جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

N° d'ordre : 97 /2023

Centre des Etudes Doctorales : Sciences et Techniques et Sciences Médicales
Formation Doctorale : Sciences de l'Ingénieur, Sciences Physiques, Mathématiques et Informatique
Discipline : Génie électrique
Spécialité : Télécommunications et Systèmes embarqués
Laboratoire : Systèmes Intelligents, Géoressources et Energies Renouvelables

Thèse pour l'obtention de DOCTORAT

Conception, optimisation et implémentation parallèle et temps réel des codes LDPC sur des plateformes embarquées

Préparée par : Mr Benhayoun Mhammed

Soutenue le 09/12/2023 devant le jury :

Nom et Prénom	Grade	Etablissement	Qualité
Najia ES-SBAI	PES	Faculté des Sciences et Techniques de Fès	Présidente
ABDELAZIZ AHAITOUF	PES	Faculté Polydisciplinaire de Taza	Rapporteur & examinateur
Mounia ZAIM	PH	Ecole Supérieure de Technologie de Salé	Rapporteur & examinateur
Hiba CHOUGRAD	PH	Ecole Nationale des Sciences Appliquées de Fès	Rapporteur & examinateur
Abdessalam AIT MADI	PH	Ecole Nationale des Sciences Appliquées de Kenitra	Examinateur
Malika ALAMI MARKTANI	PH	Ecole Nationale des Sciences Appliquées de Fès	Examinateur
Anass MANSOURI	PES	Ecole Nationale des Sciences Appliquées de Fès	Co-directeur de Thèse
Ali AHAITOUF	PES	Faculté des Sciences et Techniques de Fès	Directeur de thèse

Remerciement

Avant tout, j'adresse mes vifs remerciements à mes directeurs de thèse M. Ali Ahaitouf, professeur à l'FST de Fès, et M. Anass Mansouri, professeur à l'ENSA de Fès, pour leurs conseils, leur aide ainsi que leurs encouragements, sans lesquels je n'aurais jamais pu arriver au terme de ce travail. Merci encore une fois, pour le temps que vous avez consacré à mon encadrement.

Au nom de monsieur **Ali AHAITOUF,** je tiens à remercier aussi tous les membres du laboratoire Systèmes Intelligents, Géo-ressources et Energies Renouvelables **(SIGER)** qu'il préside.

Mes remerciements vont de même aux membres du jury : La présidente Mme Najia ES-SBAI, Les professeurs M. ABDELAZIZ AHAITOUF, Mme Mounia ZAIM et Mme Hiba CHOUGRAD pour avoir accepté d'être rapporteurs de ce travail ; les professeurs Mr ABDESSALAM AIT MADI et Mme Malika ALAMI MARKTANI en tant qu'examinateurs.

J'adresse un remerciement à tous ceux qui ont de près ou de loin contribué à la réalisation de ce projet.

Mes remerciements s'adressent enfin à ma mère, mon père, ma sœur, mes deux frères, mon épouse, mes trois enfant Kenza, Meryem et Omar, pour m'avoir soutenu et encouragé durant ces années d'épreuves.

Résumé

Les codes LDPC (Low-Density Parity-Check) sont des codes correcteurs d'erreurs qui sont largement utilisés dans les systèmes de communication numériques actuels. La complexité de leur algorithme itératif de référence LLR-BP (Log Likelihood Ratio Belief Propagation) influence significativement la latence de décodage. Afin de réduire au maximum cette latence, nous avons proposé dans cette thèse une nouvelle implémentation logicielle sur des systèmes embraqués parallèles de l'algorithme de référence et de ces algorithmes dérivés notamment le MS (Min Sum), le NMSA (Normalised MS).

La première contribution est l'étude et l'analyse des performances en termes de correction d'erreurs et en termes de complexité calculatoire des différents algorithmes de décodage LDPC. Cette étude est réalisée sur un environnement de simulation et de test que nous avons développé afin de valider les algorithmes de décodage LDPC, de simuler les différents paramètres des codes LDPC et de tracer le comportement de l'algorithme au sein du processeur et de la mémoire cache.

La deuxième contribution consiste en la proposition d'une méthode d'ordonnancement optimisée des algorithmes de décodage LDPC permettant d'accélérer la correction d'erreur et d'assurer un traitement parallèle pour profiter des plateformes multiprocesseurs.

La troisième contribution consiste en l'optimisation de la structure de données échangées lors du procédé de décodage LDPC et l'optimisation de l'implémentation sur une plateforme embarquée à base de processeur quadricœurs ARM, ce qui a permis de réduire la latence à moins de 150µs avec un taux d'erreur binaire de moins de 10^{-9}. Les travaux proposés, dans le cadre de cette thèse, peuvent être intégrés dans les communications ultra fiables à petite latence (Ultra relaible low latency communications URLLC) utilisées pour répondre aux exigences de communication des applications telles que la conduite autonome, la télé-chirurgie et l'industrie 4.0.

Mots clés : Décodeurs LDPC, URLLC, LLR-BP, MSA, NMSA

Abstract

Low-Density Parity-Check (LDPC) codes are error correction codes widely used in modern digital communication systems. The complexity of the iterative reference algorithm LLR-BP (Log Likelihood Ratio Belief Propagation) has a significant influence on decoding latency. In order to minimise this latency, we have proposed in this thesis a new software implementation on parallel-embedded systems of both the reference algorithm and its derived algorithms, including MS (Min Sum) and NMSA (Normalised MS).

The first contribution is the performance investigation and analysis in terms of error correction and calculation complexity of the various LDPC decoding algorithms. This study is performed on simulation and check environment developed to validate the LDPC decoding algorithms, simulate the different parameters of the LDPC codes and monitor algorithm behavior within processor and cache memory.

In the second contribution, we propose a new optimized scheduling method for LDPC decoding algorithms to speed up error correction and ensure parallel processing to take advantage of multiprocessor platforms.

In the third contribution, we propose a new memory organization of the several messages exchanged during the LDPC decoding process, and we propose an efficient implementation on an embedded platform based on ARM quad-core processors, which allowed latency reductions to less than 1ms with a bit error rate of less than 10-9. The proposed work can be integrated into ultra-reliable low-latency communications (URLLC), used to meet the communication requirements of applications such as self-driving cars, tele-surgery and industry 4.0.

Keywords: LDPC decoder, URLLC, LLR-BP, MSA, NMSA

Table des matières

Introduction générale 1

Chapitre I : Les codes LDPC 6

1 Les codes correcteurs d'erreurs 6

1.1 Chaîne de communications numériques 6

1.2 Codage Canal 8

1.2.1 Familles de Codes Canal 8

1.2.2 Historique des codes correcteurs d'erreurs 9

1.3 Caractéristiques des codes en blocs linéaires 12

1.3.1 Taille et rendement d'un code en bloc 12

1.3.2 Loi de codage d'un code en bloc 13

1.4 Encodage par la matrice génératrice G : 13

1.4.1 Matrice génératrice G 13

1.4.2 Forme systématique de la matrice génératrice G 14

1.5 Décodage par matrice de contrôle de parité H 15

1.5.1 Matrice de contrôle de parité H 15

1.5.2 Représentation sous forme de graphe de la matrice de parité 16

1.5.3 Equations de contrôle de parité 17

1.5.4 Détection des erreurs par syndrome : 18

1.5.5 Distance minimal de Hamming 18

1.6 Evaluation du pouvoir de corrections d'un code 20

2 Les Codes LDPC 22

2.1 Construction des codes LDPC 24

2.1.1 Influence des cycles courts sur le procédé de décodage 25

2.1.2 Codes Quasi-Cycliques LDPC 26

2.2 Encodage des codes LDPC 27

3 Décodage des codes LDPC 29

3.1 Décodage à décision dure 29

3.1.1 Algorithme Bit Flipping 30

3.1.2 Algorithme Bit Flipping amélioré et dérivés 30

3.2 Décodage à décision souple par propagation de croyances 31

3.2.1 L'algorithme LLR-BP : 31

3.2.2 LLR pour un canal de transmission AWGN 34

3.3 BP Revue de la littérature des différents algorithmes d'approximation de l'algorithme LLR- 35

3.3.1 Algorithme Min Sum 35

3.3.2 Algorithmes Normalized MS et Offset MS 38

3.3.3 Algorithme Self Adjustable Offset Min-Sum *39*
3.3.4 Algorithme MS premier min *40*
3.3.5 Algorithme variable weighted MS *41*
3.4 Etude comparative des complexités de calcul des différentes approximations de l'algorithme LLR BP 42
3.5 Ordonnancement des algorithmes à propagation de croyance 43
3.5.1 Ordonnancement par inondation *43*
3.5.2 Ordonnancement brassé *44*
3.5.3 Ordonnancement dynamique *45*
3.6 Etude comparative des Complexités des algorithmes d'ordonnancement 48
Conclusion : 49
Chapitre II : Etude et analyse de la complexité d'implémentation du décodeur LDPC 50
1 Architecture de l'environnement de test du décodeur LDPC 50
1.1 Bloc générateur de séquence et images tests 51
1.2 Bloc Codage LDPC 52
1.2.1 Calcul de la Matrice génératrice G *52*
1.2.2 Modélisation de l'encodage LDPC *53*
1.3 Bloc générateur de bruit AWNG (Box Muller) 54
1.4 Bloc Moniteur 56
1.5 Architecture du bloc de décodage LDPC 57
2 Structuration des données : 60
2.1 Organisation matricielle 60
2.2 Organisation par méthode de stockage des matrices creuses (sparse matrix) 61
2.3 Organisation en Graphe 63
3 Résultats des simulations 65
3.1 Performance d'une communication avec codage et sans codage 66
3.2 Effet de nombre maximal d'itération sur les performances de décodage 67
3.3 Effet du rendement du code R sur les performances de décodage 68
3.4 Effet de la taille du bloc d'information sur les performances de décodage 69
3.5 Effet des algorithmes de mise à jour des messages C2V sur les performances de décodage 70
3.6 Effet des méthodes d'ordonnancement sur la vitesse de convergence et les performances de décodage : 71
4 Outils et Résultats des profilages 73
4.1 Outils logiciels de profilage 73
4.2 Résultats de profilage 74
4.2.1 Effet de la structure de donnée sur le nombre de cycles CPU consommés *74*

