الجمـــهوريـــة الجـــزائريـــة الديـمقراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT

كـلـيـــــة الـعـلـــــوم
FACULTE DES SCIENCES

قســـــم الرياضيات و الإعلام الآلي
DEPARTEMENT DE MATHEMATIQUE ET INFORMATIQUE

Mémoire de MASTER 2

Domaine : Mathématique informatique
Filière : Informatique
Option : Réseaux, Systèmes et Applications Reparties

Par :
Sersab Fattoum

THEME

Implémentation Parallèle des Codes Turbo

Soutenu publiquement devant le jury composé de :

Président : Mr.Allaoui.T
Examinateur : Mr.Regab.M
Examinateur : Mr.Chaib.N
Encadreur : Mme. Chouireb.F
Co-encadreur : Mr. Chellali.S

Année Universitaire 2011 – 2012

RESUME

Résumé - Dans la correction des erreurs (FEC), les turbo-codes convolutifs ont été introduits pour augmenter la capacité de correction d'erreur. Cependant, le décodage de ces codes est un processus itératif nécessitant un temps de calcul très important. Ainsi, afin de parvenir à un débit élevé et réduire la latence, on doit avoir recours à la parallélisation de ces turbocodes.

Dans ce mémoire, nous avons commencé par décrire les bases théoriques des turbocodes convolutifs, ainsi que leurs algorithmes de décodage, et en particulier l'algorithme MAP. Ensuite, nous avons décrit une famille des turbocodes à roulettes permettant la parallélisation du processus de décodage. Cette méthode est basée sur deux idées principales : le codage de chaque dimension par P codes convolutifs récursifs circulaires indépendants et des contraintes sur la structure de l'entrelaceur qui permet de décoder en parallèle les P codes convolutifs dans chaque dimension. La construction des codes constituants et de l'entrelaceur est bien détaillée et analysée dans ce travail. Les résultats obtenus sont satisfaisant et prometteurs.

Mots clés: *Code correcteur d'erreurs, Turbo codes convolutif, Parallèle, Turbo codes à roulettes(CRSC) , Haut débit.*

Abstract - In forward error correction (FEC), convolutional turbo codes were introduced to increase error correction capability. The decoding of these codes is an iterative process requiring high computation rate and latency. Thus, in order to achieve high throughput and to reduce latency, crucial in emerging digital communication applications parallel implementations.

In this memoire, we describe the basics of turbo convolutional code and its decoding MAP algorithm. After that, we explain the principal details of a technique called slice turbo codes, this family is based on two ideas: the encoding of each dimension with P independent tail-biting codes and a constrained interleaver structure that allows parallel decoding of the P independent codewords in each dimension. The interleaver and the P decoders components are widely described in this work. Results of parallel turbocode implementation are well satisfactory.

Key words: *Forward error correction, Convolutif turbo codes, Parallel, Slice turbo codes(CRSC), High speed.*

DEDICACE

Je Dédie Ce Mémoire A :

Mes Chers Parents

&

Ma Famille Sersab.

REMERCIEMENT

Merci au bon dieu qui nous a permis de réaliser modeste travail, Je tiens à remercier à notre promotrice ***Mme.CHouireb Fatima****, de nous avoir proposé ce sujet, de nous avoir encadrées, pour ses remarques pertinentes et ses conseils judicieux qu'elle nous a octroyés le long de notre travail.*

Nos sincères remerciements vont aussi au ***Mr.Chellali.S****,* ***Mr.Nouri.N*** *et* ***Mr. Mokhtari.A****.*

J'exprime se remercie aux membres de jury pour avoir accepté de juger ce travail.

Nos remerciements à tous les enseignants qui ont dirigé et conseillés durant notre formation.

En fin, se profondes salutation à la promotion de 2eme année master, et à tous ceux qui je aide de prés ou de loin dans ce travail.

TABLE DES MATIERES

LISTE DES FIGURES ET TABLEAUX

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

ALU	**A**rithmétique **L**ogique **U**nité
APP	**A P**osteriori **P**robabilté
ASK	**A**mplutide **S**hift **K**eying
ARP	**A**lmost **R**egular **P**ermutation
AWGN	**A**dditive **W**hite **G**aussian **N**oise
BPSK	**B**inary **P**hase **S**hift **K**eying
BCJR	**B**ahl **C**ock **J**elinek **R**aviv
CDC	**C**ontrol **D**ata **C**oorporation
CRSC	**C**onvolutifs **R**écursifs **S**ystématiques **C**irculaires
DRP	**D**ithered **R**elative **P**rime
FEC	**F**orward **E**rror **C**orrection
LLR	**L**og **L**ikelihood **R**atio
ML	**M**aximum **L**ikelihood
MAP	**M**aximum **A P**osteriori
MIMD	**M**ultiple **I**nstruction **M**ultiple **D**ata
MISD	**M**ultiple **I**nstruction **S**ingle **D**ata
MPMD	**M**ultiple **P**rogram **M**ultiple **D**ata
PSK	**P**hase **S**hift **K**eying
QAM	**Q**uadrature **A**mplitude **M**odulation
QPP	**Q**uadratic **P**ermutation **P**olynomial
RSC	**R**écursive **S**ystématique **C**ode
SISD	**S**ingle **I**nstruction **S**ingle **D**ata
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SNR	**S**ignal to **N**oise **R**atio
SPMD	**S**ingle **P**rogram **M**ultiple **D**ata
SISO	**S**oft **I**nput **S**oft **O**utput
SOVA	**S**oft **O**utput **V**iterbi **A**lgorithm
TEB	**T**aux d'**E**rreur **B**inaire

NOTATIONS

CODAGE ET DECODAGE

V	Longueur du registre à décalage du code convolutif.
R	Rendement du code.
Π	Entrelaceur, fonction d'ntrelacement.
K	Longueur de la séquence d'information d.
M	Nombre de bits du symbole à coder à chaque instant i.
N	Bits de sortie $n > m$.
d'	Séquence d'estimation de d .
U	Séquence reçue correspondant à la séquence émise d.
$\mathbf{v_1}$	Séquence reçue correspondant à la séquence de la redondance émise y_1.
$\mathbf{v_2}$	Séquence reçue correspondant à la séquence de la redondance émise y_2.
y_1	Séquence de la redondance sortie du premier codeur.
y_2	Séquence de la redondance sortie du deuxième codeur.
d^i	Bit d'information à coder à l'instant i.
y_1^i	Bit de redondance à l'instant i du premier codeur.
y_2^i	Bit de redondance à l'instant i du deuxiéme codeur.
$\alpha^i(s')$	Métrique récurrente aller à l'instant i de l'état s'.
α^i	Vecteur de la métrique récurrente aller à l'instant i.
$\beta^{i+1}(s)$	Métrique d'état récurrent retour à l'instant i+1 de l'état s.

β^{i+1}	Vecteur du métrique récurrent retour à l'instant i+1.
$\gamma^i(s', s)$	Métrique de branche de l'état s' à l'instant i à l'état s à l'instant i+1.
γ^i	Vecteur de la métrique de branche à l'instant i.
$\gamma_e^i(s', s)$	Métrique de branche extrinsèque de l'état s' à l'instant i à l'état s à l'instant i+1.
γ_e^i	Vecteur de la métrique de branche extrinsèque à l'instant i.
P_{Eb}	Probabilité d'erreur par élément binaire transmis.
$P(x)$	Probabilité d'un symbole x .
$P_e(x)$	Probabilité extrinsèque d'un symbole x.
$P_a(x)$	Probabilité a priori d'un symbole x.
σ	Déviation du bruit.
s	Etat du code convolutif.
s^i	Etat du code convolutif à l'instant i.
s^c	Etat circulaire.
$LLR_{1\to2}^e$	L'information extrinséque fournie par le premier décodeur $SISO_1$ et transmise au seconde décodeur $SISO_2$
$\mathbf{A}$	Matrice d'état du codeur.
$\mathbf{B}$	Matrice d'entrée du codeur.
$\mathbf{I}$	Matrice identité.

MATHEMATIQUES

exp (x)	Opération exponentielle de x avec $x \epsilon \mathbb{R}$.
log (x)	Opération logarithmique de x avec $x \epsilon \mathbb{R}$.
$\langle a, b \rangle$	Multiplication vectorielle entre deux vecteurs a et b.
$\|a - b\|$	Distance entre deux vecteurs a et b.

INTRODUCTION GENERALE

Un système de communications numériques est un moyen de transport de l'information d'un usager à un autre en général éloigné. Il consiste à faire subir à l'information émise plusieurs opérations comme le codage de source, le cryptage et le codage du canal avant la transmission. Le dernier type de codage sert à minimiser l'impact du bruit, ce type de codage est aussi appelé codage correcteurs d'erreurs.

Les codes correcteurs d'erreurs (codage canal) sont une des solutions permettant d'améliorer la qualité des communications numériques. Le principe du codage canal est d'introduire de la redondance dans la séquence d'information binaire afin de corriger les erreurs de transmission durant la réception de l'information. Deux grandes classes des codes correcteurs d'erreurs existent : les codes convolutifs et les codes en blocs. Au début des années 90, une nouvelle famille de code correcteur d'erreurs a été découverte par Claude Berrou et Alain Glavieux : les turbocodes.

Les turbocodes sont d'excellents correcteurs d'erreurs, ils ont actuellement un pouvoir de correction inégalé mais en contre partie ils sont couteux en temps de traitement et en complexité. Le principe du turbo codeur est l'utilisation conjointe de deux codeurs convolutifs récursifs en série ou en parallèle et d'un entrelaceur qui consiste à permuter une séquence de bits de manière à ce que deux symboles proches à l'origine soient le plus éloignés possibles l'un de l'autre après permutation. Si le codage est relativement simple, le décodage est plus complexe, les turbo décodeurs sont construits par une association de deux décodeurs composants simples connectés par une fonction de permutation temporelle. Ces décodeurs composants collaborent par un échange itératif des messages probabilistes associés aux symboles reçus afin de corriger des erreurs de transmission.

Le turbo décodage est de plus en plus proposé dans les nouvelles et les futurs systèmes de communication numériques, par exemple, les applications de communication à fibre optiques RAO permettent déjà dans certains pays (Japon, Corée, Etats-Unis) de disposer d'une connexion internet à très haut débit en prolongeant la fibre optique jusque chez l'abonné. L'implantation de tels réseaux nécessite des infrastructures à faibles coûts qui dégradent la qualité des signaux transmis. Pour faire face à ces exigences, Les implémentations turbo-décodeur doivent être massivement parallèles.

Dans ce contexte, l'objectif principal de ce travail est l'implémentation parallèle des codes turbo afin réduire le temps de traitement du turbo décodage et par conséquent améliorer le temps de communication qui est un facteur vital surtout dans certains services comme la tété-opération (télé-chirurgie, tété-navigation,…).

Pour se faire, nous avons organisé ce mémoire en trois chapitres :

- Le premier chapitre vise à définir le domaine d'application considérée dans le cadre de ce travail, et plus spécifiquement les codes correcteurs d'erreurs ECC. Dans un premier temps, nous nous intéressons à préciser la place de l'application dans une chaîne de transmission numérique. Nous détaillons ensuite le principe des codes convolutifs, ainsi que et leurs algorithmes de décodage, en particulier l'algorithme de veterbi qui est un algorithme très utilisé. les turbocodes et leur algorithme de décodages Maximum A Posteriori (MAP) seront décrits dans une deuxième partie de ce chapitre.
- Dans le deuxième chapitre, la première section expose les notions générales sur le parallélisme ainsi que ses caractéristiques principales. Ensuite, dans la deuxième section, nous parlons de la programmation parallèle sous Matlab et nous donnons quelques exemples simples de programmes parallèles. Dans la dernière partie de ce chapitre, nous détaillons une solution architecturale reposant sur un turbo décodage entièrement parallèle ; c'est le turbo code à roulette.
- Une mise en application de cette étude théorique, les résultats de simulation, les commentaires et l'interprétation apparaitrons dans le dernier chapitre.

A la fin de ce mémoire, nous terminerons notre travail par une conclusion générale et nous donnerons une idée sur les perspectives.

LES CODES CORRECTEURS D'ERREURS : TURBO-CODES

1.1 Introduction

La communication numérique est définie comme étant l'infrastructure nécessaire à la transmission fiable d'un message à partir d'une source vers un destinataire. Dans ce contexte, nous nous limitons ici à l'essentiel de la modélisation d'une chaîne de communication numérique et nous insistons plus particulièrement sur les aspects du codage canal. Dans ce chapitre, nous définissons dans un premier temps le modèle d'une chaîne de communication numérique en décrivant brièvement ses principaux composants. Ensuite, nous allons détailler les principes des codes convolutifs et des codes turbo, et étudier l'algorithme de décodage itératif des turbocodes convolutifs.

1.2 Modèle d'une chaîne de communication numérique

Une chaine de communication permet de transmettre une quantité d'information donnée avec le moins d'erreurs possible. Les données peuvent être numériques – un fichier texte ou une image, ou analogiques – une source sonore. Une chaîne de transmission modélise les différentes étapes autorisant le transfert d'une information d'une source vers un destinataire. 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. A 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. Un système de transmission numérique peut être schématisé de la façon suivante (figure 1.1) :

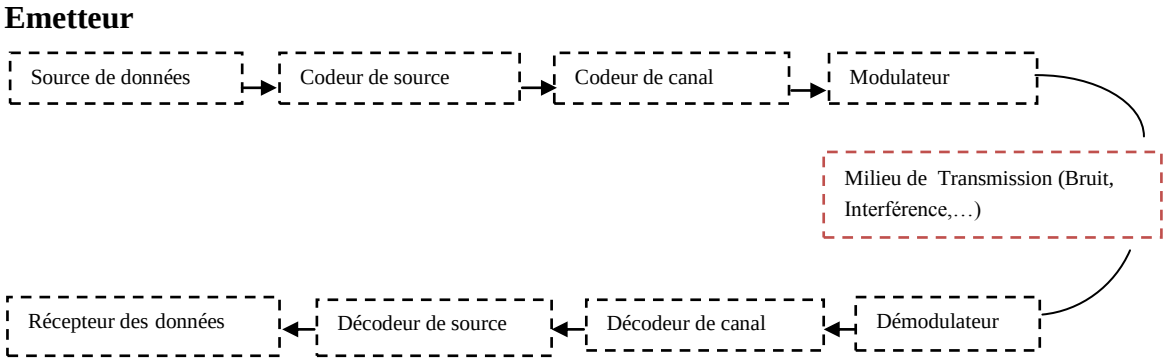


Figure 1.1: Modèle de communication numérique.

1.2.1 Codage et décodage de source

Le codage source consiste d'une part à convertir le message continu en une séquence numérique par un convertisseur analogique numérique et à transformer le message de la source en une séquence d'information de façon à minimiser la taille du message en éliminant les redondances naturelles de l'information source. Comme exemple de codage source, nous citons le codage de Huffman. Le but de cette opération est d'optimiser les ressources nécessaires à la transmission (temps, puissance, bande passante, etc.). Le décodage source consiste à reconstituer, par l'application de l'algorithme de décodage source (décompression), l'information originale [1].

1.2.2 Codage et décodage de canal

Le codage par codes correcteurs d'erreurs FEC (Forward Error Correction) appelé également 'codage canal' est utilisé afin d'ajouter de la redondance à l'information transmise dans le but de la protéger contre les perturbations survenues lors de la manipulation physique de l'information sur le canal. Le taux d'erreurs binaire TEB d'un message est le rapport du nombre de bits erronés par le nombre de bits du message. En 1948, Shannon énonce le théorème fondamental de la théorie de l'information qui associe à chaque canal une capacité représentant le maximum d'information transmissible sur ce canal (exprimé en bits par seconde). « *Si le débit d'information à l'entrée du canal est inférieur à la capacité, alors il est possible de transmettre le contenu de l'information sur le canal avec une probabilité d'erreur aussi petite que souhaitée* » [2].

Ce théorème est d'une importance fondamentale dans le monde des communications numériques. Il donne une limite supérieure que l'on peut atteindre en termes de taux de transmission des données faibles pour un canal de transmission. Shannon a utilisé un code *aléatoire* de longueur infinie. En pratique, ce code n'existe pas. Depuis, les chercheurs s'efforcent de construire des codes correcteurs d'erreurs de longueurs finies s'approchant du code aléatoire, pour être capable d'atteindre au maximum la limite du théorème de Shannon.

Il y a deux grandes familles de code :

- **les codes en blocs**, qui assurent une indépendance du codage à chaque bloc.
- **les codes convolutifs**, 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.

Au niveau du récepteur, Le décodage canal consiste dans un premier temps à détecter la présence d'erreurs dans l'information et puis dans un deuxième temps de les corriger.

1.2.3 Modulateur et démodulateur

La modulation est la transformation d'un message à transmettre en un signal adapté à la transmission sur un support physique. Le modulateur génère un signal porteur, dont la forme d'onde peut être soit une suite d'impulsions soit une onde sinusoïdale.

Le message à moduler est numérique, nous parlons alors de 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 $T = \frac{mR}{D_b}$, où R représente le rendement qui est défini par :

$$R = \frac{nombre\ de\ bits\ d'information}{nombre\ de\ bits\ du\ code\ transmis}$$

Dans le cas d'une onde sinusoïdale, le message peut être porté par la phase PSK (Phase Shift Keying), amplitude ASK (Amplitude Shift Keying), fréquence FSK et amplitude et phase QAM (Quadrature Amplitude Modulation).

Les objectifs de la modulation sont:

- une transposition dans un domaine de fréquences adapté au support de transmission.
- une meilleure protection du signal contre le bruit.
- une transmission simultanée de messages dans les bandes de fréquences adjacentes, pour une meilleure utilisation du support.

A la sortie du canal, Le démodulateur effectue le travail inverse du modulateur, c'est-à-dire qu'il convertit le signal reçu en train binaire.

1.2.4 Canal de transmission

Le canal ou milieu de transmission correspond au lien physique qui relie le modulateur et le démodulateur. Il peut s'agir d'un câble coaxial, d'une fibre optique, de l'espace libre, d'un support de stockage de données, etc. Dans le milieu de transmission, le signal modulé subit des perturbations telles que du bruit (le canal Gaussien), des atténuations (le canal de Rayleigh), des interférences entre symboles (les canaux multi_trajets) ou bien encore des interférences entre utilisateurs (canal multi-utilisateurs).

Ces perturbations modifient les propriétés physiques du signal transmis et génèrent potentiellement des erreurs de transmission.

Le canal de transmission est caractérisé par sa capacité et par sa bande passante [3]. Chaque bit reçu a une probabilité p d'être faux et $1-p$ d'être correct, sont présentées par son graphe de transition sur la figure 1.2.

