK).

4.4.2 Partie opérative

L'instruction d'addition peut être utilisée dans deux modes différents :

- un mode pour calculer des métriques de récursions : add m
- un mode pour calculer les informations (extrinsèques et décisions) : add i

Chaque nœud d'addition choisit ses opérandes en fonction du mode retenu et les registres de configuration de treillis (RT) et met le résultat du calcul dans le registre RADD du noeud.

De la même manière, les instructions max1 et max2 utilisent les deux mêmes modes de fonctionnement pour faire des réaliser soit les comparaison-sélections sur les lignes (max

m) soit sur les colonnes (max i). L'instruction max1 n'effectue qu'une comparaison-sélection (2 sorties par nœud max) tandis que l'instruction max2 en cascade deux (une sortie par nœud max). Ces instructions doivent être répétées autant que nécessaire pour obtenir soit les métriques de récursion soit l'information extrinsèque.

Le jeu d'instruction de base contient également une instruction DECISION pour générer les décisions dures sur les symboles traités dans les unités A et B.

4.4.3 Gestion des accès mémoires et des entrées-sorties

Pour être complet, le jeu d'instruction doit également fournir des instructions gérant les entrée-sorties du programme. Notre jeu d'instruction permet d'avoir des accès parallèles pour :

- charger les données entrantes du décodeur (LD_DATA),
- charger les initialisations des métriques de récursions (LD_REC)
- charger la configuration du treillis (LD_CONFIG),
- sauvegarder les métriques de récursions courantes pour de futurs initialisations (ST_REC),
- gérer les accès aux mémoires d'échange entre les deux unités de calcul BCJR (LD_CROSS, LD_ST, ST_2),
- envoyer les paquets d'information extrinsèque et les décisions dures (ST_EXT, DEC).

4.5 Exemples d'application : code double binaire huit états

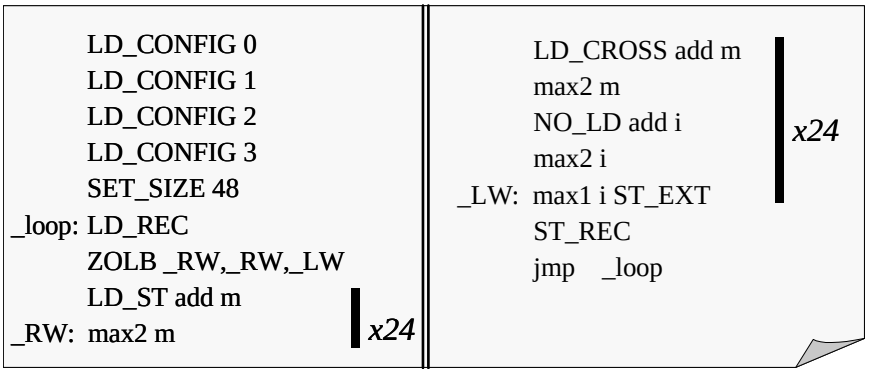

Figure 4.5 — Exemple de programme pour le code double binaire du standard Wimax

La figure 4.5 représente le code assembleur utilisé pour décoder un sous-bloc de 48 symboles encodé au standard WiMAX. Les quatre premières instructions (LD_CONFIG) configurent le processeur pour le treillis double binaire huit états correspondant à ce standard. La taille de bloc est ensuite fixée à 48, puis les registres de métriques d'états sont initialisés depuis les mémoires (LD_REC). Ensuite, les boucles papillon sont initialisées par l'instruction ZOLB et les marqueurs _RW et _LW. La première boucle, i.e. les deux instructions jusqu'à _RW, calcule les métriques de récursions : chargement des données entrantes et calcul des métriques de branches (LD_), sauvegarde des métriques d'états courante dans les mémoires d'échange (_ST), et finalement calcul des nouvelles métriques d'état (add m, puis max2 m).

Deux opérations max sont nécessaires puisque quatre transitions arrivent dans un état d'un code double binaire. La deuxième boucle, i.e. les cinq instructions de _RW à _LW, calcule ensuite des métriques de récursions comme dans la boucle précédente sauf qu'au lieu de sauvegarder les métriques d'état dans la mémoire d'échange (_ST) on récupère les métriques croisés dans la mémoire d'échanges (_CROSS). Ensuite les informations extrinsèques sont calculées grâce aux informations déjà chargées (NO_LOAD). Cette tâche nécessite 3 opérations max pour un code de 8 états. Le résultat est envoyé sous forme de paquet vers l'espace mémoire (ST_EXT). Avant de reboucler sur la prochaine itération, l'ASIP exporte ses métriques de récursions (ST_REC).

Si on regarde le temps d'exécution pour traiter N symboles, la première boucle du schéma papillon s'exécute en 2*N/2 cycles, tandis que la seconde nécessite 5*N/2 cycles. Soit un total d'environ 3,5 cycles par symbole ou 1,75 cycles par bit.

Notons que le temps d'émission d'un information extrinsèque $\#cycle\_ext$ vaut 5 , car il y a 5 instructions dans la deuxième boucle du papillon. Ce résultat, couplé au $c_{pipe\_ext} = 4$ de la section 4.3.4.1, permet de respecter la contrainte d'efficacité pour le décodage combiné (voir équation 4.5).

Ce résultat est valable également pour des codes simple binaire huit états auxquels on appliquerait une compaction de treillis, c'est-à-dire ayant un rendement de codage supérieur ou égal à 1/2.

4.6 Résultats d'implantation

4.6.1 Environnement de conception

Dans le cadre de cette thèse, nous avons utilisé la suite d'outils *Processor Designer* de *CoWare*. Cet outil permet à partir d'une description du processeur dans le langage de description d'architecture LISA de générer automatiquement des modèles du processeur (VHDL, Verilog, ou SystemC) pour la synthèse logique et l'intégration système d'une part et de fournir les outils de développement logiciel (simulateur, compilateur, assembleur, dévermineur and éditeur de liens) d'autres part.

A partir de la description RTL du processeur, les synthèses logiques ont été réalisées sur plusieurs cibles. Sur cible ASIC, le processeur a été synthétisé avec l'outil Design Compiler de Synopsys. Sur cible FPGA, il a été synthétisé avec l'outil XST de Xilinx.

4.6.2 Fréquence, surface et débits associés

4.6.2.1 Synthèse ASIC

Le tableau 4.2 résume les résultats de synthèse obtenus avec les librairies 90 et 180 nm de STmicroelectronics dans des conditions allant du pire cas (faible tension et température élevée) au cas standard (tension de fonctionnement et température courante). Outre la fréquence de fonctionnement du décodeur et son débit associé, le tableau fournit aussi la surface du processeur en mm^2 et en millier de portes logiques sans intégrer la surface des mémoires. La mémoire associée à un processeur représente environ 20 Kb représentant environ 0,18mm^2 en ST90nm.

Ce tableau indique également pour chaque librairie l'efficacité normalisée de la synthèse (i.e. le ratio débit-surface). On remarque ainsi que, pour un jeu de conditions donné, la

Librairie	Conditions (V ;T)	Fréquence (en MHz)	Débit associé (en Mbps)	Surface		Débit/Surface
				(en mm^2)	(en portes)	Normalisé
ST 180nm	(1,2V ;125°C)	83	47,4	0,844	68,7 K	0,38
	(1,55V ;85°C)	125	71,4	0,793	64,5 K	0,61
	(1,6V ;0°C)	100	57,1	0,689	56,1 K	0,56
		200	114,3	0,768	62,5 K	1
		340	194,3	0,891	72,5 K	1,47
		400	228,6	1,121	91,2 K	1,37
ST 90nm	(0,9V ;105°C)	250	142,9	0,212	48,2 K	0,52
		400	228,6	0,277	63,1 K	0,64
		475	271,4	0,297	67,5 K	0,71
	(1V ;25°C)	200	114,3	0,199	45,3 K	0,44
		500	285,7	0,221	50,3 K	1
		667	381,1	0,273	62,1 K	1,08
		950	542,9	0,367	83,5 K	1,14

Tableau 4.2 — Surfaces, fréquences et débits associés pour différentes synthèses en technologie ASIC

synthèse sera d'autant plus efficace que la fréquence obtenue sera élevée. Ainsi en condition standard avec une librairie 90nm, le processeur peut fonctionner avec une fréquence d'horloge de 950MHz pour décoder un code convolutif à 542,9 Mbps. Le chemin critique est situé dans l'étage de génération des métriques de branches.

La répartition des différentes unités sur l'ensemble de la surface est fournie par la table 4.3. L'espace est majoritairement occupé par les nœuds de calculs add et max, ainsi que par le générateur de métriques de branches et le banc de registres.

Bloc			**Surface** (en %)
Accès mémoires			1,71
Banc de registres			23,51
	FE		0,61
	FE_DC		0,60
	DC		0,26
	DC_OPF		0,48
	OPF		0,75
	OPF_BM		0,98
Pipeline	BM		8,12
	BM_EX		1,06
	EX	Noeud ALU	36,3
		Noeud max	15,58
		total	54,22
	EX_ST		0,65
	ST		0,38
		total	68,27

Tableau 4.3 — Répartition matérielle des ressources du processeur sans ses mémoires

4.6.2.2 Synthèse FPGA

Le tableau 4.4 récapitule les résultats de synthèse obtenus sur différentes cibles FPGA Xilinx. Les résultats montrent que le circuit a besoin d'un équivalent de 19000 LUTs à 4 entrées et de 17 blocs de RAM de 18kbits chacun.

Cible FPGA	**Fréquence** (en MHz)	**Débit associé** (en Mbps)	**Surface**	
			(en LUT[1])	(en %)
Virtex II Pro 30	110	62,8	18695	68
Virtex 4 FX140	117	66,8	19178	16
Virtex 4 L200	117	66,8	19178	11
Virtex 5 LX50	188	107,4	11133	41
Virtex 5 LX220	188	107,4	11133	8

Tableau 4.4 — Surfaces, fréquences et débits associés pour différentes synthèses FPGA

4.6.3 Comparaison

Le tableau 4.5 récapitule les performances obtenues pour divers implantations d'un décodeur log-MAP dans un turbo-décodeur UMTS. Les résultats sont généralement données pour des synthèses dans le pire cas d'une technologie. On peut observer que les solutions ASIP atteignent des débits proches des solutions dédiées grâce à un nombre de cycle par bit par SISO relativement bas. Parmi ces solutions ASIP, l'architecture proposée présente sensiblement de meilleures performances en utilisant la technique de compaction de treillis. Comparée à l'ASIP *FlexiTrep* [119], on obtient : 10% de performances en plus malgré une technologie moins avantageuse ; une surface équivalente environ identique ; et une longueur de pipeline plus courte permettant de supporter de nombreuses extensions sans les pénaliser et de supporter le décodage combiné.

Il convient également de noter que ce tableau comparatif n'intègre pas les surplus de calcul qui peuvent être engendrés par les structures de contrôle du programme (boucles, instructions multi-cycles, branchements), par les communications (temps d'accès aux mémoires partagées, échanges d'informations extrinsèques, initialisation des fenêtres) et par des limitations algorithmiques. Par exemple, le recours obligatoire dans certaines architectures à la méthode d'initialisation par acquisition engendre une complexité calculatoire supérieure (d'environ 15 % supérieure dans [124]). Grâce à ces structures de contrôle spécialisées, à ses mémoires locales et ses mémoires passé-futur dédiées à l'initialisation, les performances réelles de notre ASIP sont moins affectées par ces surplus de calcul que la plupart des processeurs existants. En conséquence, le ratio cycles/bit/SISO réel de notre processeur est très proche du ration idéal indiqué dans le tableau.

4.7 D'autres choix sont possibles...

Ce paragraphe décrit sommairement d'autres architectures répondant à d'autres choix de conception, notamment sur l'exploitation du parallélisme. Le point de comparaison avec le processeur proposé sera la synthèse ASIC 90nm dans les pires conditions (0,9V ;105°C) fournissant à une fréquence de 400MHz un débit de 228Mbps pour un équivalent portes logiques de 63Kportes.

[1]Les LUTs des FPGAs Xilinx ont 4 entrées jusqu'à la série Virtex4 et 6 entrées pour la série Virtex5.

4.7.1 L'extension performance

L'idée de cette architecture est d'étendre l'ASIP proposé pour exploiter au mieux le parallélisme d'instruction et augmenter les performances, mais en conservant la possibilité de faire du décodage combiné même un peu dégradé.

La méthode la plus efficace consiste à dédoubler l'étage *EX*. Le dédoublement de l'étage *EX* permet d'attribuer une matrice d'opérateurs ACS sur le premier étage d'exécution (c'est-à-dire les additionneurs du processeur actuel et le premier étage d'opérateur maximum) et d'attribuer une matrice d'opérateurs CSCS sur le second étage d'exécution (soit le deuxième étage de la matrice max actuel + 8 nouveaux comparateurs). Cette légère modification engendre un surcoût de complexité assez faible (environ 10%) mais permet de fusionner les opérations ACS et max dans un seul instruction (Figure 4.6). Ainsi pour un symbole, cet ASIP n'a besoin que d'un cycle pour générer les récursions et un autre cycle pour générer l'information extrinsèque. La première partie du papillon, qui ne génère que les récursions, a néanmoins besoins de deux instructions (dont une instruction sans opérations NOP) car l'instruction de récursion est réalisée sur deux étages donc en deux cycles.

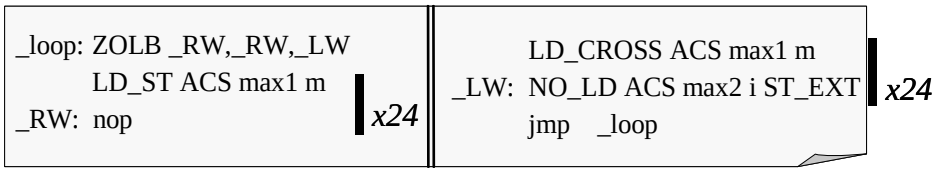

Figure 4.6 — Compaction du jeu d'instructions

Globalement, cet ASIP nécessite 2 cycles/symbole à une fréquence de 350MHz. La fréquence est un peu plus faible que dans l'ASIP original car le chemin critique est déplacé de l'étage *BM* à l'étage *EX1*. Le processeur peut donc atteindre un débit de 350Mbps soit près de 50% de performance en plus pour l'ASIP à 7 étages. Seulement avec $c_{pipe\_ext} = 5$ et $\#cycle\_ext = 2$, ce processeur inflige au décodage combiné un temps de propagation supérieur à 3 (c.f. équation 4.4), ce qui limite l'intérêt du décodage combiné (c.f. chapitre 3).

Architecture (type)	Cible	Fréquence (en MHz)	Cycles/bit/SISO	Débit / SISO (en Mbps)
XiRisc [112] (Processeur reconfigurable)	-	100	380	0,3
XTENSA [117] (processeur RISC extensible)	180 nm	133	9	14
Tigersharc TS201 [113] (DSP spécialisé)	-	600	8,25	72
Matériel dédié [106]	Virtex II	150	1	150
Décodeur Xilinx 3GPP [111]	Virtex5	310	1	310
ASIP FlexiTrep [119]	65 nm	400	2	200
ASIP proposé :				
sans compaction de treillis	90 nm	475	3,5	135
avec compaction de treillis	90 nm	475	1,75	271

Tableau 4.5 — Comparaison de différentes implantations de turbo-décodeur UMTS

4.7.2 La performance à tout prix

Pour rechercher la performance à tout prix, il faut maximiser le parallélisme des métriques BCJR et plus particulièrement le parallélisme des calculs BCJR. Il faut donc prévoir 3 unités de calculs BCJR organisées autour du schéma papillon aller. Un processeur, favorisant les codes doubles binaires, pourra ainsi réaliser les récursions en un cycle si on le dote d'une matrice d'opérateurs ACSCS dans un premier étage d'exécution (figure 4.7). Dans cette architecture, l'unité de génération des extrinsèques est partagée entre les étages EX1 et EX2, ce qui allonge la longueur du pipeline véhiculant l'information extrinsèque à $c_{pipe\_ext} = 5$. Avec seulement un cycle nécessaire par symbole ($\#cycle\_ext = 1$) et une fréquence estimée autour de 250MHz dans le premier étage d'exécution, cette architecture peut atteindre un débit de 500Mbps. En contrepartie, le coût en silicium est d'au moins 50% supérieur et le processeur empêche un décodage combiné efficace car le temps de propagation est d'au moins 5 temps d'émission (c.f. équation 4.4).

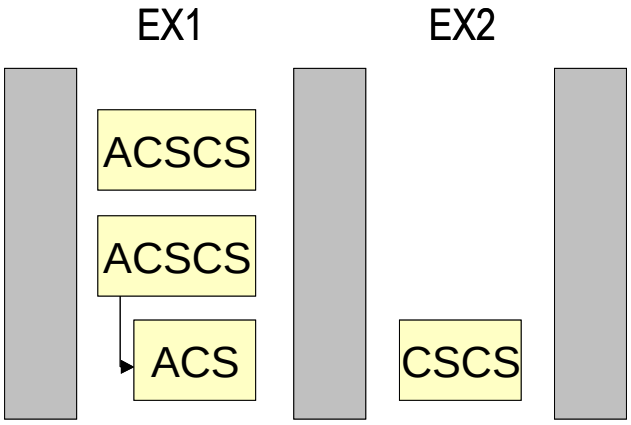

Figure 4.7 — Pipeline de l'ASIP "performance à tout prix"

4.7.3 Le meilleur rapport complexité-performance

Le principal problème de l'architecture précédente est la longueur du chemin critique due à la mise à jour de ces récursions en un seul cycle. Pour casser ce chemin critique et augmenter la fréquence du processeur, il faudrait pouvoir intercaler plusieurs instructions entre deux instructions pour un calcul récursif, de sorte que les registres de récursions puissent être mis à jour. En se basant sur cette idée, le présent processeur utilise 3 étages d'exécution pour réaliser l'opérateur ACSCS d'une récursion double binaire (plus un quatrième pour les calculs d'information) et séquentialise dans une instruction sur trois cycles les trois calculs BCJR (figure 4.8). L'instruction multi-cycle réalise dans l'ordre : la récursion retour, la recursion aller, puis la génération de l'information extrinsèque. Ainsi, les registres de la récursion retour seront à jour à l'issue de l'instruction pour recommencer une nouvelle instruction et les registres de la récursions aller seront à jour à l'issue du premier cycle de l'instruction suivante Notons qu'un retour des registres de pipeline (bypass en anglais) est utilisé à la sortie du premier étage d'exécution pour permettre durant le troisième cycle (celui de la génération d'information extrinsèque) de réutiliser les métriques obtenues lors du deuxième cycle (i.e. les sommes $\alpha + \gamma$ de chaque transition).

Ces modifications appliquées seules seraient insuffisantes pour monter en fréquence à cause

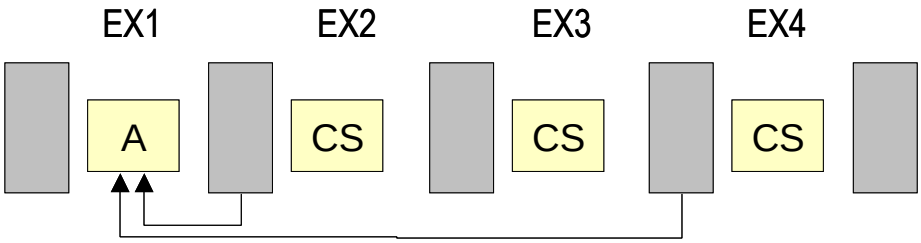

Figure 4.8 — Pipeline de l'ASIP sans parallélisme de calcul BCJR

de la génération des métriques de branches. C'est pourquoi il faut également casser l'étage *BM* en 3 étages *BM1*, *BM2* et *BM3* où les deux premiers étages génèrent la partie des métriques de branches provenant du canal tandis que le dernier étage y ajoute l'information extrinsèque *a priori*. De cette manière le paramètre c_{pipe_ext}, permettant d'évaluer l'architecture pour un décodage combiné efficace, vaut 9 puisqu'une information extrinsèque traverse alors les étages *OPF, BM1, BM2, BM3, EX1, EX2, EX3, EX4* et *ST*. Avec 3 cycles nécessaires pour générer une information extrinsèque ($\#cycle\_ext$), ce processeur inflige au décodage combiné un temps de propagation supérieur à 3,67 (c.f. équation 4.4). En revanche, ces avantages sont nombreux puisqu'on peut raisonnablement atteindre un fréquence de 600 MHz, soit un débit de 400Mbps, avec une réduction de complexité de l'ordre de 25% grâce à l'utilisation d'une unique unité de calcul BCJR.