4.2.2 Effet des algorithmes de mise à jour et de l'ordonnancement sur le nombre de cycles CPU consommés ... *78*
Conclusion : ... 82
Chapitre III : Proposition d'un nouvel algorithme d'ordonnancement à faible complexité ... 83
1 Ordonnancement par nœuds vicinaux ... 83
2 Détection et élimination des cycles d'ordre 4 : ... 85
3 Nœud de variable le plus influent : ... 88
4 Evaluation des performances de l'algorithme proposé ... 91
4.1 Performances en correction d'erreurs ... 91
4.2 Analyse de la complexité : ... 95
Chapitre IV : Implémentation des décodeurs LDPC sur des plateformes embarquées multi-cœurs ... 97
1 Revue des implémentations des décodeurs LDPC : ... 97
2 Architecture proposée : ... 99
2.1 Structuration des données dans la littérature ... 100
2.2 Optimisation de la structuration des données ... 102
2.3 Optimisation de l'algorithme par ordre d'appel dans le processeur ... 106
2.4 Structuration des données par mémorisation de la matrice d'expansion ... 110
2.5 Evaluation des cycles d'exécution des différentes structurations des données proposées 113
3 Modèle de calcul parallèle proposé ... 116
4 Implémentation logicielle du décodeur LDPC sur des plateformes multicoeurs ... 117
4.1 Bibliothèque d'implémentation parallèle Open MP (Open Multi-Processing) ... 117
4.2 Implémentation sur la plateforme HPC MARWAN ... 118
4.3 Implémentation sur processeur ARM quad-core Cortex-A72 ... 118
4.4 Résultat de latence sur les plateformes multicœurs ... 119
Conclusion : ... 122
Conclusion Générale & Perspectives ... 123
Annexe A : Liste des contributions ... 127
1 Publications indexées ... 127
2 Conférences internationales ... 127
3 Workshop internationaux ... 127
Annexe B : Démonstration de l'équation de mise à jour des messages C2V ... 128

Liste des figures

Figure 1. 1 : Chaine de communication numérique 6
Figure 1. 2 : Répartition des éléments d'un mot code systématique 15
Figure 1. 3 : (a) exemple de matrice H, (c) graphe de Tanner correspondant 17
Figure 1. 4 : impact de la distance minimal de Hamming sur le pouvoir de détection et de correction des erreurs 19
Figure 1. 5 : Evaluation des performances de décodage 21
Figure 1. 6 : Graphe d'une matrice H de grande taille avec des ensembles de contrôle de parité qui se chevauchent 22
Figure 1. 7 : Graphe d'une matrice H de grande taille avec l'approche LDPC 23
Figure 1. 8 : Cycles dans la matrice H. (a) d'ordre 4, (b) d'ordre 6 et (c) d'ordre 8 25
Figure 1. 9: Exemple de matrice d'expansion de dimension 12x96 de la norme Wimax 27
Figure 1. 10: Matrice 2304 x 1152 de la norme Wimax 27
Figure 1. 11: Calcul des messages extrinsèques et des informations a posteriori 31
Figure 1. 12: Représentation graphique de la fonction involutive Ø(x) 35
Figure 1. 13: Représentation des zones de surestimation de la fonction $\emptyset x$ 37
Figure 1. 14: Représentation graphique de la fonction non-linéaire $fx = ln(1 + e - x)$ et son interpolation sur 5 régions. 40
Figure 1. 15: Ordonnancement par inondation 44
Figure 1. 16: Illustration de l'ordonnancement Horizontal shuffle 45
Figure 1. 17: Illustration de l'ordonnancement Vertical shuffle 45

Figure 2. 1 : Architecture de l'environnement de test et de simulation 51
Figure 2. 2 Générateur de séquence à partir d'images tests 52
Figure 2. 3: Etapes de calcul de la matrice génératrice G 53
Figure 2. 4 : Bloc encodage LDPC et modulation BPSK 53
Figure 2. 5 : Comparaison des échantillons aléatoires générés par la fonction Box-Muller et la répartition théorique de la fonction gaussienne 55
Figure 2. 6 : Bloc de modélisation du canal de communication 55
Figure 2. 7 : Modélisation du bloc moniteur 56
Figure 2. 8 : Modélisation du bloc décodeur LDPC 57
Figure 2. 9 : Diagramme des fonctions de l'implémentation du décodeur LDPC 59
Figure 2. 10 Structuration des données par tableaux 62
Figure 2. 11 : Structure de données utilisée par la méthode de Graphe 63
Figure 2. 12 : Exemple de structuration des données par Graphe de la matrice (5, 10) 64
Figure 2. 13 : BER avec codage LDPC et sans codage vs SNR 66
Figure 2. 14 :BER en fonction des itérations maximales 67
Figure 2. 15 : BER en fonction du rendement R du codage 68
Figure 2. 16 : BER en fonction de la taille K du bloc d'information 69
Figure 2. 17 : BER en fonction de l'algorithme de mise à jour des messages C2V 70
Figure 2. 18 : Convergence de décodage LDPC en fonction de la méthode d'ordonnancement 71
Figure 2. 19 : BER en fonction de la méthode d'ordonnancement 72
Figure 2. 20 : Itérations moyennes lors de décodage en fonctions des différents algorithmes de mise à jour et d'ordonnancement 72
Figure 2. 21 : Résultats de simulation de la mémoire cache généré par l'outil valgrind 76
Figure 2. 22 : Diagramme en secteurs des Résultats obtenues par l'outil Valgrind 76
Figure 2. 23 : Histogramme des Résultats obtenues par l'outil Valgrind 78
Figure 2. 24 : Histogramme du nombre de cycles CPU par itération des différents algorithmes de décodage LDPC 79

Figure 2. 25 : Histogramme du nombre de cycles CPU de décodage des différents algorithmes d'ordonnancement 80
Figure 2. 26 : BER de l'algorithme NMSA en fonction de diverses valeurs α 81
Figure 2. 27 : Méthode de dichotomie appliquée sur l'intervalle [0 1] pour déterminer la valeur optimale de α 81

Figure 3. 1 : Sous-graphe du nœud de variable vj 83
Figure 3. 2 Exemple des nœuds vacinaux du nœud de variable v_1 mis en évidence en utilisant les équations de parité 84
Figure 3. 3 : Cycle d'ordre 4 entre le nœud de variable vj et ces nœuds vicinaux 85
Figure 3. 4 : Représentation d'un cycle d'ordre 4 dans la matrice H 85
Figure 3. 5 : Cycles d'ordre 4 dans la matrice (5, 10) 87
Figure 3. 6 : Etapes de l'algorithme proposé LWNS 89
Figure 3. 7 : Ordonnancement parallèle en couches (a) les nœuds de variables v_j and v_i sont parallèles $v(vi)v(vj) = \emptyset$. (b) les nœuds de variable v_j and v_i ne sont pas parallèles $vviv(vj) = vk4$ 91
Figure 3. 8 : BER et FER vs Eb/N0 des algorithmes de décodage: inondation, Brassé HS, IVRBP, OVRBP et LWNS pour les codes LDPC irrégulier (576, 288), (1152, 576) et (576, 144). 92

Figure 4. 1 Structuration des données proposée dans l'état de l'art 101
Figure 4. 2 Classement des éléments non nul par ligne 103
Figure 4. 3 Tableau de structure mémorisant les messages C2V et V2C triés par ligne 103
Figure 4. 4 Tableau de structure mémorisant pour chaque nœud de contrôle son degré d_c et le point de départ dans le tableau MEM 103
Figure 4. 5 Méthode d'accès pour le calcul des messages C2V en utilisant les deux tableau MEM et Cm 104
Figure 4. 6 Tableau Add de la matrice (5, 10) 104
Figure 4. 7 tableau Vn de la matrice (5, 10) 105
Figure 4. 8 Classement des éléments non nul par colonne 105
Figure 4. 9 Méthode d'accès pour le calcul des messages V2C en utilisant les tableaux MEM, Add et Vn 106
Figure 4. 10 : Mise à jour de la valeur LLR 107
Figure 4. 11 : Tableau MEM après l'optimisation proposée 109
Figure 4. 12 : Tableau L mémorisant les LLR des nœuds de variable 109
Figure 4. 13 : Table Col des index des colonnes par ligne de la matrice (5, 10) 109
Figure 4. 14 : Interconnexions de différentes données lors du processus de décodage en utilisant l'optimisation proposée 110
Figure 4. 15 Tableau Col et Cm nécessaire pour la mémorisation de la matrice (12, 6) 112
Figure 4. 16 : Représentation de la matrice d'expansion H_{bm} 112
Figure 4. 17 Résultats de profilage des différentes structurations de la matrice (9216, 4608) 114
Figure 4. 18 Résultats de profilage des différentes structurations de la matrice (2304, 1152) 114
Figure 4. 19 Résultats de profilage des différentes structurations de la matrice (576, 288) 115
Figure 4. 20 : Modèles de calcul parallèle pour LDPC Quasi-cyclique 116
Figure 4. 21 : Carte raspberry Pi 4 119
Figure 4. 22 : Comparaison de la latence en (ms) entre le traitement en série et le traitement en parallèle sur plateforme MARWAN pour les différentes tailles de code. 120
Figure 4. 23 : Comparaison de la latence en (ms) entre le traitement en série et le traitement en parallèle sur la plate-forme Raspberry Pi 4 pour les différentes tailles de code 120

Liste des Tableaux

Tableau 1. 1: Opérations élémentaires dans le corps $F2N$ 13
Tableau 1. 2: Approximation de la fonction non-linéaire $fx = ln(1 + e - x)$ par 5 fonctions linéaires 40
Tableau 1. 3: Complexité calculatoire pour la mise à jour d'un seul nœud de contrôle..... 42
Tableau 1. 4:Complexité équivalente des différents algorithmes d'ordonnancement..... 49

Tableau 2. 1 Dimensions des matrices creuses avec pourcentage des éléments non nuls..... 61
Tableau 2. 2 Taille des types de donnée..... 64
Tableau 2. 3 Comparaison des complexités des différentes méthodes de structuration des données. . 65
Tableau 2. 4 : Résultats de profilages des trois méthodes d'organisation des données..... 77
Tableau 2. 5 : Résultats de profilages des algorithmes de décodage LDPC..... 78