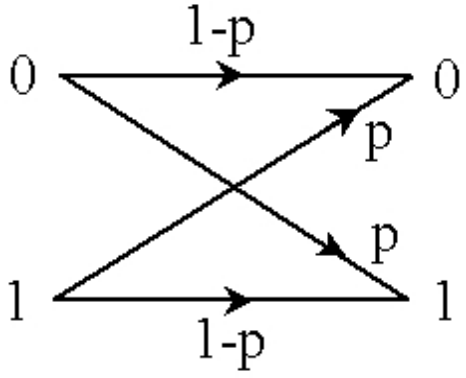


Figure 1.2 : Canal binaire symétrique de probabilité d'erreur p .

1.2.5 Performance : Gain de codage

La qualité d'une transmission numérique est caractérisée par la probabilité d'occurrence d'erreur par élément binaire transmis, elle est notée P_{Eb} . Cette probabilité est fonction du rapport signal sur bruit SNR (Signal to Noise Ratio) E_b/B . Le tracé de la courbe reliant les points de la P_{Eb} (à rapport signal/bruit donné) reflète directement la qualité de la transmission. L'obtention de ces points nécessite des simulations complexes qui permettent de mesurer le P_{Eb} à travers le taux d'erreur binaire (TEB). L'estimation du TEB est obtenue par simulation de la transmission de N symboles binaires et l'évaluation après décodage du rapport n_e / N où n_e est le nombre de symboles erronés après décodage en réception. La pratique a montré que l'obtention d'au moins une centaine d'erreurs est nécessaire pour avoir une estimation correcte de la P_{Eb} [4].

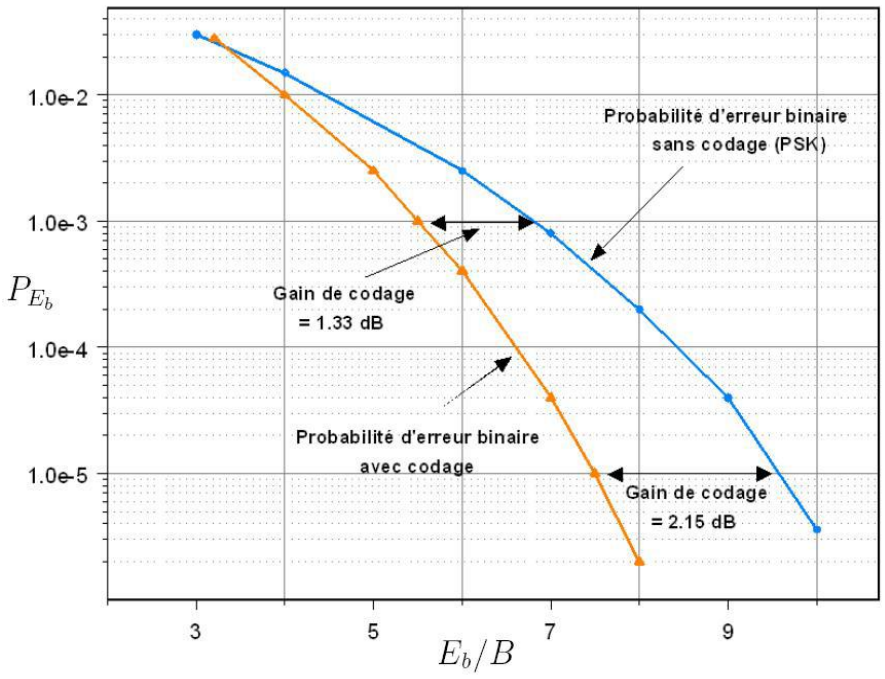


Figure 1.3 : P_{Eb} en fonction du E_b/B pour modulation PSK.

L'efficacité du code correcteur d'erreur est déterminée en effectuant la comparaison des courbes en sortie du décodeur de canal et en sortie du canal (en l'absence de codage). La distance entre les deux tracés donne le gain de codage G (figure 1.3), il s'exprime en décibel dB. G représente l'économie d'énergie induite par l'utilisation d'un codage canal. Il est également possible de le voir comme une amélioration de la qualité de transmission.

1.3 Les turbocodes convolutifs

1.3.1 Codes convolutionnels

Le principe des codes convolutifs, inventés par Peter Elias en 1954, est non plus de découper le message en blocs finis, mais de le considérer comme une séquence semi-infinie de symboles. Un code convolutif de rendement $R = \frac{m}{n}$ est une fonction linéaire qui transforme à chaque instant i un vecteur d'entrée d^i de m bits en 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 $s^i = s_1^i s_2^i \dots s_v^i$ définit un état du codeur parmi les 2^v états possibles. Les valeurs de sorties sont obtenues par combinaison linéaire (avec l'opérateur « ou exclusif » comme additionneur) des bits d'entrées à l'instant *i*, d^i , et de mémorisation s_k^i. La figure 1.4.a montre un exemple simple de codeur convolutif ou d^i est le bit d'entrée et $C^i = \left(c_1^i c_2^i\right)$ est le mot de code obtenu à la sortie de ce codeur.

Un code convolutif 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 (figure1.4.b), le mot codé par ce schéma à l'instant *i* est défini par $C^i = (d^i y^i)$.

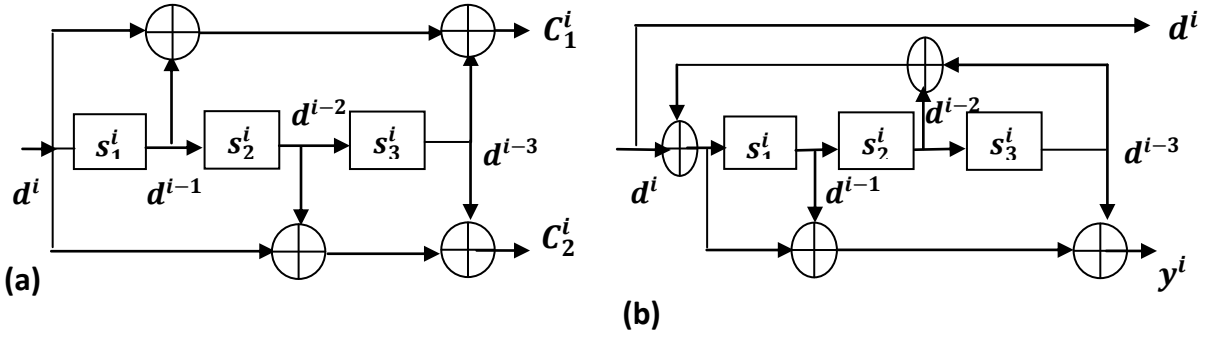


Figure 1.4: Code convolutif :
(a) code convolutif non systématique et non récursif, (b) code convolutif systématique récursif (RSC).

Il existe trois manières de représenter un code convolutif : diagramme en arbre, en treillis et diagramme d'états, **La représentation en treillis** est la plus utilisée, elle met en évidence le paramètre temporel. Chaque nœud $s^i = s_1^i s_2^i \dots s_v^i$ correspond à un état particulier du codeur à un instant i où i représente l'indice du temps. Chaque branche est indexée par les bits de sortie du codeur $C_1^i C_2^i$. Si le bit d'entrée d^i =1 la branche sera représentée par un trait continu, sinon elle sera représentée par un trait discontinu. Comme nous le verrons dans la figure 1.5.

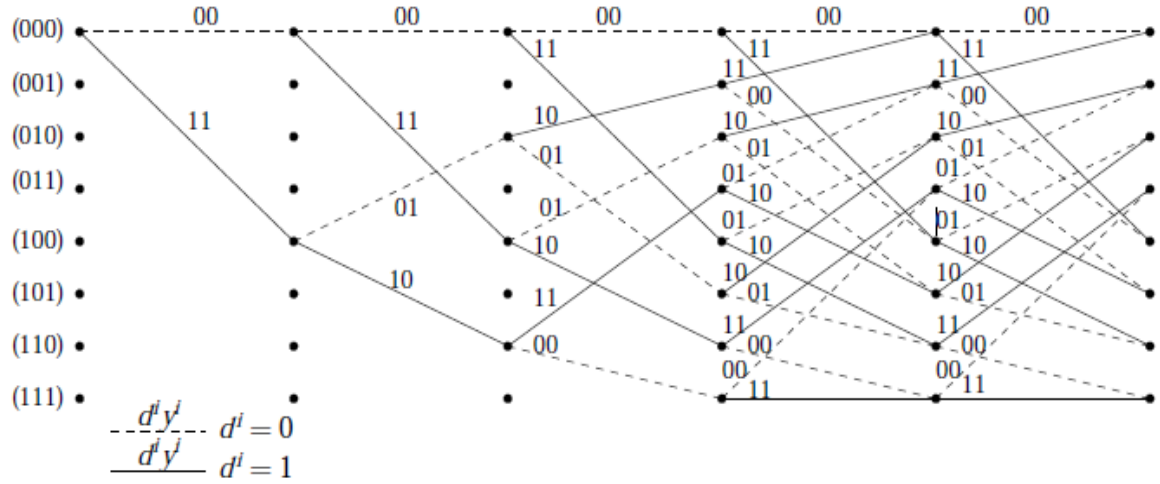
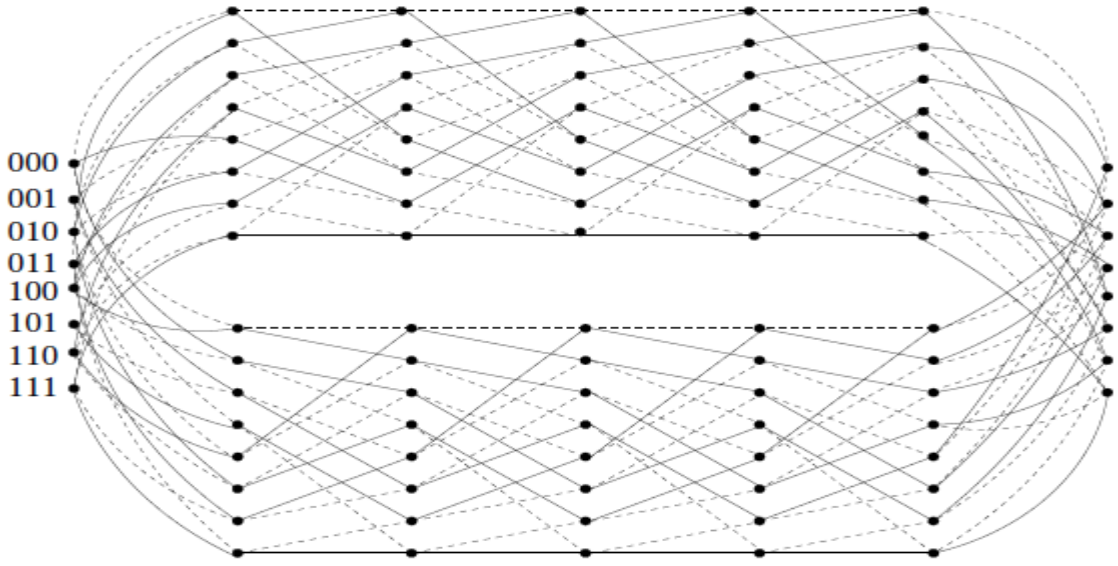


Figure 1.5 : Diagramme en treillis associé au codeur convolutif de la figure 1.4.b.

Pour les codes utilisés en pratique, le rendement naturel d'un code convolutif est assez faible et généralement inférieur à 2/3. Pour autoriser des rendements plus élevés, on utilise **la technique du poinçonnage (perforation)** 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 un bit de code qui est exclu ou perforé et un '1' un bit de code qui est inclus dans la séquence de bits transmis. Ainsi, un code de rendement 1/2 poinçonné avec le motif $\begin{matrix} C_1^i \\ C_2^i \end{matrix} \begin{pmatrix} 1\ 1\ 0 \\ 1\ 0\ 1 \end{pmatrix}$ aboutit à un rendement de 3/4. L'avantage principal de cette technique se situe au niveau du décodeur dont la structure est inchangée. La seule contrainte consiste à dépoinçonner le code en amont du décodeur en remplaçant les données poinçonnées par des valeurs neutres pour le décodeur.

Fermeture de treillis ; un mot de code convolutif n'a pas de longueur fixe, cependant pour des raisons pratiques dans la plupart des applications il est nécessaire d'imposer une longueur maximale fixe pour le mot de code. A cet effet, il y a deux manières, la première technique consiste à forcer le codeur à retourner dans un état final connu en ajoutant à la séquence d'information m bits appelés bits de queue et dont les valeurs sont nulles. Cette méthode a l'inconvénient d'impliquer une diminution du rendement. La deuxième méthode évite ce problème en rendant le code circulaire, il ne suffit plus de rajouter une séquence de m bits mais de les calculer de façon à mettre le registre d'état à zéro [5].

Codes convolutifs systématique récursifs circulaires ; une technique pour terminer les treillis des codes convolutifs sans les effets de bord : le tail-biting. Elle consiste à rendre le treillis de décodage circulaire, c'est-à-dire à faire en sorte que l'état de départ et l'état final du codeur soient identiques. Cet état est alors appelé état de circulation. Cette technique de fermeture appliquée aux codes récursifs génère un code dit convolutif systématique récursif circulaire (CSRC), le treillis d'un tel code est illustré par la figure 1.6 [6].



,

Figure 1.6 : Treillis d'un code CRSC à 8 états.

Pour expliquer la construction d'un tel code, il est nécessaire de revenir aux équations fondamentales qui régissent le fonctionnement d'un codeur. Il est possible d'établir une relation entre l'état s^{i+1} du codeur à un instant *i+1,* son état s^i et la donnée entrante d^i à un instant *i* :

$$\mathrm{s}^{\mathrm{i}+1} = \mathbf{A}\mathrm{s}^{\mathrm{i}} + \mathbf{B}\mathrm{d}^{\mathrm{i}} \tag{1.1}$$

Où **A** est la matrice d'état et **B** celle d'entrée. Dans le cas du code systématique récursif retenu dans la figure 1.4.b, les matrices **A** et **B** sont définies comme suit :

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad : \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.2}$$

Si le codeur est initialisé à l'état 0 (s^0=000), l'état final s_0^k obtenu à la fin d'une trame à longueur *k* est :

$$\mathrm{s}_0^{\mathrm{k}} = \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathbf{A}^{\mathrm{j}-1}\mathbf{B}\mathrm{d}^{\mathrm{k}-\mathrm{j}} \tag{1.3}$$

Lorsqu'il est initialisé dans un état quelconque s^c , l'état final s^k s'exprime de la manière suivante :

$$\mathrm{s}^{\mathrm{k}} = \mathbf{A}^{\mathrm{k}}\,\mathrm{s}^{\mathrm{c}} + \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathbf{A}^{\mathrm{j}-1}\mathbf{B}\mathrm{d}^{\mathrm{k}-\mathrm{j}} \tag{1.4}$$

Pour que cet état s^k soit égal à l'état de départ s^c et que celui-ci devienne donc l'état de circulation, il faut et il suffit que :

$$(\mathbf{I} - \mathbf{A}^{\mathbf{k}})\,\mathrm{s}^{\mathrm{c}} = \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathbf{A}^{\mathrm{j}-1}\mathbf{B}\mathrm{d}^{\mathrm{k}-\mathrm{j}} \tag{1.5}$$

Où **I** est la matrice identité de dimension $v \times v$.

Ainsi, en introduisant l'état s_0^k du codeur après initialisation à 0, l'état circulaire a l'expression :

$$s^c = \left(\mathbf{I} - \mathbf{A}^k\right)^{-1} s_0^k \qquad (1.6)$$

L'état circulaire est obtenu si seulement si la matrice ($\mathbf{I} - \mathbf{A}^k$) est *inversible.*

En conclusion, s'il existe, l'état circulaire s'obtient en deux étapes. La première consiste à coder la trame de k bits en entrée après avoir initialisé le codeur à l'état zéro et à conserver l'état de terminaison. La seconde se résume à déduire l'état de circulation du précédent état de terminaison et d'une table (obtenu par l'inversion de $\mathbf{I} - \mathbf{A}^k$).

Prenons pour exemple le code systématique récursif précédemment utilisé.

Ce type de code est récursif si ($\mathbf{I} - \mathbf{A}^k$) est inversible. Ce codeur possède la propriété que $\mathbf{A}^7$ soit égale à **I**. alors, si la longueur de trame est multiple de 7 (=$2^v - 1$), ($\mathbf{I} - \mathbf{A}^k$) devient une matrice nulle – non inversible. Pour les autres longueurs, la matrice ($\mathbf{I} - \mathbf{A}^k$) est non nulle, et sa matrice inverse existe. La table suivante résume les états circulaires du code-convolutif. Ils sont calculés en fonction de la longueur de la trame d'entrée et de l'état s_0^k correspondant. Ces états sont exprimés dans la base de 10.

K mod7 / s_0^k	1	2	3	4	5	6
0	0	0	0	0	0	0
1	6	4	6	2	5	7
2	3	5	4	6	7	1
3	5	1	7	4	2	6
4	7	2	1	5	6	3
5	1	6	2	7	3	4
6	4	7	5	3	1	2
7	2	3	6	1	4	5

Table 1.1 : Tableau des états circulaires du code convolutif présenté sur la figure 1.4.b.

Une méthode simple pour le codage selon un treillis circulaire se résumé en cinq étapes, après vérification de l'existence de l'état de circulation :

1. Initialiser le codeur à l'état 0 ;
2. Coder la trame pour obtenir l'état final s_0^k ;
3. Calculer l'état s^c de circulation à partir des tables déjà calculées et stockées ;
4. Initialiser le codeur à l'état s^c ;
5. Coder la trame et transmettre les redondances calculées.

1.3.1.1 Exemple numérique

a) Codeur des codes convolutionnels

Dans cet exemple (figure 1.7), à un élément binaire (é.b) d'entrée correspond deux éléments binaires de sortie c_1^i et c_2^i $(R = 1/2)$. La taille du registre est $v = 2$ et les bits d'entrées sont 1001. c_1^i et c_2^i sont déterminés par les équations suivantes :

$$c_1^i = d^i + d^{i-1} + d^{i-2}\ , \quad c_2^i = d^i + d^{i-2}$$

Soit en transformée en z :

$$C_1(z) = (1 + z^{-1} + z^{-2})d(z)\ , \qquad C_2(z) = (1 + z^{-2})d(z)$$

On remplace souvent z^{-1} par D.

On appelle les polynômes générateurs dans cet exemple : $1+ D+D^2$ et $1+D^2$

qui peuvent être exprimés en octal par : [111]=7, [101]=5

G(D) = (1 + D + D^2, 1 + D^2) est la matrice génératrice (on peut écrire également G = (7, 5) ou bien encore G = (111, 101)).