4.8 Conclusion

Ce chapitre présente la conception d'un processeur dédié à l'algorithme BCJR tel qu'il est utilisé dans les turbocodes. La première étape de conception consiste à analyser les besoins en flexibilité et en performance, notamment grâce à l'étude du parallélisme du chapitre 3. Cette étape se conclue par des décisions de conception donnant une idée globale de l'architecture. Dans le cas présenté, l'architecture de l'ASIP met en oeuvre le parallélisme du premier niveau de la classification en tant que parallélisme d'instruction tandis que le parallélisme des deux autres niveaux est considéré comme du parallélisme de tâche. Ainsi l'ASIP intègre deux unités de récusions pouvant traiter chacune jusqu'à 32 transitions de manière pipelinée. Il intègre également des interfaces spécifiques pour gérer les échanges d'information entre les tâches, c'est-à-dire les valeurs d'initialisation par passage de message pour l'exploitation du parallélisme de sous-bloc et les informations extrinsèques pour l'exploitation du parallélisme de décodeur composant. Pour ce dernier, l'influence de la longueur du pipeline du processeur a été mise en exergue, impliquant le choix d'un pipeline court de 6 étages. Le pipeline ainsi que la stratégie de contrôle sont également présentés et complétés par une description du jeu d'instruction de l'ASIP.

Outre l'architecture de l'ASIP, le chapitre s'attache aussi à évaluer les performances et la complexité du processeur et montre, par comparaison avec des implantations existantes, la pertinence de la solution proposée, puisque le processeur présente des performances proches des solutions complètement dédiées, tout en préservant une flexibilité lui assurant le support de l'ensemble des standards existants. En guise de perspectives et en cherchant d'autres compromis performance-architecture, le chapitre présente d'autres solutions architecturales libérées de la contrainte de pipeline court imposée pour maximiser les performances du décodage combiné.

La prise en compte de contraintes sur la conception des interfaces de communication entre tâches font de l'ASIP décrit dans ce chapitre un candidat idéal pour son intégration dans une plate-forme multiprocesseur flexible et performante. C'est pourquoi ce processeur sera la base constituante de la plate-forme multiprocesseur présenté dans le prochain chapitre.

CHAPITRE

5 Plate-forme multiprocesseur pour turbo-décodage haut-débit

POUR atteindre des très haut-débits tout en conservant la flexibilité, le recours à une architecture multiprocesseur est impératif. Seul, un processeur, même extrêmement spécialisé à une application comme celui présenté au chapitre précédent, voit son débit limité car il n'exploite que le parallélisme d'une tâche à la fois. L'étude sur le parallélisme des turbocodes convolutifs développée au chapitre 3 a révélée un fort intérêt pour le parallélisme de tâches lorsque les tâches sont des décodeurs SISO BCJR. Ainsi l'utilisation de ce parallélisme de tâche s'impose naturellement pour constituer une plate-forme multiprocesseur.

Ce chapitre présente une plate-forme multi-ASIP de sa description jusqu'à son prototypage sur une carte d'émulation. Dans un premier temps, la plate-forme multiprocesseur est décrite au travers des choix de composants, d'organisation de ces composants, en insistant plus particulièrement sur les réseaux de communication et la gestion des mémoires. Les performances de cette plate-forme sont ensuite discutées autour des métriques débit et surface et comparées aux architectures existantes. Finalement, le chapitre se termine sur la présentation du flot de prototypage qui a permis de mettre en oeuvre cette plate-forme sur une carte à base de circuits FPGA et d'en valider le fonctionnement.

5.1 Organisation de la plate-forme multiprocesseurs

Selon la classification établie dans le chapitre 3, pour paralléliser un turbocode convolutif, il est possible d'exploiter le parallélisme des métriques BCJR, le parallélisme de décodeur BCJR-SISO et le parallélisme de turbo décodeur. Au vue de l'étude d'efficacité menée au chapitre 3, le parallélisme de turbo-décodeur n'a pas été retenu. Pour exploiter au mieux le parallélisme des métriques BCJR, nous proposons d'utiliser l'ASIP présenté dans le chapitre 4. Les travaux présentés dans ce cinquième chapitre exploitent uniquement le parallélisme de décodeur BCJR-SISO, c'est-à-dire le parallélisme de sous-bloc et le parallélisme de décodeur composant, par la proposition d'une plate-forme multi-ASIP. Dans ce contexte, il convient maintenant de s'interroger sur la gestion des ressources au sein de la plate-forme, notamment sur comment répartir les tâches (i.e. les décodeurs BCJR-SISO) sur les ASIPs, comment les faire communiquer entre elles et comment organiser les plans mémoires.

5.1.1 Répartition des décodeurs BCJR-SISO sur les processeurs

Le décodage multiprocesseur proposé repose à la fois sur le parallélisme de sous-bloc et sur le parallélisme de décodeur composant. Ainsi, on pourra identifier un décodeur BCJR-SISO grâce au couple (n,m) identifiant le $n^{ième}$ sous-bloc extrait du bloc à décoder par le $m^{ième}$ code composant. Comme, dans la pratique, les codes convolutifs n'utilisent que deux décodeurs composants, ce choix de parallélisme nécessite $2P$ décodeurs BCJR-SISO avec P le degré de parallélisme de sous-bloc. Dans notre plate-forme, chacune de ces $2P$ tâches est affectée à un ASIP distinct. Ainsi comme le montre l'illustration générale de la plate-forme (figure 5.1), P ASIPs sont regroupés pour traiter chaque décodeur composant.

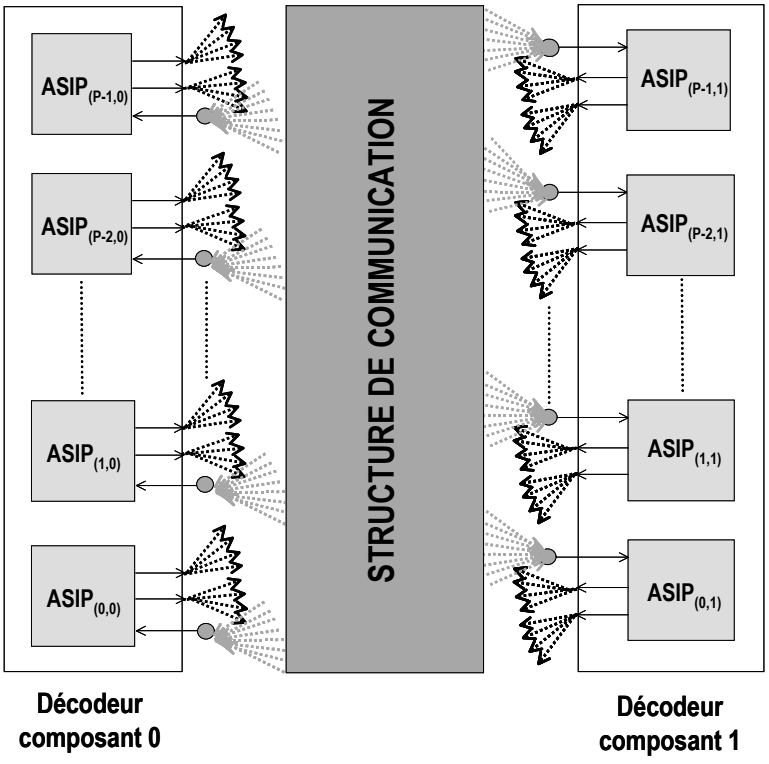

Figure 5.1 — Vision générale de la plate-forme multi-ASIP

5.1.2 Structures de communications

Au delà de l'intégration des ASIPs, cette figure montre également le besoin de structures de communication pour gérer les échanges importants de données entre les processeurs. Deux types d'échanges y sont mis en évidence : les communications entre les ASIPs travaillant sur des décodeurs composants différents et les communications entre les ASIPs travaillant sur un même décodeur composant.

Les communications entre les ASIPs travaillant sur des décodeurs composants différents représentent les échanges d'informations extrinsèques entre les décodeurs composants. En conséquence, la structure de communication associée doit gérer la fonction d'entrelacement du turbocode, c'est-à-dire d'un point de vue matériel le routage des informations extrinsèques générées par les ASIPs d'un décodeur composant vers les mémoires d'information extrinsèque des ASIPs de l'autre décodeur composant. Cet échange d'information peut donc être décomposé en deux flux uni-directionnel allant du décodeur composant 0 au décodeur composant 1 et vice versa. Du fait du parallélisme de décodeur composant (ou plus précisément du décodage combiné), ces deux flux co-habitent dans le réseau. Un réseau composé de deux sous-réseaux uni-directionnels est dans ces conditions plus appropriée qu'un réseau bi-directionnel.

Afin de garantir le bon fonctionnement du processus itératif, les échanges d'information doivent être réalisés dans un temps le plus court possible. Comme on l'a vu au chapitre 3, la convergence du processus itératif avec le décodage combiné est d'autant plus rapide que le rapport temps de propagation de l'information extrinsèque t_p sur temps entre deux émissions successives d'informations extrinsèques t_e est faible. Les simulations (figure 3.17) nous ont montrées que l'efficacité du décodage combiné restait raisonnable pour :

$$\frac{t_p}{t_e} < 3 \tag{5.1}$$

Or, au chapitre 4, on a réécrit ce ratio en fonction du nombre de cycles nécessaire pour émettre une information extrinsèque ($\#cycle\_ext$), la fréquence de fonctionnement du décodeur BCJR-SISO f, la profondeur de pipeline entre l'entrée et la sortie d'une information extrinsèque (c_{pipe_ext}) et le temps de traversé du réseau ($t_{réseau}$) :

$$\frac{t_p}{t_e} = \frac{t_{réseau}.f}{\#cycle\_ext} + \frac{c_{pipe\_ext} - 1}{\#cycle\_ext} + 1 \tag{5.2}$$

Comme l'ASIP utilisé a un $\#cycle\_ext$ de 5 et un c_{pipe_ext} de 4 (cf. section 4.5), on peut en déduire :

$$t_{réseau} < \frac{7}{f} \tag{5.3}$$

Il faudra donc que le temps moyen de propagation de la structure de communication gérant les informations extrinsèques respectent cette contrainte.

Les communications entre les ASIPs travaillant sur un même décodeur composant proviennent des initialisations de métriques d'état d'un sous-bloc. D'après les résultats du chapitre 3 (section 3), un turbo-décodeur utilisant le parallélisme de sous-bloc présente de meilleurs résultats avec la technique d'initialisation par passage de message. Donc, au sein d'un même décodeur composant, chaque ASIP doit pouvoir communiquer avec les deux ASIPs traitant des sous-blocs voisins du sien.

L'ensemble de ces communications peut être pris en charge par l'ASIP qui intègre les interfaces nécessaires aux échanges d'informations extrinsèques et aux échanges de métriques d'état.

5.1.3 Gestion de la mémoire

Par nature, le turbo-décodeur doit traiter des données en provenance du démodulateur rassemblées sous la forme d'un paquet (ou trame). Or par conception, notre ASIP utilise une mémoire dédiée pour les données entrantes. Son utilisation dans un contexte multiprocesseur impose ainsi une phase de distribution du paquet de données vers différentes mémoires distribuées. Avec cette approche distribuée, chaque ASIP se voit attribuer un sous-bloc de taille identique (au symbole près) à celle des autres sous-blocs. Au niveau du processeur, cette approche ne présente que des avantages : accès rapide aux données (mémoire locale) et en temps constant puisqu'il est le seul lecteur. En contrepartie, les difficultés rencontrées avec cette approche sont au niveau système.

Premièrement, la contrainte de mémoire entrante dédiée à l'ASIP couplée avec l'exécution parallèle des décodeurs composants implique la duplication de la partie de ces mémoires entrantes correspondant aux informations systématiques. En effet, pour traiter un symbole, chaque décodeur composant utilise l'information systématique correspondant au symbole ainsi que l'information de redondance associée au décodeur. Les deux décodeurs composants travaillant en parallèle, il existe donc deux ASIPs différents qui traiteront de la partie systématique de cette information. La manière la plus simple pour traiter ce problème consiste à dupliquer ces informations systématiques dans les mémoires concernées durant la phase de distribution. Le surcoût engendré sur la mémoire de donnée entrante est de 33% puisqu'un symbole nécessite alors 8 informations (4 données entrantes sont traitées par ASIP) au lieu de 6 informations pour un turbo-décodage optimisé d'un code double binaire de rendement un tiers (2 informations systématiques plus 2 redondances par décodeur composant).

Bien que nous ne l'utiliserons pas par la suite, il est possible d'éviter cette duplication de mémoire en changeant un peu l'exécution du décodage combiné. L'idée, introduite dans [125], consiste à ne fournir les informations systématiques qu'à l'un des décodeurs composants. Ce décodeur passe ensuite les informations systématiques à l'autre décodeur composant en l'ajoutant à l'information extrinsèque. Pour le sens retour de l'itération, l'information est renvoyée privée de l'information entrante (et donc sans l'information systématique qu'il ne faut pas renvoyer au premier décodeur). Notons que pour pouvoir utiliser ce schéma en conservant les propriétés de convergence du décodage itératif, il est obligatoire de réaliser une première itération sans exploiter le parallélisme de décodeur composant pour permettre l'initialisation du deuxième décodeur composant avec les informations systématiques. Cette contrainte représente la perte d'une demi-itération dans la convergence du processus itératif.

La deuxième difficulté intervenant sur la mémoire des données entrantes provient de la gestion des entrée-sorties. Tel que le système a été décrit jusqu'à maintenant il ne dispose que d'une seule mémoire (distribuée) pour stocker le paquet entrant. Cela signifie que le chargement et la distribution d'un nouveau paquet est bloquant pour le décodage multiprocesseur et vice versa. Comme montré dans [125] et [126], cela peut fortement limiter les débits de décodage. Ainsi notre choix s'est orienté vers une solution avec deux plans mémoires afin d'effectuer simultanément le décodage et la gestion des entrées-sorties.

Si on considère maintenant les mémoires d'informations extrinsèques toujours selon le template de la figure 5.1, le décodage combiné impose l'utilisation de 2 bancs de P mémoires

d'informations extrinsèques, i.e. un banc par décodeur composant, alors qu'un seul banc suffit sans décodage combiné, car, dans ce cas, le banc est utilisé alternativement par les décodeurs composants.

En fait, il est également possible de n'utiliser qu'un seul banc mémoire pour échanger l'information extrinsèque dans le cas d'un décodage combiné à condition de gérer les conflits de consistance pouvant intervenir (voir figure 3.15). En effet, pour tout symbole n'ayant pas de conflit de consistance, un décodeur composant est assuré de trouver dans le banc mémoire l'information inscrite par l'autre décodeur composant, ce qui assure l'échange itératif d'information entre les décodeurs composants et en conséquence l'intégrité du résultat. En revanche, en cas de conflit de consistance, un des décodeurs composants accède dans le banc mémoire à l'information extrinsèque qu'il avait précédemment stocké. Donc l'échange itératif est rompu. Pour conserver l'échange itératif et préserver l'intégrité du résultat, une mémoire secondaire est nécessaire pour doubler la portion de mémoire principale associée à des symboles sujets à un conflit de consistance. En cas de conflit, l'un des décodeurs composants accède à la mémoire secondaire et l'autre à la mémoire principale. De la sorte, seule la portion de mémoire présentant des conflits de consistance, petite devant la taille du banc mémoire d'information extrinsèque, a besoin d'être dupliqué, ce qui réduit grandement l'encombrement de cette mémoire. Un tel mécanisme a déjà été présenté dans [127] pour éviter les conflits de consistance. L'adressage de la mémoire secondaire y est générée par l'intermédiaire d'une table de hachage. Dans notre cas, la taille de la mémoire secondaire dépend du nombre de conflits de consistance et le choix de la fonction de hachage dépendra de leurs positions. Selon le chapitre 3, nombre et positions des conflits de consistance dépendent de l'entrelaceur du turbocode et des différents choix de parallélisme. Cela rend cette méthode très pertinente pour une implantation complètement dédiée. En revanche, pour une implantation partiellement flexible, il faudrait supporter différentes fonctions de hachage (une par cas d'usage) avec une mémoire secondaire taillée pour le cas d'usage ayant le plus de conflits de consistance. Puisque nous cherchons à disposer d'une flexibilité totale d'implantation, ce mécanisme n'a pas été utilisé et nous avons utilisé deux bancs mémoires séparés pour l'information extrinsèque.

5.2 Structures de communications

Pour illustrer l'implantation des structures de communications, la figure 5.2 présente une architecture regroupant 4 ASIPs, deux par décodeurs composants. Cette figure montre trois structures de communications différentes : l'interface d'entrée-sortie, le réseau de métrique d'état et le réseau d'information extrinsèque.

5.2.1 Gestion des entrées-sorties

Le réseau de gestion des entrée-sorties a comme son nom l'indique deux missions :

- permettre de faire la liaison entre le flux de données entrant dans le turbo-décodeur et les mémoires de données entrantes de chaque ASIP, comme décrit dans la section précédente.
- récupérer les décisions dures du décodeur composant 1, les réordonner et les transmettre vers la sortie du turbo-décodeur.

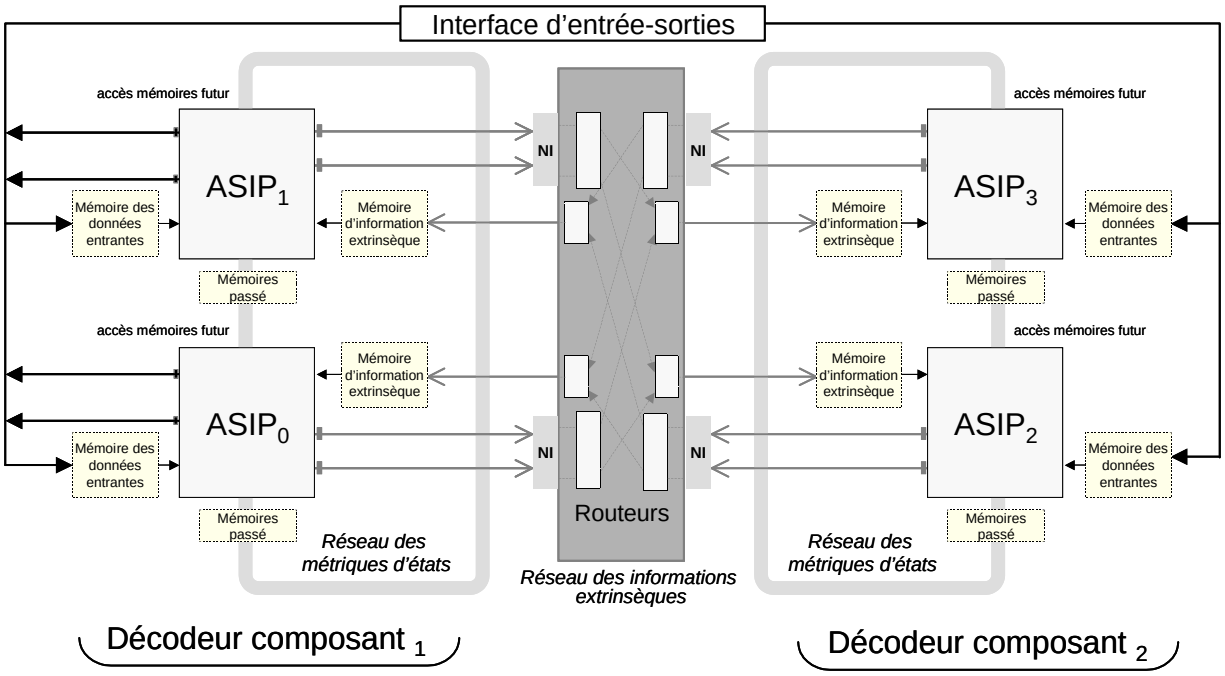

Figure 5.2 — Architecture de turbo-décodage multiprocesseur à 4 ASIPs

5.2.2 Gestion des métriques d'états

Le réseau des métriques d'états permet les échanges d'informations entre des ASIPs voisins au sein d'un décodeur composant. Grâce à ces échanges, un ASIP peut initialiser avec la technique du passage de message les métriques d'état au début et à la fin du sous-bloc traité par cet ASIP. Comme cela fût décrit dans le chapitre 4, les ASIPs accèdent à ces valeurs d'initialisations à l'adresse 0 des mémoires passé (physiquement associée à l'ASIP) et futur (physiquement associée à l'ASIP voisin). Si le mécanisme de fenêtrage n'est pas utilisé, ces mémoires peuvent même être remplacée par des registres tampons et dans ce cas le réseau des métriques d'états se résume à un jeu de registres tampons entre processeurs voisins.

5.2.3 Gestion des informations extrinsèques

Le réseau des informations extrinsèques se base sur un assemblage de routeurs pour véhiculer les informations extrinsèques d'un décodeur composant à l'autre (cet assemblage peut en fait être séparé en deux sous-ensemble puisque deux flux uni-directionnels distincts existent). Plus précisément, l'information est transportée d'un port d'émission d'un ASIP appartenant à un décodeur composant vers une mémoire d'information extrinsèque rattachée à un ASIP de l'autre décodeur composant.