Tableau 3. 1 : index des nœuds participants aux cycles d'ordre 4 de la matrice (5, 10)..... 87
Tableau 3. 2 : Nombre de cycle et degré de chaque nœud participant aux cycles d'ordre 4 dans la matrice (5, 10) 88
Tableau 3. 3 : Comparaison des performances de l'algorithme proposé LWNS et des algorithmes de la littérature : 93
Tableau 3. 4 Nombre de mise à jour des messages C2V et V2C de l'algorithme proposé 95
Tableau 3. 5 : Comparaison des complexités des différents algorithmes existants dans la littérature et celui l'algorithme proposé..... 96

Tableau 4. 1 : Méthode de calcul des lignes et colonnes de la matrice H (12, 6) à partir des coefficient de permutation de la matrice d'expansion H_{bm} (4, 2)..... 112
Tableau 4. 2 Gain en terme du nombre de cycle CPU et de taille mémoire par rapport aux travaux précédents..... 115
Tableau 4. 3 : Performances de décodage sur les deux plateformes 121

Liste des acronymes et abréviations

3GPP	Third Generation Partnership Project
5G	Cinquième génération de normes de communication mobile
ARQ	Automatic Repeat Request
AWGN	Additif White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit Error Ratio
BF	Bit Flipping
BP	Belief propagation
BPSK	Binary Phase Shift Keying
C2V	Control node to Variable node message
CRC	Codage de Redondance Cyclique
DVBS2	Digital Video Broadcasting - Satellite Second Generation
eMBB	Enhanced Mobile Broadband
FEC	Forward Error Correction
FER	Frame Error Ratio
GDBF	Gradient Descent Bit Flipping
HS	Horizontal Shuffle
IA	Intelligence Artificielle
IDS	Informed Dynamic Scheduling
IoT	Internet of Things
IVC RBP	Informed Variable-to-Check Residual Belief Propagation
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood-Ratio
LUT	Look Up Table
LWNS	Layered Vicinal Variable Nodes Scheduling
mMTC	Massive Machine-Type Communications
MS	Min Sum
MWBF	Modified Weighted Bit Flipping
NMSA	Normalized Min-Sum Algorithm
OMS	Offset Min-Sum
OVC RBP	Oscillating Variable-to-Check Residual belief Propagation
PWL	Piece Wise linear
RBP	Residuel belief propagation
RRWGDBF	Reliability Ratio Weighted Gradient Descent Bit Flipping
RS	Reed Solomon
SAOMS	Self Adjustable Offset Min-Sum
SDR	Software Defined Radio
smMS	Second min MS

SNR	Signal-to-Noise Ratio
SVNF	Silent variable node free
URLLC	Ultra-Reliable Low-Latency Communications
V2C	Variable node to control node message
V2X	Vehicle-to-Everything
VC-RBP	Variable to Check Residuel belief propagation
VS	Vertical Shuddle
vwMS	Variable weighted MS
WBF	Weighted Bit Flipping
Wimax	Worldwide I+A2:B49nteroperability for Microwave Access

Introduction générale

Contexte

La communication ultra-fiable à faible latence, en anglais, "Ultra-Reliable Low-Latency Communications" (URLLC) est une fonctionnalité clé parmi les trois services fournis par la cinquième génération de normes de communication mobile (5G) développée par le 3GPP (The Third Generation Partnership Project), l'organisme qui produit et publie les spécifications techniques pour les réseaux mobiles. Ce type de communication est conçu pour fournir une connectivité fiable et ultra-rapide à faible latence (délai entre l'émission et la réception des données) destiné aux applications nécessitant une communication en temps réel.

Les deux autres services qui font partie de la 5G sont :

- La communication mobile à large bande améliorée (eMBB enhanced Mobile Broadband) est un service qui offre un transfert de données à haut débit pour des applications telles que la diffusion en continu des vidéos ou la réalité virtuelle. eMBB est destinée aux applications qui nécessitent un très haut de débit, mais où la latence est moins critique.
- La communication massive pour les objets connectés (mMTC massive Machine-Type Communications), est un service qui offre une prise en charge d'un grand nombre d'appareils avec des débits de données faibles, tels que des capteurs ou des objets connectés à Internet, en anglais, Internet of Things (IoT). mMTC est destiné aux applications avec une grande densité de connectivité (nombre de dispositifs connectés par unité de surface), mais où le débit et la latence sont moins critiques.

Dans cette thèse on s'intéresse aux communications de type URLLC, destinées aux applications où les retards et les échecs de transmission de données pourraient avoir des conséquences graves, telles que la conduite automatique des voitures, la téléchirurgie, l'indusie 4.0, etc…

Dans le scénario de la conduite automatique des voitures, où les véhicules circulent sans aucune intervention humaine, l'échange sans fil d'informations entre le véhicule autonome et n'importe quel objet de son environnement, permet d'améliorer le pilotage des véhicules et la sécurité routière. Cependant, il est nécessaire que le véhicule autonome qui roule à une vitesse donnée, soit capable d'envoyer et de recevoir sans erreurs ses informations en quelques milliseconde afin de réagir correctement et à temps.

Un deuxième domaine applicatif des communications de type URLLC est la téléchirurgie où un chirurgien réalise une intervention chirurgicale à distance, permettant ainsi de mutualiser les ressources et d'éviter le déplacement des patients entre les villes ou entre les pays. Cependant, la téléchirurgie présente également des défis liés aux problèmes de latence et de fiabilité de transmission entre le chirurgien et le robot situé dans un autre endroit géographique, ce qui peut affecter la précision et la sécurité des interventions.

Un autre exemple est celui de la quatrième révolution industrielle 4.0, avec un ordinateur qui contrôle des automates en leur envoyant chaque 2ms des commandes sous forme de paquet de 32 octets. Le débit nécessaire dans ce cas n'est pas très important, en revanche, un paquet ne doit pas mettre plus de 2ms pour arriver à destination.

Ces trois domaines applicatifs partagent les mêmes exigences de communication en termes de fiabilités et de latence. Dans sa version 17 de Mars 2022 [1], Le 3GPP a défini pour l'URLLC, un objectif de 1ms de latence avec une fiabilité de 99,999% pour un paquet de 32 octets, autrement dit une erreur d'un seul bit est admise pour chaque 10^5 bits envoyés.

Il existe plusieurs contraintes à prendre en compte pour garantir ces objectifs, notamment :

- Les erreurs de transmission : il s'agit d'altération involontaire de l'information transmise lors d'une communication numérique. Elle peut être causée par des perturbations sur le canal de communication (par exemple des interférences électromagnétiques, des perturbations atmosphériques ou des pertes de signal), des erreurs de synchronisation, ou encore des problèmes liés à la qualité des équipements utilisés. Ces erreurs ont un effet négatif sur la qualité du signal, rendant difficile ou impossible l'interprétation du message reçu.
- Capacité de traitement : Le traitement des données, notamment les différentes techniques de traitement de signal telles que la modulation/démodulation et le codage/décodage canal, sont également une contrainte importante dans les

communications URLLC, car il peut affecter la latence et la fiabilité de la communication.

- Coût : La mise en place d'infrastructures de communication URLLC peut être coûteuse, en particulier pour les applications nécessitant des connexions hautement fiables et à faible latence.

Parmi les différentes techniques de codage canal opérant comme des codes correcteurs d'erreur dans une communication numérique, les codes LDPC (Low Density Parity Check), connus par leur pouvoir très élevé de correction d'erreurs. Ces codes ont été adoptés par les nouvelles normes de communication comme : La DVBS2 (Digital Video Broadcasting - Satellite Second Generation) utilisée pour la diffusion de la télévision numérique de haute définition à un très haut débit par des satellites géostationnaires. Le Wimax (Worldwide Interoperability for Microwave Access) utilisé pour fournir un accès à Internet haut débit à des zones rurales ou éloignées où les connexions filaires sont difficiles ou impossibles. La 5G pour les communications de type eMBB qui assure un haut débit mobile.

Problématique & objectifs

Les algorithmes de décodage LDPC sont des algorithmes itératifs avec une complexité calculatoire très importante rendant ainsi très délicate leur utilisation pour les applications exigeantes en latence. Leur implémentation matérielle sur des circuits dédiés avec des fréquences d'horloge élevées permet de réduire la latence de décodage, en contrepartie le changement ou la mise à jour de la partie radio d'une chaine de communication nécessite dans ce cas le changement de toute la partie matérielle avec des incidences financières très élevées. Il devient donc nécessaire d'envisager une implémentation logiciel (en anglais Software) sur des architectures numériques programmables, ce qui permet au même équipement d'avoir la capacité de supporter divers algorithmes de traitement de signal et d'avoir la possibilité de modifier et de mettre à jour une fonctionnalité en téléchargeant et en exécutant une nouvelle version logicielle sans changer le matériel. Ce modèle est soutenu par l'émergence de processeurs avec une grande puissance de calcul et des systèmes embarqués multi-cœurs avec des systèmes d'exploitation temps réel permettant des calculs parallèles très puissants. Cependant, les programmeurs doivent concevoir leur programme de manière à ce que le compilateur soit capable de le décomposer en calculs parallèles, en tenant compte des contraintes matérielles de l'architecture du système embarqué, telles que l'accès partagé à la mémoire.

Dans ce contexte, l'objectif principal de cette thèse est de concevoir une implémentation logicielle des décodeurs LDPC sur des processeurs généralistes et sur des systèmes embarqués multicœurs.

Les principales contributions de cette thèse

En absence de modèle mathématique pouvant décrire les performances des décodeurs LDPC en termes de correction d'erreurs, la première contribution consiste à concevoir un environnement de test et de simulation fonctionnelle modélisant une chaine de communication numérique complète avec un codage et un décodage LDPC. Cet environnement permettra à la fois :

- De valider fonctionnellement les différentes solutions et optimisations proposées.
- D'évaluer les performances des décodeurs LDPC en termes de correction d'erreur en termes de taux d'erreur binaire (BER Bit Error Ratio) et en termes de taux d'erreurs de trame ou de paquet (FER Frame Error Ratio), qui représentent respectivement le nombre de bits erronés reçus par rapport au nombre total de bits transmis et le nombre de paquet erronées reçues par rapport au nombre total de trames transmises.
- De tracer et d'analyser le comportement en processeur et en mémoire cache des différents blocs constituant le décodeur.