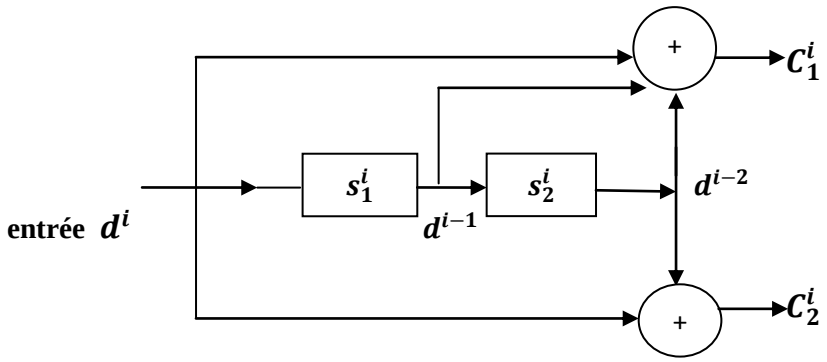


Figure 1.7 : Exemple simple de code convolutif (R=1/2, v =2).

Le tableau ci-dessous donne les résultats du codage obtenu pour le codeur convolutif de la figure 1.7. Quand on a en entrée la séquence de bits 1001 les bits qui vont être réellement transmis sont : 11101111.

Etape	Bit d'entrée	Etat du registre $s_1^i s_2^i$	Calcul	Bits de sortie
1	1	00	C_1^i=1+0+0 C_2^i=1+0	11
2	0	10	C_1^i=0+1+0 C_2^i=0+0	10
3	0	01	C_1^i=0+0+1 C_2^i=0+1	11
4	1	00	C_1^i=1+0+0 C_2^i=1+0	11

Table 1.2 : Résultat du codage du codeur convolutif de la figure1.7.

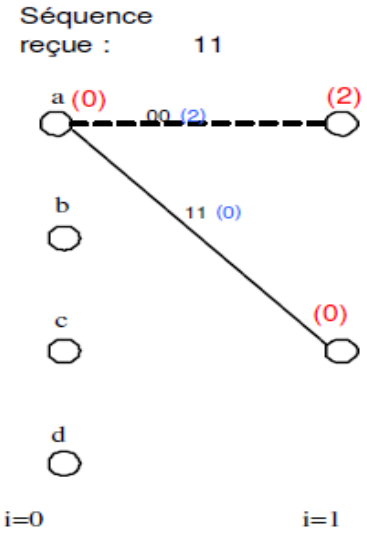
b) Décodage des codes convolutionnels : Algorithme de Viterbi

Le décodage le plus courant est basé sur l'algorithme de Viterbi qui est proposé en 1967 par A. J. Viterbi, il est basé sur le principe du maximum de vraisemblance. Cet algorithme est une méthode optimale de décodage pour les codes convolutifs. Il consiste à rechercher dans l'arbre le chemin qui correspond à la séquence la plus probable, c'est-à-dire celle qui est à la distance minimale de la séquence reçue ou encore la séquence la plus probable [7]. Dans l'exemple qui suit (basé sur le codeur déjà décrit) on suppose que le démodulateur fournit des 0 et des 1(décision dure). A chaque fois qu'un groupe de deux éléments binaires arrive, on examine toutes les branches possibles du treillis, on calcule la distance de Hamming[1] entre les éléments binaire affectés aux branches et les éléments binaire reçus, on ne garde que les branches donnant lieu à une distance minimale (ces branches forment le chemin survivant), et on affecte l'extrémité du chemin survivant d'une métrique égale à la somme de la métrique précédente et de la distance de la branche retenue.

[1] La distance de Hamming d(x,y) entre deux mots x et y est le nombre de positions de coordonnées qui diffèrent entre x et y. Exemple : A={0,1}, x=(10110) y=(11011) d(x,y)=3.

Dans notre exemple les données avant codage sont 1001, la séquence après codage est 11 10 11 11, alors que les données reçues sont 11 00 11 11, une erreur survient dans la transmission du troisième bit.

Les figures ci-dessous montrent les différentes étapes de décodage par l'algorithme de Viterbi. Sur la branche, est représentée la distance entre le couple d'é.b. reçu et codé en bleu. Sur le nœud nous avons le poids en rouge ; somme du dernier poids et de la distance.

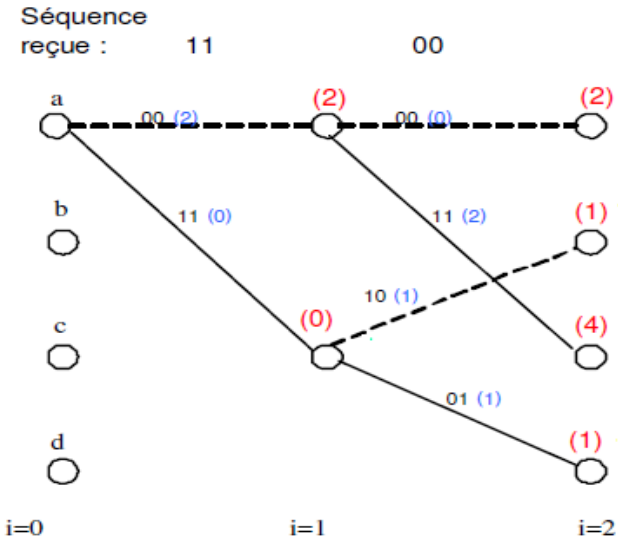


données avant codage : 1001

après codage : 11 10 11 11

erreur de transmission

données reçues : 11 00 11 11



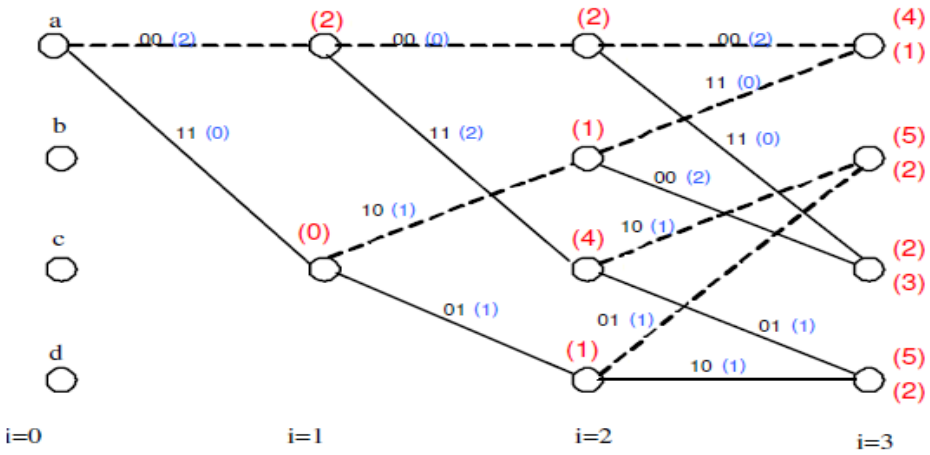
sur la branche :

distance entre le couple d'é.b. reçu et codé

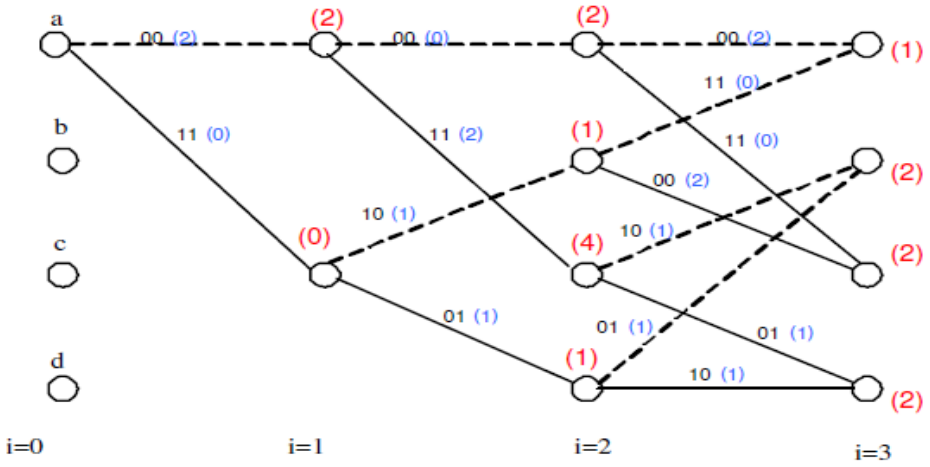
sur le nœud : poids

somme du dernier poids et de la distance

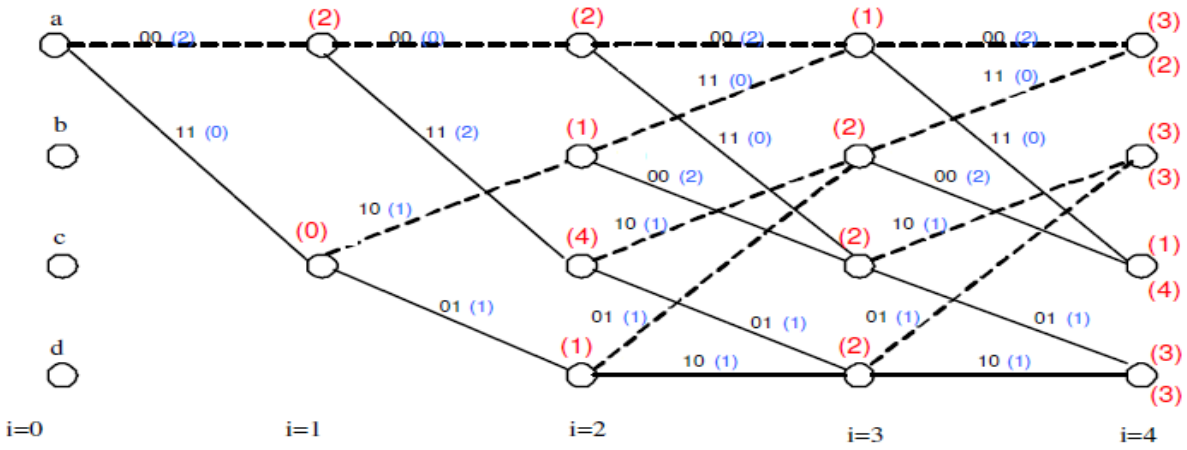
Séquence reçue : 11 00 11

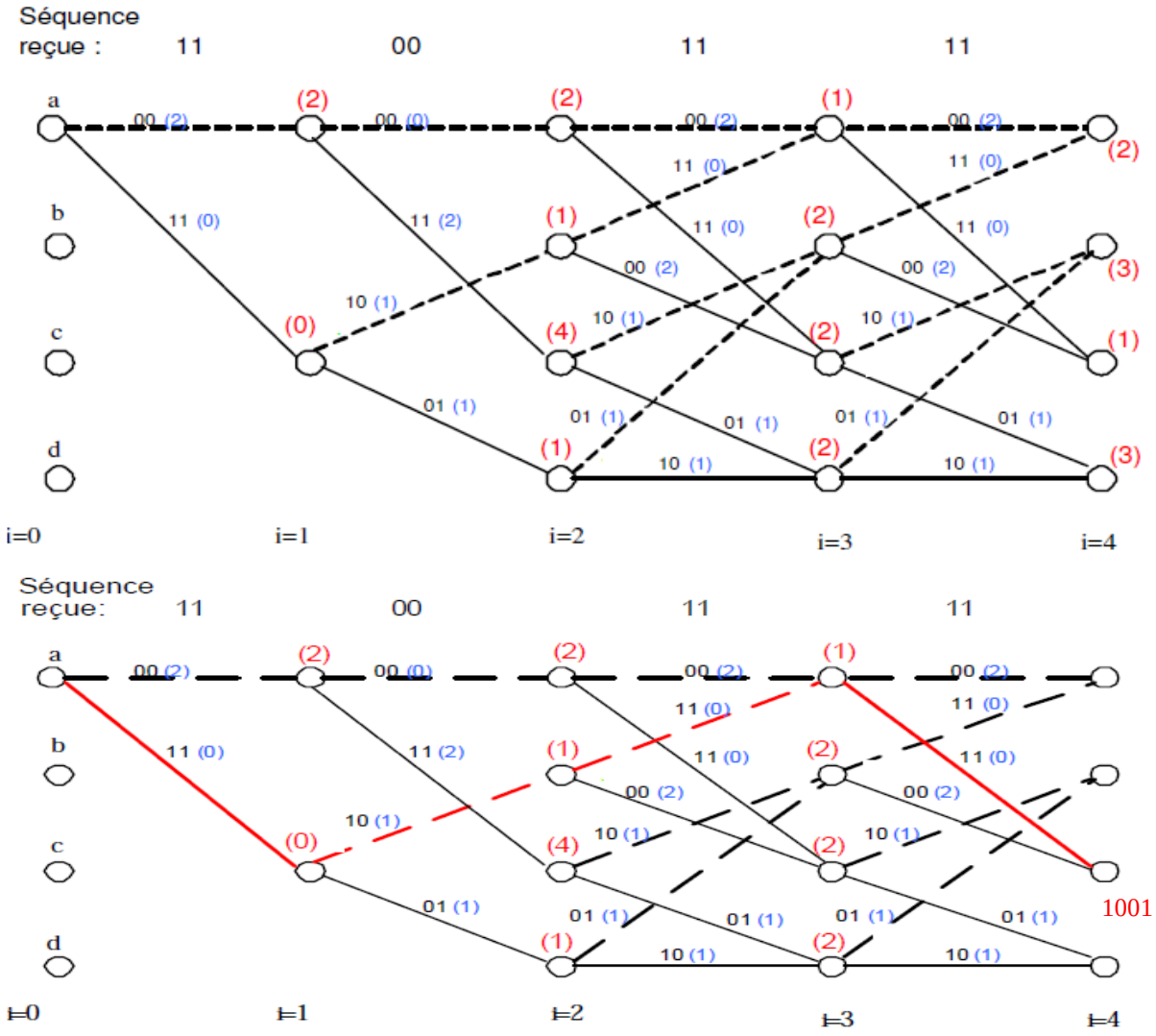


Séquence reçue : 11 00 11



Séquence reçue : 11 00 11 11





Chemin retenu

Les branches en rouge représentent le chemin retenu (formé par les branches donnant lieu à une distance minimale), la séquence décodée est alors : 1001 qui correspond bien à la séquence émise.

1.3.2 Codes entrelacés

L'entrelacement ou opération de génération des permutations est utilisé pour éviter les erreurs consécutives qui dépasseraient le pouvoir de correction du code correcteur d'erreurs. Il consiste à modifier l'ordre de transmission des symboles de manière à éloigner au maximum les bits consécutifs faux [8], l'intérêt premier de son utilisation dans des codes concaténés. On peut citer les entrelaceurs ARP (Almost Regular Permutation) qui sont utilisés dans de nombreux standards, les entrelaceurs DRP (Dithered Relative Prime) qui atteignent le gain d'entrelacement optimal et également les entrelaceurs QPP (Quadratic Permutation Polynomial) qui, en ayant des performances très proches du gain d'entrelacement optimale, assurent également un entrelacement sans collision, ce qui est primordial dans les implantations matérielles [1].

1.3.3 Codes concaténés

La concaténation de codes permet d'augmenter la puissance des systèmes de codage au prix d'une augmentation de la complexité globale [6]. L'idée, introduite par Forney, consiste à bâtir un code possédant un pouvoir de correction suffisant à partir de plusieurs codes simples [9]. La concaténation peut se faire de trois façons : parallèle, série ou hybride (parallèle et série) et sur deux ou plusieurs niveaux [10].

Selon la proposition originale de Forney, l'information est codée deux fois ; une première fois par le premier code appelé code externe, puis une seconde fois par le deuxième, dit code interne [11], comme illustrée sur la figure 1.8.a. On parle alors dans ce cas d'une concaténation série. 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 comme sur la figure 1.8.b. Avec l'apparition des turbo codes, une nouvelle structure a été introduite : la concaténation parallèle (figure 1.8.c). Cette structure associée à des codeurs systématiques permet de générer plusieurs sources de redondance différentes pour le même message.

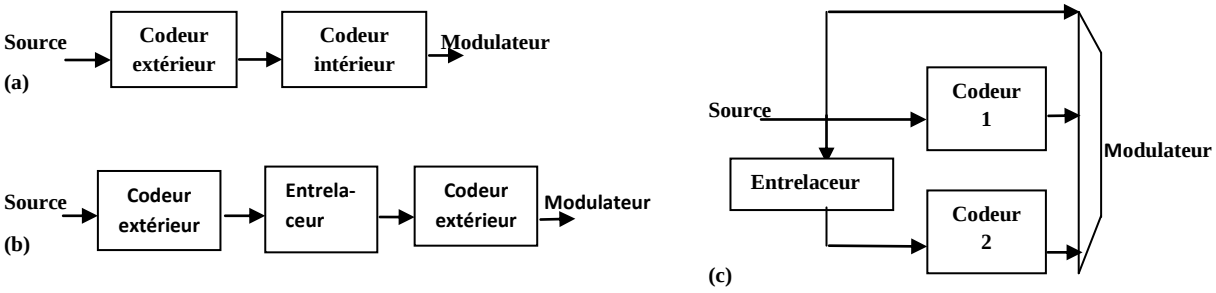


Figure 1.8: Structure de concaténation : (a)série original, (b) série, (c) parallèle.

1.3.4 Principe turbo

Aujourd'hui, les codes turbo sont considérés comme la meilleure méthode d'arrangement pour la correction d'erreur dans la transmission d'information. Un turbo code est la concaténation d'un ou plusieurs codes (convolutifs ou blocs) séparés par des blocs d'entrelacements, l'entrelaceur est un élément essentiel dans le codage turbo. Son principe de décodage se base sur un algorithme, *itératif*, où deux unités échangent des informations au cours de chaque itération afin d'améliorer la capacité de correction d'erreurs de chaque unité. Après plusieurs itérations, les deux unités convergent vers un même mot de code, qui est identique au mot de code transmis.

Le premier schéma de turbo-codage convolutif proposée par C.Berrou en 1993, A.Glavieux et P.Thitimajshima de l'École nationale supérieure des télécommunications de Bretagne (ENST Bretagne) est présenté en figure 1.9. Ce code est construit par une concaténation en parallèle de deux codes convolutifs récursifs et systématiques (RSC) identiques *RSC1* et *RSC2* séparés par un bloc d'entrelacement qui effectue une permutation sur la séquence d'information. Chaque codeur constituant est un codeur convolutif comme celui illustré sur la figure 1.4.b.

Ce type de code est appelé "turbocode" par référence à son principe de décodage itératif. La séquence d'information d'entrée est codée deux fois par les deux codeurs élémentaires. L'entrelaceur **Π** est placé avant le deuxième codeur et effectue une permutation sur la séquence d'information. Alors, les deux codeurs composants codent la même séquence d'information mais en ordres différents.