De part le support du schéma papillon, chaque ASIP peut envoyer sur le réseau jusqu'à deux paquets d'informations extrinsèques par instruction d'émission exécutée. Chaque paquet contient une en-tête contenant l'adresse du symbole (supposé double binaire) dans l'espace d'adressage du décodeur composant et un corps contenant les quatre informations extrinsèques correspondant aux quatre valeurs possibles pour un symbole double binaire. Un paquet est ensuite transmis de l'ASIP à une interface réseau (NI) associée pour réaliser la fonction d'entrelacement. Chaque NI regénère alors, à partir d'une table stockant la fonction d'en-

trelacement utilisée, une nouvelle en-tête contenant l'information de routage vers la mémoire de destination et également l'adresse dans cette mémoire. Afin de faciliter ultérieurement la reconfiguration de la fonction d'entrelacement, une version modifiée de l'ASIP a également été développée pour intégrer ces interfaces réseaux.

Le réseau en lui-même est basé sur des routeurs, qui intègrent des mécanismes de tamponnage et qui peuvent supporter jusqu'à deux ports d'entrée et deux ports de sortie. La figure 5.2 représente une topologie d'assemblage possible pour une architecture à quatre ASIPs. Cette topologie respecte aisément l'équation 5.3, puisque la traversée du réseau ne nécessite que le passage de deux routeurs et d'une interface réseau. Pour des architectures intégrant plus de processeurs, le respect de cet équation devient plus difficile, car le temps moyen de traversée d'un réseau croît au minimum logarithmiquement avec son nombre de ports d'entrée N [19]. Par exemple, pour des réseaux basés sur des routeurs deux entrées deux sorties, le temps moyen de traversée d'un réseau mesh évolue en $o(\sqrt{N-1})$ et celui d'un réseau en anneau bi-directionel évolue en o(N/4). De fait, avec ces réseaux, l'extensibilité de la plate-forme multiprocesseur serait limitée à de faible N afin de respecter l'équation 5.3. Pour bénéficier d'une extensibilité maximale, le réseau d'information extrinsèque doit donc disposer d'une topologie offrant un temps moyen de traversée minimum, c'est-à-dire logarithmique. De nombreuses topologies le permettent notamment comme les topologies shuffle-exchange ou les topologies multi-étages.

Nous avons retenues ces dernières et notamment les topologies Benĕs et butterfly [128], qui s'avèrent très appropriées pour gérer des flux unidirectionnels car elles offrent une bande passante constante au cours des échanges. La topologie butterfly, illustrée figure 5.3, offre un temps de traversée correspondant au passage de l'interface réseau et de lg(N) routeurs si aucune collision n'intervient dans le réseau. En cas de collision, c'est-à-dire les deux données traitées par le routeur veulent emprunter la même sortie, l'une est tamponnée dans le routeur tandis que l'autre est transférée vers le routeur suivant. Le temps de traversée moyen, i.e. tenant compte des retards due aux collisions, a été observé proche de lg(N)+1 lorsque les paquets injectés ciblent avec une probabilité uniforme les mémoires de destinations, comme c'est le cas avec les entrelaceurs utilisés dans les turbocodes.

Les sorties du réseau des informations extrinsèques sont directement reliées aux mémoires d'informations extrinsèques. Chaque sortie dispose en fait de deux ports pour accéder à la mémoire en écriture.

5.3 Résultats de synthèse

Dans cette section, la surface et le débit de l'architecture multi-ASIP sont évalués afin de vérifier l'efficacité globale de la plate-forme. **Les synthèses logiques ont été réalisées avec la technologie ST 90nm dans les pires conditions (0,9V ; 105°C)**. Dans ces conditions, nous avons considéré que l'ASIP fonctionne à une fréquence de 400 MHz et occupe 0,277 mm^2. La plate-forme étudiée utilise le réseau Butterfly présenté dans la section précédente.

5.3.1 Surface

Les résultats sont présentés pour une plate-forme pouvant supporter des trames turbocodées ayant jusqu'à 2048 bits ou 1024 symboles double binaire, ce qui suffit par exemple à supporter toutes les variantes du turbocode du standard DVB-RCS. Cette contrainte nous permet de fixer la taille maximum des mémoires. D'autre part, comme un ASIP travaille sur

une fenêtre de 64 symboles, cette contrainte fixe également le nombre de fenêtres associées à chaque ASIP.

Le tableau 5.1 détaille, pour des plate-formes allant du simple processeur à une architecture à 32 ASIPs, les surfaces occupées par l'ensemble des ASIPs, par le réseau d'information extrinsèque Butterfly, par l'ensemble des mémoires (c'est-à-dire les mémoires d'instructions, d'informations extrinsèques, d'échanges de métriques interne et externe, de données entrantes incluant le tampon d'entrée, de configuration et les mémoires d'entrelacement contenant l'information de routage dans le réseau Butterfly) et finalement par la plate-forme dans son ensemble.

Nombre d'ASIP (# fenêtres par ASIP)	1 (16)	4 (8)	8(4)	16(2)	32(1)
Surface des ASIPs (en mm^2)	0,28	1,11	2,21	4,42	8,85
Nombre de routeurs dans le réseau Butterfly	-	8	24	64	160
Surface du réseau Butterfly (en mm^2)	-	0,10	0,39	1,16	3,09
Surface des mémoires (en mm^2)	0,94	1,65	2,16	3,04	4,61
Surface totale de la plate-forme multi-ASIP (en mm^2)	1,22	2,86	4,76	8,63	16,56

Tableau 5.1 — Surfaces obtenues pour des plate-formes multi-ASIPs de turbo-décodage gérant des tailles de trames jusqu'à 2048 bits d'information

Dans ce tableau, on retrouve l'évolution de la complexité du réseau Butterfly en o(N.lgN). On constate également que la surface des mémoires varie presque linéairement en fonction du

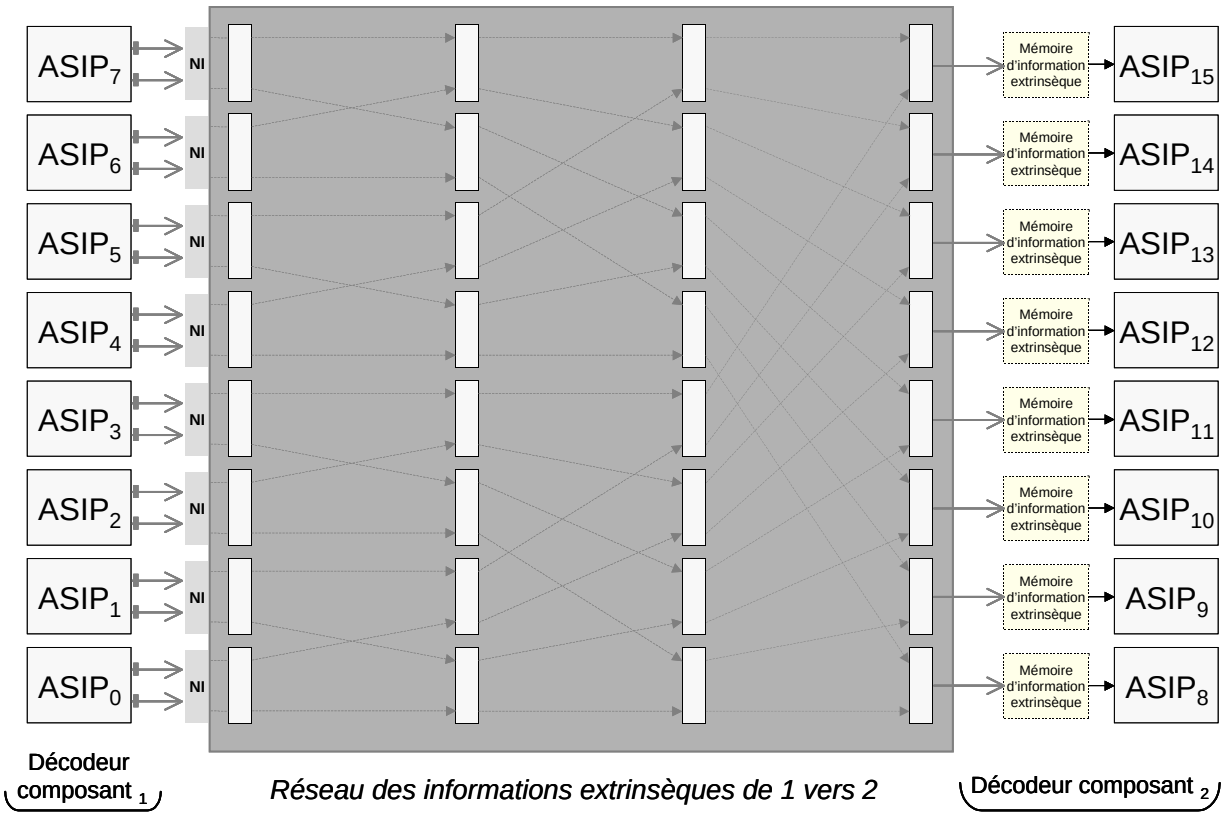

Figure 5.3 — Moitié du réseau des informations extrinsèques organisé selon la topologie butterfly

nombre d'ASIP de la plate-forme. Ceci s'explique en partie par la duplication des mémoires internes à l'ASIP (c'est-à-dire les mémoires d'instructions, d'échanges de métriques interne et de configuration) qui occupent une surface de 0,076 mm^2 par ASIP. Les autres mémoires conservent la même capacité de stockage d'une configuration à l'autre[1], mais l'évolution de leur surface provient d'un fractionnement différent du plan mémoire. D'une manière générale, l'utilisation de grandes mémoires (configurations avec peu de ASIPs) réduit la logique d'adressage de la mémoire et donc la surface.

En observant l'évolution de la surface de l'ensemble de la plate-forme, on constate une évolution plutôt linéaire en fonction du nombre d'ASIP. En revanche, en terme de proportions, on observe de grandes disparités entre la configuration monoprocesseur où la mémoire constitue à elle seule 80% de la complexité et la configuration à 32 ASIPs où les mémoires n'occupent plus qu'un quart de la surface totale, qui est occupée sur plus de sa moitié par les ASIPs.

5.3.2 Débit

L'évaluation des débits est réalisée sur une trame MPEG de 1504 bits codée par le code double binaire du standard DVB-RCS et décodée sur des fenêtres de 47 symboles. Par ailleurs, les résultats sont obtenues de manière à avoir des performances de correction, pour chaque configuration, similaire à celles d'une architecture n'utilisant pas de parallélisme de décodeur BCJR-SISO effectuant 5 itérations. Pour cela, le nombre d'itération est fixé selon les valeurs des efficacités de parallélisme de sous-bloc et de parallélisme de décodeur composant (voir chapitre 3). Noter également que l'efficacité de parallélisme de décodeur composant a été obtenue en tenant compte de la latence du réseau d'information extrinsèque dont elle est dépendante. Le débit d'information utile est calculé en fonction de la fréquence de fonctionnement f, du nombre de cycles par bit ($cycles$), du nombre d'itérations nécessaire pour obtenir les performances désirées (it) et également du nombre d'ASIP N.

$$Débit = \frac{f}{2.it.cycles}.N \tag{5.4}$$

Le tableau 5.2 récapitule l'ensemble des valeurs permettant d'aboutir à une estimation du débit pour les différentes configurations multi-ASIP déjà abordées. Le programme des ASIPs permet de décoder les trames à la cadence moyenne de 3,6 cycles par symbole par itération.

Nombre d'ASIP (# fenêtres par ASIP)	1 (16)	4 (8)	8(4)	16(2)	32(1)
Latence du réseau d'information extrinsèque (en cycles)	2	4	5	6	7
efficacité du décodage combiné	-	0,72	0,74	0,75	0,74
efficacité du parallélisme de sous-bloc	1	0,97	0,92	0,82	0,71
# itérations	5	7	7	8	9
débit d'information utile (en Mbps) @90nm	22	64	127	222	395

Tableau 5.2 — Débits obtenus pour des plate-formes multi-ASIPs de turbo-décodage décodant une trame de 1504 bits codée avec le standard DVB-RCS

En couplant la faible latence du réseau Butterfly avec la longueur maîtrisée c_{pipe_ext} de traversée de l'information extrinsèque de l'ASIP, on obtient le temps de propagation nécessaire

[1]Sauf la configuration avec un seul ASIP dont la capacité des mémoires d'information extrinsèque est de moitié inférieure en l'absence de décodage combiné.

pour mettre à jour une information extrinsèque. Ce temps de propagation est suffisamment court pour permettre un décodage combiné efficace. A partir de la figure 3.17, on peut alors estimer l'efficacité du décodage combiné au alentour de 0,75 pour toutes les configurations étudiées ce qui se traduit par l'ajout de 2 itérations par rapport à un décodage sans parallélisme de décodeur composant. Par ailleurs, l'usage du parallélisme de sous-bloc dont l'efficacité de parallélisme décroît avec le nombre d'ASIP (voir figure 3.10) rajoute également pour les architectures avec 16 et 32 ASIPs respectivement une et deux itérations. Cet impact négatif du parallélisme de sous-bloc, dont l'efficacité suit une loi d'Amdahl, est atténué dans ces exemples par l'utilisation du décodage combiné, qui divise par deux le degré de parallélisme de sous-bloc.

Cette propriété assure à notre décodeur multi-ASIP une augmentation du débit plus linéaire que ce qui existent avec le décodage série conventionnel dans la littérature [125] [110]. En effet, avec le décodage série conventionnel, le débit d'une architecture multiprocesseur subit rapidement une loi d'Amdalh soit à cause des communications d'entrée-sortie pouvant bloquer le décodage (problème résolu dans notre cas par adjonction d'un espace mémoire double sur la mémoire des données entrantes), soit à cause des communications nécessaires pour propager l'information extrinsèque entre deux itérations (problème intimement lié dans notre cas à l'efficacité du décodage combiné) ou soit à cause des dégradations dues au parallélisme de sous-bloc (le degré de ce parallélisme est divisé par deux avec le décodage combiné).

La combinaison de ses avantages nous permet d'atteindre sur le plate-forme à 32 ASIPs un débit utile de 395Mbps à une fréquence de fonctionnement de 400MHz. Ces performances sont très supérieures à celles observées pour d'autres architectures programmables et très proches de celles d'architectures dédiées. Dans [125], une plate-forme multi-ASIP est proposée pour atteindre le haut-débit de manière flexible. Seulement, le débit n'est que de 22,84 Mbps dans une technologie ASIC de 180 nm et en utilisant 16 processeurs (réalisant chacun un SISO) et 5 itérations. Dans le même contexte, notre plate-forme atteint un débit de 222 Mbps en technologie 90nm, soit, ramené à des technologies comparables, des performances environ cinq fois supérieur. Toujours dans les mêmes conditions (16 SISOs et 5 itérations), la plate-forme multi-SISO dédiée proposée dans [110] atteint un débit de 340 Mbps dans une technologie 130 nm soit un débit comparable d'environ deux fois supérieur à notre plate-forme.

5.3.3 Efficacité globale de la plate-forme

Nombre d'ASIP (# fenêtres par ASIP)	4 (8)	8(4)	16(2)	32(1)
débit d'information (en Mbps) @90nm	64	127	222	395
Surface totale de la plate-forme multi-ASIP (en mm^2)	2,86	4,76	8,63	16,56
Ratio Débit/Surface	1,24	1,48	1,43	1,32

Tableau 5.3 — Efficacité des plate-formes multi-ASIPs de turbo-décodage utilisant le décodage combiné sur une trame de 1504 bits codée avec le standard DVB-RCS

Les tableaux 5.3 et 5.4 résument les résultats obtenus en surface et en débit pour respectivement des architectures utilisant le décodage combiné et ne l'utilisant pas. Selon le critère d'efficacité débit/surface normalisé par rapport à la configuration avec un ASIP, on s'aperçoit qu'il est plus efficace de ne pas utiliser le décodage combiné si le nombre de processeurs est faible. En revanche, avec 32 processeurs, l'architecture devient plus efficace (25% d'écart

de performance) avec le décodage combiné. Notons également que toutes les configurations présentées sont plus efficaces selon ce critère que la solution monoASIP.

Nombre d'ASIP (# fenêtres par ASIP)	1 (16)	4 (4)	8 (2)	16 (1)	32 ($\frac{1}{2}$)
débit d'information (en Mbps) @90nm	22	84	140	241	337
Surface totale de la plate-forme multi-ASIP (en mm^2)	1,22	2,76	4,8	8,68	17.41
Ratio Débit/Surface	1	1,69	1,62	1,53	1,07

Tableau 5.4 — Efficacité des plate-formes multi-ASIPs de turbo-décodage sans le décodage combiné sur une trame de 1504 bits codée avec le standard DVB-RCS

Les résultats sans décodage combiné intègrent dans la fréquence de fonctionnement les cycles d'attente nécessaires pour véhiculer l'information extrinsèque entre les itérations. Dans ces conditions, l'ASIP décode une trame au rythme de 3,8 cycles/symbole au lieu de 3,6 cycles/symbole.

5.4 Prototypage FPGA

5.4.1 Description du système de prototypage

Le turbo-décodeur multi-ASIP est validé par l'intermédiaire d'une carte FPGA pilotée par un PC hôte, qui affichera les résultats.

5.4.1.1 La carte

Le prototypage a été réalisé sur une carte conçue par la société Dinigroup comprenant 6 FPGAs Virtex 5 LX330 de Xilinx, offrant suffisamment de ressources pour prototyper des réseaux jusqu'à 32 processeurs [129]. Côté logique, chaque FPGA, gravé en 65nm, dispose de 288 blocs de RAM de 36kbits (soit environ 10Mbits de RAM) et également 51840 "slices" contenant chacun 4 LUTs à 6 entrées et 4 bascules flip-flop. Côté connectique, la carte dispose de trois interfaces utilisables pour établir un dialogue avec un PC hôte :

- une interface USB, qui permet un accès direct aux six FPGAs par l'intermédiaire d'un bus central avec un débit plutôt lent de 80 Mbps en lecture (i.e. de la carte vers le PC hôte) et 32 Mbps en écriture.
- une interface PCI, qui offre un débit maximum de 700Mbps en lecture et 350Mbps en écriture vers un FPGA de la carte.
- deux interfaces Ethernet, qui offrent chacune un débit de 1Gbps entre le PC hôte et un FPGA, à condition d'implanter dans le FPGA la pile réseau complète du protocole Ethernet.

5.4.1.2 Définition de l'interfaçage carte/PC hôte

Avant de valider le système par prototypage, il convient de poser clairement les interfaces entre ce qui va être exécuté sur la carte et ce qui va être exécuté sur le PC hôte. La figure 5.4 regroupe l'ensemble des tâches à exécuter pour observer le résultat du turbo-décodeur, ainsi que les débits nécessaires au fonctionnement de ces tâches où D_u.désigne le débit utile, R le rendement du turbocode, q la quantification utilisée sur l'entrée du turbo-décodeur. Ainsi, pour une plate-forme de turbo-décodage avec seulement 8 ASIPs à la fréquence de 150

MHz, en plaçant l'interface autour du turbo-décodeur en position 1 et en considérant $R = \frac{1}{3}$ et $q = 8$, les besoins en bande passante s'élèveraient à 1,1 Gbps en écriture et 48 Mbps en lecture. Avec les ressources de la carte, cet interfaçage impose de facto une limitation du débit, qui se traduira par réduction de fréquence sur la plate-forme prototypée. Cet interfaçage ne peut donc raisonnablement pas être utilisé pour valider le côté haut-débit de la plate-forme. Néanmoins c'est la solution la plus simple pour valider le bon fonctionnement du décodeur, puisque toutes les autres parties du système s'exécutent sur le PC hôte avec du logiciel validé.

Pour un prototype haut-débit, il faut revoir l'interfaçage et notamment réduire les besoins dispendieux de l'interface en écriture. Pour cela, il faut au moins intégrer un générateur de bruit matériel pour repousser l'interface en 2. Dans cet optique, nous pensions utilisé le générateur basé sur la méthode de Box-Muller proposé dans [130]. Cependant, des expériences haut-débit utilisant ce générateur ont montrées des dégradations de performances de correction dues aux imperfections du générateur [131]. A l'heure d'écriture de ces lignes, un autre générateur, qui produit une distribution gaussienne plus proche de l'idéale grâce à la méthode de Wallace [131], est en développement et sera intégré dans des prototypes ultérieurs.

Si on suppose acquis l'intégration du générateur de bruit sur la carte, il y a alors peu de raisons de ne pas déplacer l'interface un peu plus vers le PC hôte. En effet, la faible complexité du bloc turbocodeur permettrait une implantation rapide et peu coûteuse de ce bloc avec à la clé une diminution supplémentaire du débit en écriture de l'interface. En regardant encore en amont, on s'aperçoit que les blocs générateur d'information utile et détecteur d'erreur sont également très peu complexes (quelques bascules et comparateurs) et de ce fait facilement intégrables. Avec une interface repoussée en position 4, de sorte d'intégrer tous les blocs sur la carte, le PC hôte n'a alors plus qu'à gérer les configurations des différents blocs et l'affichage des taux d'erreur.