Ensuite, la deuxième contribution consiste en la proposition d'une nouvelle méthode d'ordonnancement des différentes étapes de décodage LDPC, permettant de réduire la complexité calculatoire et de paralléliser le lancement des calculs et d'améliorer les performances en termes de corrections d'erreurs.

La troisième contribution consiste en l'implémentation optimisée de la solution proposée en exploitant la capacité de calculs parallèles des plateformes embarquées multicoeurs, tout en gardant les mêmes performances en termes de correction d'erreur.

Organisation du manuscrit

Afin de détailler ces trois contributions, le rapport de cette thèse sera structuré comme suit :

Le premier chapitre présentera brièvement les différents codes correcteurs d'erreurs et les notions du codage canal. Il introduira ensuite les codes LDPC et le principe de décodage itératif des deux classes d'algorithmes de décodage, à décision souple et à décision dure.

Ce chapitre présentera également les notions théoriques permettant de mettre en évidence les contraintes qui augmentent la complexité du processus de décodage LDPC, ainsi que les différentes solutions proposées dans la littérature pour chaque contrainte dans le but de diminuer la complexité de décodage.

Le deuxième chapitre détaillera notre méthodologie d'implémentation de l'environnement de simulation, portable, flexible et dynamique avec les différents algorithmes utilisés pour modéliser toutes les composantes de la chaine de communication numérique (générateur de séquence, modulation BPSK, codage LDPC, générateur de bruit blanc gaussien, décodage LDPC), ainsi que le bloc qui permettra de tracer les performances en terme de BER et FER. Les méthodes de validation fonctionnelle de chaque bloc seront aussi présentées avec les outils de profilage et d'implémentation parallèle utilisés dans ce travail. Enfin les résultats des simulations et de profilage des différents algorithmes et paramètres des codes LDPC seront présentés et analysés.

Le troisième chapitre se focalisera sur une nouvelle méthode d'ordonnancement fixe des algorithmes à décision souple afin de réduire la complexité et d'avoir les mêmes performances en termes de correction d'erreurs des algorithmes à ordonnancements dynamiques connu par leur grande complexité de calcul. Nous présentons aussi, les résultats de simulation et analyse de complexités de la méthode proposée par rapport à celles proposées dans la littérature.

Le quatrième chapitre sera consacré à présenter une nouvelle optimisation de l'algorithme de décodage LDPC, de la structure de données compacte utilisées pour la mémorisation des données, ainsi que l'architecture parallèle proposée. Nous présenterons les résultats des implémentations sur processeur généraliste multicœurs. Ce chapitre détaillera aussi notre implémentation de l'algorithme optimisé sur un système embarqué multicoeurs à base de processeur ARM. Nous présenterons les différentes latences de décodage obtenues sur les différents systèmes embarqués utilisés.

Le manuscrit se termine par une conclusion générale et perspectives qui rappellent les principaux résultats obtenus lors de ce travail et les pistes de recherche pour des futurs travaux.

Chapitre I : Les codes LDPC

Les codes LDPC font partie des codes correcteurs d'erreurs en bloc linéaires opérant sur des blocs de bits de taille fixe. Ainsi, ils partagent les mêmes caractéristiques des codes en bloc linéaires sauf qu'ils sont définit à partir d'une matrice de parité de faible densité, et que le procédé de décodage est itératif avec un algorithme de décodage permettant d'assurer de bonnes performances en termes de correction d'erreurs.

Avant de détailler et d'analyser la complexité des algorithmes de décodage LDPC, ce chapitre présentera les caractéristiques en communs des différents codes correcteurs d'erreurs et celles spécifiques aux codes LDPC.

1 Les codes correcteurs d'erreurs

1.1 Chaîne de communications numériques

Les systèmes de communications numériques transmettent de l'information d'une source vers un ou plusieurs destinataires. Malgré leur hétérogénéité, tous les systèmes de communication sont basés sur un modèle commun proposé par Claude Shannon [2], qui a proposé de modéliser un système de communication avec cinq composants principaux : une source d'information, un émetteur, un canal, un récepteur et une destination.

Ces composants de communications numériques peuvent se décomposer en blocs consécutifs comme illustré sur la figure 1.1.

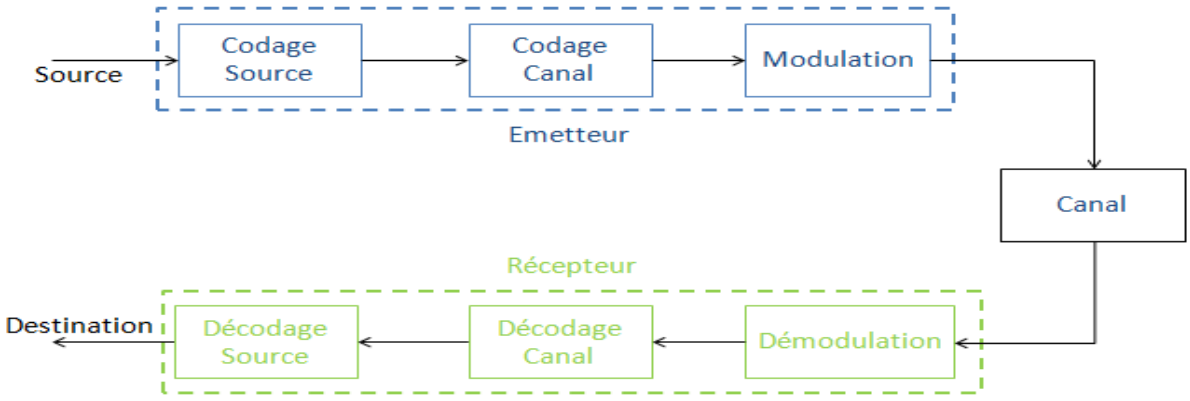


Figure 1. 1 : Chaine de communication numérique

Le signal d'information (par exemple la voix, la vidéo ou les données) est d'abord échantillonné et quantifié pour former une séquence en format binaire puis il passe par le codage source (ou la compression des données) afin de réduire la quantité d'information à transmettre, la décompression des données sera réalisée à la réception par le décodage source qui permettra de récupérer tous le signal d'information. Dans le cas où on n'utilise pas de codage canal, le modulateur transforme le flux binaire en signaux continus adapté au canal de communication.

Le canal de communication est le support physique qui est utilisé pour envoyer le signal de l'émetteur au récepteur. Le canal peut être l'atmosphère pour les communications sans fil (cas des signaux hertziens), une ligne filaire (câble en cuivre) ou un câble en fibre optique. Au cours de la transmission à travers le canal, les signaux physiques peuvent subir de nombreuses perturbations. Ces perturbations sont issues du bruit lié à l'environnement et à la nature du canal, ainsi qu'au bruit thermique des composants électroniques des équipements de transmission. Ces perturbations dégradent la qualité des signaux transmis.

A la réception, le démodulateur réalise la fonction opposée à celle du modulateur, à partir des signaux continus reçus, il estime la valeur binaire des bits reçus. Ces estimations peuvent être erronées si les perturbations induites par le canal sont trop importantes. Cela pose un problème car les données binaires reçues par le destinataire peuvent ne pas être identiques à celles émises par la source. Pour rendre la transmission de données numériques plus fiable, un codage canal est introduit dans la chaine de communication, du côté émetteur, un codage est réalisé dans le but d'ajouter de la redondance à l'information binaire transmise. La redondance et l'information utile sont liées par une loi de codage canal. Ce sont ces redondances et la loi de codage qui serviront au décodeur, au niveau de la réception, de reconstituer le signal d'origine.

Cependant, l'ajout de la redondance augmente le nombre de bits à transmettre par le canal et engendre un temps de traitement supplémentaire au niveau du codeur et du décodeur ce qui diminue la rapidité du système, ainsi, un équilibre est à assurer entre la rapidité de décodage et le gain lié à la correction des erreurs. Le choix d'un code correcteur d'erreur et de ses paramètres est déterminant pour assurer cet équilibre selon le domaine d'application et du type de canal de transmission.

1.2 Codage Canal

1.2.1 Familles de Codes Canal

La correction des erreurs de transmission se base sur l'ajout d'une quantité de bits appelés bits de redondance au message initial, le message obtenu après codage est appelé mot code, deux grandes familles de codes correcteurs d'erreurs ont été introduites, selon la méthode utilisée pour produire un mot code :

- Les codes en blocs : le message à transmettre est découpé en plusieurs blocs de taille fixe, ce type de code traite chaque bloc d'information indépendamment des autres. Il existe deux catégories de codes en bloc : les codes en bloc linéaires et les codes en bloc non-linéaires. Les codes en bloc non-linéaires ne sont jamais employés dans les applications pratiques et ne sont pas beaucoup étudiés. Les codes en bloc linéaires se subdivisent en codes non cycliques et en codes cycliques.
- Les codes convolutifs : La sortie d'un codeur convolutif dépend de l'information courante à coder ainsi que de l'information précédente et du mot code précédent.

Du côté récepteur, deux modes de correction des erreurs sont possibles :

- Correction par retransmission (ARQ: Automatic Repeat Request) : L'objectif de ce mode est l'ajout d'une petite quantité de redondance au message, de manière à permettre seulement la détection d'éventuelles erreurs de transmission. Dans le cas d'une détection d'erreurs, le décodeur demande à la source de retransmettre du message erroné. Le principal désavantage du mode ARQ est le temps nécessaire pour la demande et la retransmission du paquet.
- Auto-correction (FEC: Forward Error Correction) : Dans ce mode, la redondance introduite permet de détecter et de corriger, au niveau du décodeur, un nombre fini d'erreurs. La quantité de redondance nécessaire dans ce cas est plus grande que pour le mode ARQ, ce qui constitue le principal désavantage du mode FEC.

Un mode mixte ou hybride (ARQ-FEC) permettant de bénéficier des avantages des deux approches peut être envisagé pour les applications supportant un retard de retransmission. Grâce à la redondance fournie par le mode FEC, le système cherche d'éventuelles erreurs qui

sont ensuite corrigées. Si le taux d'erreurs est supérieur à celui supportable par la méthode FEC, la méthode ARQ intervient en exigeant la retransmission du message.