La séquence d'information **d** est fournie au premier codeur RSC_1 et aussi transmise directement comme séquence systématique. Le codeur RSC_1 produit la première séquence de redondance $\mathbf{y_1}$. Parallèlement, la séquence d'information d'entrée est permutée par l'entrelaceur **Π**. La séquence entrelacée est ensuite codée par le second codeur RSC_2. Ce codeur fournit la seconde séquence de redondance $\mathbf{y_2}$. Le symbole de turbocode possède des bits systématiques et des bits de parité générés par les deux codeurs RSC_1 et RSC_2. Le mot codé par ce schéma à l'instant *i* est défini par $C^i = (d^i y_1^i y_2^i)$.

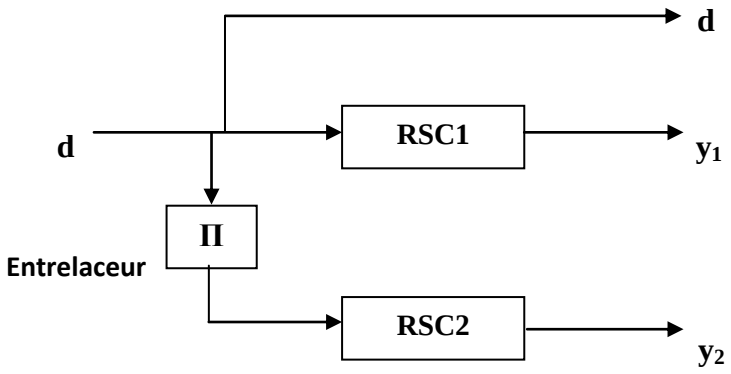


Figure 1.9 : Le principe d'un turbo-codeur.

1.3.4.1 Décodage itératif d'un turbocode

A la réception, le signal bruité est démodulé et transmis au décodeur. L'algorithme optimal de décodage Maximum de vraisemblance (Maximum Likelihood – ML), est appliqué pour décoder le vecteur de données reçues basé sur la structure en treillis. Cependant, en raison de l'entrelaceur, le treillis du turbocode possède un grand nombre d'états. Ceci complique l'algorithme de décodage ML qui s'avère impossible à implémenter en pratique pour une grande longueur d'entrelaceur. C'est la raison pour laquelle, chaque code composant de la structure est décodé séparément par un décodeur. Les informations d'entrée et de sortie de chaque décodeur sont relatives à celles du codeur correspondant. Ainsi, un décodeur peut profiter de l'information de sortie de l'autre décodeur comme information d'entrée. Cet échange d'information est considéré comme le principe de décodage itératif, et les informations échangées sont nommées *informations extrinsèques*.

Il faut souligner que les informations d'entrée de chaque décodeur ainsi que les informations échangées entre les décodeurs sont naturellement pondérées.

Chaque décodeur est désigné comme un décodeur à entrées et sorties pondérées SISO (Soft Input Soft Output). Les sorties soft sont typiquement représentées par des termes appelés logarithmes du rapport de vraisemblance LLR (Log Likelihood Ratio) [6].

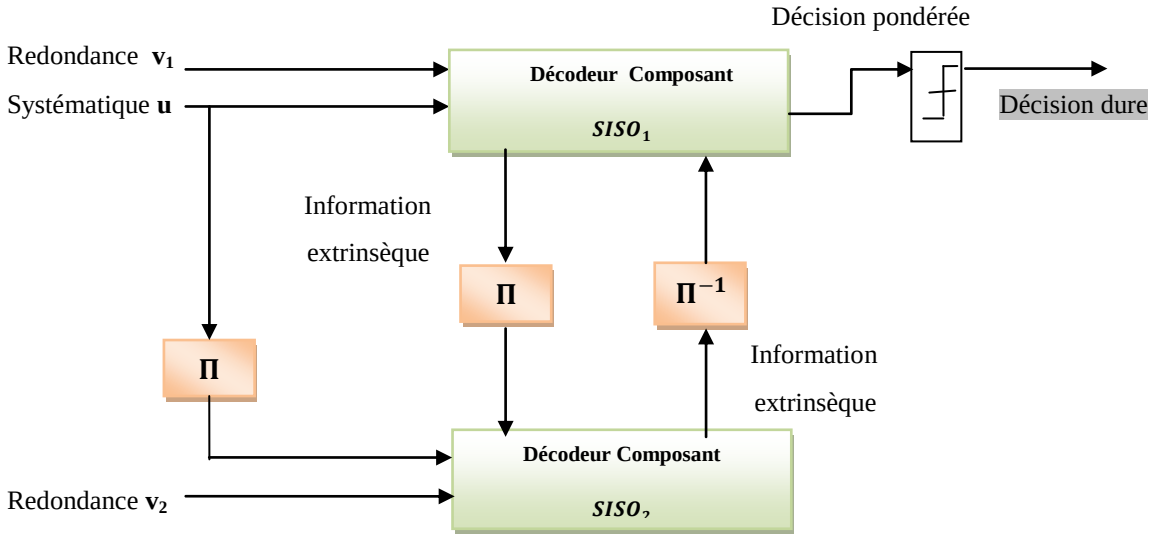


Figure 1.10 : Le principe d'un turbo-décodeur.

Le décodeur de la figure 1.10 fonctionne itérativement. Il est composé de deux décodeurs SISO, de deux entrelaceurs et d'un désentrelaceur. **u** est la séquence d'information reçue correspondant à la séquence d'information émise **d**, $\mathbf{v_1}$ et $\mathbf{v_2}$ sont les séquences d'informations bruitée associées respectivement aux séquences de redondances $\mathbf{y_1}$ et $\mathbf{y_2}$.

L'information échangée entre les décodeurs élémentaires est décrite sous la forme d'une probabilité conditionnelle du bit d'information émis sur les observations à l'entrée du décodeur. Dans le cas du code binaire, l'information extrinsèque du bit d^i avec $i = 1..k$ est formulée comme $\mathrm{P_e}(d^i = 1|\mathbf{u}, \mathbf{v_1})$ ou $\mathrm{P_e}(d^i = 0|\mathbf{u}, \mathbf{v_1})$. Les grandeurs de cette quantité sont très souvent considérées sous forme logarithme telles que des Logarithme de rapport de vraisemblance (LLR).

Par exemple, l'information extrinsèque fournie par le premier décodeur $SISO_1$ et transmise au second décodeur $SISO_2$ est :

$$LLR_{1\to 2}^{e}(d^i) = ln\left(\frac{\mathrm{P_e}(\mathrm{d^i} = 1|\mathbf{u}, \mathbf{v_1})}{\mathrm{P_e}(\mathrm{d^i} = 0|\mathbf{u}, \mathbf{v_1})}\right) \quad (1.7)$$

Les informations extrinsèques produites par un décodeur sont ensuite considérées comme de l'information *a priori* en entrée de l'autre décodeur.

Dans la première itération, le premier décodeur prend seulement les valeurs d'entrées soft par le canal (bits systématiques, bits de parité et les LLR d'information a priori mis à zéro) :

- Le décodeur $SISO_2$ réalise le décodage à partir de trois contributions : la séquence de redondance $\mathbf{v_2}$, la séquence d'information $\mathbf{u}$ entrelacée qui correspond au processus de codage du second codeur RSC, l'information extrinsèque entrelacée sortie du premier décodeur $SISO_1$. Ensuite le décodeur $SISO_2$ produit une sortie pondérée qui correspond à la fiabilité de chaque bit décodé. A partir de cette information, l'information extrinsèque correspondant à la séquence $\mathbf{v_2}$ est produite, entrelacée et transmise au décodeur $SISO_1$.
- Le décodeur $SISO_1$ utilise l'information extrinsèque fournie par le décodeur $SISO_2$ afin de décoder la séquence d'information basée sur ses propres séquences d'observation : $\mathbf{u}$ et $\mathbf{v_1}$. Ce décodeur va ensuite produire une information fiable sur chaque bit décodé correspondant à la séquence $\mathbf{v_1}$. Cette information est par la suite entrelacée et utilisée par le décodeur $SISO_2$ lors de l'itération suivante.

Ce cycle est répété, et pour chaque itération le taux d'erreur par bit (BER) s'améliore. Le turbo décodeur prend une décision sur la sortie du premier décodeur $SISO_1$ comme la décision globale.

1.3.4.2 Décodage des codes turbo : Algorithme MAP

Historiquement, 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 Bahl-Cock-Jelinek-Raviv (BCJR), aussi appelé Maximum A Posteriori (MAP) ou algorithme Aller-Retour comme le log-MAP ou le max-log-MAP. Dans la pratique, les algorithmes log-MAP et max-log-MAP sont préférés pour leur complexité maîtrisée et leur bonne performance par rapport au décodage optimal. La suite de cette section décrit l'algorithme MAP pour décoder les codes convolutifs.

Le principe du Maximum A Posteriori [12], a chaque instant *i*, l'algorithme MAP fournit des probabilités *A Posteriori* - APP $\mathrm{P}\left(d^i = j \middle| \mathbf{u}, \mathbf{v_1}\right)$ correspondant à chaque symbole émis $d^i = 0..\ 2^m - 1$. Le décodeur prend la décision dure d^i qui maximise ces probabilités APP. Pour chaque valeur $j = 0..\ 2^m - 1$, ces probabilités peuvent s'exprimer en fonction des vraisemblances conjointes $\mathrm{P}(d^i = j, \mathbf{u}, \mathbf{v_1})$:

$$\mathrm{P}\left(\mathrm{d^i} = \mathrm{j} \middle| \mathbf{u}, \mathbf{v_1}\right) = \frac{\mathrm{P}\left(\mathrm{d^i} = \mathrm{j}, \mathbf{u}, \mathbf{v_1}\right)}{\sum_{\mathrm{k}=0}^{2\mathrm{m}-1} \mathrm{P}(\mathrm{d^i} = \mathrm{k}, \mathbf{u}, \mathbf{v_1})} \tag{1.8}$$

En pratique, on calcule les vraisemblances conjointes $\mathrm{P}(d^i = j, \mathbf{u}, \mathbf{v_1})$ pour $j = 0..\ 2^m - 1$ (m=2 en binaire), et ensuite, chaque APP est obtenue par normalisation. Lorsque d^i prend seulement les valeurs binaires $j = (0,1)$. Alors :

$$\mathrm{P}\left(\mathrm{d^i} = \mathrm{j} \middle| \mathbf{u}, \mathbf{v_1}\right) = \frac{\mathrm{P}(\mathrm{d^i} = \mathrm{j}, \mathbf{u}, \mathbf{v_1})}{\mathrm{P}(\mathbf{u}, \mathbf{v_1})} = \frac{\mathrm{P}(\mathrm{d^i} = \mathrm{j}, \mathbf{u}, \mathbf{v_1})}{\mathrm{P}(\mathrm{d^i} = 1, \mathbf{u}, \mathbf{v_1}) + \mathrm{P}(\mathrm{d^i} = 0, \mathbf{u}, \mathbf{v_1})} \tag{1.9}$$

Les probabilités conjointes $\mathrm{P}(d^i = j, \mathbf{u}, \mathbf{v_1})$ sont calculées comme la somme des probabilités conjointes possibles de toutes les transitions du treillis de l'état s' à l'état s correspondant au symbole d'entrée $d^i = j$:

$$\mathrm{P}\left(\mathrm{d^i} = \mathrm{j}, \mathbf{u}, \mathbf{v_1}\right) = \sum_{(\mathrm{s}', s)/\mathrm{d^i}(\mathrm{s}', s) = \mathrm{j}} \mathrm{P}\left(\mathrm{s^i} = \mathrm{s}', \mathrm{s^{i+1}} = \mathrm{s}, \mathbf{u}, \mathbf{v_1}\right) \tag{1.10}$$

Bahl [13] a montré que la structure en treillis permettait de décomposer les probabilités conjointes $\mathrm{P}(s^i = s', s^{i+1} = s, \mathbf{u}, \mathbf{v_1})$ en trois probabilités différentes correspondant aux trois métriques, conformément à l'équation (1.11) et structuré à la figure 1.11.

- Métrique récurrente aller (forward) $\alpha^i(s')$ c'est la probabilité que le treillis est en état s' à l'instant i et la séquence reçue par le canal à cet instant est $\mathbf{u}^{k<i}, \mathbf{v_1}^{k<i}$: $\alpha^i(s') = \mathrm{P}(s^i = s', \mathbf{u}^{k<i}, \mathbf{v_1}^{k<i})$.
- Métrique récurrente *retour* (backward) $\beta^{i+1}(s)$ c'est la probabilité que le treillis est en état s à l'instant $i+1$ et la séquence future à recevoir par le canal est $\mathbf{u}^{k>i}, \mathbf{v_1}^{k>i}$: $\beta^{i+1}(s) = \mathrm{P}(\mathbf{u}^{k>i}, \mathbf{v_1}^{k>i} | s^{i+1} = s)$.
- Métrique $\gamma^i(s', s)$ c'est la probabilité que le treillis était en état s' à l'instant i, déplaçant vers l'état s à l'instant $i+1$ et la séquence reçue par le canal pour cette transition est $\mathbf{u}^i, \mathbf{v_1}^i$: $\gamma^i(s', s) = \mathrm{P}(s^{i+1} = s, \mathbf{u}^i, \mathbf{v_1}^i | s^i = s')$.

$$\mathrm{P}(s^i = s', s^{i+1} = s, \mathbf{u}, \mathbf{v_1}) = \alpha^i(s')\gamma^i(s', s)\beta^{i+1}(s) \qquad (1.11)$$

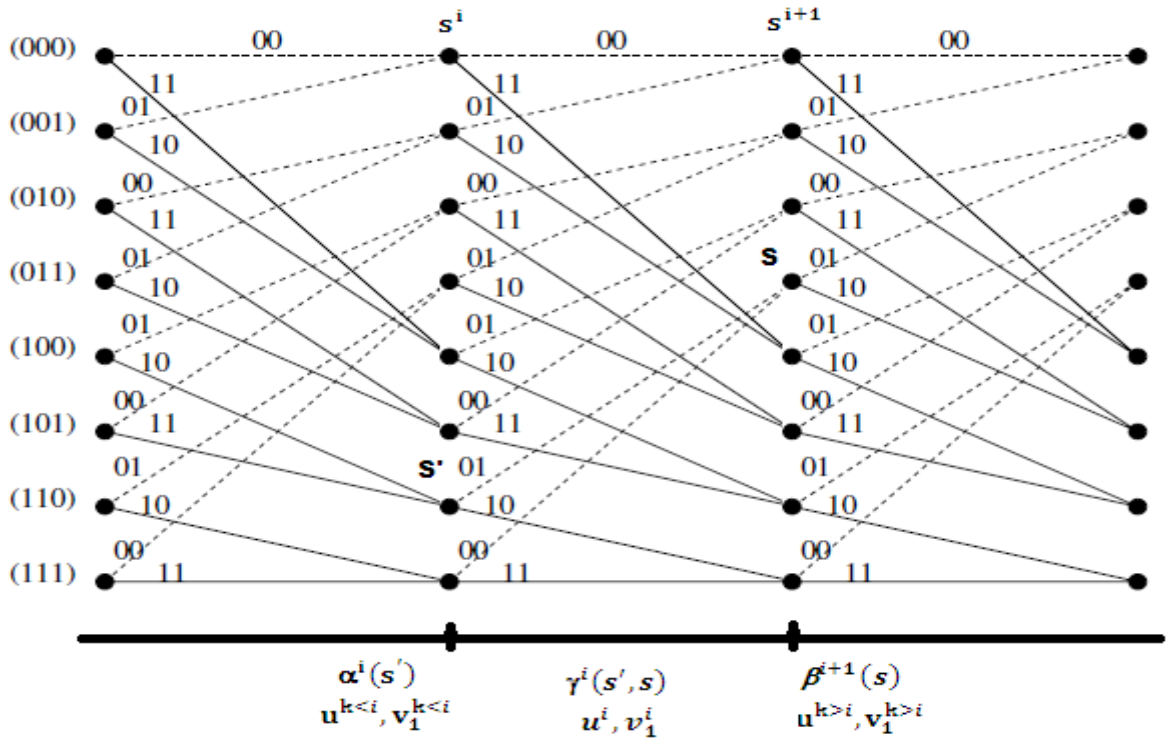


Figure 1.11 : Treillis d'un décodeur montrant les trois métriques de probabilité.

A partir des équations (1.8) et (1.11), les probabilités *a posteriori* APP s'expriment :

$$P(d^i = j \,|\mathbf{u}, \mathbf{v_1}) = \frac{\sum_{(s',s)/d^i(s',s)=j} \alpha^i(s')\gamma^i(s',s)\beta^{i+1}(s)}{\sum_{(s',s)} \alpha^i(s')\gamma^i(s',s)\beta^{i+1}(s)} \tag{1.12}$$

Les métriques récurrentes aller et retour en équation (1.11) sont récursivement calculées :

$$\begin{aligned} \alpha^i(s') &= \sum_{s'=0}^{2^v-1} \alpha^{i-1}(s')\gamma^{i-1}(s',s) \quad \text{pour } i = 1..k \\ \beta^{i+1}(s) &= \sum_{s=0}^{2^v-1} \beta^{i+1}(s)\gamma^{i}(s,s') \qquad \text{pour } i = k-1..0 \end{aligned} \tag{1.3}$$

Afin de calculer les probabilités récurrentes de l'équation (1.13), une connaissance de l'état de départ et de l'état final du codage est nécessaire. Par exemple, si l'état s_0 de départ du codeur est connu, alors, α_0 (s_0) = 1 et α_0 (s)= 0 pour tout $s \neq 0$. Si l'état est inconnu, les métriques récurrentes $\alpha_0(s)$ sont initialisées à la même probabilité. La même règle est également appliquée pour l'état final s_k du treillis. De plus, afin d'éviter le problème de débordement, les métriques récurrentes sont régulièrement normalisées.

La métrique de vraisemblance d'une branche de l'équation (1.11) est nulle si aucune transition entre les états s' et s existe. Dans le cas contraire, elle peut s'exprimer comme :

$$\gamma^i(s',s) = P_a(d^i = j)P(\mathbf{u}^i, \mathbf{v_1^i}|d^i, y_1^i) \tag{1.14}$$

où $P_a(d^i = j)$ est la probabilité *a priori* correspondant au symbole émis $d^i = j$ qui dépend de la statistique de la source. Dans le cas d'une source équiprobable, tous les symboles possèdent la même probabilité de transmission.

Alors, $P_a(d^i = j) = 1/2$ pour une source binaire équiprobable.