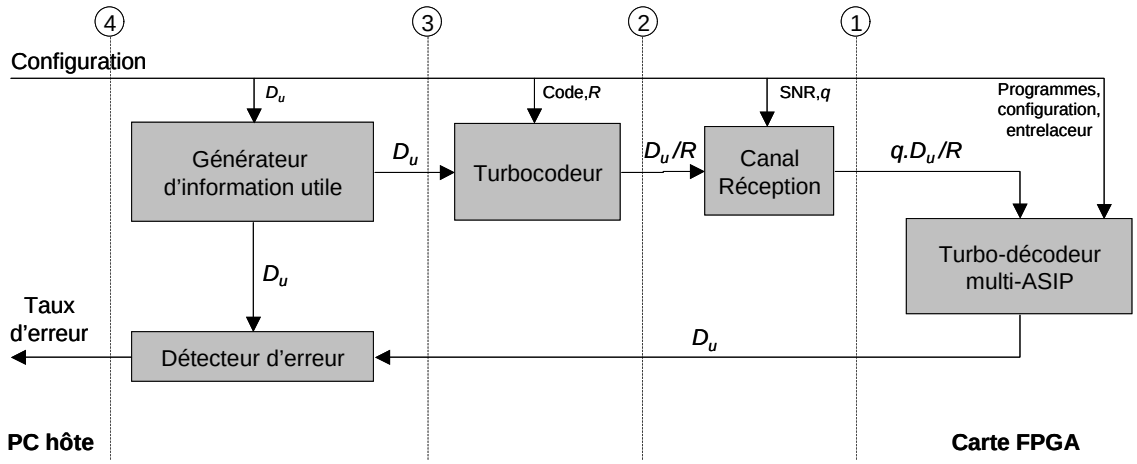

Figure 5.4 — Schéma bloc de prototype incluant les débits entre les blocs et les interfaces possibles

Pour résumer, deux interfaces présentent un intérêt pour le prototypage du turbo-décodeur multi-ASIP. La première interface (1) permet de valider rapidement le fonctionnement de la plate-forme tandis que la seconde interface intéressante (4) peut, grâce à des communications réduites avec le PC hôte, valider les performances haut-débit de la plate-forme et vérifier les performances asymptotiques des codes qui y sont programmés.

5.4.2 Flot de validation et résultats de prototypage

Cette section présente le flot de validation de la plate-forme dans son ensemble. Dans un premier temps, nous expliquerons comment le fonctionnement du processeur a été validé. Ensuite, nous présenterons un premier test de validation de la plate-forme entière en plaçant l'interface carte/PC hôte autour du turbo-décodeur. Finalement nous présenterons le test embarquant l'ensemble de la chaîne de transmission sur le FPGA.

5.4.2.1 Vérification de l'ASIP

La figure 5.5 représente l'ensemble du flot de vérification et de prototypage mis en oeuvre pour assurer le bon fonctionnement de l'ASIP. La partie correspondant au flot de vérification y est représentée en gris clair et celle correspondant au flot de prototypage en gris foncé. Ces flots s'entremêlent entre les différents niveaux d'abstraction utilisés.

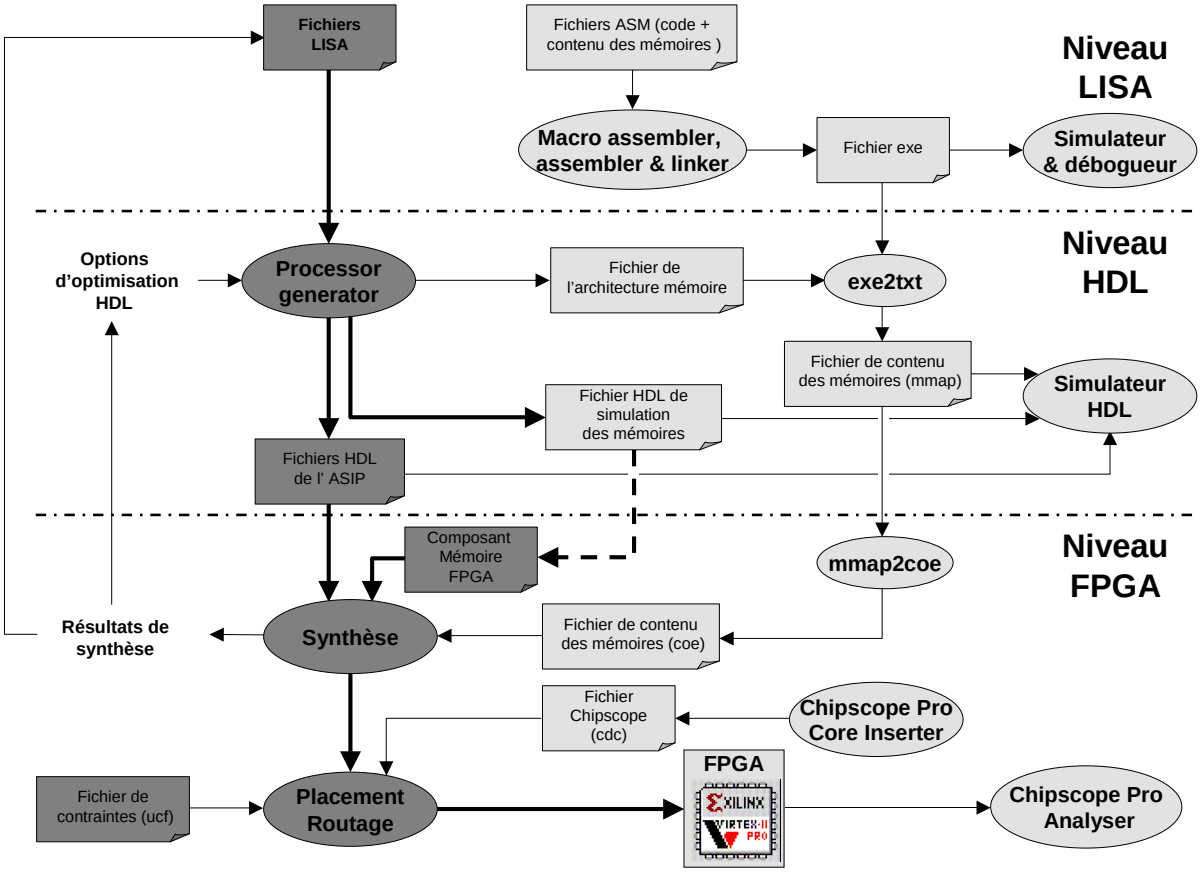

Figure 5.5 — Flot de vérification et de prototypage d'un ASIP avec la suite Processor Generator de CoWare

Le point d'entrée du flot utilise une description matérielle du processeur dans le langage LISA et des descriptions logicielles en assembleur qui incluent un programme pour l'ASIP et des contenus de mémoire initiaux issus d'un modèle C de référence. La suite d'outils *CoWare* permet ensuite de simuler et déboguer ces descriptions afin d'obtenir des résultats en accord avec le modèle C de référence.

Quand les descriptions conviennent, on peut passer au niveau de description RTL HDL. *Processor Generator* génère automatiquement l'architecture matérielle en HDL à partir du modèle LISA. La qualité du code RTL HDL généré dépend du code LISA fourni et des options d'optimisations retenues pour la génération de code (affectant les résultats en surface ou en fréquence de fonctionnement). Plus le code LISA est précis (par exemple sur le dimensionnement des ressources ou en explicitant le partage des ressources), meilleure est la qualité du code généré.

Le code HDL généré peut ensuite être vérifié par simulation en effectuant les mêmes tests qu'au niveau LISA. Cette étape de vérification est facilitée par un utilitaire (exe2txt) qui convertit le contenu des mémoires dans le format exécutable LISA vers le format utilisé (mmap) par les fichiers HDL de simulation de mémoire.

Pour passer ensuite au modèle FPGA, il faut substituer aux fichiers HDL de simulation de mémoire, qui ne sont pas synthétisables, des composants mémoires du FPGA correctement paramétrés (profondeur, largeur, latence, nombre de ports) et interfacés avec le code HDL de l'ASIP. Pour des FPGAs Xilinx, les composants mémoires peuvent être de la RAM distribuée sur la logique accessible sans cycle d'attente ou des blocs de RAM embarqués dans le FPGA accessible avec un cycle d'attente. Les choix des mémoires appropriées sont réalisés en fonction des contraintes sur les ressources. Toutes modifications de ces choix impliquent des modifications dans le code LISA pour intégrer les temps d'accès dans le pipeline du processeur. Le modèle HDL incluant ces composants mémoires est synthétisable et permet donc de caractériser le processeur en termes de surface et fréquence de fonctionnement. Si les résultats ne conviennent pas, des modifications peuvent être opérées sur le code LISA (par exemple, pour casser les chemins critiques et équilibrer les étages du pipeline ou regrouper des blocs matériels pour diminuer la surface) ou sur le choix des options de génération du code HDL. Il n'est cependant pas raisonnable d'envisager des modifications dans le code HDL à cause du peu de lisibilité du code HDL généré. Lorsque le code généré convient, il peut servir à programmer le FPGA après les étapes de placement-routage.

La vérification sur le FPGA effectue toujours les mêmes tests qu'aux niveaux LISA et HDL. Le contenu initial des composants mémoires du FPGA peut être chargé lors de la programmation du FPGA en joignant des fichiers d'initialisation (extension .coe) que l'on génère automatiquement à partir d'un script de conversion (mmap2coe). La vérification sur puce du processeur est ensuite réalisé grâce aux outils *ChipScope Pro* de *Xilinx.* L'outil de post-synthèse *ChipScope Pro Core Inserter* permet de générer une netlist ajoutant au circuit utilisateur un analyseur logique embarqué et un contrôleur de communication entre l'analyseur et le port JTAG du FPGA. L'analyseur logique peut capturer des signaux internes du circuit (sélectionnés dans le fichier .cdc) au moyen de signaux déclencheurs fixés par l'utilisateur. Les signaux capturés sont ensuite stockés dans les blocs RAM embarqués. Les signaux monitorés peuvent ensuite être lus au cours de l'exécution grâce à l'outil *ChipScope Pro Analyzer* et comparés à ceux du modèle de référence.

5.4.2.2 Validation fonctionnelle d'une plate-forme de turbo-décodage

Une fois le processeur validé au travers d'une large batterie de test, on peut raisonnablement l'intégrer dans une plate-forme multi-ASIP. Pour le prototypage décrit par la suite, la plate-forme multi-ASIP considérée utilise 8 ASIPs organisés autour d'un réseau butterfly en répartissant 4 ASIPs par décodeur composant.

L'intégration de cette plate-forme multi-ASIP de turbo-décodage peut générer un grand nombre d'erreurs potentielles au vue du nombre d'interfaces à connecter : 16 interfaces entre

les processeurs et le réseau d'information extrinsèque, 16 interfaces entre le réseau d'information extrinsèque et les mémoires d'information extrinsèque, 32 interfaces entre les processeurs et les mémoires d'échanges de métriques d'état (passé ou futur), 8 interfaces d'entrée entre l'extérieur et les mémoires des données entrantes et 4 interfaces de sortie entre les ASIPs associés au décodeur composant 1 et l'extérieur. C'est pourquoi nous avons commencé à valider cette plate-forme autour d'un test simple.

Ce premier test n'intègre donc aucun aspect système (pas d'interruption, pas de gestion des configurations...) et se contente de vérifier l'aptitude de la plate-forme à décoder correctement une seule trame. Le test réalisé utilise une trame ATM de 53 octets codée par un code double binaire (standard DVB-RCS) avec un rendement de codage $R = \frac{1}{3}$. Sur cette trame, on peut donc répartir le traitement à 53 symboles (106 bits) par processeur (4 processeurs par décodeur composant). Cette répartition ne nécessite donc pas l'utilisation du mécanisme de fenêtrage nécessaire quand plus de 64 symboles sont affectés à un processeur.

Simple, ce test ne permet pourtant pas une vérification bit à bit entre le modèle de référence C et la description VHDL de la plate-forme à cause principalement des échanges d'informations extrinsèques. Dans le modèle de référence C, ces échanges ne peuvent être différés que de manière statique. C'est-à-dire que le temps de propagation est le même pour tous les échanges d'information extrinsèque. Or le modèle HDL intègre un réseau de communication pour les informations extrinsèques dont le temps de traversée n'est pas constant à cause des collisions possibles. Du coup, certains échanges peuvent se produire dans un modèle et pas dans l'autre. Dans ces conditions, les processus itératifs des deux modèles se terminent avec des valeurs différentes pour les informations extrinsèques bien que les décisions obtenues soient les mêmes. En conséquence, le test ne permet pas de vérifier pleinement la validité du code HDL de la plate-forme. Le test permet certes de résoudre la plupart des problèmes en se fiant d'une part aux décisions finales du décodeur et d'autres part aux valeurs prises par les informations extrinsèques, car ces dernières doivent être proches de celles du modèle C. Pour aboutir à un modèle HDL correct, le test a été réalisé sur plusieurs trames obtenues dans des conditions différentes : trame sans bruit (i.e. obtenue à partir d'un $\frac{E_b}{N_0}$ très grand) pour une validation sommaire, trames avec beaucoup d'erreurs ($\frac{E_b}{N_0}$ autour de 0,5) pour vérifier que les erreurs sont aux mêmes endroits dans les deux décodeurs et que les valeurs des informations extrinsèques sont proches entre les deux modèles. Ces simulations du modèle HDL nous ont permis d'avoir une bonne confiance en sa validité au point de le prendre comme référence pour le prototypage. Néanmoins, il faut garder à l'esprit que seul un tracé de taux d'erreur permet de valider le modèle HDL. Or les vitesses de simulateurs HDL sont trop lentes pour permettre cette validation dont nous reparlerons dans la section suivante.

Le prototypage de ce modèle est séquencé en trois étapes : d'abord le contenu des mémoires des données entrantes des huit processeurs[2] est chargé sur la carte depuis le PC hôte via l'interface PCI ; Dans un deuxième temps, l'exécution de l'ensemble de la plate-forme est autorisée et toutes les informations extrinsèques générées sont récupérées toujours via l'interface PCI ; Lorsque le décodage de la trame se termine, la plate-forme émet les décisions dures qui sont également récupérées sur le PC hôte via l'interface PCI.

Nous avons abouti à un prototype en accord avec le modèle HDL qui fonctionne à une fréquence de fonctionnement de 135 MHz pour une surface occupée de 111700 LUTs soit 53% d'un FPGA de la carte. Cette configuration de prototype permet donc de décoder une trame en huit itérations au débit de 37,5 Mbps.

[2]Ces données sont évidemment issues du modèle C de référence.

5.4.2.3 Validation haut-débit du turbo-décodeur

Pour atteindre le débit théorique de notre plate-forme multi-ASIP sur un flux de trames, le test mis en oeuvre dans la section précédente n'est pas adapté, car la connectique entre la carte et le PC hôte ne peut supporter le choix d'interfaçage autour du turbo-décodeur (c.f. section 5.4.1.2). L'étude de la connectique de la carte nous a montré qu'il faut déplacer l'interface de manière à intégrer l'ensemble de la chaîne de transmission sur la carte pour ne plus être limité par la connectique entre la carte et le PC hôte.

Par ailleurs, il faut également repenser le séquencement du prototype. Le test précédent réalisait séquentiellement d'abord l'initialisation des mémoires pour une trame, puis le décodage de cette trame, puis la détection d'erreurs sur cette trame. Or le débit théorique de la plate-forme n'est calculé que sur la partie décodage de trame. Pour l'atteindre, il faut donc exécuter en parallèle l'initialisation des mémoires et la détection d'erreurs. La partie décodage de trame est évidement la plus longue de ces trois tâches, c'est pourquoi la synchronisation des tâches est réalisée grâce à un signal indiquant la fin du décodage d'une trame. Comme le montre la figure 5.6, ce séquencement nécessite l'utilisation de deux bancs mémoires pour les données issues du canal. Ainsi une trame codée et bruitée peut être stockée dans l'un des bancs et dans le même temps, la trame présente dans l'autre banc peut être décodée. Les accès aux bancs sont permutés à chaque synchronisation. Notons que la synchronisation est également utilisée pour conserver les trames non codées jusqu'au détecteur d'erreurs.

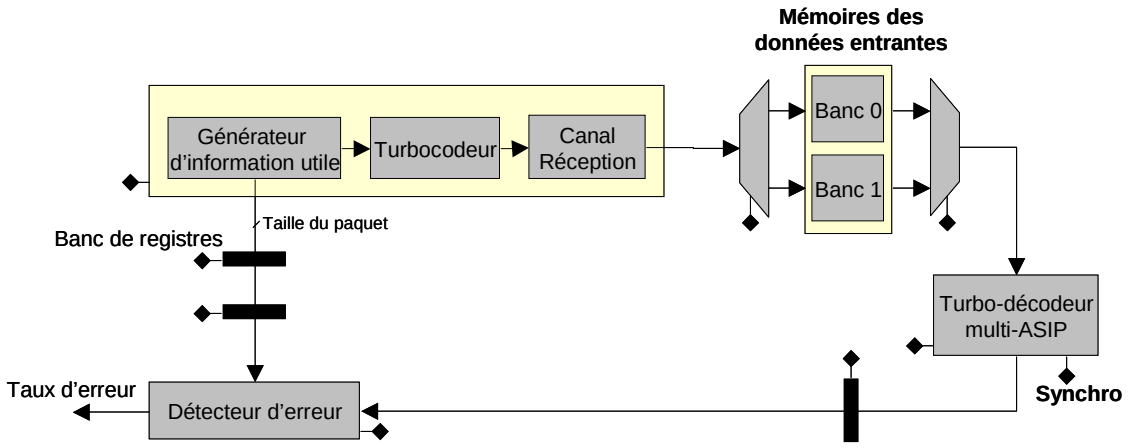

Figure 5.6 — Séquencement du prototype haut-débit

Grâce au débit de décodage autorisé par la carte, ce prototype permet également de valider les performances de décodage de la plate-forme avec des courbes pouvant descendre à des taux d'erreur binaire sûr[3] jusqu'à des valeurs de 10^{-10} en une demie-heure.

A l'heure de l'écriture de ces lignes, la mise en oeuvre de ce test est en cours de réalisation. La courbe de taux d'erreur concluant cette expérience sera intégrée dans la version finale de ce manuscrit.

[3]On considère un taux d'erreur sûr lorsque au moins 100 paquets sont erronés.

5.5 Conclusion

Ce chapitre présente une plate-forme multiprocesseur permettant de décoder des turbocodes à très haut-débit. Grâce notamment à l'utilisation d'un ASIP dédié au décodage des turbocodes et de réseaux de communication adaptés, la plate-forme présente également la flexibilité nécessaire pour supporter tous les turbocodes convolutifs proposés dans les standards existants et émergeants. Dans un premier temps, nous avons montré comment les composants s'organisent dans cette plate-forme et notamment comment les tâches sont affectées aux ASIPs, comment les ressources mémoires sont gérées au sein de la plate-forme et comment les communications s'articulent entre les ASIPs et avec l'extérieur de la plate-forme.

Suite à cette description de la plate-forme, cette dernière a été évaluée au vue de sa complexité et de son débit de décodage. Avec une surface de 16,56 mm^2 dans la technologie ST 90nm, notre plate-forme avec 32 ASIPs peut décoder un flux de trame MPEG codé avec la norme DVB-RCS au débit de 395 Mbps. Ce débit est d'autant plus impressionnant que les standards actuels les plus exigeants ne nécessitent "que" 150 Mbps. De plus, les performances obtenues par notre plate-forme ont été comparées à d'autres implantations et se sont montrées bien plus proches des performances d'un circuit dédié que de celles d'architectures programmables.

Pour valider complètement ces résultats, un prototype avec 8 ASIPs a été mis en oeuvre sur une carte intégrant 6 FPGAs Virtex5 LX330 de Xilinx. Le prototype peut atteindre le débit de 37,5 Mbps en occupant simplement 53% des ressources logiques d'un des FPGAs. Ainsi un large champ d'amélioration s'ouvre derrière cette première génération de prototype pour mieux exploiter les ressources de prototypage d'une part et pour intégrer au prototype les mécanismes systèmes qui permettront de programmer ou reprogrammer dynamiquement la plate-forme depuis l'extérieur.

CHAPITRE

6 Caractérisation des échanges d'informations extrinsèques lors du turbo-décodage et applications

AU-DELÀ des calculs et de la latence du décodage, les itérations d'un turbo-décodeur impliquent également des échanges intensifs d'informations extrinsèques entre les décodeurs composants. Dans la littérature, ces échanges ont été analysés à de multiples fins. L'analyse la plus utilisée a été proposée par Ten Brink sous le nom de diagramme EXIT pour analyser la convergence d'un processus itératif [53]. D'autres études sur la non linéarité de ces échanges [55] sont utilisées pour analyser le fonctionnement chaotique de ces échanges et proposer des corrections (ou post-processing) améliorant la convergence du processus. La connaissance sur les informations extrinsèques échangées dans le processus itératif est également mise à profit afin d'établir des critères d'arrêt pour le processus itératif. La littérature est riche sur ce sujet avec pour les plus connus le critère *CE* de cross entropie [132], le critère SCR (Sign Change Ratio) [50], le critère SDR (Sign Difference Ratio) [133] et d'autres encore tel que le Min-LLR dans [134]. A ces critères, dont le but est d'arrêter les trames convergées, sont venues s'adjoindre des critères de détection d'erreur toujours basés sur l'étude des informations extrinsèques [135], des mécanismes de répétition (ARQ) [136] et plus récemment une nouvelle catégorie de critère d'arrêt a été introduite [137] de manière à arrêter non plus le processus dans son ensemble, mais simplement les symboles ayant convergés. Ce dernier type de critère permet non seulement d'arrêter plus vite le processus que les critères classiques, mais permet en plus d'éviter des échanges inutiles d'informations lors des dernières itérations.