1.2.2 Historique des codes correcteurs d'erreurs

En 1948, Shannon a évalué la capacité d'un canal de propagation en fonction du rapport signal sur bruit, en anglais, Signal-to-Noise Ratio (SNR) et de sa largeur de bande ce qui a permis de définir un débit binaire limite (appelé capacité limite de Shannon, exprimé en bit/s) avec lequel une transmission peut être assuré sans erreur sur un canal bruité [2]. La formule de la limite de Shannon, établit une borne supérieure théorique pour la quantité maximale d'information qu'il est possible de transmettre à travers un canal de communication avec une certaine bande passante et en présence de bruit. La formule de la limite de Shannon est donnée par [2] :

$$C = B \times log_2(1 + \frac{S}{N}) \quad (1.1)$$

Où :

- C représente la capacité du canal en bits par seconde (bit/s).
- B est la bande passante du canal en hertz (Hz).
- S est la puissance du signal utile (signal power) en watts (W).
- N est la puissance du bruit (noise power) en watts (W).
- $\log_2$ représente le logarithme en base 2, il est utilisé dans la formule pour exprimer les quantités d'information en termes de bits par seconde.

Toutefois, dans ses travaux, Shannon n'a pas indiqué la méthode de construction de codage permettant d'atteindre cette capacité limite.

Depuis lors, plusieurs recherches ont été réalisées afin de concevoir des codes correcteurs d'erreurs, dans le but d'atteindre la limite théorique de Shannon et d'avoir un bon compromis performance/complexité.

Codes à répétition :

Le premier codage canal utilisé était le codage à répétition, qui consiste à envoyer plusieurs copies de la même information, le décodage est réalisé en comparant les différentes copies bit par bit, le décodage est moins complexe dans ce cas. Cependant, plusieurs bits sont

utilisés rien que pour protéger un seul bit de l'information utile. Aussi si la même erreur survient au niveau de plusieurs copies du message, l'erreur ne peut pas être corrigée.

Codes par bits de parités :

Pour réduire au maximum le nombre de bit de protection, le codage par bit de parité est introduit, ce type de codage consiste à ajouter un seul bit de parité pour protéger l'ensemble des bits de l'information à transmettre. Le bit de parité est calculé de façon à assurer la nullité modulo 2 de la somme des bits, à la réception cela permet de détecter les erreurs simples sans pouvoir les corrigés. Cependant ce type de codage ne permet de détecter qu'une seule erreur sans pouvoir la corriger, car la nullité modulo 2 est assurée lorsqu'un nombre pair d'erreurs survient.

Pour augmenter le pouvoir de détection, plusieurs bits de contrôle de parité peuvent être ajoutés de telles sortes que chaque bit de contrôle de parité protège un sous ensemble de bits de l'information utile. Le bit de contrôle de parité ainsi que le sous ensemble de bit qu'il protège est appelé un ensemble de contrôle de parité. Cependant, nous rencontrons le même problème si un nombre pair d'erreurs se produisent dans un même ensemble de contrôle de parité, dans ce cas, les bits erronés ne peuvent pas être détectés.

Codes de Hamming :

Dans les années 1950, Richard Hamming [3] a proposé une méthode qui permet de corriger des erreurs multiples, sa méthode se base sur l'utilisation des ensembles de contrôle de parité qui se chevauchant, permettant ainsi de détecter même un nombre paire d'erreur produit sur un même ensemble de contrôle de parité. Ce code permet aussi de corriger une seule erreur par paquet. Si deux erreurs sont détectées l'émetteur demande à la source de retransmettre l'information. Depuis lors, tous les codes correcteurs d'erreur ont adopté ce principe de chevauchement entre les ensembles de contrôle de parité.

Codes Cycliques :

Dans les années 1960, le codage de redondance cyclique (CRC) a été découvert par W. Wesley Peterson [4], la particularité de ce type de code est que le vecteur obtenu à partir des permutations circulaires du mot code est aussi un mot code. Ainsi l'information est représentée sous forme de polynôme auquel on ajoute un polynôme qui représente la redondance tel que le message obtenu soit divisible par le polynôme générateur, le décodage est réalisé en vérifiant si le polynôme correspondant au message reçu est divisible par le polynôme générateur, ce codage permet la détection de toutes les erreurs de transmission, mais ne permet pas de corriger aucune erreur.

Code BCH :

Les codes BCH [5] sont des codes cycliques, ils portent le nom de leurs inventeurs Bose, Ray-Chaudhuri et Hocquenghem, ces codes cycliques sont construits de façon à permettre la correction d'un nombre d'erreur défini au préalable, lors de décodage le calcul du polynôme localisateur d'erreurs et du polynôme évaluateur d'erreur est nécessaire pour pouvoir corriger les erreurs.

Code Reed-Solomon :

Les codes Reed-Solomon (RS) [6] sont des codes cycliques et plus précisément des codes BCH non binaires. Ainsi, le codage est réalisé avec un polynôme générateur dont les coefficients sont des symboles (ensemble de bit) et non des valeurs binaires, le décodage est réalisé avec les mêmes méthodes de décodage BCH mais adapté aux symboles non binaires.

Code convolutifs :

Les codes convolutifs, introduits en 1955 par Elias [7], diffèrent d'un code bloc par le fait que le mot code en sortie du codeur ne dépend pas seulement des k entrées à un instant donné mais aussi des m mots code précédents, le codeur est représenté en treillis. Il s'agit d'une représentation graphique en deux dimension, l'axe horizontal représente le temps, et l'axe vertical représente les différents états possibles, ce qui permet de mettre en évidence le paramètre temporel, le décodage est réalisé principalement en utilisant l'algorithme de Viterbi proposé en 1967 par A. J. Viterbi [8], il permet de calculer la valeur la plus probable des états étant donnée les valeurs reçues depuis le canal. Sa complexité augmentant exponentiellement avec la longueur du code

Les turbo-codes :

Les turbo-codes ont été découverts par C. Berrou et al. en 1993 [9]. Un turbo-code est la concaténation d'un ou de plusieurs codes convolutifs séparés par des blocs de permutations. Une détection par maximum de vraisemblance est envisagée pour le décodage des turbo codes avec l'introduction pour la première fois du principe itératif de décodage qui consiste à exécuter un ensemble d'itérations jusqu'à l'obtention d'un mot code valide.

Les codes LDPC :

Ce principe de décodage itératif a permis la redécouverte en 1995 par David MacKay [10] des codes LDPC (Low-Density Parity-Check), découverts initialement par Robert G.

Gallager en 1962 [11] qui a proposé une méthode générale pour construire des codes LDPC pseudo aléatoires, dont la complexité n'était pas supportée par la technologie de l'époque. En 1981 Tanner [12] leur a donné une nouvelle interprétation plus simple sous forme de graphique, ce qui a permis à David MacKay avec l'avènement de décodage itératif, la monté des vitesses d'horloge des processeurs et la miniaturisation des circuits intégrés de proposer une construction des codes LDPC et de montrer qu'ils ont de très bonne performance en terme de correction d'erreurs de transmission. Actuellement, les codes LDPC sont utilisés par les nouvelles normes de communication numérique telles que WiMAX, DVB-S2, la 5G etc.

D'autres codes existent tels que les codes raptor [13], les codes polaires [14], etc… L'objectif de cette section n'est pas d'être exhaustif mais de donner un aperçu sur l'historique des codes correcteurs d'erreur. Cette thèse se concentrera sur les codes LDPC, ces codes ont été choisis parce qu'ils sont connus pour être proches de la limite de Shannon, pour leurs performances en correction d'erreur. Cependant, malgré leur adoption par des normes de communication numérique d'actualité, leur complexité de décodage pose toujours un défi d'implémentation pour d'autres normes de communication plus exigent en latence.

1.3 Caractéristiques des codes en blocs linéaires

1.3.1 Taille et rendement d'un code en bloc

Le principe du codage en bloc noté C(N, K), consiste à découper l'information utile à transmettre en blocs de longueur K écrits dans l'alphabet $\{0, 1\}$. Ensuite, chaque bloc d'information utile U de K bits est transformé en un bloc d'information codée C de N bits, appelé mot code, moyennent l'ajout de $N - K$ bits de redondances.

Le rapport $R = \frac{K}{N} \in]0, 1[$ représente le rendement de codage, c'est le rapport entre le nombre de bits de l'information utile et le nombre total de bits transmis, ainsi plus le rendement d'un code est faible, plus le nombre de bits de redondance est important et par conséquence plus son pouvoir de correction est élevé. Par contre, le nombre de bits d'information utile est faible ce qui diminue le débit utile.

1.3.2 Loi de codage d'un code en bloc

La loi de codage qui permet de transformer un bloc d'information de K bits en un mot code de N bits, peut être représenté par une application g définit de F_2^K à F_2^N, où F_2^K et F_2^N sont des corps de vecteurs binaires de longueur respectivement K et N. Cette application est injective du fait que chaque vecteur mot code *C* de F_2^N à au plus un antécédent par g, ce qui revient à dire que deux vecteurs de messages utiles U_1 et U_2 de F_2^K ne peuvent pas avoir la même image par g, il existe donc 2^K mots code parmi les 2^N vecteurs de F_2^N. Un code en bloc C (N, K) est dit linéaire si et seulement si l'application g est linéaire, autrement dit, elle vérifie les deux conditions ci-dessous :

$$\forall\, (U_1, U_2) \in\ F_2^K\ \ g(U_1 \oplus\ U_2) =\ g(U_1) \oplus\ g(U_2) \qquad (1.\ 2)$$

$$\forall\, (U_1) \in\ F_2^K, \qquad \forall\, \gamma \in \{0, 1\} \quad g(\,\gamma \bullet U_1) =\ \gamma \bullet g(U_1) \qquad (1.\ 3)$$

En F_2^K et F_2^N, l'addition est équivalente à un ou exclusif logique $\oplus$ et la multiplication est équivalente à un Et logique « • » (voir tableau 1.1).