En outre, dans le cas d'un canal gaussien Additive White Gaussian Noise- AWGN, la probabilité $P(\mathbf{u}^i, \mathbf{v_1^i}|d^i, y_1^i)$ est donnée par :

$$P(\mathbf{u}^i, \mathbf{v_1^i}|d^i, y_1^i) = \frac{1}{2\pi\,\sigma^2} \exp\left(-\frac{\|\mathbf{u}^i - d^i\|^2 + \|\mathbf{v_1^i} - y_1^i\|^2}{2\sigma^2}\right) \tag{1.15}$$

avec σ^2 la variance du bruit additif blanc gaussien. En pratique, seuls les termes spécifiques correspondant à la transition considérée et qui ne s'éliminent pas par la division dans l'expression (1.9) sont retenus. Par conséquent, l'expression (1.15) est réécrite comme suit :

$$\mathrm{P}\left(\mathbf{u}^{\mathbf{i}}, \mathbf{v}_{\mathbf{1}}^{\mathbf{i}} \middle| \mathrm{d}^{\mathrm{i}}, \mathrm{y}_1^{\mathrm{i}}\right) = \exp\left(\frac{\langle \mathbf{u}^{\mathrm{i}}, \mathrm{d}^{\mathrm{i}} \rangle + \langle \mathbf{v}_{\mathbf{1}}^{\mathrm{i}}, \mathrm{y}_1^{\mathrm{i}} \rangle}{\sigma^2}\right) \tag{1.16}$$

Dans le cadre du turbo décodage, le turbo-décodeur demande aux deux décodeurs convolutifs d'échanger des informations extrinsèques. C'est pourquoi, l'algorithme MAP doit être modifié pour produire l'information extrinsèque. Pour cela, l'expression (1.14) doit tenir également compte de l'information extrinsèque produite par l'autre décodeur élémentaire :

$$\gamma^{\mathrm{i}}(\mathrm{s}', \mathrm{s}) = \mathrm{P}_{\mathrm{a}}\left(\mathrm{d}^{\mathrm{i}} = \mathrm{j}\right) \mathrm{P}_{\mathrm{e}}^{\mathrm{in}}\left(\mathrm{d}^{\mathrm{i}} = \mathrm{j} \,|\mathbf{u}, \mathbf{v_2}\right) \mathrm{P}\left(\mathbf{u}^{\mathrm{i}}, \mathbf{v}_{\mathbf{1}}^{\mathrm{i}} \middle| \mathrm{d}^{\mathrm{i}}, \mathrm{y}_1^{\mathrm{i}}\right) \tag{1.17}$$

Où l'information *a priori* $P_a\left(d^i = j\right)$ est remplacée par $P_a\left(d^i = j\right) P_e^{in}\left(d^i = j \,|\mathbf{u}, \mathbf{v_2}\right)$ et $P_e^{in}\left(\mathrm{d}^{\mathrm{i}} = \mathrm{j} \,|\mathbf{u}, \mathbf{v_2}\right)$ est l'information extrinsèque fournie par le second décodeur composant.

L'information extrinsèque générée par ce décodeur élémentaire est exprimée par :

$$\mathrm{P}_{\mathrm{e}}^{\mathrm{out}}\left(\mathrm{d}^{\mathrm{i}} = \mathrm{j} \,|\mathbf{u}, \mathbf{v_1}\right) = \frac{\sum_{(\mathrm{s}', \mathrm{s})/\mathrm{d}^{\mathrm{i}}(\mathrm{s}', \mathrm{s}) = \mathrm{j}} \alpha^{\mathrm{i}}(\mathrm{s}')\, \gamma_{\mathrm{e}}^{\mathrm{i}}(\mathrm{s}', \mathrm{s}) \beta^{\mathrm{i}+1}(\mathrm{s})}{\sum_{(\mathrm{s}', \mathrm{s})} \alpha^{\mathrm{i}}(\mathrm{s}') \gamma_{\mathrm{e}}^{\mathrm{i}}(\mathrm{s}', \mathrm{s}) \beta^{\mathrm{i}+1}(\mathrm{s})} \tag{1.18}$$

avec $\gamma_e^i(s', s)$ la métrique de branche modifiée qui ne tient plus compte des informations disponibles dans le décodeur auquel l'information extrinsèque est adressée. Dans le cas du turbo décodage des codes convolutifs concaténé en parallèle, cette métrique est obtenue pour un canal AWGN à partir de l'expression (1.16) en éliminant l'information systématique :

$$\gamma_{\mathrm{e}}^{\mathrm{i}}(\mathrm{s}', \mathrm{s}) = \exp\left(\frac{\langle \mathbf{v}_{\mathbf{1}}^{\mathrm{i}}, \mathrm{y}_1^{\mathrm{i}} \rangle}{\sigma^2}\right) \tag{1.19}$$

1.4 Conclusion

Nous avons décrit dans ce chapitre les bases nécessaires des turbocodes convolutifs et de leur décodage, pour cela, nous avons passé en revue les différents composants d'une chaîne de communication numérique. Un turbo code convolutif est une concaténation de codes de type récursif systématique convolutif (RSC) séparés par des générateurs de permutations. Le principe turbo ou traitement itératif, consiste à faire dialoguer deux décodeurs qui travaillent dans deux domaines différents. Les échanges d'informations s'effectuent de manière probabiliste jusqu'à ce que les deux décodeurs aient convergés vers une solution. Pour le turbo décodage, nous avons décrit l'algorithme de décodage « MAP », nous avons également introduit le principe du maximum a posteriori qui est la base de l'algorithme MAP.

EXPLOITATION DU PARALLELISME DANS LES TURBO-CODES CONVOLUTIFS

2.1 Introduction

Les nouvelles technologies de communication exigent des normes de plus en plus strictes en termes de qualité de service. Le Turbo décodage est de plus en plus proposé dans les nouvelles et les futurs systèmes de communication numériques, par exemple, les applications de communication à fibre optiques, les communications sans fil, et le stockage. En pratique, les modèles turbo-décodeur, comme celui utilisé pour la norme IEEE802.16 e, Long Term Evolution (LTE) ou des normes IEEE802.11, exigent des hauts débits pour la transmission des données (plusieurs centaines de mégabits par seconde) et une faible latence (dix ms ou presque). Pour faire face à ces exigences, les implémentations des turbo-décodeur doivent être massivement parallèles.

Dans la première partie de ce chapitre, nous décrivons brièvement quelques notions de base sur le parallélisme avec des exemples de programmation parallèle en Matlab. Ensuite, dans une seconde partie, nous explorons les niveaux du parallélisme dans les turbocodes.

2.2 Notions sur le parallélisme

Le parallélisme d'un système d'information est défini comme l'exécution simultanée de plusieurs traitements dans le but d'améliorer les performances d'un système [1]. Dans le domaine des architectures des processeurs, Il y a deux approches différentes :

- Les architectures multiprocesseurs permettent un parallélisme des tâches, où un processus peut être exécuté sur chaque processeur, la machine ILLIAC-IV de 8×8 processeurs est la première machine parallèle conçue à la fin des années 60 par Daniel Slotnik pour la NASA.

- Les architectures vectorielles sont spécialisées dans le traitement rapide des boucles. Dans ces architectures, on a pipeliné l'Unité Arithmétique et Logique. Ceci permet de recouvrir le temps d'exécution des instructions. L'ordinateur STAR 100 est le premier ordinateur vectoriel introduit 1973 par CDC (Control Data Coorporation)

Au milieu des années 60, Flynn a proposé un modèle simpliste pour classer les ordinateurs, en fonction des flots d'instructions et de données [14]. Il a ainsi rangé tous les ordinateurs en quatre grandes catégories :

SISD (Single Instruction Single Data), c'est-à-dire un seul flot d'instructions et un seul flot de données. Un seul processeur (unité centrale) travaille de manière séquentielle sur des informations en mémoire qui constituent à la fois les données et les programmes : une seule unité de commande traitant une seule séquence d'instructions et une seule unité d'exécution Unité Arithmétique et Logique (ALU) traitant une unique séquence de données. C'est la machine monoprocesseur classique telle que la décrit John Von Neumann (la figure 2.1)

SIMD (Single Instruction Multiple Data), c'est-à-dire à un flot d'instruction est associé plusieurs flots de données. L'unité de commande envoie une instruction à toutes les ALU qui exécutent l'instruction pas à pas sur des données locales. Le réseau d'interconnexions permet aux résultats d'être envoyés vers une autre ALU qui pourra les utiliser comme opérande dans une instruction suivante. C'est la machine vectorielle (la figure 2.2.a).

MISD (Multiple Instruction Single Data), affectant un flot de données sur plusieurs instructions en parallèle, il n'existe pour l'instant aucune machine architecturée sur ce modèle.

MIMD (Multiple Instruction Multiple Data), qui traite plusieurs flots de données à partir de plusieurs flots d'instructions. Plusieurs processeurs disposent de leurs propres programmes indépendants (chaque processeur exécute ses propres instructions et opère sur ses propres données). C'est la machine multiprocesseur (la figure 2.2.b).

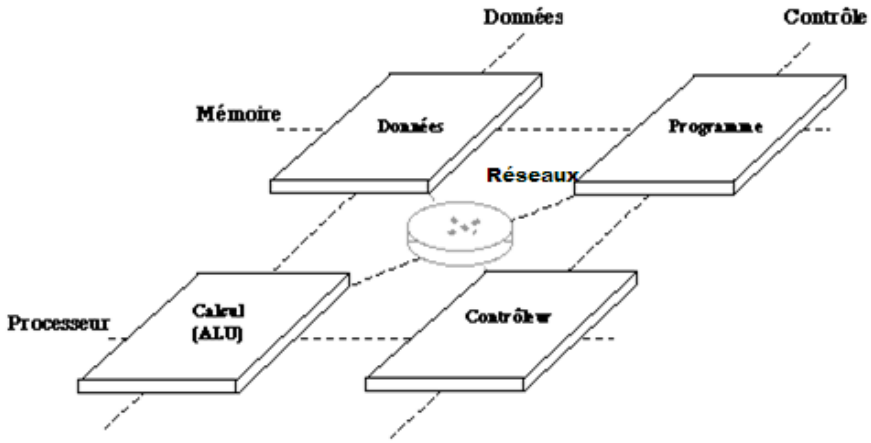


Figure 2.1 : Architecture SISD.

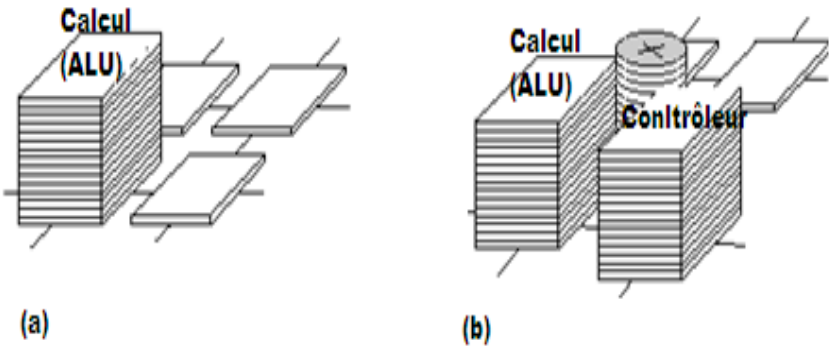


Figure 2.2: Architecture :(a) SIMD,(b) MIMD.

Modèles de programmation SPMD (Single Program Multiple Data), tous les processeurs exécutent le même programme. Il ya deux classifications de mémoire :

-mémoire partagé : tous les processeurs ont accès à l'ensemble de la mémoire (la figure 2.3).

-mémoire distribué : chaque processeur possède sa propre mémoire. Il n'a pas accès à celle des autres, il faut gérer l'échange de messages (la figure 2.4) [15].

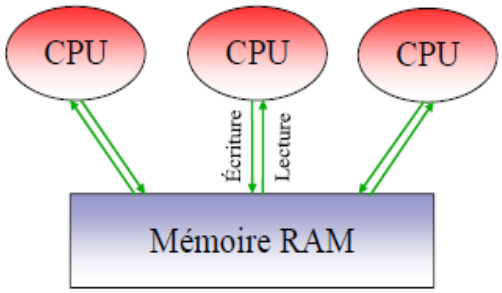


Figure 2.3 : Mémoire partagé.

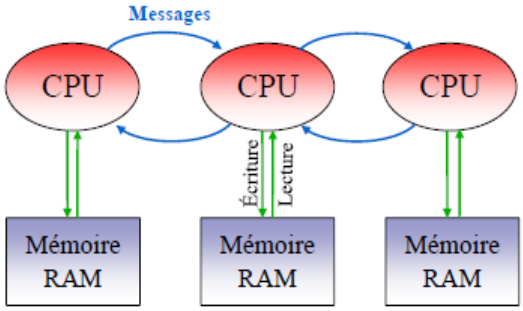
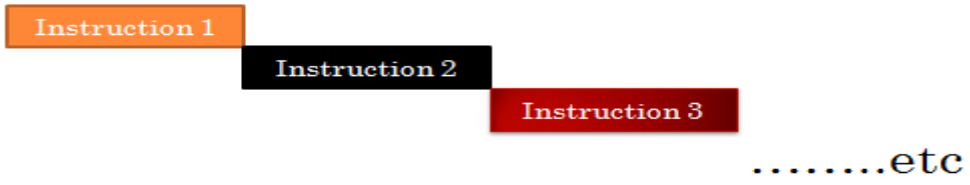


Figure 2.4 : Mémoire distribué.

2.2.1 Architecture à base de pipeline

Dans une structure d'un microprocesseur, le traitement des instructions en assembleur se font les unes à la suite des autres. Tant que l'exécution d'une instruction n'est pas terminée, l'instruction suivante n'est pas exécutée [16].



Si le temps d'exécution d'un processus est T_p, l'exécution séquentielle de m processus prend un temps T_t : $T_t = mT_p$ [17].

Les machines PipeLine permettent de résoudre ce problème, ils sont des extensions du modèle de Von Neumann où on a pipeliné l'unité d'exécution des instructions. Ceci permet de recouvrir le temps d'extraction des instructions [18] comme illustré dans la figure 2.5.

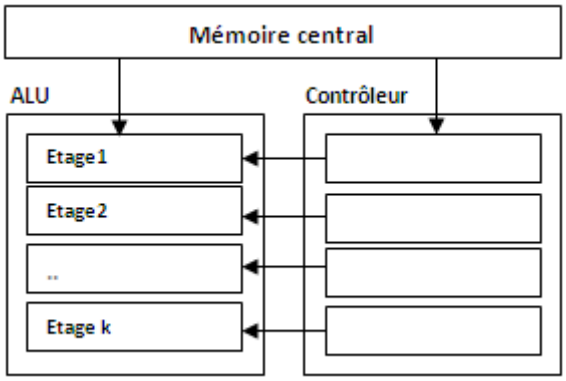


Figure 2.5 : Architecture PipeLine.

Pour comprendre le mécanisme du pipeline, il est nécessaire au préalable de comprendre les phases d'exécution d'une instruction **(**lecture d'une instruction, décodage, exécution et écriture du résultat).

Les instructions sont organisées en file d'attente dans la mémoire, et sont chargées les unes après les autres. Grâce au pipeline, le traitement des instructions nécessite au maximum les 4 étapes précédentes. Dans la mesure où l'ordre de ces étapes est invariable, il est possible de créer dans le processeur un certain nombre de circuits spécialisés pour chacune de ces phases.

L'objectif du pipeline est d'être capable de réaliser chaque étape en parallèle avec les étapes amont et aval, c'est-à-dire de pouvoir lire une instruction lorsque la précédente est en cours de décodage , que celle d'avant est en cours d'exécution , que celle située encore précédemment accède à la mémoire et enfin que la première de la série est déjà en cours d'écriture dans les registres .

2.2.2 Caractérisation des architectures

La caractérisation d'une architecture peut être menée dans un but de comparaison avec d'autres architectures ou bien simplement pour montrer que celle-ci est en mesure d'atteindre un niveau de performance requis ; Le débit, la complexité, l'efficacité et latence sont utiles pour quantifier les performances d'une architecture.

Le débit : dans bien des cas, une des premières caractéristiques d'un système de communication que l'on cherche à estimer est son débit binaire. Il s'agit de la quantité d'information binaire que ledit système peut traiter ou générer par unité de temps [3].

L'efficacité : dans un contexte très haut débit, la principale contrainte est naturellement de concevoir des architectures capables d'atteindre des débits de traitement importants. Néanmoins, comme nous venons de le voir, considérer le débit d'un système sans se soucier de sa complexité n'est pas judicieux. A titre d'exemple, considérons deux architectures A_0 et A_1 ayant des débits de traitement respectifs de D_0 et D_1 ainsi que des complexités C_0 et C_1. Si A_1 est deux fois plus rapide ($D_1 = 2 \times D_0$) mais quatre fois plus complexe $C_1 = 4 \times C_0$ que A_0, il va alors de soi que l'architecture A_0 est plus intéressante. Ceci montre qu'au delà du débit et de la complexité d'une architecture il faut tenir compte du rapport entre ces deux métriques. Autrement dit, dans un environnement très haut débit, tout l'art du concepteur est de trouver des architectures qui maximisent les débits de traitement tout en minimisant le coût en surface. Ce compromis est atteint lorsque l'on maximise la métrique d'efficacité qui est définie comme : $E = \frac{D}{C}$ [10].

La complexité : la complexité est une notion si largement utilisée qu'il est essentiel de correctement la définir dans le contexte de l'étude. Considérée comme une métrique, elle quantifie la difficulté inhérente à un processus. En algorithmique, la théorie de la complexité a permis de formaliser ce concept qui s'est ensuite implicitement appliqué a l'électronique.

L'objet de la complexité algorithmique est de pouvoir comparer les performances des différents algorithmes. Néanmoins, faire une comparaison objective implique de s'affranchir de tout paramètre matériel et technologique. En effet deux algorithmes identiques qui s'exécutent sur des machines différentes (type de processeur, temps d'accès mémoire, compilateur utilisé…) ont des temps d'exécution différents [3].

La latence : classiquement, la latence se définie comme le retard temporel induit par l'architecture entre ses entrées et ses sorties. Dans cette étude, nous définissons la latence comme la durée s'écoulant entre la réception d'un symbole et l'émission du même symbole décodé par le décodeur. La latence d'une architecture parallèle doit être inférieure à celle d'une architecture séquentielle [1].

2.3 Programmation parallèle sous Matlab

La boite à outils « Parallel Computing Toolbox » de Matlab permet de résoudre des problèmes qui font appel aux calculs et gourmands en données à l'aide de processeurs multicœurs, GPU et clusters d'ordinateurs. Les constructions de haut niveau (boucle-`For` parallèle, types de tableaux spéciaux et algorithmes numériques parallélisés) permettent de paralléliser les applications MATLAB. Il est possible d'utiliser la boîte à outils avec Simulink pour exécuter plusieurs simulations d'un modèle en parallèle. La boîte à outils fournit douze workers (moteurs de calcul MATLAB) pour exécuter les applications localement sur un ordinateur de bureau multicœur. Sans changer le code, nous pouvons exécuter la même application sur un cluster d'ordinateurs ou une grille (avec MATLAB Distributed Computing Server). La boite à outils « Parallel Computing Toolbox » peut assurer quatre types de parallélisme :

- boucle parfor
- distributed jobs
- parallel jobs
- Single Program Multiple Data (SPMD)

Dans notre travail on s'est intéressé au premier et au quatrième type de parallélisme.