Dans le chapitre précédent, nous constations la part de plus en plus importante des structures de communications dans la complexité d'un turbo-décodeur haut-débit. Cette complexité est due aux contraintes cumulées qui reposent sur la structure de communication gérant les informations extrinsèques : une bande passante énorme (près de 500 fois le débit utile pour un code double binaire) avec les ressources de routage associées ; une gestion des conflits en temps réel pour conserver une flexibilité totale vis-à-vis du choix de l'entrelaceur ; un temps d'acheminement le plus court possible pour limiter les latences ralentissant le décodeur. En s'attaquant séparément à chaque contrainte, on peut réduire la complexité des communications. Cependant le problème n'est pas traité à la source. A t'on vraiment besoin d'échanger autant d'information ? Toutes les informations extrinsèques ont-elles la même importance ?

Pour répondre à cette question, il faut être en mesure de quantifier l'utilité des communications, où plus précisément l'apport d'un symbole dans la convergence du processus itératif.

Cette caractérisation d'un aspect encore vierge de l'information extrinsèque sera abordée dans la première section de ce chapitre. En fournissant un ordonnancement sur l'utilité d'un symbole, elle nous permettra par la suite diverses applications comme notamment la compression du flux d'échange d'informations extrinsèques ou sa prioritarisation. La première application est une réponse directe à notre problème initial, car en compressant le flux d'échange, on rend possible des réductions sur la complexité de routage, sur la consommation, voire même sur le délai d'acheminement de la structure de communication. La deuxième application offre un moyen de moduler les flux de communications au sein de la structure de communication, ce qui peut être utiliser pour minimiser les ressources.

6.1 Caractérisation des échanges d'informations extrinsèques lors du turbo-décodage

6.1.1 Evaluer la fiabilité des informations extrinsèques

Notons d_k le $k^{ième}$ symbole d'information de la trame, et $L_m^{(i)}\left(d_k = \widetilde{d}_k\right)$ le logarithme de la probabilité extrinsèque fournit par le $m^{ième}$ décodeur à la $i^{ième}$ itération pour l'estimation $\widetilde{d}_k$ du symbole d_k. Notons également $\widehat{d}_k$ l'estimation la plus probable du symbole d_k.

On peut alors définir la métrique de fiabilité $\Delta_m^{(i)}(d_k)$ de l'information extrinsèque du symbole d_k issue du $m^{ième}$ décodeur à la $i^{ième}$ itération comme :

$$\Delta_m^{(i)}(d_k) = L_m^{(i)}\left(d_k = \widehat{d}_k\right) - L_m^{(i)}\left(d_k \neq \widehat{d}_k\right) \tag{6.1}$$

Remarquons que, dans le cas simple binaire, cette métrique de fiabilité correspond simplement à la valeur absolue du LLR du symbole. Dans les cas non binaire, le deuxième terme est plus difficille à appréhender :

$$\begin{aligned} L_m^{(i)}\left(d_k \neq \widehat{d}_k\right) &= \ln \sum_{\widetilde{d}_k \neq \widehat{d}_k} P_m^{(i)}\left(\widetilde{d}_k\right) = \max_{\widetilde{d}_k \neq \widehat{d}_k}{}^{*}\left(L_m^{(i)}\left(d_k = \widetilde{d}_k\right)\right) \\ &\approx \max_{\widetilde{d}_k \neq \widehat{d}_k}\left(L_m^{(i)}\left(d_k = \widetilde{d}_k\right)\right) = L_m^{(i)}\left(d_k = \breve{d}_k\right) \end{aligned} \tag{6.2}$$

Ce terme est néanmoins approximable avec des dégradations négligeables en utilisant le second symbole le plus probable $\breve{d}_k$et on peut donc simplifier la métrique de fiabilité avec :

$$\Delta_m^{(i)}(d_k) = L_m^{(i)}\left(d_k = \widehat{d}_k\right) - L_m^{(i)}\left(d_k = \breve{d}_k\right) \tag{6.3}$$

D'après [53], l'information extrinsèque sous sa forme logarithmique peut être approchée par une distribution gaussienne. En conséquence, la métrique de fiabilité proposée peut également être approchée par une distribution gausienne. La figure 6.1 illustre ce résultat en représentant les distributions de la fiabilité au fil des itérations. Outre la forme gaussienne des distributions, on remarque que la fiabilité moyenne augmente d'itération en itération jusqu'à plafonner pour les itérations assurant la convergence du processus itératif.

Ces distributions de fiabilité ont été obtenues sur un grand nombre de trames en laissant le décodeur fonctionnait jusqu'à 100 itérations. Ainsi chacune de ces distributions comporte à la fois des trames ayant déjà convergées et d'autres n'ayant pas convergées. Or la distribution de fiabilité est intimement liée avec la convergence du processus itératif. Pour une distribution de fiabilité à plus de 3 itérations, le faible taux d'erreur noie la distribution de fiabilité des trames non convergées dans la distribution de fiabilité globale. Le critère du génie, correspondant à l'arrêt du processus itératif dès que la décision observée sur la trame est bonne, permet d'isoler les trames non convergées au fil des itérations. La figure 6.2 représente la fiabilité des informations extrinsèques sur les trames non convergées en utilisant le critère du génie.

Par comparaison avec la figure 6.1, il est flagrant que les trames non convergées présentent une fiabilité moyenne beaucoup plus basse que celle des trames convergées. Cette métrique de fiabilité constitue donc un excellent indicateur de la convergence du processus itératif. Ce fait

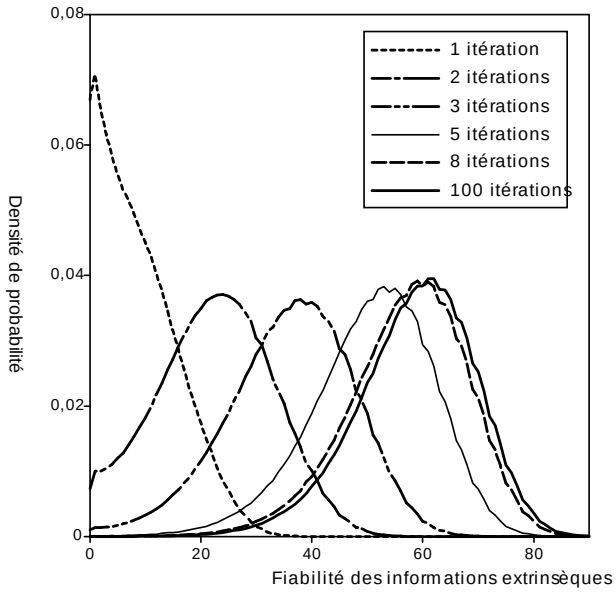

Figure 6.1 — Distributions de la fiabilité pour différentes itérations d'un code double binaire type DVB-RCS, quantification de 6 bits, R=6/7, sans critère d'arrêt

est depuis longtemps utilisé dans les critères d'arrêt comme le Min-LLR [134] qui décide de l'arrêt d'une trame en fonction de la valeur minimum prise par les LLRs (c-à-d des fiabilités) des bits de cette trame. L'utilisation de ce critère a été récemment proposé [133] pour stopper des échanges d'information extrinsèque au cours d'un processus itératif. Evidement, ce critère est sans effet lors de la première itération.

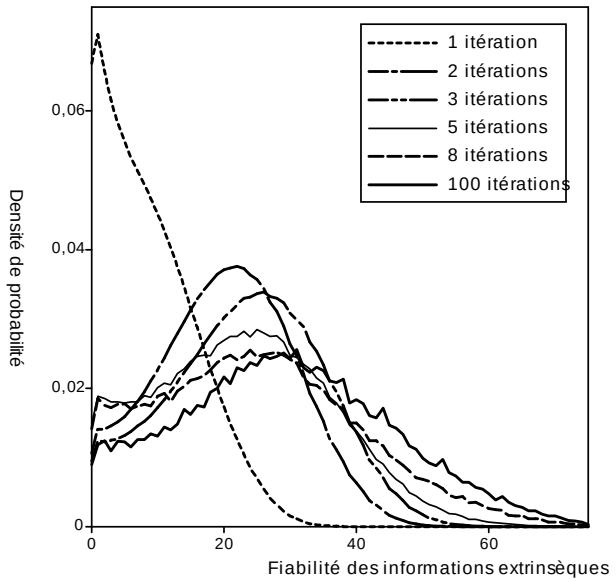

Figure 6.2 — Distributions de la fiabilité pour différentes itérations d'un code double binaire type DVB-RCS, quantification de 6 bits, R=6/7, critère d'arrêt du génie

Dans la suite de ce chapitre, nous exploiterons cette métrique pour déterminer l'impor-

tance d'une information extrinsèque sur la convergence du processus itératif.

6.1.2 Evaluer l'importance d'une information extrinsèque pour le processus itératif

Dans le cadre d'un processus itératif, qui n'a pas encore convergé, il est légitime de se demander ce que les informations échangées au cours du processus itératif apportent à la convergence et également de les caractériser. Avec une telle caractérisation, on pourrait ensuite classer les informations extrinsèques par ordre d'importance et, à la clé de ce classement, envisager une compression du flux des échanges d'information extrinsèques avec ou sans pertes. Dans cette section, nous définirons quelques règles permettant de juger de la pertinence de l'échange d'une paire d'information extrinsèque, et en se basant sur ces règles nous proposerons des critères d'ordonnancement pour ces paires.

6.1.2.1 Les règles de bases

A partir des diverses expérimentations menées pour évaluer la contribution d'une paire d'information extrinsèque à la convergence du processus itératif, les règles suivantes ont été dégagées :

- L'échange d'information est nécessaire lorsque les deux décodeurs composants prennent des décisions dures différentes sur un symbole.
- L'échange d'information est superflue lorsque les informations générées sur un symbole sont considérées comme fiables des deux côtés.
- L'échange d'information est également superflue lorsque les deux informations générées sur un symbole ont des fiabilités très proches.

Cette liste de règles, bien qu'empirique et probablement non exhaustive, donne une idée sur la forme des informations aidant à la convergence. Les deux premières règles partent en fait d'un principe plus général, à savoir qu'un échange présente un intérêt si et seulement si au moins une composante n'est pas fiable. Ainsi la règle 1 peut être revue comme le fait que le décodeur se trompant a une fiabilité relativement faible donc la paire d'information est importante.

De ce principe, on peut extraire un premier critère Ψ pour caractériser la contribution au processus itératif d'une paire d'information extrinsèque :

$$\Psi^{(i)}_{m,m'}\left(d_k\right) = \min\left(\Delta^{(i)}_{m}\left(d_k\right), \Delta^{(i)}_{m'}\left(d_k\right)\right) \tag{6.4}$$

A l'usage, ce critère s'est avéré très puissant permettant des gains de compression sans pertes allant jusqu'à 80 % des flux échangés sur les dernières itérations à condition d'utiliser des seuils pertinents. En effet, le gros problème de ce critère provient du choix des seuils décidant si une paire d'information extrinsèque est fiable ou non. Pour obtenir de bons résultats, ces seuils doivent être fixés pour chaque itération (évolution suivant la valeur moyenne de la répartition de la fiabilité), à chaque SNR et pour chaque rendement. De plus, les gains en compression observables lors des premières itérations sont relativement faibles se limitant dans nos recherches entre 5 et 7%. Ces raisons nous ont poussé à chercher un critère moins dépendant des seuils et plus performant lors des premières itérations.

La troisième règle repose sur une philosophie très différente des deux premières. Selon la théorie de l'information, une information est jugée d'autant plus importante que sa probabilité d'apparition est faible. Ainsi une information répétée est considérée comme peu importante. Donc quand nos deux décodeurs ont des informations extrinsèques avec des fiabilités proches, ils se répètent une information presque identique et l'échange d'information en devient superflue. Par définition, cette règle repose donc peu sur la notion de fiabilité puisque finalement seul compte l'écart relatif entre deux fiabilités. Dis autrement, on ne dépend plus de la moyenne des distributions de fiabilité, mais uniquement de son écart-type, qui varie un peu moins.

6.1.2.2 Le critère SRD

Le critère SRD (Symbol Reliability Difference), ou en français critère d'écart de fiabilité des symboles, combine en réalité un critère sur les décisions dures (règle 1) avec un critère utilisant des informations souples (règle 3). Originalement, le critère dur, appelé critère SDR (Sign Difference Ratio) [133], effectue ,dans un décodeur composant, la comparaison entre les décisions binaires (le signe de la décision souple) obtenues sur les informations extrinsèques *a priori* et celles obtenues sur les informations extrinsèques *a posteriori*. Le critère est considéré comme positif si les décisions sont de même signe, c'est-à-dire que les deux décodeurs prennent la même décision sur ce bit. Afin de supporter des codes non-binaire, nous avons étendu ce critère en vérifiant non plus une égalité de signe, mais une égalité sur les symboles les plus fiables. Si le critère est faux pour un symbole, les décodeurs composants n'ont pas encore convergé pour ce symbole et les échanges d'information extrinsèque sur ce symbole sont donc primordiaux pour une éventuelle convergence. Dans le cas contraire, le critère dur ne permet pas de juger de l'importance des informations extrinsèques échangées pour ce symbole. C'est pourquoi nous avons ajouté un critère souple pour en juger. Le critère souple proposé s'exprime comme la différence de fiabilité entre deux décodeurs composants m et m' sur le symbole d_k.

$$\Phi^{(i)}_{m,m'}\left(d_k\right) = \left|\Delta^{(i)}_{m}\left(d_k\right) - \Delta^{(i)}_{m'}\left(d_k\right)\right| \tag{6.5}$$

Par définition, ce critère souple évalue, à chaque itération, l'ambiguïté d'accord entre les décodeurs sur le symbole d_k. Et comme il respecte la règle 3, ce critère ordonne les informations extrinsèques de manière à refléter l'importance de leurs contributions à la convergence. Notons également que le critère souple n'a de sens que si le critère dur est respecté. En effet, dans l'hypothèse contraire, l'écart d'accord entre les décodeurs devrait être défini comme la somme des fiabilités.

Les distributions obtenues sur plusieurs itérations pour ce critère souple (figure 6.3) montrent, comme annoncé dans la section précédente, une faible dépendance au nombre d'itérations

Dans la suite, ces distributions seront utilisées pour fixer les seuils de décision pour le critère afin soit de compresser le flux d'information extrinsèque soit de fixer des priorités à l'information extrinsèque.

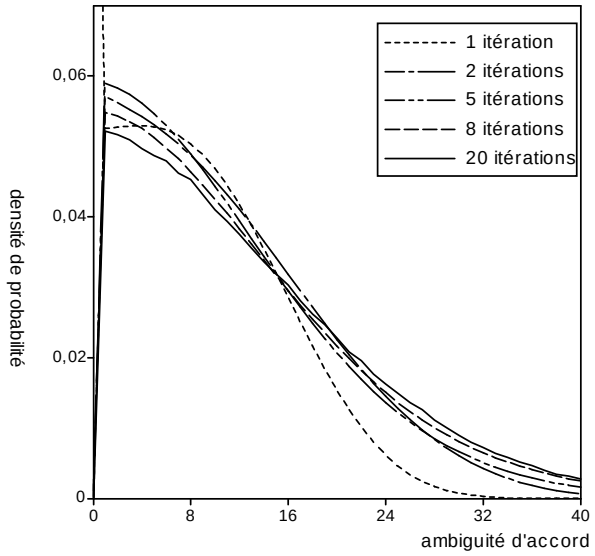

Figure 6.3 — Distributions de l'ambiguité d'accord (critère souple) pour différentes itérations d'un code double binaire type DVB-RCS, quantification de 6 bits, R=6/7, critère d'arrêt du génie

6.2 Réduction des communications à toutes les itérations

Dans cette section, nous montrerons la capacité du critère SRD à réduire les communications dans la structure de communication gérant les informations extrinsèques, ce qui peut entendre une réduction de la bande passante de la structure et une réduction des accès aux mémoires d'informations extrinsèques. La réduction de communication ne peut s'opérer que si un seuil est adjoint au critère SRD afin de ne pas échanger les informations extrinsèques sous ce seuil. Donc dans un premier temps, il faut chercher à mesurer, s'il existe, le seuil au-delà duquel des dégradations de performances apparaissent. De fait, ce seuil caractérise la compression sans pertes, puisque au-delà de ce seuil les taux d'erreur obtenus seront dégradés par rapport à la solution originale. La figure 6.4 représente le taux d'erreur paquet obtenus pour un rendement $\frac{1}{2}$ en seuillant à différentes valeurs. On remarque d'emblée que les écarts entre la courbe de référence et les courbes illustrant une compression sont presque constants à tout SNR, ce qui signifie que le choix du seuil se fait en tout indépendance vis à vis du SNR. Pour cette figure, on peut considérer que le seuil à 4 constitue le seuil sans pertes et que le seuil à 8 représente le seuil maximum présentant une dégradation acceptable (moins de 0,1dB). Au delà de ce seuil, à cause du caractère fortement non linéaire des dégradations, les performances obtenues s'éloigne dramatiquement des performances initiales. D'une manière générale, on notera que la progressivité de la dégradation avec l'augmentation du seuil confirme la capacité du critère SRD à détecter les informations extrinsèques les moins utiles à la convergence du processus itératif.

Pour un rendement de $\frac{6}{7}$ (figure 6.5), on obtient une compression quasi sans pertes avec un seuil fixé à 3 et de bonnes performances (moins de 0,05 dB de dégradation) avec un seuil fixé à 4. Au-delà, les performances sont déjà trop dégradées. On résumera ces deux figures en constatant que le choix du seuil est principalement lié au rendement du code, puisque les dégradations pour un seuil donné sont à peu près constantes à différentes valeurs de SNR.

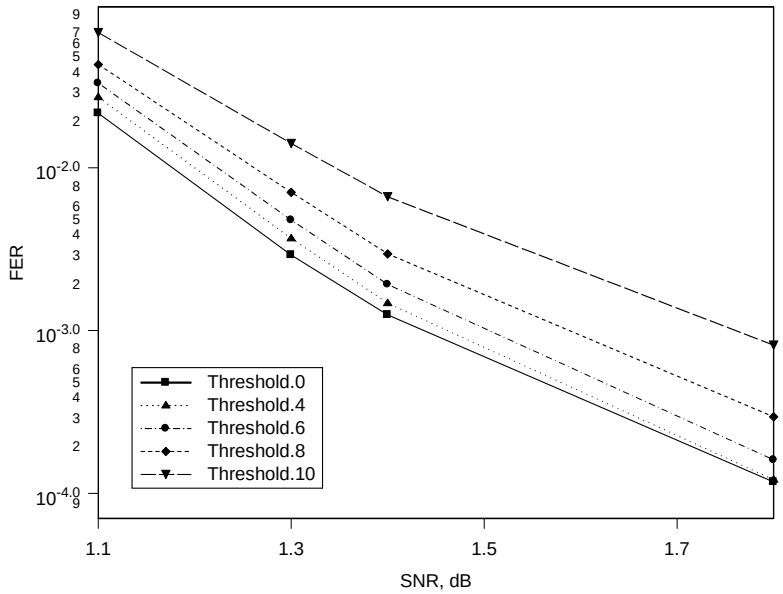

Figure 6.4 — Taux d'erreur paquet obtenu par filtrage des informations extrinsèques sous divers seuils, Code DVB-RCS, max-log-MAP, décodage combiné, R=$\frac{1}{2}$, paquet de 188 octets, critère d'arrêt du génie

Un fois le seuil fixé, on peut évaluer la compression avec des pertes négligeables obtenue avec le critère SRD. D'un point de vue statistique, on peut dégager deux informations importantes. Premièrement, la compression moyenne obtenue sur un grand nombre de paquets nous indique le gain global en communication, qui se répercute directement sur le nombre d'accès mémoires et sur la consommation. Deuxièmement, il est intéressant d'observer la compression minimum obtenue pour un paquet sur un grand nombre de paquets, car elle indique les gains que l'on peut réaliser sur les ressources de routage sans dégrader les propriétés de la structure de communication. D'après la figure 6.6, avec un seuil de 4 pour le rendement $\frac{6}{7}$, le taux de compression moyen se situe autour de 25% à partir de la deuxième itération et le taux de compression minimum autour de 18%. Les résultats obtenus sont légèrement meilleurs pour la première itération (30% en moyenne et 22% minimum), ce qui s'explique facilement car la distribution du critère pour la première itération est plus ramassée autour de 0, éliminant du coup plus d'informations extrinsèques. En analysant la figure 6.6 au vue du SNR, on s'aperçoit que les taux de compression diminuent très légèrement avec le SNR passant de 26% pour un SNR de 3,4dB à 23% pour un SNR de 4,2dB. Analytiquement, ce phénomène s'explique comme suit : en augmentant le SNR, on augmente légèrement l'écart-type de la métrique de fiabilité ; en conséquence, la distribution de l'ambiguité d'accord s'étale ; il a donc moins d'information extrinsèque sous le seuil ; soit moins de compression.