Tableau 1. 1: Opérations élémentaires dans le corps F_2^N

a	B	a $\oplus$ b	a•b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

1.4 Encodage par la matrice génératrice G :

1.4.1 Matrice génératrice G

Dans le cas des codes en bloc linéaires, le passage d'un vecteur d'information de longueur K, $U(u_0, \ldots, u_{K-1})$ vers un mot code de longueur N, $C(x_0, \ldots, x_{N-1})$, peut être réalisé moyennant la multiplication du vecteur d'information *U* par une matrice génératrice G de taille $K \times N$. Cette matrice génératrice est formée par K mots code disposés par ligne.

En effet, les propriétés (1. 2) et (1. 3) de l'application linéaire définissant un code en bloc linéaire permettent de déduire les propriétés ci-dessous :

- Le vecteur NULL est un mot code.
- La somme binaire de deux mots codes est un mot code.
- Le produit d'un scalaire par un mot code est aussi un mot de code.

Ce qui coïncide avec la définition d'un sous espace vectoriel. Ainsi, les 2^K mots code forment un sous espace vectoriel de dimension K de l'espace vectoriel F_2^N. Il existe donc, K mots code $g_0, g_1, \ldots, g_{K-1}$ de F_2^N linéairement indépendants qui forment une base de ce sous espace vectoriel. Par conséquence, chaque mot code C s'écrit sous forme de combinaison linéaire des mots code de la base $g_0, g_1, \ldots, g_{K-1}$, comme suit :

$$C = u_0 \bullet g_0 \oplus u_1 \bullet g_1 \oplus \ldots \oplus u_{K-1} \bullet g_{K-1} \qquad (1.\ 4)$$

Où les scalaires $(u_0, \ldots, u_{K-1})$ représentent les bits du vecteurs d'information U, ainsi l'équation devient :

$$C = (u_0, u_1 \ \ldots \ u_{K-1})\ o \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{K-1} \end{bmatrix} = \mathrm{U}\ o \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{K-1} \end{bmatrix} \qquad (1.\ 5)$$

Où le symbole « o » désigne le produit qui permet le passage à l'équation (1. 4). Les K mots codes constituant la base du sous espace vectoriel C (N, K) disposés en ligne forme une matrice génératrice G définie par :

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{K-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,N-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{K-1,0} & g_{K-1,1} & \cdots & p_{K-1,N-1} \end{bmatrix} \qquad (1.6)$$

Les 2^K vecteurs mot code de dimension N, n'est que l'image de la matrice génératrice G définie par :

$$C(N,K) = Img(G) = \{C \in 2^N, tel\ que\ \forall\ U \in 2^K\ ,\ C = U\ o\ G\} \qquad (1.\ 7)$$

1.4.2 Forme systématique de la matrice génératrice G

Un codage est dit systématique lorsque les K bits d'information sont présents dans K positions déterminés du mot code, en général les K premiers bits du mot code. Ainsi, K sorties du codage sont exactement égales à ses entrées et par conséquence ne nécessitent pas de calcul, après le processus de décodage, l'extraction de l'information utile se fait sans aucun calcul supplémentaire, ce qui permet de réduire la complexité de codage et de décodage.

La matrice génératrice G d'un code systématique, s'écrit sous la forme d'une sous matrice identité I_{KxK} et une sous matrice $P_{K\times(N-K)}$:

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{K-1} \end{bmatrix} = \begin{bmatrix} \underbrace{\begin{matrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{matrix}}_{\text{Sous matrice } I_{KxK}} & \underbrace{\begin{matrix} p_{0,0} & p_{0,1} & \cdots & p_{0,N-K-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,N-K-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K-1,0} & p_{K-1,1} & \cdots & p_{K-1,N-K-1} \end{matrix}}_{\text{Sous matrice } P_{K\times(N-K)}} \end{bmatrix} \quad (1.8)$$

Dans ce cas, tous les mots code générés contiennent au début les K bits d'information suivis par les N-K bits de redondance comme illustré dans la figure 1.2.

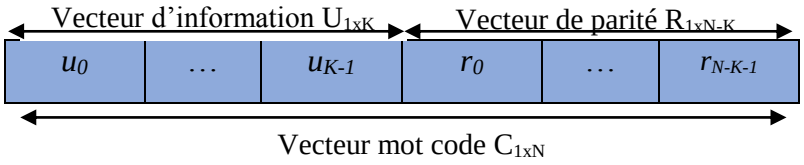


Figure 1. 2 : Répartition des éléments d'un mot code systématique

Ainsi, le calcul d'un mot code C, est réduit au calcul des bits de parité ($r_0\ r_1 \dots r_{N-K-1}$) selon l'équation ci-dessous :

$$R_{1\times N-K} = U_{1\times K}\ o\ P_{K\times N-K} \quad (1.9)$$

1.5 Décodage par matrice de contrôle de parité H

1.5.1 Matrice de contrôle de parité H

Etant donné un code en bloc linéaire C (N, K) dont les mots code sont générés par une matrice génératrice G, la matrice de contrôle de parité H permettant la détection des erreurs de transmission est toute matrice de dimension $N-K\ \times N$ telle que :

$$G\ o\ H^t = 0 \quad (1.10)$$

L'existence d'une telle matrice pour n'importe quel code linéaire découle des propriétés de l'algèbre linéaire, en effet, on dit que deux sous espaces vectoriels F_2^r et F_2^t de l'espace vectoriel F_2^N sont orthogonaux si :

- $r + t = N$
- Tous les vecteurs de F_2^r sont orthogonaux à tous les vecteurs F_2^t

Donc, il existe un sous espace vectoriel de dimension $N - K$ orthogonal au sous espace vectoriel C(N, K) = F_2^K, et par la suite il existe $N - K$ vecteurs de longueur N, h_0, h_1, . . ., h_{N-K-1}, qui définissent une base de ce sous espace vectoriel. La matrice de contrôle de parité H est la matrice constituée par ces N – K vecteurs de base disposés par ligne et on a pour tout mot code C de C(N, K) :

$$C \; o \; H^t = 0 \tag{1.11}$$

Etant donnée une matrice génératrice de la forme systématique (1.8), la matrice de contrôle de parité $\mathbf{H}_{N-K\times N}$ du code C(N, K) est calculée à partir de la matrice génératrice G par l'équation (1.11) ci-dessous :

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{N-K-1} \end{bmatrix} = \Big[\underbrace{\begin{matrix} p_{0,0} & p_{1,0} & \cdots & p_{K-1,0} \\ p_{0,1} & p_{1,1} & \cdots & p_{K-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0,N-K-1} & p_{1,N-K-1} & \cdots & p_{K-1,N-K-1} \end{matrix}}_{\text{sous matrice } P^T_{(N-K)\times K}} \; \underbrace{\begin{matrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{matrix}}_{\text{sous matrice } I_{N-K}} \Big] \tag{1.12}$$

Où P^T est la matrice transposée de P, sa dimension est $(N-K)\times K$, et I_{N-K} est la sous matrice identité.

1.5.2 Représentation sous forme de graphe de la matrice de parité

La matrice de contrôle de parité d'un code en bloc binaire peut également être décrite en utilisant un graphe de Tanner [12] contenant N nœuds de variables correspondant aux N colonnes de la matrice H et donc aux bits du mot code C(x_1, …, x_N) et N – K nœuds de contrôles correspondant aux N – K lignes de la matrice H. Les nœuds de variables sont reliés aux nœuds de contrôles via des branches bidirectionnelles, un nœud de variable v_j est relié à un nœud de contrôle c_i par une branche, si et seulement si, l'élément correspondant à la j^{ème} ligne et la i^{ème} colonne de la matrice de contrôle de parité est non nul ($H_{ij} = 1$). La figure 1.3, représente la matrice de contrôle de parité (a) et le graphe de Tanner correspondant (b).

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
c_1	0	0	1	0	0	1	1	1	0	0	0	0
c_2	1	1	0	0	1	0	0	0	0	0	0	1
c_3	0	0	0	1	0	0	0	0	1	1	1	0
c_4	0	1	0	0	0	1	1	0	0	1	0	0
c_5	1	0	1	0	0	0	0	1	0	0	1	0
c_6	0	0	0	1	1	0	0	0	1	0	0	1
c_7	1	0	0	1	1	0	1	0	0	0	0	0
c_8	0	0	0	0	0	1	0	1	0	0	1	1
c_9	0	1	1	0	0	0	0	0	1	1	0	0

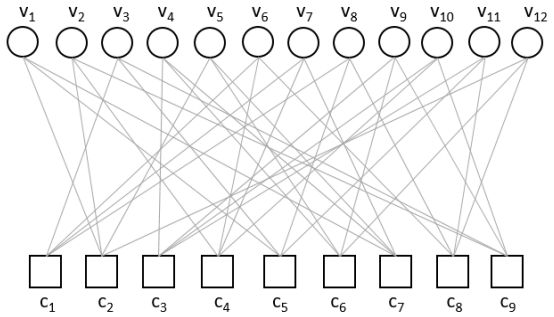


Figure 1. 3 : (a) exemple de matrice H, (c) graphe de Tanner correspondant

1.5.3 Equations de contrôle de parité

L'équation (1.11) peut être vue comme un système linéaire de *N* - *K* équations de parité. Tous les mots code C de forme systématique (u_0, …, u_{K-1}, x_0, …, x_{N-K-1}) de F_2^N devraient vérifier les *N*-*K* équations de parité, sinon le mot code C n'est pas un mot code valide. Les équations de parités sont calculées à partir de l'équation (1.13) ci-dessous :

$$C\ o\ H^t = (u_0\ , \dots , u_{K-1}, x_0\ , \dots , x_{N-K-1})\ o \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,N-K-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,N-K-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K-1,0} & p_{K-1,1} & \cdots & p_{K-1,N-K-1} \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = 0 \qquad (1.13)$$

On trouve le système d'équation de parité suivant, où la sommation $\sum$ est une sommation binaire équivalente à un ou exclusif logique $\oplus$:

$$\begin{cases} x_0 = \sum_{i=0}^{i=K-1} u_i \bullet p_{i,0} \\ x_1 = \sum_{i=0}^{i=K-1} u_i \bullet p_{i,1} \\ \vdots \\ \vdots \\ x_{N-K-1} = \sum_{i=0}^{i=K-1} u_i \bullet p_{i,N-K-1} \end{cases} \qquad (1.14)$$

1.5.4 Détection des erreurs par syndrome :

Nous considérons un mot de code X = (x_1, …, x_N) transmis via un canal de transmission bruité à entrée binaire, nous désignons Y = (y_1, …, y_N) ∈ 2^N la séquence reçue du canal de transmission.