Utilisation de la commande parfor pour les programmes parallèles, la façon la plus simple de parallélisation d'un programme MATLAB se concentre sur les boucles dans le programme. Si une boucle for est adapté pour l'exécution en parallèle, on remplace simplement le mot for par le mot parfor. Lorsque le programme MATLAB est exécuté, et si les workers ont été mis à disposition par la commande matlabpool, alors le travail dans chaque boucle parfor sera distribué entre ces workers [20].

Chaque exécution du contenu d'une boucle parfor est une itération. Les workers de MATLAB exécutent les itérations indépendamment les uns des d'autres sans aucun ordre particulier, il n'ya aucune garantie que les itérations sont synchronisées en aucune façon, ni aucune nécessité pour cela. Si le nombre de travailleurs est égal au nombre d'itérations de la boucle, chaque travailleur effectue une itération de la boucle. S'il ya plus d'itérations que les travailleurs, certains travailleurs effectuent plus d'une itération de boucle, dans ce cas, un

travailleur pourrait recevoir de multiples itérations à la fois pour réduire le temps de communication.

Dans cette boucle, par exemple, la variable angle est une variable temporaire. MATLAB est capable de gérer ce calcul en parallèle, sans aucune action particulière de l'utilisateur, qui a tout simplement à remplacer for par parfor:

```
for i = 1 : n
  angle = ( i - 1 ) * pi / ( n - 1 );
  t(i) = cos (angle);
end
```

Nous ne pouvons pas utiliser une boucle parfor quand une itération dans la boucle dépend des résultats d'autres itérations. Chaque itération doit être indépendante de toutes les autres. Dans l'exemple simple ci-dessous, Nous ne pouvons pas remplacer la boucle for par une boucle parfor car chaque itération dépend du résultat de l'itération qui la précède :

```
dx = 0,25;
x = zéros (1, n);
for  i = 2: n
   x (i) = x (i-1) + dx;
end
```

Utilisation de SPMD, la déclaration de SPMD (Single Program Multiple Data) dans un code permet de définir un bloc de code qui s'exécute simultanément des workers multiples. La forme générale de la déclaration de SPMD est la suivante :

```
spmd
<statements>
End
```

Par exemple, on peut créer une matrice aléatoire sur trois workers par:

```
matlabpool
spmd (3)
  R = rand(4,4);
end
matlabpool close
```

2.3.1 Exemple simple de calcul des nombres premiers

a) Utilisation de la boucle parfor

Comme premier exemple, on a choisi un programme simple qui compte le nombre des nombres premiers entre 1 et N, en utilisant l'instruction parfor avec un client (Matlab) et quatre workers.

```
function total = nb_premier_parfor( n )
%**************************************************************************
%  Résultats à trouver
%                N       TOTAL
%
%                1           0
%               10           4
%              100          25
%            1,000         168
%           10,000       1,229
%          100,000       9,592
%        1,000,000      78,498
%       10,000,000     664,579
%      100,000,000   5,761,455
%    1,000,000,000  50,847,534

  total = 0;
  parfor i = 2 : n
    premier = 1;
    for j = 2 : sqrt (i)
      if ( mod ( i, j ) == 0 )
        premier = 0;
        break
      end
    end
    total = total + premier;
  end
  return
end
```

Le code suivant fait appel à cette fonction :

```
%% PREMIER _POOL utilise la commande MATLABPOOL pour exécuter le code des
%% nombres premiers

matlabpool open local 4
  n = 10000000;
  fprintf (1, '\n');
  tic
  total = nb_premier_parfor (n);
  toc
  fprintf (1, '\n');
  fprintf (1, '  Total est %d\n', total);
matlabpool close
```

Les résultats d'exécution de ce code pour un seul worker et pour 4 workers sont donnés ci-dessous.

```
>> PREMIER_POOL
Starting matlabpool using the 'local' configuration ... connected to 1 labs.

Elapsed time is 168.789174 seconds.

  Total est 664579
Sending a stop signal to all the labs ... stopped.
```

Figure 2.6 : Résultat pour un seul worker.

```
>> PREMIER_POOL
Starting matlabpool using the 'local' configuration ... connected to 4 labs.

Elapsed time is 27.464193 seconds.

  Total est 664579
Sending a stop signal to all the labs ... stopped.
```

Figure 2.7 : Résultat pour quatre workers.

A partir de ces résultats, nous remarquons que le temps d'éxécution avec 4 workers est devenu très court ; il est divisé par quatre (le nombre des workers).

b) Utilisation de SPMD

Cette fois-ci, le code qui permet de compter le nombre de nombres premiers entre 1 et N utilise l'instruction spmd.

```
function total = nb_premier_spmd ( n )
 spmd
  if ( 0 )
   np = ( n * ( labindex - 1 ) ) / numlabs + 1;      % La première méthode choisit des  intervalles égaux
   ng = ( n *   labindex      ) / numlabs;
  else
   np = floor ( ( n * sqrt ( labindex - 1 ) ) / sqrt ( numlabs ) ) + 1;       % la deuxième méthode  divise la
   ng = floor ( ( n * sqrt ( labindex     ) ) / sqrt ( numlabs ) );           % quantité de travail entre workers
  end
  if ( np == 1 )
   np = 2;
  end
  fprintf ( 1, '  Range = [%d,%d]\n', np, ng );
%
%  chaque worker cherche le nombre des nombres premier dans sa  rangé.
%
  total_part = 0;
  for i = np : ng
    premier = 1;
    for j = 2 : i - 1
      if ( mod ( i, j ) == 0 )
        premier = 0;
        break
      end
    end
    total_part = total_part + premier;
  end
 %
 %  Somme des valeurs dans chaque  worker.
 %  La somme elle-même est  seulement dans les workers, pas dans le client.
 %
  total_spmd = gplus ( total_part );
 end
%
%  Le client peut retirer la somme  à partir de l'un des workers.
%
 total = total_spmd{1};
 return
end
```

Les résultats de l'exécution de ce code sont donnés ci-dessous.

```
>> PREMIER_POOL
Starting matlabpool using the 'local' configuration ... connected to 1 labs.

Lab 1:
    Range = [2,10000000]

Elapsed time is 168.789174 seconds.


  Total est 664579
Sending a stop signal to all the labs ... stopped.
```

Figure 2.8 : Résultat pour un seul worker.

```
>> PREMIER_POOL
Starting matlabpool using the 'local' configuration ... connected to 4 labs.

Lab 1:
    Range = [2,5000000]
Lab 3:
    Range = [7071068,8660254]
Lab 4:
    Range = [8660255,10000000]
Lab 2:
    Range = [5000001,7071067]


Elapsed time is 27.464193 seconds.


  Total est 664579
Sending a stop signal to all the labs ... stopped.
```

Figure 2.9 : Résultat pour quatre workers.

2.3.2 Exemple de l'application du parallélisme pour l'amélioration du contraste d'une image

La commande spmd peut être utilisée dans le traitent d'une image. Le client Matlab lit l'image, et comme ce dernier est noir et blanc, on obtient une matrice à deux dimensions qui sera distribuée par colonnes. Les quatre workers assurent l'augmentation du contraste sur des parties distinctes de cette image (4 parties verticales). A la fin le client rassemble et affiche les résultats, comme c'est montré sur la figure 2.10.

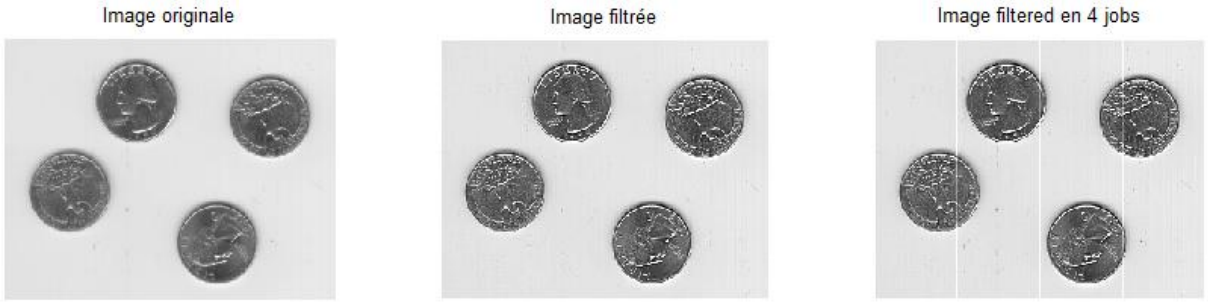


Figure 2.10 : Résultat de l'amélioration du contraste d'une image par quatre workers.

2.3.3 Débruitage d'une image en parallèle en utilisant SPMD

Dans cet exemple qui utilise la commande spmd, une bonne image est endommagé par l'ajout un bruit de type « poivre et sel ». Le résultat est une image bruité, mais il y a encore beaucoup d'information originale. L'approche consiste à diviser l'image en couleur dans les plans R, G et B, et chaque plan sera traité par un worker.

Le client lit l'image, l'utilisation de la commande spmd se fait plus naturellement en distribuant l'image, de sorte que chaque job reçoit une section distincte. Une fois le traitement terminé. Le client peut remonter l'image.

Figure 2.11 : Résultat du débruitage d'une image RGB par trois workers.

2.4 Exploitation du parallélisme dans les turbo-codes convolutifs

Le processus de décodage d'un turbocode avec l'algorithme MAP 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 MAP. 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.

Dans notre travail, on s'intéresse aux deux derniers niveaux de granularité.

2.4.1 Parallélisme du sous-bloc

Ce type de parallélisme consiste à utiliser des décodeurs SISO multiples. Chaque trame est divisée en p sous-blocs, ensuite chaque sous-bloc et traité par un décodeur SISO-BCJR [21]. Chacun des décodeurs SISO doit donc être initialisé de manière correcte afin de pouvoir casser cette dépendance de données (la figure 2.12). Ceci peut être réalisé par la méthode d'initialisation (par treillis circulaire).

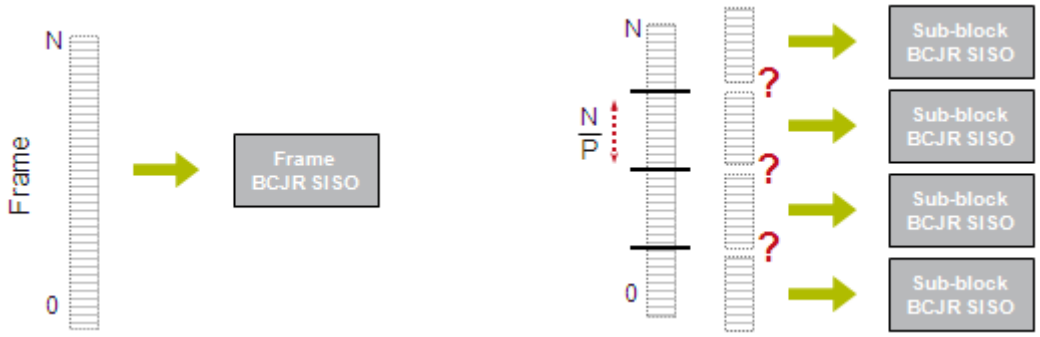


Figure 2.12 : Parallèlisme du sous-bloc.

2.4.2 Parallélisme au niveau de turbo-décodeur complet

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 [1]. Le parallélisme de trame ne présente aucune limitation de parallélisme [21].

2.4.3 Turbo code à roulettes

Dans cette section, nous décrivons les turbocodes à roulettes qui sont une famille de code permettant la parallélisation du codage et du décodage turbo en utilisant le principe du parallélisme du sous-bloc décrit ci-dessus.

En turbo-codes à roulettes la trame d'information de N symboles est découpée en *P* blocs de *M* symboles chacun, avec *N=M*P*, ainsi le turbo-code est noté (*N, M, P*). L'opération de codage est d'abord effectuée dans l'ordre naturel afin de générer la redondance dans la première dimension. Chaque bloc est alors codé de manière indépendante par un code CRSC (code Convolutif Récursif Systématique Circulaire). Ces codes constituants sont appelés roulettes par analogie avec la technique de fermeture du treillis « circulaire » utilisée. La trame d'information est ensuite permutée par un entrelaceur de taille *N* symboles. La trame entrelacée est également découpée en *P* blocs de longueur *M* et chaque bloc est codé indépendamment par un code CRSC pour produire la redondance de la deuxième dimension.

L'entrelaceur est construit conjointement à l'organisation de la mémoire de façon à permettre le décodage en parallèle des *P* roulettes. En clair, sa structure permet de lire et écrire, à chaque cycle symbole *k*, les *P* données nécessaires aux *P* décodeurs des *P* bancs mémoires $MB_1, MB_2, MB_3, \dots, MB_p$ sans conflits.

En effet, une seule lecture peut être effectuée au même instant pour effectuer *P* accès en parallèle, *P* bancs mémoires sont alors nécessaires [22].

2.4.3.1 Construction de l'entrelaceur

a) Structure d'entrelaceur

La figure 2.13 présente la structure d'entrelacement utilisée pour construire un code à roulettes garantissant le parallélisme de décodage.

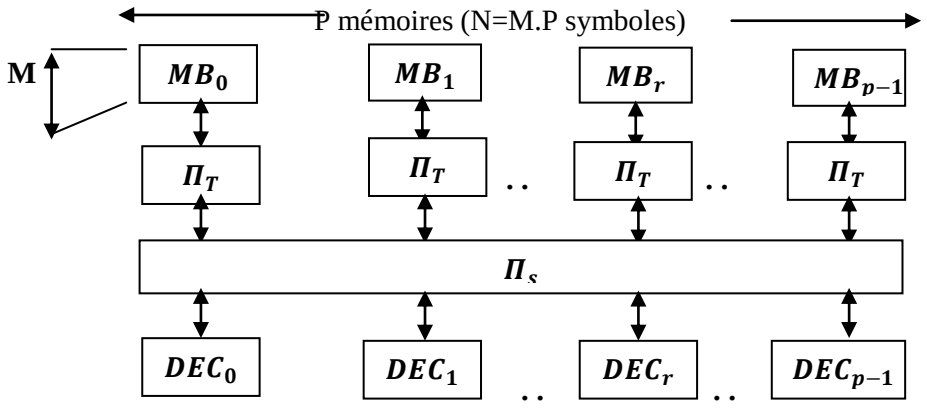


Figure 2.13: Structure de l'entrelaceur.

Dans l'ordre naturel, le codage s'effectue par blocs indépendants de *M* symboles consécutifs. Ainsi, le symbole d'indice *j* intervient dans la roulette *j / M* à l'indice temporel *j* mod *M.* De même, dans l'ordre entrelacé, le symbole d'indice *k* intervient dans la roulette $r = k / M$ à l'indice temporel $t = k \bmod M$, soit $k = M.r + t$ avec $r = 0..P-1$ et $t = 0..M-1$. La fonction d'entrelacement Π associe à chaque indice *k* de l'ordre entrelacé, le symbole d'indice $\Pi(k) = \Pi(t,r)$ correspondant dans l'ordre naturel. La fonction d'entrelacement peut se décomposer en deux niveaux suivant : une permutation spatiale Π_S (t, r) et une permutation temporelle Π_T (t, r), comme défini :

$$\Pi(k) = \Pi(t, r) = \Pi_S (t, r) + \Pi_T (t, r)$$

Ainsi, le symbole k de l'ordre entrelacé est lu du banc mémoire Π_S (t, r) à l'adresse Π_T (t, r). Lorsque l'on décode la première dimension du code, la trame est lue dans l'ordre naturel et donc les permutations spatiale et temporelle sont remplacées par la fonction Identité [23].

b) Un exemple simple

L'exemple suivant permet de clarifier la construction de l'entrelaceur. Il consiste à construire un simple code (18,6,3). Choisissons la permutation temporelle $\Pi_T(t)$ ={1,4,3,2,5,0} et pour la permutation spatiale un décalage circulaire d'amplitude $A(t \bmod 3)$, où la roulette d'indice r est associée au banc mémoire d'indice $\Pi_S(t, r) = (A(t \bmod 3) + r) \bmod 3$ avec $A(t \bmod 3) = \{2,0,1\}$.

L'entrelaceur pour les 3 indices temporels t=0 (2.a), t=1 (2.b) et t=5 (2.c) est illustré sur la figure 2.14. Les 18 symboles dans l'ordre naturel sont divisés en 3 roulettes de 6 symboles correspondant à la première dimension. Dans la deuxième dimension, à l'indice temporel t, les symboles $\Pi_T(t)$ des roulettes de la première dimension sont sélectionnés et permutés par la permutation spatiale $\Pi_S(t, r)$.

Par exemple, à l'indice temporel t=0, les symboles à l'indice $\Pi_T(0)$=1 sont lus dans chaque roulette. Ils sont décalés d'une amplitude $A(0 \bmod 3) = 2$. Ainsi, les symboles 0 des roulettes 0,1 et 2 de la première dimension vont respectivement dans les roulettes 2,0 et 1 de la seconde dimension.

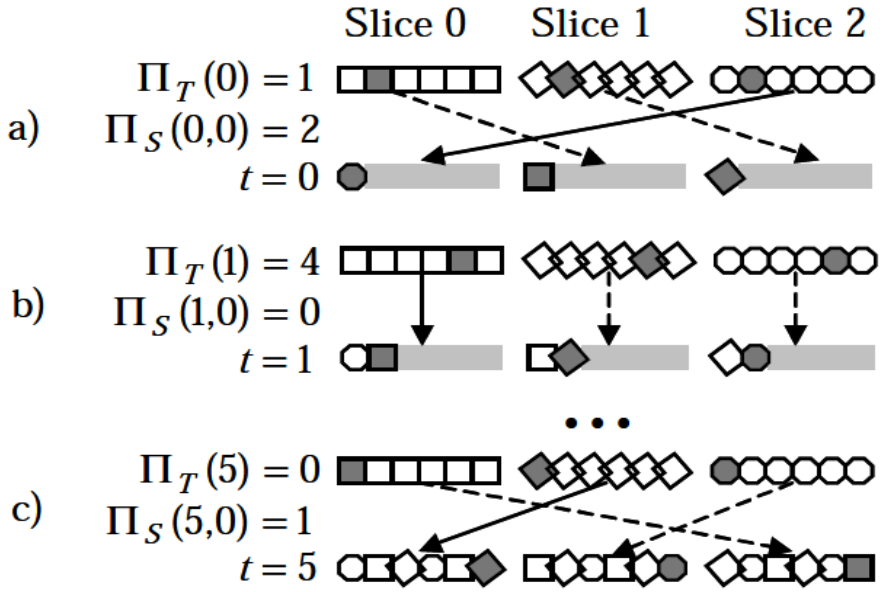


Figure 2.14 : Un exemple simple d'un code (18,6,3).