La même analyse peut être menée avec le rendement ou avec le nombre d'itération. Une diminution du rendement augmente fortement la dynamique de la métrique de fiabilité, ce qui aboutit par le même cheminement à une bien moins bonne compression à seuil égal. Même en ajustant les seuils de manière à avoir une dégradation constante pour tous les rendements, la compression avec le critère SRD s'avère toujours meilleure pour les rendements les plus forts.

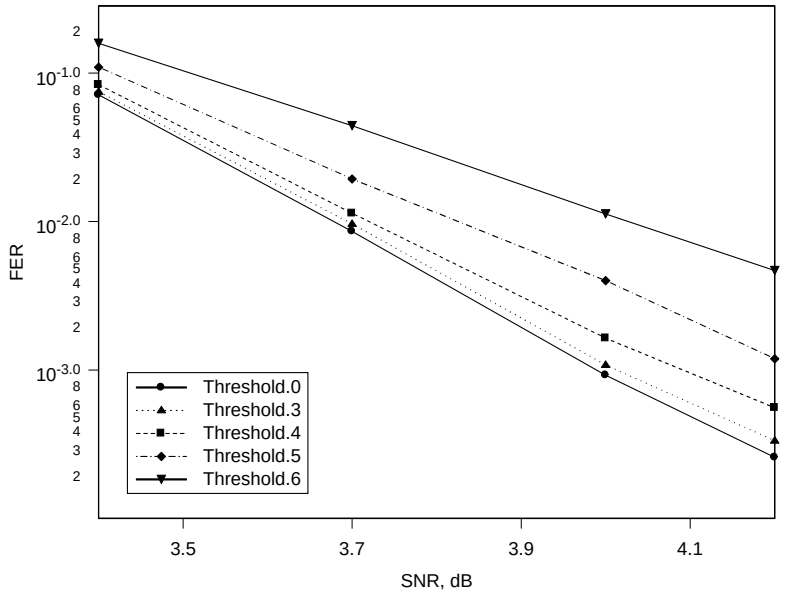

Figure 6.5 — Taux d'erreur paquet obtenu par filtrage des informations extrinsèques sous divers seuils, Code DVB-RCS, max-log-MAP, décodage combiné, R=$\frac{6}{7}$, paquet de 188 octets, critère d'arrêt du génie

Ainsi, dans le cas d'un rendement $\frac{1}{2}$ avec un seuil fixé à 8, la compression moyenne obtenue est de 20% au lieu de 25% pour le rendement $\frac{6}{7}$.

6.3 Adapter la qualité de service (QoS) d'un réseau sur puce

L'objet de cette section est de proposer un nouveau paradigme pour le réseau transportant les informations extrinsèques. Normalement le réseau doit "transporter sans penser" les paquets reçus et de fait les traiter tous de la même manière. Dans la réalité, cet équité est un non-sens, car chaque paquet comporte un information différente ne nécessitant pas forcément le même traitement. Dans cette optique, nous considérerons par la suite qu'un réseau doit "transporter le trafic le plus prioritaire avec la meilleure qualité de service", ce qui nécessite au préalable de créer des priorités exploitables par un réseau sur puce. D'une certaine manière, ce nouveau paradigme permet de transférer du contrôle de l'algorithme vers le réseau de communication, puisque le réseau de communication doit être en mesure de gérer plusieurs flux de données de priorités différentes. Certes ce transfert peut représenter un léger surcoût matériel et n'améliore en rien le transport du flux global, qui est régit par les caractéristiques imposées par le réseau (topologie, routage,...) telles que le taux de pertes de paquet ou le temps moyen de propagation, mais en contrepartie on peut adapter à chaque flux une QoS différente comme par exemple un taux de pertes des paquets lié à l'importance du paquet ou un délai d'acheminement lié à l'urgence du transport.

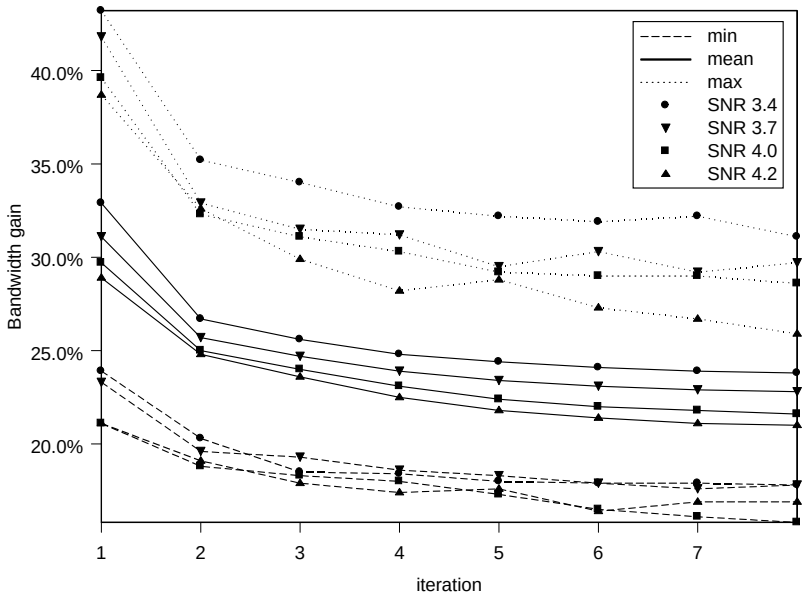

Figure 6.6 — Compression minimum, moyenne et maximum du flux d'information extrinsèque au fil des itérations pour différents SNR, code DVB-RCS, max-log-MAP, décodage combiné, R=$\frac{6}{7}$, paquet de 188 octets, critère d'arrêt du génie

6.3.1 Le taux de pertes des paquets

Autoriser un réseau à perdre des paquets permet d'économiser nettement les ressources à mettre en oeuvre au sein du réseau, notamment sur les ressources de mémorisation. Cette économie de complexité est d'autant plus intéressante que la complexité du circuit étudié est dominé par les communications. Avec plus de 20 % de surface dédiée aux communications, notre architecture multi-ASIP (avec au moins 32 ASIPs) peut prétendre à de telles économies. Reste néanmoins à s'accorder sur les paquets que l'on s'autorise à perdre pour réaliser ces économies.

Grâce au critère SRD, il est possible d'une part de détecter les informations les plus importantes avec le critère dur et d'autre part d'ordonner avec le critère souple les paquets d'information extrinsèque d'un moins important que l'on peut perdre au très important qu'il faut absolument conserver. En utilisant plusieurs seuils pour ce critère souple, on peut alors créer plusieurs flux d'information ayant chacun une priorité associée. Du côté algorithmique, la seule contrainte pour fixer les seuils impose une limite haute pour la priorité la plus basse supérieure au seuil de compression sans pertes. Le non-respect de cette contrainte signifierait que l'on accepte l'envoie d'un flux inutile, qui aurait pu être filtrer à la source.

Du côté réseau, une étude métrologique sur l'architecture réseau est nécessaire pour obtenir les proportions optimales de chaque flux par rapport au flux global. Couplées à la distribution du critère souple SRD, ces proportions optimales permettent alors de définir les seuils les mieux adaptés au réseau. Par exemple, pour un réseau ayant une répartition optimale de 4 sous-flux en (22%,15%,45%,18%) et le même code qu'à la section précédente, d'après la distribution du critère souple SRD (figure 6.7), on considèrera qu'une information

extrinsèque sur le symbole d_k fait partie du flux de priorité 0 si $0 \leqslant \Phi(d_k) \leqslant 5$, de priorité 1 si $5 < \Phi(d_k) \leqslant 8$, de priorité 2 si $8 < \Phi(d_k) \leqslant 24$, et de priorité 3 si $24 < \Phi(d_k)$ ou si le critère souple n'est pas respecté (on parle alors de priorité 3 dur). Comme le critère souple suit le taux d'erreur binaire, il est presque toujours respecté dans le cas de la $8^{ième}$ itération. Ainsi le seuillage (5,8,24) proposé respecte la répartition de flux optimale mesurée.

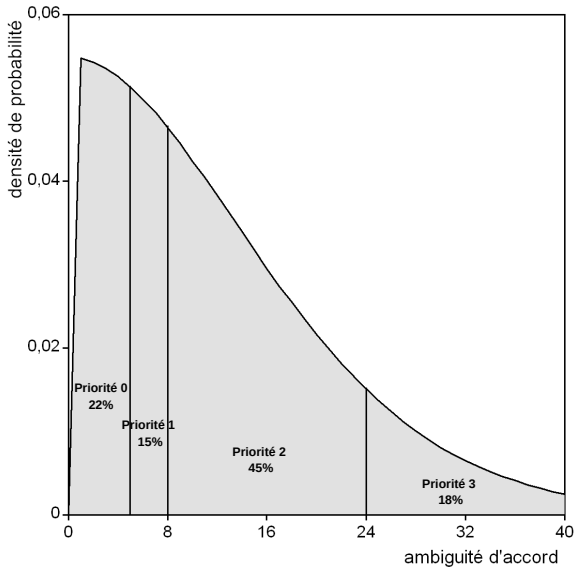

Figure 6.7 — Relation entre proportions des flux et distribution de l'ambiguité d'accord (critère souple) pour la $8^{ième}$ itération d'un code double binaire type DVB-RCS, quantification de 6 bits, R=6/7, critère d'arrêt du génie

Abandonnons ce seuillage au profit d'un seuillage linéaire avec des seuils à 4, 8 et 12 pour un rendement $\frac{6}{7}$. Nous avons, dans ce cas, étudié l'évolution de la répartition des sous-flux au cours des itérations en filtrant une partie des sous-flux (voir figure 6.8) tout en gardant un oeil sur les dégradations engendrées sur la convergence (voir figure 6.9).

On s'est aperçu que le filtrage du flux de priorité 0 seul permet un taux de compression[1] de l'ordre de 20% (voir figure 6.8.b) sans grande dégradation. Cependant, on peut observer sur la figure 6.9 que la convergence, mesurée par le nombre moyen d'itération, est ralentie d'environ 30% par rapport à la solution sans filtrage. En filtrant à la fois les flux de priorités 0 et 1 (voir figure 6.8.c), le taux de compression monte péniblement au dessus de 30% avec une détérioration inacceptable du TEP. Un filtrage plus conséquent (par exemple, des flux de priorités 0,1 et 2 comme sur la figure 6.8.d) n'apporte pas beaucoup plus à la compression et altère encore plus le taux d'erreur paquet obtenu.

Ces mauvaises performances proviennent de la définition même de la partie souple du critère SRD. A cause de son caractère relatif, le critère Φ ne permet pas un ordonnancement uniforme selon les fiabilités des décodeurs composants comme l'illustre le schéma 6.10.

Par exemple, lorsque Φ est petit devant la fiabilité moyenne des deux décodeurs composants (figure 6.10.a), le critère juge l'information extrinsèque peu importante, à juste titre au vu de la troisième règle de la section 6.1.2.1 (les deux décodeurs portent une information

[1] C'est-à-dire le rapport entre le nombre d'informations extrinsèques émises et celui des informations extrinsèques générées.

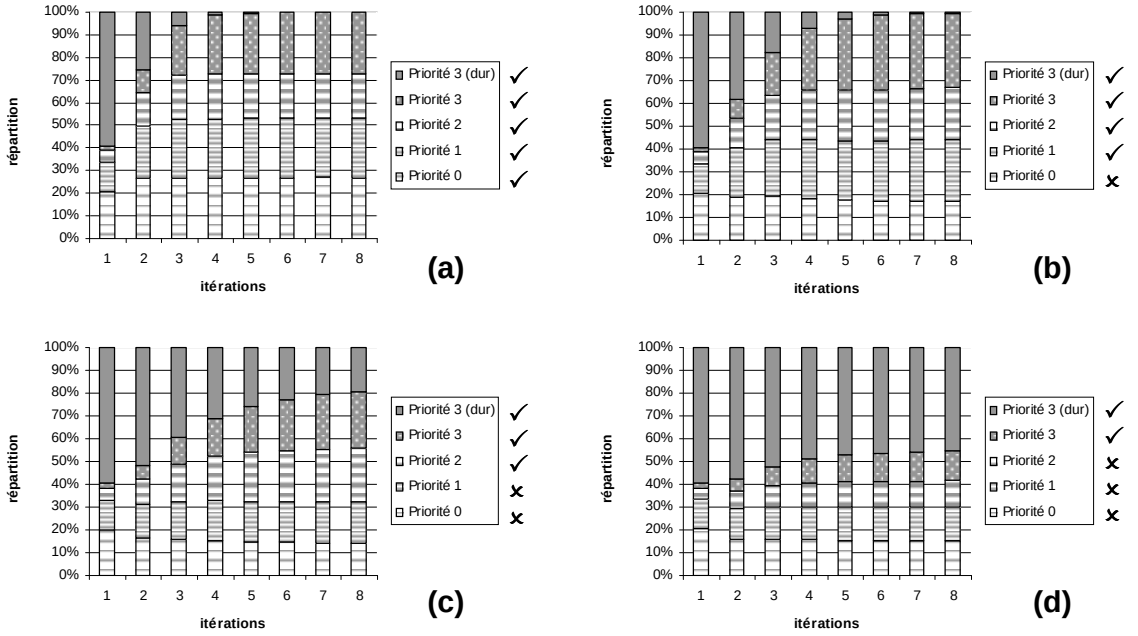

Figure 6.8 — Répartitions des sous-flux pour un seuillage linéaire (4,8,12) sur le critère SRD : (a) décodage référence, (b) décodage avec flux 0 filtré, (c) décodage avec flux 0 et 1 filtrés, (d) décodage avec flux 0, 1 et 2 filtrés. DVB-RCS, 6 bits, R=$\frac{6}{7}$, $\frac{Eb}{N_0}$ =4dB

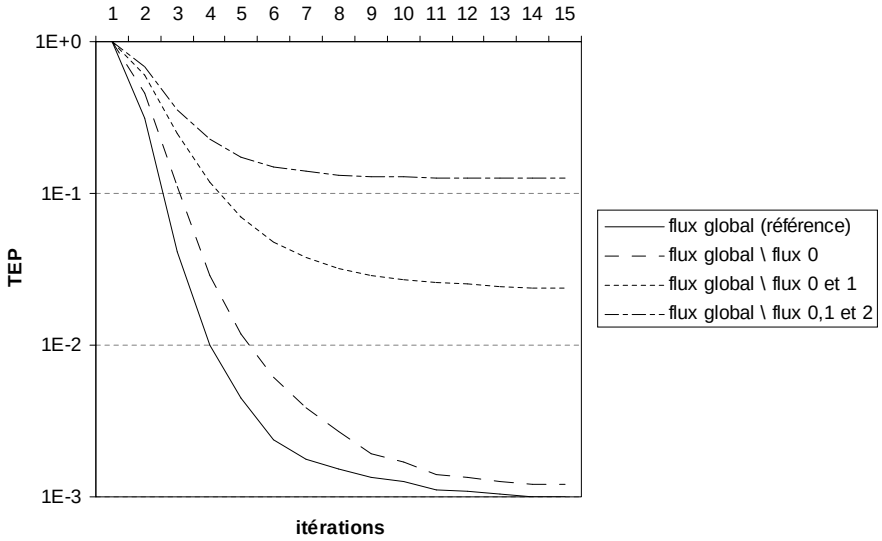

Figure 6.9 — Dégradation du TEP des décodeurs privés de certains sous-flux pour un seuillage linéaire (4,8,12) sur le critère SRD, DVB-RCS, 6 bits, R=$\frac{6}{7}$, $\frac{Eb}{N_0}$ =4dB

vraiment proche). Lorsque Φ est grand devant cette même fiabilité moyenne, le critère juge l'information extrinsèque très importante. Dans ce cas, le discernement du critère est encore correct, car un des deux décodeurs n'est peu fiable. Entre ces deux extrêmes, le critère n'est plus un bon indicateur de l'importance des informations extrinsèques (figure 6.10.c et d). Le

critère donne par exemple $\Phi_d > \Phi_c$, pourtant la fiabilité minimum du cas de figure d est supérieure à celle du cas de figure c, ce qui signifie que le cas c devrait être plus important (selon la règle 2).

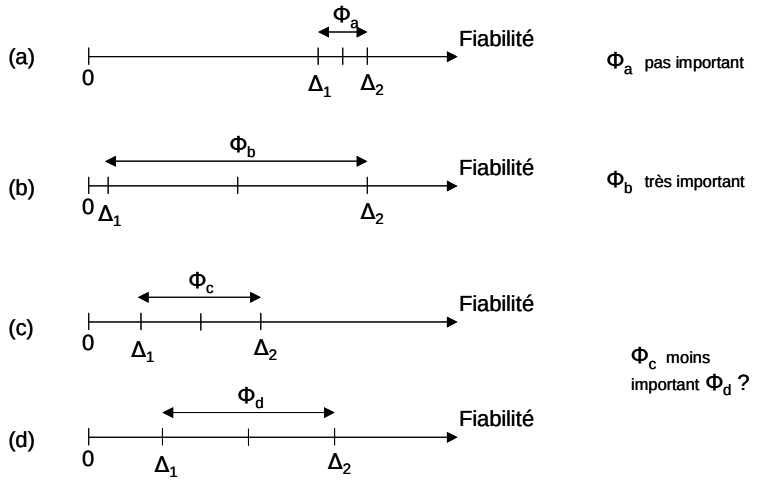

Figure 6.10 — Validité du critère SRD dans différents cas de figure

En définitive, la non-uniformité du critère Φ nous explique pourquoi ce critère ne permet pas de moduler de manière satisfaisante l'information extrinsèque avec des priorités, alors qu'il est efficace pour compresser le flux d'information extrinsèque.

6.3.2 Le délai d'acheminement

Considérons maintenant la QoS non plus sous l'angle de l'importance des données mais sous celui de l'urgence de leur transport. Ainsi nous considérons par la suite un réseau sans pertes où les données les moins prioritaires sont retardées ou déviées en cas de collision au profit de données plus urgentes. Cela implique évidemment un réseau avec un temps de propagation variable, de sorte que chaque flux puisse avoir un délai d'acheminement différent et notamment que le flux le plus prioritaire est un délai de propagation très proche du délai de propagation minimum du réseau. Dans notre contexte de turbo-décodage, on a vu au chapitre 3 que les performances du turbo-décodeur, de part l'efficacité du décodage combiné, sont directement liées au temps de propagation de la structure de communication. Cette dépendance plutôt gênante est générée par la portion des symboles qui n'arrivent pas à être échangés au cours d'une itération. En passant ces paquets d'information extrinsèque (ou symbole) en flux prioritaire, il est possible d'échanger plus d'information par itération et donc d'améliorer les performances du décodage.

Commençons par quantifier l'urgence d'un paquet. La relation d'ordonnancement urgence repose sur le temps dont dispose un paquet de son émission par un décodeur composant à son utilisation par l'autre décodeur composant. Comme on l'a vu au chapitre 3, ce temps peut être observé directement sur le masque de l'entrelaceur. Par exemple, pour un décodage combiné papillon, il s'agit de la distance minimum aux diagonales. La figure 6.11 nous servira d'illustration pour montrer comment ce critère de distance peut être utilisé pour échanger plus d'information par itération. Cette figure représente le masque d'un entrelaceur de 100 symboles tirés aléatoirement. Initialement, on suppose que le réseau de communication à un temps moyen de propagation de 8. Dans ces conditions, on aura 27% de symboles ne

pouvant bénéficier de l'effet du décodage combiné. Supposons maintenant que le réseau gère deux flux de communication en véhiculant un flux prioritaire d'environ 20% du trafic avec un temps moyen de propagation de 4 et un flux secondaire avec un temps moyen de propagation de 9. Globalement le temps de propagation est toujours de 8. Sur notre masque, il y a 14 symboles nécessitant un temps de transport inférieur à 4 et 17 symboles ayant temps maximal de transport entre 4 et 9. Le flux prioritaire est défini avec les symboles ayant un temps maximal de transport entre 4 et 9. Comme ils ne peuvent pas être véhiculés à temps, les symboles ayant un temps de transport inférieur à 4 sont intégrés dans le flux secondaire avec les symboles ayant un temps de transport supérieur ou égal à 9

En utilisant le réseau avec les priorités, le taux de symboles ne pouvant bénéficier de l'effet du décodage combiné passe de 27% à 14%. En conséquence, la convergence du processus itératif est améliorée par l'utilisation de cette priorité sur le délai d'acheminement.