Le principe de détection des erreurs par syndrome est de calculer le produit matriciel entre le transposée du message reçu Y et la matrice *H* en utilisant l'équation (1.11) :

$$S = H \circ Y^T \tag{1.15}$$

Si le vecteur syndrome S de l'équation (1.15) est nul, alors toutes les équations de parités sont vérifiées et le message reçu est un mot code valide sinon le message contient des erreurs. Cependant, trouver un mot de code valide ne signifie pas que le message reçu est identique à celui qui a été généré lors du codage. En effet, plusieurs erreurs de transmission peuvent éventuellement faire aboutir le processus sur un mot de code valide, mais pas nécessairement celui envoyé par l'émetteur. Il est donc important d'avoir des mots de code suffisamment éloignés les uns des autres pour éviter cette confusion lors du décodage.

1.5.5 Distance minimal de Hamming

Pour quantifier l'éloignement inter mots code, On utilise la distance de Hamming, (x, y), définie comme le nombre de positions pour lesquelles deux vecteurs x et y ont des valeurs binaires diffèrent. Pour un code C(N, K), la distance minimale de Hamming, noté d_{min}, est la valeur minimale de toutes les distances de Hamming calculées entre tous les mots code deux à deux.

$$d_{min} = min\{d(x, y) \mid x, y \in C(n, K), \ x \neq y\} \tag{1.16}$$

Dans le procédé de décodage basé sur la correction par proximité de vecteur, le message reçu détecté comme erroné converge vers le mot de code le plus proche au sens de la distance de Hamming. Par conséquence, la distance minimale du code permet d'évaluer la capacité de correction du processus de décodage, en effet, le décodage est capable de corriger au plus un nombre *e* d'erreurs [12, 13], donné par l'équation (1.17) :

$$e = E\left[\frac{d_{min} - 1}{2}\right] \tag{1.17}$$

Où E[] est la partie entière.

Dans le cas où le nombre d'erreurs par message reçu est supérieur à *e*, le message reçu se trouve plus proche d'un autre mot code valide autre que celui envoyé par l'émetteur et ainsi, le procédé de décodage converge vers le mot code le plus proche au sens de distance de Hamming.

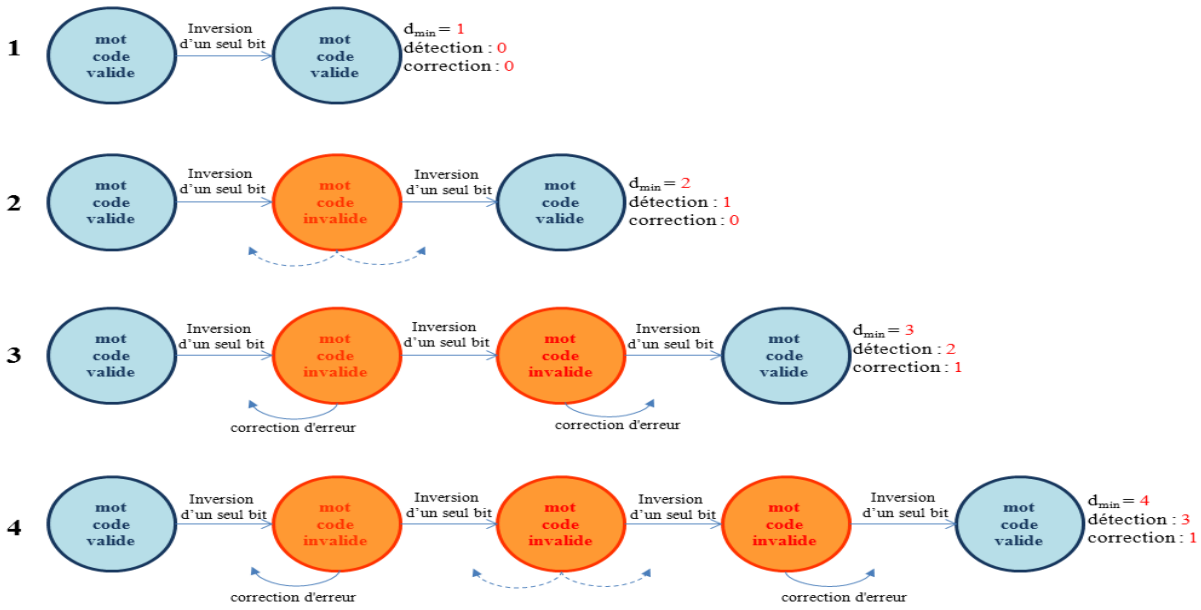


Figure 1. 4 : impact de la distance minimal de Hamming sur le pouvoir de détection et de correction des erreurs

La figure 1.4, illustre l'effet de la distance minimale de Hamming sur le pouvoir de correction des erreurs. Dans le premier cas, un seul bit inversé donne un autre mot code valide, la distance minimale de Hamming est égale à 1, ce qui ne permet ni de détecter ni de corriger aucune erreur. Dans le 4ème cas, la distance minimale est de 4, le nombre de mots code invalides qui peuvent être détectés est de 3, tandis que le nombre de mots code invalides qui peuvent converger vers un mot code valide sont seulement 2, car le 3ème mot code invalide se trouve est équidistant de deux mots code valide.

Le calcul de la distance minimale de Hamming nécessite la comparaison bit par bit de tous les vecteurs mots code deux à deux, ainsi le nombre de comparaison à réaliser est donnée par :

$$N \times C_{2^K}^2 = N \times \frac{(2^K)!}{2! \times (2^K - 2)!} = N \times 2^{(K-1)} \times (2^K - 1) \quad (1.18)$$

Où $C_{2^K}^2$ est le nombre de paires de mots code à calculer leur distance parmi 2^K mots code.

Ce qui devient impraticable dès que K est grand. Néanmoins, Pour les codes linéaires on profite des propriétés de la distance minimale sans calculer sa valeur. En effet, pour les codes linéaires, étant donné que le vecteur 0 est un mot code, la distance minimale de Hamming est égale au plus petit poids non nul d'un mot code, où le poids d'un vecteur binaire est défini comme le nombre d'éléments non nul dans ce vecteur. Par conséquent, cette propriété devrait être prise en considération lors de la construction des matrices G et H afin de maximiser les poids des mots code et par la suite maximiser la distance minimale, et le pouvoir de correction et de détection des erreurs.

1.6 Evaluation du pouvoir de corrections d'un code

Pour évaluer le pouvoir de correction d'un code canal, il est très courant de calculer le taux d'erreur binaire, en anglais bit error ratio (BER), ou le taux d'erreur trame, en anglais frame error ratio (FER), définis par :

$$BER = \frac{Nombre\ de\ bits\ erronés}{Nombre\ de\ bits\ émis} \quad (1.19)$$

$$FER = \frac{Nombre\ de\ trames\ erronées}{Nombre\ de\ trames\ émises} \quad (1.20)$$

Ces rapports d'erreurs sont calculés en variant le rapport signal sur bruit, en anglais signal noise ratio (SNR) exprimé en décibel (dB). Le pouvoir de correction d'un code canal est évalué en comparant le BER avec codage et sans codage.

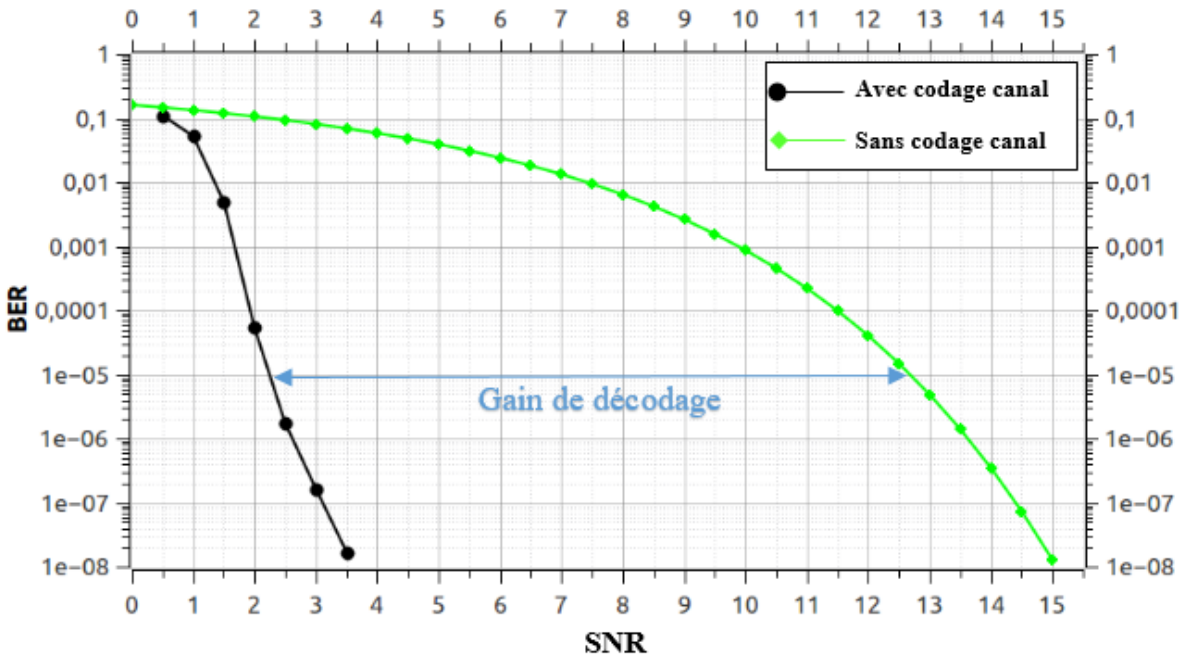


Figure 1. 5 : Evaluation des performances de décodage

La figure 1.5 illustre un exemple de différence de performance de transmission entre des systèmes de communications numériques avec codage et sans codage canal. En utilisant un codage canal le taux d'erreur binaire (BER) atteint 10^{-5} (c.-à-d. un bit erroné tous les 10^5 bits transmis) avec un rapport signal sur bruit de 2.25 dB, alors que pour une transmission sans codage canal avec le même rapport signal sur bruit de 2.25 dB, le BER n'excède pas les 10^{-1} (c.-à-d. plus d'un bit erroné tous les 10 bits transmis, sur 10^5 bits transmis on aura plus de 10000 bits erronés). Autrement dit, en utilisant un codage canal, la communication est fiable malgré que le rapport signal sur bruit soit de 2.25 dB, alors que pour avoir le même degré de fiabilité (BER à 10^{-5}), il faut que le rapport signal sur bruit soit de 12.5 dB au lieu de 2.25dB, on dit dans ce cas que le gain du codage est de 10.25 dB (12.5 – 2.25).