2.5 Conclusion

Dans ce chapitre, nous avons donné quelques généralités sur le parallélisme, nous avons parlé également de la toolbox « Parallel Computing Toolbox » de Matlab et donné quelques exemple de programmes parallèles sous Matlab.

Dans le cas des turbocodes, trois niveaux de parallélismes ont fait l'objet de beaucoup de travaux de recherche. Dans ce travail, nous nous sommes intéressés plus particulèrement au parallélisme du sous-bloc et au parallelisme du turbo-décodeur complet. Nous avons décrit les turbocodes à roulettes qui permettent d'avoir un parallélisme du sous-bloc. Les résultats de simulation de cette famille de turbocode ainsi que le parallélisme du turbo-décodeur complet vont être présentés dans le chapitre suivant.

SIMULATION, RESULTATS ET INTERPRETATION

3.1 Introduction

Dans ce chapitre, on s'intéressera à l'implémentation séquentielle et parallèle sous Matlab des turbocodes convolutifs. Leurs performances seront montrées par des résultats de simulation. Cette simulation a été faite sous Matlab dans le but de tester rapidement et efficacement les performances des différents algorithmes de Turbo codage et Turbo décodage, d'autant plus que ce logiciel renferme la boite à outil « Parallel Computing Toolbox » qui permet la programmation locale en paralléle sur une seule machine multiprocesseur ou multicœur.

3.2 Données de simulation

On peut distinguer trois parties dans la simulation : une partie pour le codeur, une partie pour le canal et une autre partie pour le décodeur. La simulation du codeur turbo est basée sur la description évoquée dans le chapitre 1. L'encodeur simulé est composé de deux RSC identiques concaténés en parallèle et séparés par un entrelaceur. Comme paramètres de simulation, nous avons choisi :

- Le type des données à transmettre (information aléatoire pour le test du BER, image comme application,..);
- l'entrelaceur utilisé est aléatoire ;
- deux valeurs de taux du code R=1/2 (avec perforation) et R=1/3 (sans perforation);
- techniques de terminaison de treillis des codes convolutifs pour forcer les deux RSCs de retourner à l'état initial 0 (ajoutant des séquences des bits(0), sans terminaison et terminaison par un état circulaire (code CRSC) ;

- les simulations sont effectuées dans le cadre d'une transmission dans un canal AWGN ;
- le décodeur simulé est composé de deux SISOs utilisant le même algorithme de décodage (Log-MAP) ;
- pour l'étude des performances, nous avons supposé une estimation parfaite de la variance du bruit dans le canal, dont la formule est donnée par $\sigma^2 = (2.R.\frac{E_b}{N_0})$;
- notons aussi que dans nos calculs du BER en fonction de E_b/N_0 , on fixé le nombre de blocs erronés à 15 pour arrêter la simulation.

3.3 Simulation

Nous avons partagé nos résultats de simulation en deux parties ; dans la première partie, nous exposons les résultats permettant l'étude des performances (test du BER) du turbo code séquentiel classique et cela en transmettant des données aléatoires. Comme application, nous avons choisi de transmettre une image couleur RGB. Dans la deuxième partie, nous passons à la parallélisation du turbocode selon les deux niveaux de parallélisme :

- Le parallélisme du sous-bloc : les turbocodes à roulettes, et
- Le parallélisme du turbo décodeur complet.

Dans cette partie, nous allons également faire l'étude des performances, la transmission d'images comme application, et une comparaison des résultats avec ceux des turbocodes séquentiels, surtout en ce qui concerne temps d'exécution et performance.

3.3.1 Turbo code séquentiel classique

3.3.1.1 Etude des performances

a) Effet du nombre d'itérations

Le nombre d'itérations est un facteur clé dans la détermination des performances des codes turbo. Il permet par le biais du processus de décodage itératif, l'amélioration continue de l'information à priori ($LLR(d^i)$) des bits d'information.

Il est clair d'après la figure 3.1 que l'augmentation du nombre d'itérations est suivie d'une amélioration considérable du taux d'erreurs binaire (BER) du code turbo considéré. Par exemple pour un rapport signal sur bruit E_b/N_0= 2 dB le taux d'erreurs binaire BER passe de l'ordre de 10^{-3} dans la troisième itération, pour atteindre l'ordre de 3.10^{-4} dans la cinquième itération et l'ordre de 2.10^{-4} dans la huitième itération.

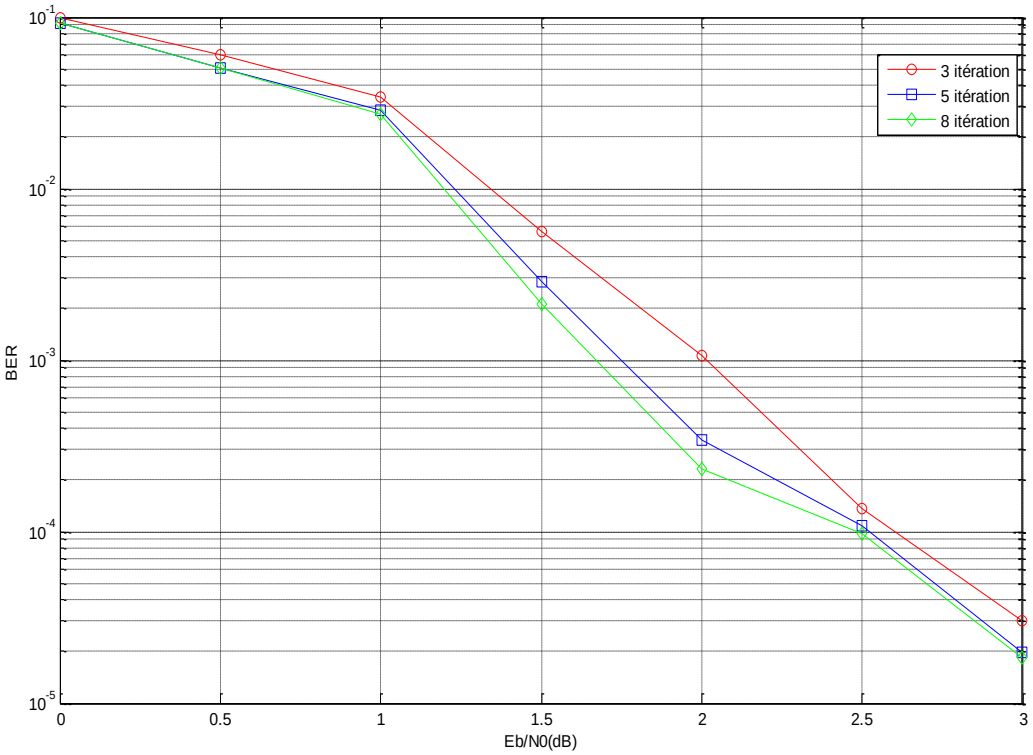


Figure 3.1 : Performance en fonction du nombre d'itérations du turbocode pour R=1/2, v=2.

Nous remarquons que la différence entre la courbe correspondant à cinq itérations et celle de huit itérations n'est pas très significative ; par exemple, si on fixe le BER à 10^{-3}, on aura un gain de code à 0.0001 dB entre les deux courbes (celle de cinq itérations et celle de huit itérations). Pour cette raison, nous choisissons cinq itérations dans le reste de nos simulations car ce nombre d'itérations présente un compromis entre les performances et le temps d'exécution.

b) Effet du polynôme générateur

Sur la figure 3.2 sont représentées les courbes du taux d'erreurs binaire (BER) pour des différents polynômes générateurs ; (g_0, g_1)=(7,5) dont la longueur du registre à est égal à v=2, le polynôme (g_0, g_1) = (15,13) de longueur v=3, et le polynôme (g_0, g_1) = (23,17) de longueur v=4. L'efficacité de la longueur de registre de décalage (v) est déterminée en effectuant la comparaison des courbes pour v=2, v = 3 et v=4 (bloc de taille 256 bits). Nous remarquons que l'augmentation de la longueur du registre améliore le taux d'erreurs binaire (BER), mais en contre partie elle consomme plus de temps ; le temps d'exécution est 2792.477023 secondes pour v =2, celui de v=3 est 16099.194253 et celui de v=4 est 41000.911697 secondes. Dans notre cas, nous avons choisi le polynôme générateur (g_0, g_1)=(7,5) dont la longueur du registre à est égal à v=2 car le facteur temps est un facteur vital.

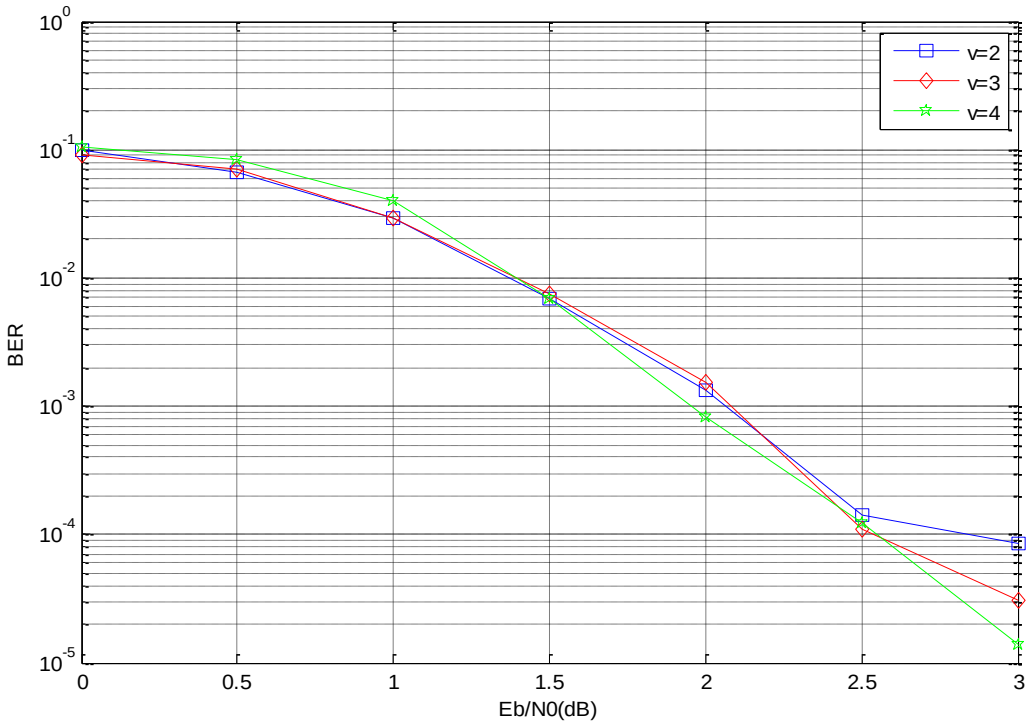


Figure 3.2 : Performances selon la longueur du registre de décalage du turbocode.

c) Effet de perforation

Le turbocodeur utilisé renferme deux RSC_s concaténés en parallèle permettant de générer des bits de parités lesquels vont être ajoutés aux bits systématiques d'information. Si on transmet le code produit sans perforation on aura un taux de code R=1/3.

Pour voir l'effet de la perforation sur les performances en BER des codes turbo, on a perforé les séquences de parités de chaque RSC pour avoir à la fin un taux de code R=1/2.

Il est clair d'après la figure 3.3 que les performances de BER avec R=1/3 (sans perforation) sont meilleures que les performances avec R=1/2 pour un bloc de longueur égale à 256 bits. Pour un BER de 10^{-3}, on aura un gain de code égale à 0.6 dB en passant du rendement R=1/3 à R=1/2 , ce qui donne une différence importante en puissance de transmission.

D'autre part, le temps d'exécution correspondant au taux du code R=1/2 est plus court que celui du taux de code R=1/3, ce qui est logique car la perforation consiste à éliminer certains bits de redondances des trames à transmettre.

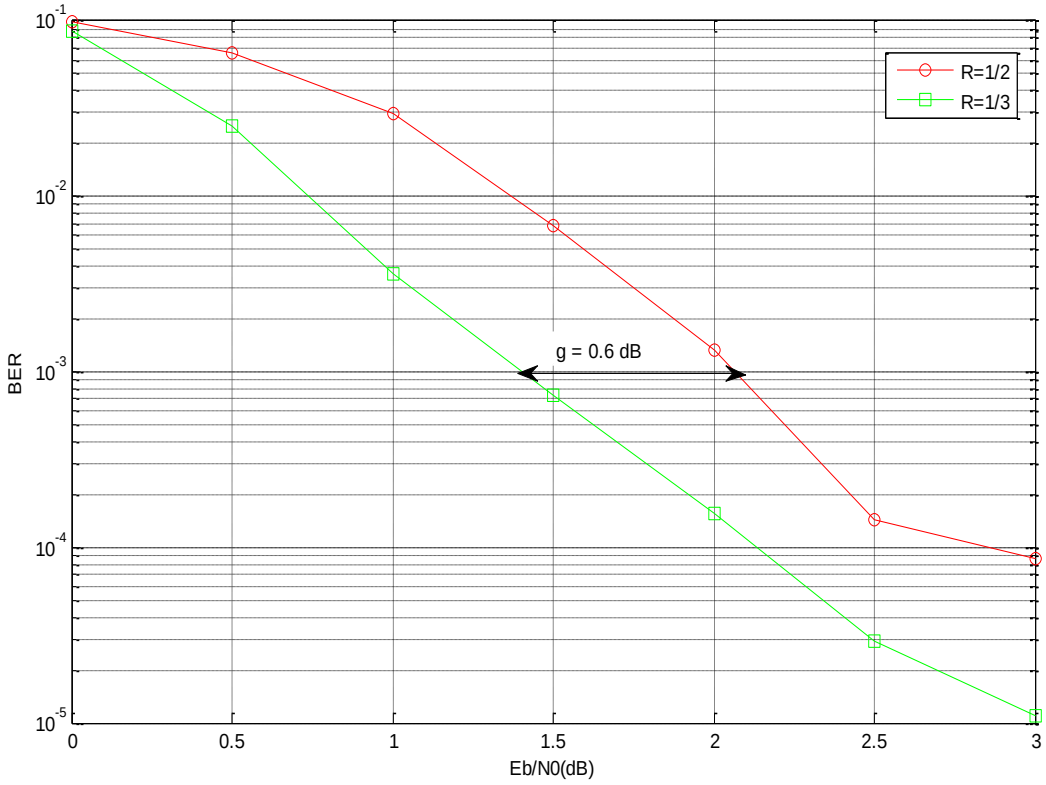


Figure 3.3 : Performances du code turbo dans le cas de perforation et sans perforation.

d) Effet de terminaison du treillis

La figure 3.4 permet de comparer le turbocode avec terminaison (terminaison de treillis des codes convolutifs pour forcer les deux RSCs de retourner à l'état initial 0 en ajoutant des séquences des bits 0) et le cas de turbocode sans terminaison de treillis.

A partir de cette figure, nous remarquons que la courbe représentant les performances du BER dans le cas du turbocode avec terminaison (courbe en bleu) est un légèrement meilleure que celle du turbocode sans terminaison (courbe en rouge), mais la différence entre ces deux courbes n'est pas très significative. En contrepartie, nous avons un gain en temps d'exécution des turbocodes sans terminaison par rapport à leurs homologues avec terminaison (temps d'exécution d'un code convolutifs avec terminaison est 2792.477023 secondes et celui des codes sans terminaison est 1348.221519 secondes).

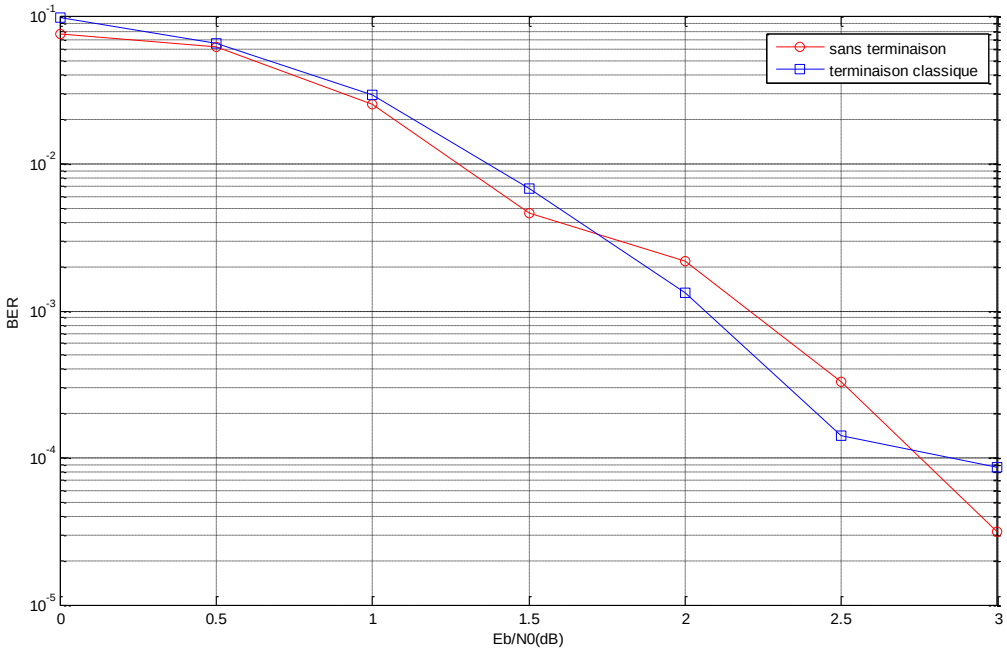


Figure 3.4 : Performances de terminaison du treillis du turbocode.

3.3.1.2 Application à la transmission d'une image

Les figures suivantes mettent en évidence le pouvoir correcteur des turbocodes sur une image couleur RGB de dimension (480, 640, 3). Nous avons simulé le cas d'un canal gaussien bruité avec un rapport signal sur bruit Eb/N0 égal à 0.5dB (figure 3.6), et un rapport 1.5dB (figure 3.7) et comparé les résultats de cette simulation.

Figure 3.5 : Image RGB originale.