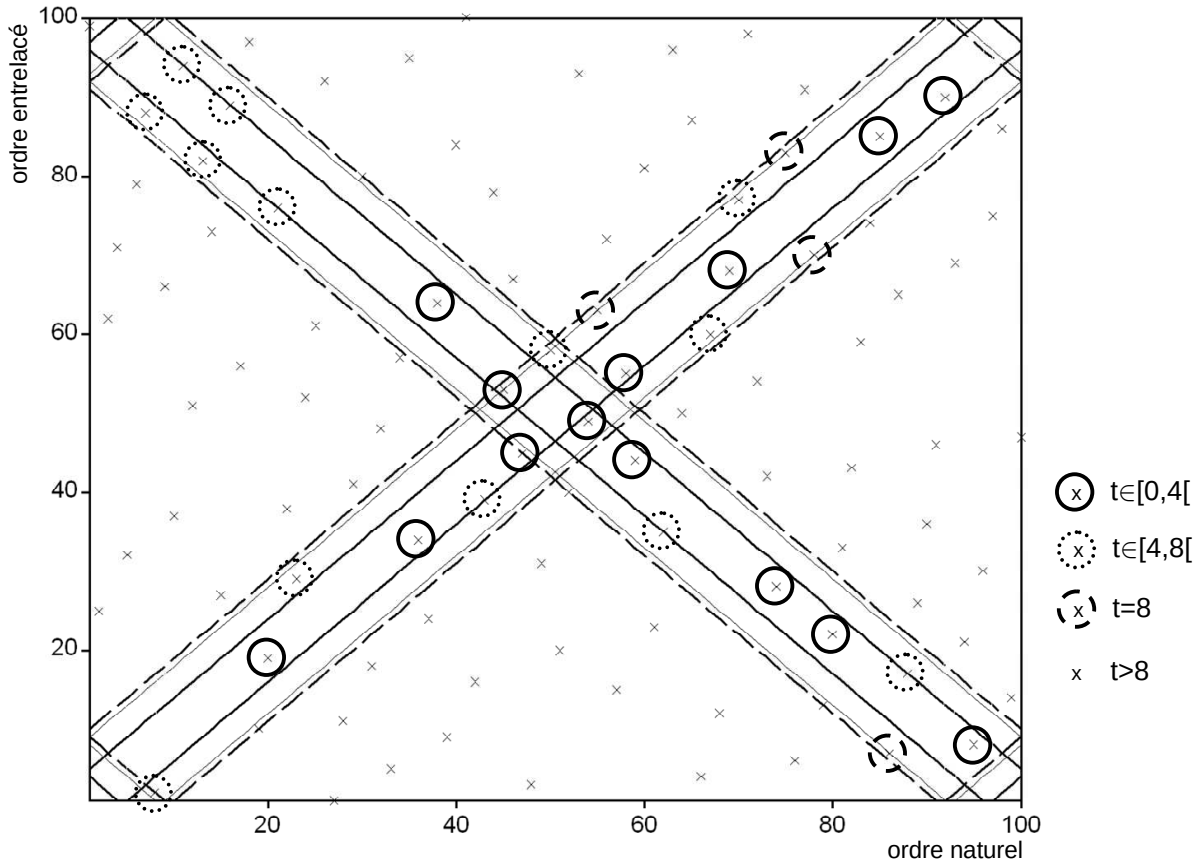

Figure 6.11 — Masques d'un entrelaceur de 100 symboles tirés aléatoirement pour le décodage combiné papillon pour des temps de propagation de 4, 8 et 9.

6.4 Conclusion

Dans un turbo-décodeur, tous les échanges d'information n'ont pas la même importance pour la convergence du processus itératif. Nous avons montré qu'il était possible de les classer selon des critères d'ordonnancement. Le critère d'ordonnancement SRD proposé nous a permis de compresser le flux d'information extrinsèque sans pertes et dès la première itération. Cette

compression présente des caractéristiques statistiques stables au fil des itérations, ce qui facilite son utilisation pour réduire la complexité matérielle. Autre avantage conséquent, le critère ne dépend ni du rapport signal à bruit ni de l'itération de décodage, mais seulement du rendement du code.

En revanche, on a vu que ce critère était clairement sous-optimal pour ordonnancer l'utilité de l'information, puisqu'il ne permet pas de compresser autant que d'autres critères. En conséquence, il n'est pas approprié pour moduler les flux d'informations extrinsèques par un système de priorités. En extension de ce travail, il serait intéressant d'observer les résultats avec le critère min-LLR, qui offre un ordonnancement plus fidèle que le critère SRD et donc devraient offrir plus de marges pour l'utilisation de priorités.

Par ailleurs, ce chapitre a été l'occasion de traiter de la qualité de service dans un réseau de communication et des moyens de l'utiliser pour rendre le turbo-décodage plus efficace soit en diminuant la complexité soit en améliorant la convergence du processus itératif.

Conclusion et perspectives

CE MÉMOIRE de thèse traite de la conception d'architecture multiprocesseur dédiée au domaine des turbo-communications et pose une première brique à cet édifice en ce qui concerne les applications de turbo-décodage convolutif. Les apports de ces travaux s'articulent autour de deux axes de recherches : la conception d'architecture multiprocesseur, ainsi que les algorithmes de turbo-communications. C'est pourquoi deux états de l'art sur ces axes amorcent ce mémoire, afin de camper au mieux le cadre de ces travaux.

Pour aboutir à une plate-forme multiprocesseur générique adaptée au turbo-décodage convolutif, nous avons conduit au préalable une étude algorithmique sur le parallélisme exploitable dans cette application. Un modèle d'efficacité, tenant à la fois compte de l'accélération d'un parallélisme et de son coût sur la complexité globale du décodeur nous a permis d'élaborer une classification des parallélismes existants sur trois niveaux : au niveau des métriques BCJR, des décodeurs BCJR-SISO et des turbo-décodeurs. Pour éclairer les relations entre les parallélismes, nous avons ensuite mené des investigations nouvelles, principalement sur le second niveau de la classification traitant des parallélismes de décodeurs BCJR-SISO. Nous avons, par exemple, montré pour le parallélisme de sous-bloc que la technique d'initialisation des décodeurs BCJR-SISO influe sur la convergence du processus itératif et que ce parallélisme suit une loi d'Amdahl le rendant inefficace à fort degré de parallélisme. Dans ces conditions de fort parallélisme de sous-bloc, nous avons également montré que le parallélisme de décodeur composant, récemment intronisé par la technique du décodage combiné ou « shuffled decoding », améliore l'efficacité du turbo-décodeur ; nous proposons aussi une optimisation de cette dernière en contraignant la conception de l'entrelaceur du turbocode. L'ensemble de ces recherches dirigées sur les parallélismes dans une application de turbo-décodage convolutif facilite la prise de décisions quant à leur choix dans une architecture matérielle.

Dès lors, nous avons pu proposer et concevoir un ASIP (processeur à jeu d'instruction spécialisé à une application) pour le décodage des codes convolutifs répondant à un juste compromis entre flexibilité et performance. En effet, la programmabilité de cet ASIP a été étudiée pour fournir la flexibilité nécessaire au support des applications de turbo-décodage convolutif. Quant à la performance de l'architecture, des débits de décodage s'élevant jusqu'à 540Mbps par itération sont assurés grâce à l'exploitation du parallélisme des métriques du BCJR. Pour plus de performances, le processeur est également équipé d'interfaces de communication dédiées au support des parallélismes de décodeur BCJR-SISO. Cela confère à notre processeur la modularité suffisante pour faciliter son intégration dans une plate-forme multi-ASIP.

Une plate-forme multi-ASIP générique adaptée aux turbo-décodeurs convolutifs a donc été présentée en utilisant comme composants ces ASIPs, des mémoires et des réseaux d'interconnexions dédiés pour lier les interfaces tout en assurant la bande passante nécessaire à ces communications. Nous avons prototypé cette plate-forme dans sa déclinaison avec huit ASIPs sur une carte d'émulation intégrant des circuits FPGA. Le prototype actuel peut atteindre

un débit de turbo-décodage de 37,5 Mbps en utilisant moins de 10% des ressources logiques de la carte.

De manière transversale à cet aboutissement, nous nous sommes intéressés aux échanges d'informations extrinsèques, dans l'idée d'en réduire le flux trop volumineux pour en obtenir des avantages dans l'implantation matérielle. En proposant un nouveau critère, nous avons ordonnancé ces échanges d'informations selon leur importance pour la convergence du processus itératif. Cela s'applique à la compression sans perte du flux d'information extrinsèque, mais aussi à l'instauration de la qualité de service par l'intermédiaire de priorités dans le réseau de communication véhiculant ce flux. Sur cette dernière application, nous constatons également qu'un système de priorités par urgence lié à la technique du décodage combiné et non par importance permet d'améliorer la convergence du processus itératif.

Perspectives

Ces travaux ont ouvert une voie vers une plate-forme multi-processeur capable de traiter n'importe quel turbo-récepteur en répondant aux exigeances de débit, de qualité de transmission et de complexité matérielle. D'autres thèses au sein du département électronique de l'ENST Bretagne ont déjà pour projet (et auront encore dans l'avenir) de poursuivre, étoffer et parachever la progression vers un turbo-récepteur universel.

A court terme, des travaux sont à faire sur le prototype pour mettre en avant sa flexibilité lors de démonstrations. Ainsi le prototype doit être doté d'aspects systèmes supplémentaires pour, par exemple, changer dynamiquement de configuration.

Des travaux légers d'extension de l'ASIP constitueraient également des perspectives à court terme. Pour les applications de turbo-décodage convolutif, le processeur pourrait être agrémenté d'une unité dédiée à la création de priorités pour chaque information (voir chapitre 6) et/ou à la gestion d'un critère d'arrêt. Par ailleurs, comme le processeur supporte l'algorithme BCJR, d'autres applications pouvant l'utiliser peuvent être supportées sous couvert de modifications sur l'ASIP. Par exemple, la turbo-égalisation repose sur l'algorithme BCJR et de récents travaux montrent qu'il peut en être de même pour les codes LDPC.

Des prolongements de ce travail à plus long terme sont également envisagés, ou envisageables, pour étendre la plate-forme à l'ensemble de la chaîne de transmission. Le cas de la turbo-démodulation, bien que non présenté dans ce mémoire, est déjà en cours d'étude. Un autre prolongement intéressant serait de supporter des algorithmes de type MMSE[2] que l'on utilise dans des applications de détection multi-utilisateurs (CDMA par exemple), de détection MIMO ou d'égalisation.

En marge du cadre de la plate-forme de turbo-communication, l'intérêt de la technique du décodage combiné, souligné pour les turbocodes dans ces travaux et également pour les codes LDPC dans les travaux précurseurs, suscite une multitude d'ouvertures possibles quant à son extension vers l'ensemble des processus itératifs.

[2]Minimum Mean Square Error

Glossaire

3GPP	3rd Generation Partnership Project
ACS	Addition Comparaison Sélection
ACSO	Addition Comparaison Sélection avec Offset
API	Application Programming Interface
ARP	Almost Regular Permutation
ARQ	Automatic Repeat reQuest
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
ATM	Asynchronous Transfer Mode
BBAG	Bruit Blanc Additif Gaussien
BCJR	Bahl-Cock-Jelinek-Raviv
BPSK	Binary Phase Shift Keying
CAO	Conception Assistée par Ordinateur
CDMA	Code Division Multiple Access
DRP	Dithered Relative Prime
DSP	Digital Signal Processor
DVB-RCS	Digital Video Broacasting Return Channel Satellite
DVB-RCT	Digital Video Broacasting Return Channel Terrestrial
EPIC	Explicit Parallel Instruction Computing
EXIT	EXtrinsic Information Transfer
FDMA	Frequency Division Multiple Access
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GALS	Globalement Asynchrone Localement Synchrone
GPP	General Purpose Processor
HDL	Hardware Description Language
IDMA	Interleaver Division Multiple Access

ILP	Instruction Level Parallelism
IP	Intellectual Proporty
LDPC	Low-Density Parity-Check
LLR	Log Likelihood Ratio
LUT	Look Up Table
MAP	Maximum A Posteriori
MIMD	Multiple Instruction Multiple Data
MIMO	Multiple Input Multiple Output
MISD	Multiple Instruction Single Data
MMSE	Minimum Mean Square Error
MPEG	Moving Picture Experts Group
NI	Network Interface
NISC	No Instruction Set Computer
NoC	Network on Chip
OCP	Open Core Protocol
OOK	On-Off Keying
PC	Personnal Computer
PCI	Peripheral Component Interconnect
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPP	Quadratic Permutation Polynomial
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfert Level
SDR	Sign Difference Ratio
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLD	System Level Design
SNR	Signal to Noise Ratio
SRD	Symbol Reliability Difference
SoC	Sytem on Chip
SOVA	Soft Output Viterbi Algorithm
TDMA	Time Division Multiple Access
TEB	Taux d'Erreur Binaire
TEP	Taux d'Erreur Paquet
TLM	Transaction Level Modeling
TLP	Task Level Parallelism
UMTS	Universal Mobile Telecommunications System

USB	Universal Serial Bus
VCI	Virtual Component Interface
VLIW	Very Long Instruction Word
ZOL	Zero Overhead Loop

Bibliographie

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture : A Quantitative Approach, Fourth Edition*. Morgan Kaufmann Publishers, 2006.

[2] M. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. C-21, pp. 948–960, Sept. 1972.

[3] G. Slavenburg, S. Rathnam, and H. Dijkstra, "The Trimedia TM-1 PCI VLIW media processor," in *Proceedings of Hot Chips VIII Symposium, IEEE CS Press,Los Alamitos,Californie*, 1996.

[4] *StarCore 140 DSP Core Reference Manual*, Rev. 3 ed., November 2001.

[5] *TMS320C5x User's Guide (Rev. D)*, Technical documents, spru056d ed., Texas Instrument, June 1998.

[6] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man, "Instruction set definition and instruction selection for ASIPs," in *High-Level Synthesis, 1994., Proceedings of the Seventh International Symposium on*, 1994, pp. 11–16.

[7] M. Reshadi, B. Gorjiara, and D. Gajski, "Utilizing horizontal and vertical parallelism with a no-instruction-set compiler for custom datapaths," in *Computer Design : VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on*, 2005, pp. 69–74.

[8] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri, "A VLIW processor with reconfigurable instruction set for embedded applications," *Solid-State Circuits, IEEE Journal of*, vol. 38, no. 11, pp. 1876–1886, 2003.

[9] H. Singh, M.-H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "MorphoSys : an integrated reconfigurable system for data-parallel and computation-intensive applications," *Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[10] R. David, D. Chillet, S. Pillement, and O. Sentieys, "DART : a dynamically reconfigurable architecture dealing with future mobile telecommunications constraints," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 156–163.

[11] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP - a self-reconfigurable data processing architecture," *The Journal of Supercomputing*, vol. 26, no. 2, pp. 167–184, 2003.

[12] J. R. Hauser and J. Wawrzynek, "Garp : A MIPS processor with a reconfigurable coprocessor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, Eds. Los Alamitos, CA : IEEE Computer Society Press, 1997, pp. 12–21.

[13] G. Hadjiyiannis, S. Hanono, and S. Devadas, "ISDL : An instruction set description language for retargetability," in *Design Automation Conference, 1997. Proceedings of the 34th*, 1997, pp. 299–302.

[14] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," in *European Design and Test Conference, 1995. ED&TC 1995, Proceedings.*, 1995, pp. 503–507.

[15] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo, "ArchC : a SystemC-based architecture description language," in *Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on*, 2004, pp. 66–73.

[16] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (asip) using the machine description language lisa," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, 2001, pp. 625–630.

[17] L. Benini and G. De Micheli, "Networks on chips : a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[18] W. Dally and B. Towles, "Route packets, not wires : on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

[19] ——, *Principles and Practices of Interconnection Networks.* San Francisco, CA, USA : Morgan Kaufmann Publishers, 2003.

[20] A. Jantsch and H. Tenhunen, Eds., *Networks on chip.* Hingham, MA, USA : Kluwer Academic Publishers, 2003.

[21] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Zeferino, "SPIN : a scalable, packet switched, on-chip micro-network," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 70–73 suppl.

[22] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, 2005, pp. 54–63.

[23] W. Dally, "Virtual-channel flow control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 194–205, 1992.

[24] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, 2000, pp. 250–256.

[25] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, 2002, pp. 105–112.

[26] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip : concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, 2005.

[27] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 113–129, 2005.

[28] http ://www.arteris.com.

[29] S. Evain, J.-P. Diguet, and D. Houzet, "A generic CAD tool for efficient noc design," in *Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004. Proceedings of 2004 International Symposium on*, 2004, pp. 728–733.

[30] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC : Specification Language and Methodology.* Kluwer Academic Publishers, 2000.

[31] T. Grotker, *System Design with SystemC.* Norwell, MA, USA : Kluwer Academic Publishers, 2002.

[32] E. Martin, O. Sentieys, H. Dubois, and J. Philippe, "GAUT : An architectural synthesis tool for dedicated signal processors," in *Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93. European*, 1993, pp. 14–19.

[33] A. Jerraya, I. Park, and K. O'Brien, "AMICAL : An interactive high level synthesis environment," in *European Design Automation Conference*, Paris, 1993, pp. 58–62.

[34] J.-Y. Brunel, W. Kruijtzer, H. Kenter, F. Petrot, L. Pasquier, E. de Kock, and W. Smits, "COSY communication IP's," in *Design Automation Conference, 2000. Proceedings 2000. 37th*, 2000, pp. 406–409.

[35] A. Jerraya and W. Wolf, Eds., *MULTIPROCESSOR SYSTEMS-ON-CHIPS.* Morgan Kauffman Publishers, 2004.

[36] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design : orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 1523–1543, 2000.

[37] S. Pasricha, "Transaction level modelling of SoC with SystemC 2.0," in *Synopsys User Group Conference*, 2002.

[38] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Extending the transaction level modeling approach for fast communication architecture exploration," in *Design Automation Conference, 2004. Proceedings. 41st*, 2004, pp. 113–118.

[39] A. Baghdadi, D. Lyonnard, N.-E. Zergainoh, and A. Jerraya, "An efficient architecture model for systematic design of application-specific multiprocessor SoC," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, 13-16 March 2001, pp. 55–62.

[40] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-software co-design of embedded systems : the POLIS approach.* Norwell, MA, USA : Kluwer Academic Publishers, 1997.

[41] L. Cai and D. Gajski, "Transaction level modeling : an overview," in *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, 2003, pp. 19–24.

[42] S. Abdi, J. Peng, H. Yu, D. Shin, A. Gerstlauer, R. Dömer, and D. Gajski, "System-on-chip environment (SCE version 2.2.0 beta) : Tutorial," Center for Embedded Computer Systems, University of California, Irvine, Tech. Rep. Technical Report CECS-TR-03-41, July 2003.

[43] W. Cesário, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. Jerraya, and M. Diaz-Nava, "Component-based design approach for multicore SoCs," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 789–794.

[44] H. Chang, L. Cooke, M. Hunt, G. Martin, A. J. McNelly, and L. Todd, *Surviving the SOC revolution : a guide to platform-based design.* Norwell, MA, USA : Kluwer Academic Publishers, 1999.

[45] C. Shannon, "A mathematical theory of communication," Bell System Technical Journal, Tech. Rep., 1948.

[46] P. Elias, "Coding for noisy channels," *IRE Convention Record*, vol. 4, pp. pp. 37–47, 1955.

[47] J. Proakis, *Digital Communications*, fourth edition ed. McGraw-Hill, 1995.

[48] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding : Turbo-codes. 1," in *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, 1993, pp. 1064–1070 vol.2.

[49] J. Hagenauer, "The turbo principle : Tutorial introduction and state of the art," in *International Symposium on Turbo Codes*, september 1997, pp. pp. 1–11.

[50] R. Shao, S. Lin, and M. Fossorier, "Two simple stopping criteria for turbo decoding," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1117–1120, 1999.

[51] A. Shibutani, H. Suda, and F. Adachi, "Reducing average number of turbo decoding iterations," *Electronics Letters*, vol. 35, no. 9, pp. 701–702, 1999.

[52] S. ten Brink, "Convergence of iterative decoding," *Electronics Letters*, vol. 35, no. 10, pp. 806–808, 1999.

[53] ——, "Convergence behavior of iteratively decoded parallel concatenated," *Communications, IEEE Transactions on*, vol. 49, no. 10, pp. 1727–1737, 2001.

[54] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," *Selected Areas in Communications, IEEE Journal on*, vol. 19, no. 5, pp. 891–907, 2001.

[55] L. Kocarev, F. Lehmann, G. Maggio, B. Scanavino, Z. Tasev, and A. Vardy, "Nonlinear dynamics of iterative decoding systems : analysis and applications," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1366–1384, 2006.

[56] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *Communications, IEEE Transactions on*, vol. 53, no. 2, pp. 209–213, 2005.

[57] J. Hagenauer, "Source-controlled channel decoding," *Communications, IEEE Transactions on*, vol. 43, no. 9, pp. 2449–2457, 1995.

[58] S. Le Goff, A. Glavieux, and C. Berrou, "Turbo-codes and high spectral efficiency modulation," in *Communications, 1994. ICC 94, SUPERCOMM/ICC '94, Conference Record, Serving Humanity Through Communications. IEEE International Conference on*, 1994, pp. 645–649 vol.2.

[59] M. Moher, "Turbo-based multiuser detection," in *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on*, 1997, pp. 195–.

[60] S. Haykin, M. Sellathurai, Y. de Jong, and T. Willink, "Turbo-mimo for wireless communications," *Communications Magazine, IEEE*, vol. 42, no. 10, pp. 48–53, 2004.

[61] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference : Turbo-equalization," *European Transactions on Telecommunications*, vol. Vol. 6, no. 5, pp. pp. 507–511, 1995.

[62] N. Noels, C. Herzet, A. Dejonghe, V. Lottici, H. Steendam, M. Moeneclaey, M. Luise, and L. Vandendorpe, "Turbo synchronization : an em algorithm interpretation," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 4, 2003, pp. 2933–2937 vol.4.

[63] C. Berrou, C. Douillard, and M. Jezequel, "Multiple parallel concatenation of circular recursive systematic convolutional (crsc) codes," *Annales des télécommunications*, vol. Vol. 54, no. no. 3, pp. pp. 166–172, 1999.

[64] R. Fano, "A heuristic discussion of probabilistic decoding," *Information Theory, IEEE Transactions on*, vol. 9, no. 2, pp. 64–74, 1963.

[65] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260–269, 1967.

[66] J. Forney, G.D., "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[67] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," in *Global Telecommunications Conference, 1989, and Exhibition. 'Communications Technology for the 1990s and Beyond'. GLOBECOM '89., IEEE*, 1989, pp. 1680–1686 vol.3.

[68] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *Information Theory, IEEE Transactions on*, vol. 20, no. 2, pp. 284–287, 1974.

[69] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *Communications, 1995. ICC 95 Seattle, Gateway to Globalization, 1995 IEEE International Conference on*, vol. 2, 1995, pp. 1009–1013 vol.2.

[70] C. Douillard and C. Berrou, "Turbo codes with rate-m/(m+1) constituent convolutional codes," *Communications, IEEE Transactions on*, vol. 53, no. 10, pp. 1630–1638, 2005.

[71] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *Selected Areas in Communications, IEEE Journal on*, vol. 19, no. 5, pp. 871–882, 2001.

[72] G. Jeong and D. Hsia, "Optimal quantization for soft-decision turbo decoder," in *Vehicular Technology Conference, 1999. VTC 1999 - Fall. IEEE VTS 50th*, vol. 3, 1999, pp. 1620–1624 vol.3.

[73] Y. Wu and B. Woerner, "The influence of quantization and fixed point arithmetic upon the ber performance of turbo codes," in *Vehicular Technology Conference, 1999 IEEE 49th*, vol. 2, 1999, pp. 1683–1687 vol.2.

[74] E. Boutillon, W. Gross, and P. Gulak, "Vlsi architectures for the map algorithm," *Communications, IEEE Transactions on*, vol. 51, no. 2, pp. 175–185, 2003.

[75] Y. Wu, B. Woerner, and T. Blankenship, "Data width requirements in siso decoding with module normalization," *Communications, IEEE Transactions on*, vol. 49, no. 11, pp. 1861–1868, 2001.

[76] A. Hekstra, "An alternative to metric rescaling in viterbi decoders," *Communications, IEEE Transactions on*, vol. 37, no. 11, pp. 1220–1222, 1989.

[77] G. J. Forney, *"Performance of concatenated codes", Key papers in the development of coding theory*, E. Berlekamp, Ed. IEEE Press, 1974.

[78] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes : performance analysis, design, and iterative decoding," *Information Theory, IEEE Transactions on*, vol. 44, no. 3, pp. 909–926, 1998.

[79] K. Narayanan and G. Stuber, "Selective serial concatenation of turbo codes," *Communications Letters, IEEE*, vol. 1, no. 5, pp. 136–139, 1997.

[80] A. Graell i Amat, G. Montorsi, and F. Vatta, "Design and performance analysis of a new class of rate compatible serial concatenated convolutional codes," *Submitted to IEEE Transactions on Communications*, 2005.

[81] K. Gracie and M.-H. Hamon, "Turbo and turbo-like codes : Principles and applications in telecommunications," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1228–1254, 2007.

[82] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," The Telecommunications and Data Acquisition Report, Tech. Rep. pp. 56-65., 1995.

[83] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes : towards a single model," in *Communications, 2004 IEEE International Conference on*, vol. 1, 2004, pp. 341–345.

[84] S. Crozier and P. Guinand, "Distance upper bounds and true minimum distance results for turbo-codes designed with drp interleavers," in *International symposium on turbo codes and related topics No3*, 2003.

[85] O. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *Information Theory, IEEE Transactions on*, vol. 52, no. 3, pp. 1249–1253, 2006.

[86] J. Vogt and A. Finger, "Improving the max-log-map turbo decoder," *Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, 2000.

[87] P. Ould-Cheikh-Mouhamedou, Y Guinand and P. Kabal, "Enhanced max-log-app and enhanced log-app decoding for dvb-rcs," in *3rd International Symposium on Turbo codes*, 2003.

[88] C. Berrou, *Codes et turbocodes*. Springer, 2007.

[89] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS spring joint computer conference*, Atlantic City,N.J., april 1967, pp. pp. 483–485.

[90] G. Masera, G. Piccinini, M. Roch, and M. Zamboni, "Vlsi architectures for turbo codes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 7, no. 3, pp. 369–379, Sept. 1999.

[91] Y. Zhang and K. Parhi, "Parallel turbo decoding," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, 23-26 May 2004, pp. II–509–12Vol.2.

[92] G. Fettweis and H. Meyr, "Parallel viterbi algorithm implementation : breaking the acs-bottleneck," *Communications, IEEE Transactions on*, vol. 37, no. 8, pp. 785–790, 1989.

[93] M. Mansour and N. Shanbhag, "Vlsi architectures for siso-app decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 4, pp. 627–650, 2003.

[94] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (map) module to decode parallel and serial concatenated codes," TDA Progress Report, Tech. Rep., 1996.

[95] A. Viterbi and A. Viterbi, "An intuitive justification and a simplified implementation of the map decoder for convolutional codes," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 2, pp. 260–264, 1998.

[96] C. Schurgers, F. Catthoor, and M. Engels, "Memory optimization of map turbo decoder algorithms," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 2, pp. 305–312, April 2001.

[97] A. Worm, H. Lamm, and N. Wehn, "A high-speed map architecture with optimized memory size and power consumption," in *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, 2000, pp. 265–274.

[98] J. Zhang, Y. Wang, M. Fossorier, and J. Yedidia, "Replica shuffled iterative decoding," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, 4-9 Sept. 2005, pp. 454–458.

[99] T. Wolf, "Initialization of sliding windows in turbo decoders," in *3rd International Symposium on Turbo Codes and Related Topics*, Brest, France, September 2003, pp. pp. 219–222.

[100] A. Tarable and S. Benedetto, "Mapping interleaving laws to parallel turbo decoder architectures," *Communications Letters, IEEE*, vol. 8, no. 3, pp. 162–164, 2004.

[101] F. Speziali and J. Zory, "Scalable and area efficient concurrent interleaver for high throughput turbo-decoders," in *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, 31 Aug.-3 Sept. 2004, pp. 334–341.

[102] M. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A scalable system architecture for high-throughput turbo-decoders," in *Signal Processing Systems, 2002. (SIPS '02). IEEE Workshop on*, 16-18 Oct. 2002, pp. 152–158.

[103] M. Jézéquel, C. Berrou, C. Douillard, and P. P, "Characteristics of a sixteen-state turbo-encoder/decoder (turbo4)," in *International Symposium on Turbo Codes*, 1997.

[104] "Discusion privée avec claude berrou."

[105] M. Arzel, C. Lahuec, F. Seguin, D. Gnaedig, and M. Jezequel, "Semi-iterative analog turbo decoding," *Circuits and Systems I : Regular Papers, IEEE Transactions on [Circuits and Systems I : Fundamental Theory and Applications, IEEE Transactions on]*, vol. 54, no. 6, pp. 1305–1316, 2007.

[106] D. Gnaedig, E. Boutillon, M. Jézéquel, V. Gaudet, and P. Gulak, "On multiple slice turbo codes," in *third international symposium on turbo codes and related topics*, Brest , FRANCE, 2003, pp. pp. 343–346.

[107] J. Zhang, J. Zhang, Y. Wang, M. P. C. Fossorier, and J. S. Yedidia, "Iterative decoding with replicas," *Information Theory, IEEE Transactions on*, vol. 53, no. 5, pp. 1644–1663, 2007.

[108] C. Heegard and S. Wicker, *Turbo Coding*. Kluwer Academic Publishers, 1999.

[109] D. Gnaedig, E. Boutillon, J. Tousch, and M. Jézéquel, "Towards an optimal parallel decoding of turbo codes," in *4th International Symposium on Turbo-Codes and Related Topics*, 2006.

[110] G. Prescher, T. Gemmeke, and T. Noll, "A parametrizable low-power high-throughput turbo-decoder," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 5, 2005, pp. v/25–v/28 Vol. 5.

[111] Xilinx, "3gpp turbo decoder v3.1," May 2007.

[112] A. La Rosa, L. Lavagno, and C. Passerone, "Implementation of a umts turbo decoder on a dynamically reconfigurable platform," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 100–106, 2005.

[113] R. Kothandaraman and M. Lopez, "An efficient implementation of turbo decoder on adi tigersrarc ts201 dsp," in *Signal Processing and Communications, 2004. SPCOM '04. 2004 International Conference on*, 2004, pp. 344–348.

[114] P. Ituero and M. Lopez-Vallejo, "New schemes in clustered vliw processors applied to turbo decoding," in *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on*, 2006, pp. 291–296.

[115] M. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. Davis, G. Woodward, C. Nicol, and R.-H. Yan, "A unified turbo/viterbi channel decoder for 3gpp mobile wireless in 0.18-/spl mu/m cmos," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 11, pp. 1555–1564, 2002.

[116] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. Flautner, "Design and implementation of turbo decoders for software defined radio," in *Signal Processing Systems Design and Implementation, 2006. SIPS '06. IEEE Workshop on*, 2006, pp. 22–27.

[117] F. Gilbert, M. Thul, and N. Wehn, "Communication centric architectures for turbo-decoding on embedded multiprocessors," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 356–361.

[118] O. Muller, A. Baghdadi, and M. Jézéquel, "ASIP-based multiprocessor soc design for simple and double binary turbo decoding," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 6-10 March 2006, pp. 1–6.

[119] T. Vogt and N. Wehn, "A reconfigurable applcation specific instruction set processor for viterbi and log-map decoding," in *Signal Processing Systems Design and Implementation, 2006. SIPS '06. IEEE Workshop on*, 2006, pp. 142–147.

[120] K. Chugg, P. Thiennviboon, G. Dimou, P. Gray, and J. Melzer, "New class of turbo-like codes with universally good performance and high-speed decoding," in *Military Communications Conference, 2005. MILCOM 2005. IEEE*, 2005, pp. 3117–3126 Vol. 5.

[121] B. Frey and D. MacKay, "Irregular turbocodes," in *Information Theory, 2000. Proceedings. IEEE International Symposium on*, 2000, pp. 121–.

[122] H. Ma and J. Wolf, "On tail biting convolutional codes," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 34, no. 2, pp. 104–111, 1986.

[123] T. Miyauchi, K. Yamamoto, T. Yokokawa, M. Kan, Y. Mizutani, and M. Hattori, "High-performance programmable siso decoder vlsi implementation for decoding turbo codes," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, vol. 1, 2001, pp. 305–309 vol.1.

[124] T. Vogt, C. Neeb, and N. Wehn, "A reconfigurable multi-processor platform for convolutional and turbo decoding," in *Reconfigurable Communication-centric SoCs (ReCoSoC)*, Montpellier, France, 2006.

[125] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn, "A scalable system architecture for high-throughput turbo-decoders," *The Journal of VLSI Signal Processing*, vol. 39, no. 1, pp. 63–77, Jan. 2005.

[126] E. Boutillon, C. Douillard, and G. Montorsi, "Iterative decoding of concatenated convolutional codes : Implementation issues," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1201–1227, 2007.

[127] D. Gnaedig, "High-speed decoding of convolutional turbo codes," Ph.D. dissertation, Université de Bretagne Sud, ENST Bretagne, Turboconcept, 2005.

[128] H. Moussa, O. Muller, A. Baghdadi, and M. Jézéquel, "Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, 2007, pp. 1–6.

[129] "Dn9000k10pci xilinx virtex-5 based asic prototyping engine." [Online]. Available : http://www.dinigroup.com/DN9000k10PCI.php

[130] J.-L. Danger, A. Ghazel, E. Boutillon, and H. Laamari, "Efficient fpga implementation of gaussian noise generator for communication channel emulation," in *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conference on*, vol. 1, 2000, pp. 366–369 vol.1.

[131] D.-U. Lee, W. Luk, J. Villasenor, G. Zhang, and P. Leong, "A hardware gaussian noise generator using the wallace method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 911–920, 2005.

[132] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *Information Theory, IEEE Transactions on*, vol. 42, no. 2, pp. 429–445, 1996.

[133] Y. Wu, W. Ebel, and B. Woerner, "Forward computation of backward path metrics for map decoder," in *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, vol. 3, 2000, pp. 2257–2261 vol.3.

[134] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders," Jet Propulsion Laboratory, California Institute of Technology, Tech. Rep., 2000.

[135] F. Zhai and I. Fair, "Techniques for early stopping and error detection in turbo decoding," *Communications, IEEE Transactions on*, vol. 51, no. 10, pp. 1617–1623, 2003.

[136] A. Reid, T. Gulliver, and D. Taylor, "Convergence and errors in turbo-decoding," *Communications, IEEE Transactions on*, vol. 49, no. 12, pp. 2045–2051, 2001.

[137] D.-H. Kim and S. W. Kim, "Bit-level stopping in turbo decoding," in *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, vol. 3, 2003, pp. 2134–2138 vol.3.

Liste des publications

Articles de revue avec comité de lecture

[1] O. Muller, A. Baghdadi, and M. Jézéquel. Bandwidth reduction of extrinsic information exchange in turbo decoding. In *Electronics Letters*, volume 42, pages 1104–1105, Sept. 2006.

[2] Olivier Muller, Amer Baghdadi, and Michel Jézéquel. Parallel convolutional turbo decoding and interleaver design. *prepared and to be submitted shortly.*

[3] Olivier Muller, Amer Baghdadi, and Michel Jézéquel. From parallelism levels to a multi-ASIP architecture for turbo decoding. *IEEE Transactions on Very Large Scale Integration Systems*, 2007. Accepted for publication.

Conférences

[1] H. Moussa, O. Muller, A. Baghdadi, and M. Jézéquel. Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[2] O. Muller, A. Baghdadi, and M. Jézéquel. ASIP-based multiprocessor soc design for simple and double binary turbo decoding. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1–6, 6-10 March 2006.

[3] O. Muller, A. Baghdadi, and M. Jézéquel. Exploring parallel processing levels for convolutional turbo decoding. In *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, volume 2, pages 2353–2358, 24-28 April 2006.

[4] O. Muller, A. Baghdadi, and M. Jézéquel. Parallélisme et turbocodes convolutifs. In *MajecSTIC 2006 : MAnifestation des JEunes Cherchercheurs STIC, 22-24 novembre, Lorient, France*, 2006.

[5] O. Muller, A. Baghdadi, and M. Jézéquel. Flexible multi-ASIP SoC for high-throughput turbo decoders. In *Premier Colloque National du GDR SOC-SIP , 13-15 juin, Paris, France*, 2007.

[6] Olivier Muller, Amer Baghdadi, and Michel Jézéquel. On the parallelism of convolutional turbo decoding and interleaving interference. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–5, 2006.

RESUME

Les applications dans le domaine des communications numériques deviennent de plus en plus complexes et diversifiées. En témoigne l'apparition des turbo-communications qui représentent la généralisation du principe de processus itératif introduit par les turbocodes. La mise en œuvre de systèmes de turbo-communications, communément appelés turbo-récepteurs, est devenue primordiale pour atteindre les performances aujourd'hui exigées en terme de qualité de transmission. Des architectures matérielles dédiées implantant ces systèmes ont déjà vu le jour dans plusieurs équipes de recherches académiques et industrielles. Cependant, pour des exigences de flexibilité de l'implantation (pour supporter les évolutions d'une norme ou des applications multi-standards), de qualité de transmission et de haut débit de communication, des architectures multiprocesseurs adéquates deviennent incontournables.

Le sujet de cette thèse porte sur la mise en œuvre d'une plate-forme architecturale multiprocesseur générique adaptée aux turbo-récepteurs et plus particulièrement aux turbo-décodeurs convolutifs. Ainsi, le sujet gravite autour de deux axes de recherche : un axe algorithmique autour des systèmes de turbo-décodage et un autre autour de la conception numérique ces derniers.

Sur l'axe algorithmique, ces travaux présentent une étude approfondie des algorithmes de turbo-décodage autour des techniques de parallélisme. Les fondations de cette étude reposent sur une classification des parallélismes existants qui distingue les parallélismes selon leurs granularités et leurs pouvoirs d'accélération. L'analyse de cette classification a révélé la nécessité d'investiguer les parallélismes de sous-bloc et de décodeur composant pour améliorer l'efficacité de leur mise en œuvre. Les recherches menées mettent en évidence que le parallélisme de sous-bloc s'avère plus efficace avec la technique d'initialisation par passage de message. Nous avons également montré que le parallélisme de décodeur composant, grâce à la technique du décodage combiné ou « shuffled decoding » , améliore l'efficacité des architectures de turbo-décodeur fortement parallèles et que cette dernière peut être optimisée en contraignant la conception de l'entrelaceur du turbocode.

Sur l'axe architectural, ces avancées algorithmiques ont été mises à profit dans une plate-forme multiprocesseur qui exploite au mieux les compromis matériel/logiciel (i .e. performance/flexibilité) tant au niveau du calcul qu'au niveau des communications. Au niveau du calcul, un processeur ASIP (Application-Specific Instruction-set Processor) dédié au décodage des codes convolutifs a été proposé et conçu de manière à ne fournir que la flexibilité désirée, tout en conservant des performances élevées grâce à un chemin de données fortement parallélisé. Au niveau des communications, la plate-forme a été dotée de réseaux sur puce dédiés pour assurer la bande passante nécessaire aux échanges itératifs d'information. Cette plate-forme multi-ASIP flexible a été prototypée sur une carte d'émulation intégrant des circuits FPGA.

La flexibilité de la plate-forme proposée autorise le support de tous les standards de turbocodes convolutifs actuels et émergeants et peut trouver un intérêt industriel dans les domaines des télécommunications mobiles et satellitaires, de la diffusion de contenu ou de l'Internet haut-débit.

MOTS-CLES : Turbocodes, multiprocesseur, processeur dédié, parallélisme

TITLE : Generic multiprocessor architectures for high-throughput turbo-communications

ABSTRACT

Applications in the field of digital communications are becoming more and more diversified and complex. This trend is driven by the emergence of turbo-communications which generalize the principle of iterative processing introduced by the turbo-codes. Implementation of turbo-communication systems, so-called turbo-receivers, is becoming crucial to reach the nowadays performance requirements in terms of transmission quality. Several dedicated implementations of these systems have already been proposed. However, implementation requirements in flexibility (to support the continuously developing new standards and applications in this field) and in high-throughput, make resorting to new design methodologies and the proposal of a flexible turbo communication platform inevitable.

The subject of this thesis deals with the implementation of a generic multiprocessor platform dedicated to turbo-receivers et more specifically to convolutional turbo decoders. Thus, the subject evolves around two research areas : algorithmical aspects of turbo-decoding systems and architectural aspects of their numeric design.

Concerning the algorithmical part, this work presents a wide range of investigations of parallelism in convolutional turbo decoding. These investigations are based on three-level classification of parallelism techniques, which is constructed according to their granularity and acceleration abilities. Analysis of this classification reveals that sub-block parallelism and component-decoder parallelism need efforts to improve the implementation efficiency. Our researches enlighten that sub-block parallelism becomes more efficient with the message passing initialisation technique. Additionally, we show that component-decoder parallelism with shuffled decoding improves the efficiency of highly parallelized turbo-decoder architecture. Furthermore, we present how to optimise this efficiency through constraints interleaver design.

Concerning the architectural part, algorithmical results were integrated in a multiprocessor platform exploiting the tradeoffs between hardware and software (i.e. performance/flexibility) at processing and communication levels. To cope with processing tradeoffs, we propose a Application-Specific Instruction-set Processor (ASIP) dedicated to turbo-decoding of convolutional codes. The designed ASIP provides the required flexibility while enabling high-throughput thanks to a highly parallelized datapath. At the communication level, our multi-ASIP platform exploits dedicated network on chip in order to ensure the bandwidth required for iterative exchange of information. The resulting multi-ASIP platform was prototyped on emulation board based on FPGA.

The flexibility of the proposed platform enables the support of all existing and emerging standards of convolutional turbocodes, and have industrial applications in mobile and satellite-based communications, broadcasting, Internet high-throughput.