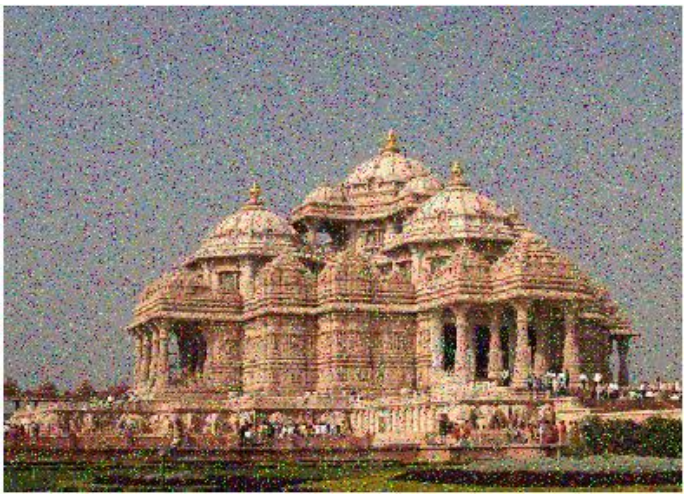


Figure 3.6 : Résultat de correction par un turbo code de l'image RGB avec Eb/N0=0.5dB.

Figure 3.7 : Résultat de correction par un turbo code de l'image RGB avec Eb/N0=1.5dB.

A partir de ces figures et en inspectant l'image reçue au niveau du récepteur dans le cas d'un rapport signal sur bruit Eb/N0 égal à 0.5dB, nous remarquons que cette image est bruitée mais elle porte toujours de l'information. Ce bruit est dû au fait que le turbodécodeur au niveau du récepteur à corrigé beaucoup d'erreurs de transmission mais il a échoué dans la correction de certaines erreurs car le canal est trop bruité. Cette image peut être améliorée en la filtrant par un simple filtre au niveau du récepteur (voir figure 3.8). Dans le cas de l'image reçue avec un rapport signal sur bruit Eb/N0 égal à 1.5 dB (voir figure 3.9), nous remarquons qu'elle est moins bruitée que la précédente ; c'est très logique car le canal renferme moins de bruits. Concernant le temps d'exécution, ce dernier est aux alentours de 16614.973670 secondes qui est un temps très important dû à la taille importante de l'image d'une part et au temps important consommé par le turbocode séquentiel (en particulier son algorithme de décodage) d'autre part.

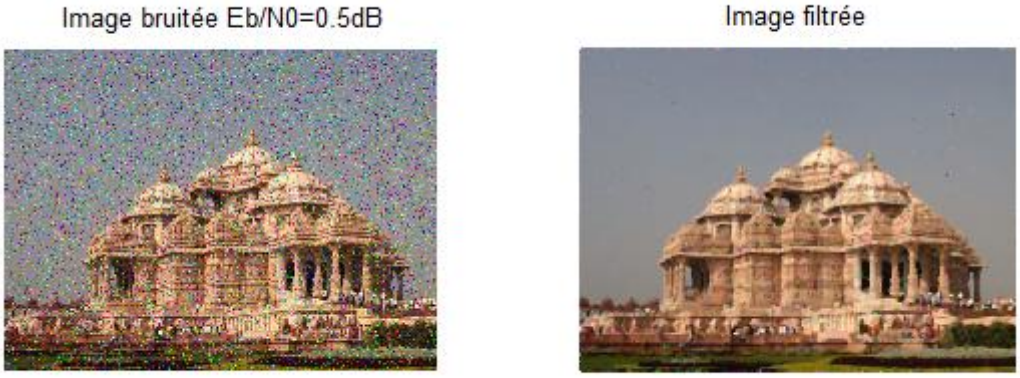


Figure 3.8 : Image reçue filtrée (cas où Eb/N0=0.5 dB).

Figure 3.9 : Image reçue filtrée (cas où Eb/N0=1.5 dB).

Le problème qui se pose alors est donc est le temps d'exécution important qui a beaucoup d'influence sur le débit de transmission, d'où la nécessité de parallélisation des turbocodes dans les nouveaux systèmes de communication numériques pour résoudre ce problème.

3.3.2 Parallélisation du turbo code

3.3.2.1 Etude des performances (test du BER)

Pour la programmation parallèle de la courbe permettant le test de BER (performances des turbocodes), nous avons proposé deux solutions ; la première fait partie du niveau de parallélisme du turbo décodeur complet. Elle repose sur le fait que pour une valeur fixée du rapport signal sur bruit Eb/N0 correspond une valeur de BER (voir figure 3.10 à titre d'exemple), et cette valeur sont complètement indépendante des valeurs précédentes de Eb/N0 et de BER. Pour cette raison nous avons réfléchi à une parallélisation dans ce sens par une boucle parfor. La deuxième solution fait partie du niveau de parallélisme du sous-bloc ; les turbocodes à roulettes bien détaillés dans le chapitre 2.

a) Parallélisation de la boucle des valeurs du BER en fonction des Eb/N0

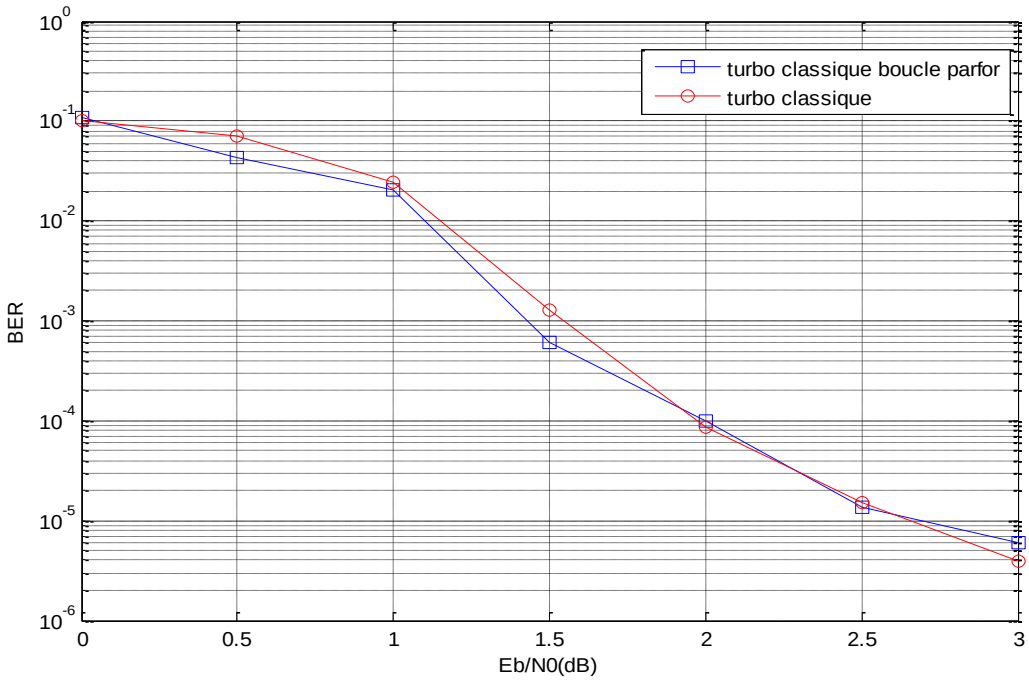


Figure 3.10 : Parallélisation de la boucle des valeurs du BER en fonction des Eb/N0.

Les résultats de simulation d'un bloc de 1024bits; en utilisant la parallélisation de la boucle des valeurs du BER donne un temps d'exécution (pour 8 processeurs) égal à 13783.817136 secondes qui est très court par rapport au temps correspondant à un turbocode classique (un seul processeur) égal à 41178.575997 secondes. Concernant les performances (figure 3.10), nous remarquons que les deux sont presque identiques.

b) Turbocode à roulettes

En appliquant le principe des turbo codes à roulettes ; la trame d'information qu'on souhaite transmettre de N symboles est découpée en P blocs de M symboles chacun, avec N=M.P (symboles).

Le tableau 3.1 récapitule les résultats concernant le temps d'exécution des turbocodes à roulettes obtenus pour une taille fixée de la trame N égale à 1024 et un nombre variable de sous-blocs. Ces résultats mettent en évidence le fait que l'augmentation du nombre des sous-blocs est suivie d'une amélioration considérable du temps d'exécution. Ceci est logique car ça signifie que la tâche de décodage est partagée entre plusieurs processeurs ce qui diminuera le temps d'exécution de cette tâche.

Nombre de sous-bloc	Taille de trame (symboles)	Temps d'exécution (s)
4	**1024**	**4118.618843**
8	**1024**	**400.021954**
16	**1024**	**115.010996**

Tableau 3.1 : Influence du nombre des sous-blocs sur le temps d'exécution des turbocodes à roulettes.

Comparaison des performances du turbocode classique et du turbocode à roulettes, nous construisons un code binaire à roulette (1024, 128, 8) de taux de code R=1/2, et nous comparons ses performances à celles d'un turbocode classique séquentiel (1024,1024,1). Le résultat est montré sur la figure 3.11. Nous remarquons que le turbocode à roulettes présente des performances meilleures que celles du turbocode classique, en plus de l'avantage du temps d'exécution court des turbocodes à roulettes qui est égal à 400,021954

secondes dans le cas de l'utilisation de 8 processeurs, alors que le turbocode classique nécessite 41178.575997 secondes d'exécution.

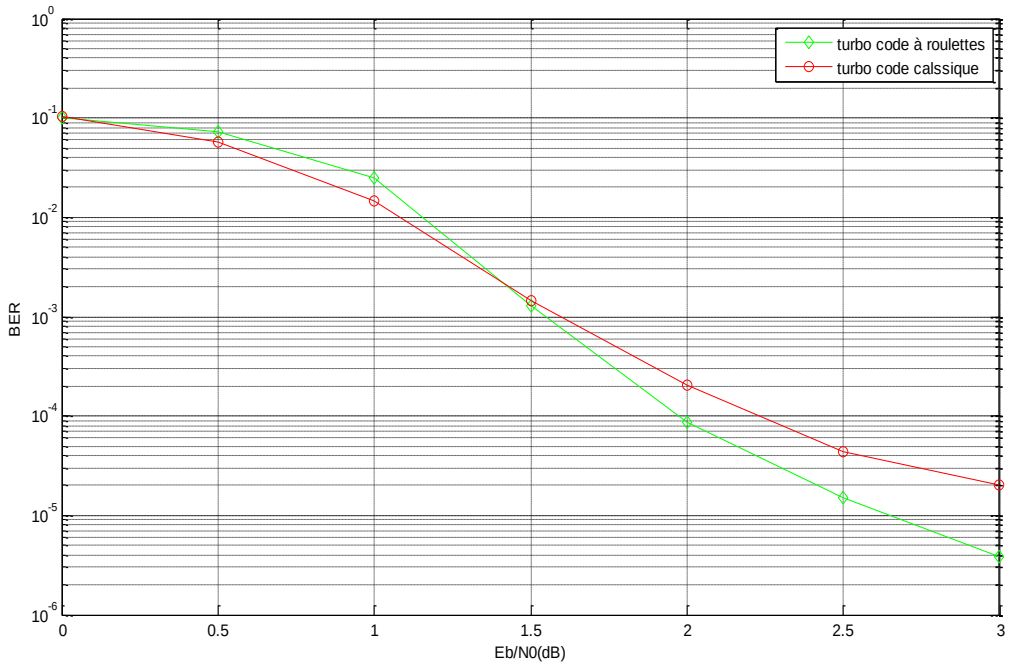


Figure 3.11 : Performances du turbocode à roulettes (1024,128,8)et celles du code classique (1024,1024,1) pour R=1/2 et 5 itérations.

3.3.2.2 Parallélisation de la transmission d'une image RGB

La transmission en parallèle d'une image peut se faire de deux manières ; la première consiste à partager (ou distribuer) cette image entre plusieurs processeurs. Chacun s'occupe d'une partie de cette image et à la fin on rassemble les différentes parties pour reconstituer l'image (ça fait partie du parallélisme du parallélisme du turbocode complet). La distribution de l'image RGB peut se faire soit en séparant les trois plans correspondants aux trois couleurs entre trois processeurs, soit en partageant cette images en plusieurs parties (8 par exemple pour augmenter le nombre de processeurs se partageant cette tâche) horizontalement ou verticalement. Chacune est une partie RGB de l'image, 8 processeurs par exemples peuvent être utilisés, chacun traitant en parallèle une partie de cette image. A la fin, ces parties vont être rassemblées pour reconstituer l'image.

Le résultat de simulation du premier cas est montré sur la figure (3.12) de temps d'exécution égale à 5764.457865 secondes, A partir de ce résultat, nous remarquons que le temps d'exécution avec trois processeurs est devenu très court par rapport un seul processeur (figure 3.7).

Figure 3.12 : Résultat de correction d'une image RGB avec Eb/N0=1.5dB par trois processeurs.

La deuxième manière est d'utiliser les turbocodes à roulettes pour transmettre cette image. Dans ce cas les processeurs ne vont par traiter des parties distinctes de cette image, mais chaque trame de l'image complète sera divisée en P sous trames et chacun des P processeurs de l'émetteur jouera le rôle d'un codeur. Au niveau du récepteur, P processeurs chacun jouant le rôle d'un décodeur vont décoder les sous-trames et à la fin on récupère l'image complète reçue.

3.4 Conclusion

Dans ce chapitre, nous avons fait la simulation des turbocodes classique ainsi que leurs performances et la programmation parallèle de ces turbocodes. La simulation montre l'influence du nombre d'itérations, la longueur de registre de décalage, le rendement et la terminaison de treillis sur les performances des turbocodes classiques, après l'interprétation des résultats, nous avons choisi les valeurs optimales des paramètres précédents qui donnent un compromis entre les performances désirées et le temps d'exécution court.

D’autre part, nous avons vu l’influence des différents niveaux de parallélisme sur la programmation parallèle de ces turbocodes point de vue performances et rapidité d’exécution.

L’architecture parallèle permet une réduction de la latence de 50% à 100 s par rapport à une architecture série classique. Les simulations des performances montrent que l'introduction du parallélisme ne dégrade pas les performances comme nous l’avons vu dans les figures (3.10, 3.11, 3.12) et l’augmentation du nombre des sous-blocs améliore le temps d’exécution.

CONCLUSION GENERALE

Ce mémoire de thèse porte sur l'étude et la programmation parallèle des turbocodes convolutifs et en particulier leurs algorithmes de décodage. Ces codes très utilisés actuellement dans le domaine des communications numériques, sont devenues primordiales afin d'atteindre des débits de transmission très élevé et réduire la latence.

Pour pouvoir atteindre de très hauts débits et pour accélérer l'exécution des turbocodes, une exploitation du parallélisme s'avère nécessaire. Trois niveaux de parallélisme existent dans la littérature ; le parallélisme au niveau des métriques utilisées dans l'algorithme MAP, le parallélisme par une exécution simultanée de plusieurs décodeurs BCJR SISO travaillant sur une même trame, et le parallélisme au niveau des Turbo-décodeurs complets pour les faire travailler sur des trames différentes. Ces trois niveaux de parallélisme permettent d'accroître le débit et réduire la latence.

Dans notre travail, on s'est intéressé plus particulièrement aux deux derniers niveaux cruciaux dans la parallélisation des turbocodes. Nous avons tout d'abords simulé le turbocode classique et dégagé les paramètres optimaux qui donnent un compromis entre performance et temps d'exécution. Ensuite, nous avons étudié et simuler les turbocodes à roulettes qui sont une architecture qui permet de paralléliser le processus de décodage en divisant chaque trame du message reçu en P sous-blocs et décoder ces différents sous-blocs en parallèle par P processeurs ce qui permet d'améliorer le temps de transmission et également les performances. Les résultats présentés dans ce mémoire sont encourageants et prometteurs.

Comme perspectives, le parallélisme doit être exploité sur tous les niveaux (niveaux bas) pour atteindre des débits plus élevés, tout en diminuant la latence globale du turbo décodage. Nous pouvons également faire l'implantation parallèle de ces architectures sur des cartes à base de circuit FPGA.

BIBLIOGRAPHIE

ARTICLES ET LIVRE

[1] Olivier Muller, *Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit*, Thèse de doctorat, ENST Bretagne, 2007.

[2] C.Shannon, *A mathematical theory or communication*, Bell system technical journal, Tech.Rep, 1948.

[3] G.Leroux, *Conception d'architectures parallèles de turbo-décodeurs de codesproduit: produits : De l'exploration à la mise en œuvre*, Thèse de doctorat, Septembre 2008.

[4] Houssein Jaber, *Conception architecturale haut débit et sûre de fonctionnement pour les codes correcteurs d'erreurs*, Thèse de doctorat, Décembre 2009.

[5] C.Berrou, C.Douillard and M.Jezequel, *Multiple parallel concatenation of Circular Recursive Systematic Convolutional (CRSC) codes*, Annales des telecommunication, 1999.

[6] C.Berrou, *Codes et Turbocodes*, Springer, 2007.

[7] A.J.Viterbi, *Error Bounds for Convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Trans, April 1967.

[8] V.Dégardin, *Analyse de la faisabilité d'une transmission de données haut débit sur le réseau électrique basse tension*, Thèse de doctorat, Décembre 2002.

[9] G.J.Forney, *Performance of concatenated codes*, Key papers in the development of coding theory, E.Berlekamp, Ed. IEEE Press, 1974.

[10] A.D.Liveris, Xiong Zixiang, C. N. Georghiades, *Distributed compression of binary sources using conventional parallel and serial concatenated convolutional codes*, Proc. IEEE DCC '03, March 2003.

[11] Jonathan Leonard Roth, *Performance optimization and parallelization of turbo decoding for software-defined radio*, September 2009.

[12] Quang Trung Dong, *Le principe du calcul stochastique appliqué au décodage des turbocodes : conception, implémentation et prototypage sur circuit FPGA*, Thèse de doctorat, Décembre 2011.

[13] 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, Mar 1974.

[14] Paul Feautrier, *Parallélisation automatique*, ENS de Lyon, Septembre 2008.

[15] Arnaud Legrand and Yves Robert, *Algorithmique parallèle*, Dunod, 2003.

[16] J.C.Dubacq, *Architecture pipeline*, IUT de Villetaneuse, *2009.*

[17] C. Germain, D.Etiemble, *Architecture des ordinateurs*, IUP Miage-FIIFO, *2008.*
.

[18] M.Siadat, C.Diou, *Architecteur des systèmes à microprocesseurs*, 2008.

[19] Brok Palen, *Matlab for research computing*, May 2011.

[20] John Burkardt,Virginia Tech , *Parallel matalab at FSU : parallel for loops* , April 2010.

[21] O.Muller,A.Baghdadi et M.Jézéquel , *Efficacité parallélisme dans le turbo –décodage convolutif*, Article, Décembre 2010.

[22] D.Gnaedig , E.Boutillion , M.Jézéquel ,V.C.Gaudet et P.G.Gulak , *Les Turbo-codes à roulettes*, Article, 1995.

[23] D.Gnaedig , E.Boutillion , M.Jezequel , *Design of three-dimensional multiple slice turbo codes*, Article, Novembre 2004.

[24] *S.Benaissa,a.Elaziza,Turbo code classique dans un systeme de communication numérique*,Mémoire de l'ingénieur d'état,2011.

WEBOGRAPHIE

[25] http://www.mathworks.com/products/parallel-computing/.