Afin de pouvoir comparer les performances de correction d'erreur des codages canal ayant des valeurs de rendement R distincts et utilisant des chaines de transmission différentes (valeur de bande passante et type de modulation différentes), on utilise le rapport E_b/N_0 (en dB) au lieu du SNR il s'agit d'une valeur normalisée du rapport signal sur bruit. E_b correspond à l'énergie moyenne par bit, son unité est le Watt.s, et N_0 correspond à la densité spectrale du bruit, c'est la puissance de bruit pour 1 Hz de bande passante, son unité est Watt/Hz.

2 Les Codes LDPC

Avec des blocs de messages très longs de plusieurs milliers de bits, le nombre des nœuds de contrôle de parité qui se chevauchent augmente et leur taille augmente aussi, et par conséquence, le système d'équations de parité (1.14) devient plus complexe à résoudre et demande des opérations de décodage plus complexes ce qui ralentisse le processus de communication.

Pour pallier à ce problème, Robert Gallager [11] a proposé une nouvelle approche du problème du codage qui se base sur deux conditions clés :

- La première consiste à utiliser des nœuds de contrôle de parité de petite taille, et ce, en réduisant le nombre des bits connectés à chaque nœud de contrôle de parité, ce qui permet de réduire le nombre des variables de chaque équation de parité du système linéaire (1.14) en gardant le même nombre d'équation. Pour quantifier cette première condition, Robert Gallager a défini une grandeur appelé degré des nœuds de contrôle de parité, noté d_c, la valeur de cette grandeur correspond au nombre des éléments non nul par ligne de la matrice de parité H.
- La deuxième consiste à garder un niveau de chevauchement suffisamment grand entre les nœuds de contrôle de parité pour bien protéger chaque bit, et ce, en assurant la connectivité de chaque bit à plusieurs nœuds de contrôle de parité y compris les bits de parités eux-mêmes, puisque tout bit connecté à un seul nœud de contrôle de parité est vulnérable. Pour quantifier cette deuxième condition, Robert Gallager a défini une grandeur appelé degré de bit, noté d_v, la valeur de cette grandeur correspond au nombre des éléments non nul par colonne de la matrice de parité H.

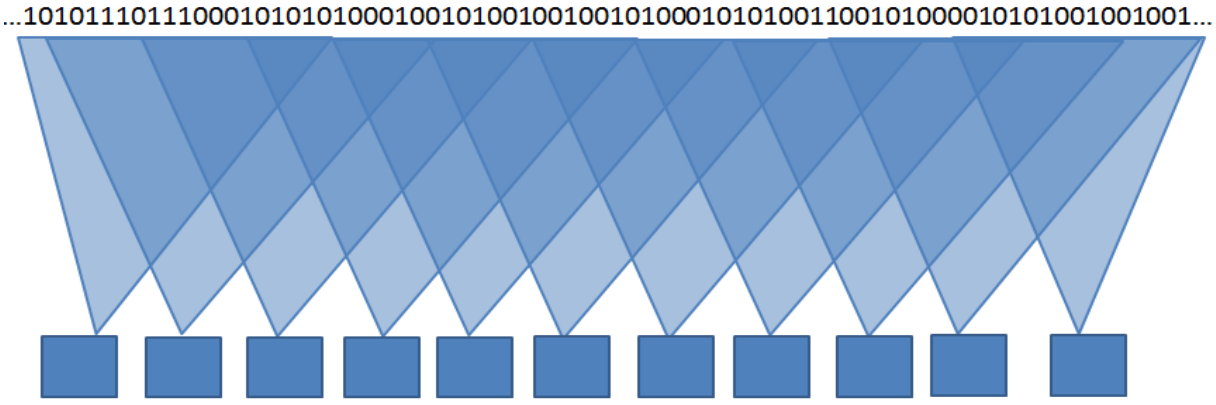


Figure 1. 6 : Graphe d'une matrice H de grande taille avec des ensembles de contrôle de parité qui se chevauchent

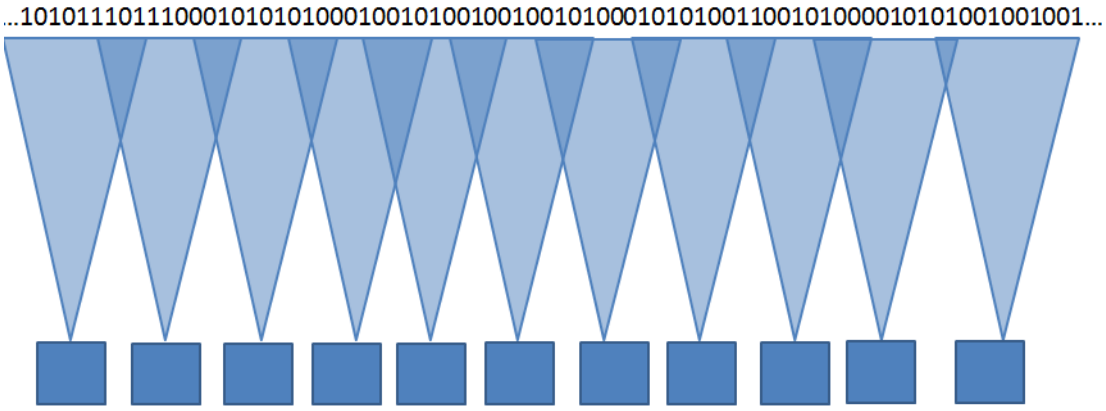


Figure 1. 7 : Graphe d'une matrice H de grande taille avec l'approche LDPC

Les figures 1.6 et 1.7, illustrent la représentation sous forme de graphe de Tanner d'une matrice H de grande taille avec et sans les deux conditions de Gallager. Les deux représentations ont le même nombre de bits et des nœuds de contrôle de parité, cependant, dans la figure 1.6 le niveau de chevauchement entre les nœuds de contrôle de parité est élevé ce qui augmente le nombre de bits participant à chaque nœud de contrôle et par conséquence la complexité de décodage. Dans la figure 1.7, le niveau de chevauchement est moins élevé ce qui permet d'avoir des nœuds de contrôle de petite taille en fixant les poids par ligne d_c et par colonne d_v à des petites valeurs.

Si toutes les colonnes ont un même poids de variable d_v, et toutes les lignes ont un même poids de contrôle d_c, le code LDPC est dit régulier. Dans le cas contraire, si les degrés des nœuds de variables et/ou des nœuds de contrôle ne sont pas constants, le code LDPC est dit irrégulier. Les codes irréguliers sont plus performants en termes de BER que les codes réguliers [15].

Le nombre de 1, noté E, dans la matrice de contrôle de parité est relié aux poids d_c et d_v comme suit :

$$E = \sum_{i=1}^{N-K} d_c(i) = \sum_{j=1}^{N} d_v(j) \tag{1.21}$$

Où $d_c(i)$, $d_v(j)$ représentent respectivement le poids d_c de la ligne i et le poids d_v de la colonne j.

Dans le cas d'un code LDPC régulier l'équation 1.21 devient :

$$E = d_c \times (N - K) = d_v \times N \tag{1.22}$$

Dans ce cas, les deux valeurs d_c et d_v sont reliés au rendement par la relation ci-dessous :

$$R = \frac{K}{N} = 1 - \frac{d_v}{d_c} \tag{1.23}$$

Par conséquence, avec des petites valeurs des poids d_c et/ou d_v, le nombre de 1 dans la matrice sera réduit par rapport au nombre de 0 et donc la densité de 1 sera plus faible, d'où le nom de code de contrôle de parité à faible densité, en anglais, Low density Parity Check (LDPC).

2.1 Construction des codes LDPC

Construire un code LDPC revient à construire une matrice de contrôle de parité H. Ainsi, pour une taille et un rendement donné, construire une matrice de contrôle de parité H revient à déterminer les poids des lignes et des colonnes ainsi que la position de tous les éléments non nuls dans la matrice de contrôle de parité. Le positionnement des éléments non nuls peut se faire d'une manière aléatoire ou structurée. Lors de la construction des matrices H des codes LDPC, les éléments ci-dessous sont pris en compte :

- Maximiser la distance minimale de Hamming du code, car cela conduit à une meilleure performance de correction d'erreur.
- Accélérer l'algorithme de décodage en minimisant les cycles courts dans le graphe de Tanner.
- Réduire les contraintes de conception telles que la taille de la mémoire disponible pour le stockage des matrices de parité, en structurant le positionnement des éléments non nul dans la matrice.

L'obtention de telle matrice peut être réalisé en appliquant des réarrangements sur une matrice obtenue aléatoirement, en utilisant la méthode d'élimination de Gauss-Jordan qui consiste à effectuer un ensemble d'opérations telles que : la permutation des lignes, le remplacement d'une ligne par le résultat de l'addition modulo 2 de deux ou plusieurs lignes, et ce, dans le but de la transformer en une matrice aléatoire à une matrice structurée.

2.1.1 Influence des cycles courts sur le procédé de décodage

Un cycle existe dans un graphe lorsqu'il y a un chemin pour quitter et revenir à un nœud sans passer par les mêmes branches. L'emplacement des éléments non nul dans la matrice de contrôle de parité H, peut engendrer des cycles qui peuvent ralentir ou bloquer le processus de décodage de converger vers un mot code valide. En adoptant la représentation par graphe d'une matrice H. Le nombre de branches traversées détermine la longueur du cycle, nommé ordre du cycle. Un graphe sans cycle est appelé un arbre. La figure 1.8 représente des cycles d'ordres différents. La figure (a) illustre un cycle d'ordre 4, la figure (b) un cycle d'ordre 6 et la figure (c) un cycle d'ordre 8.

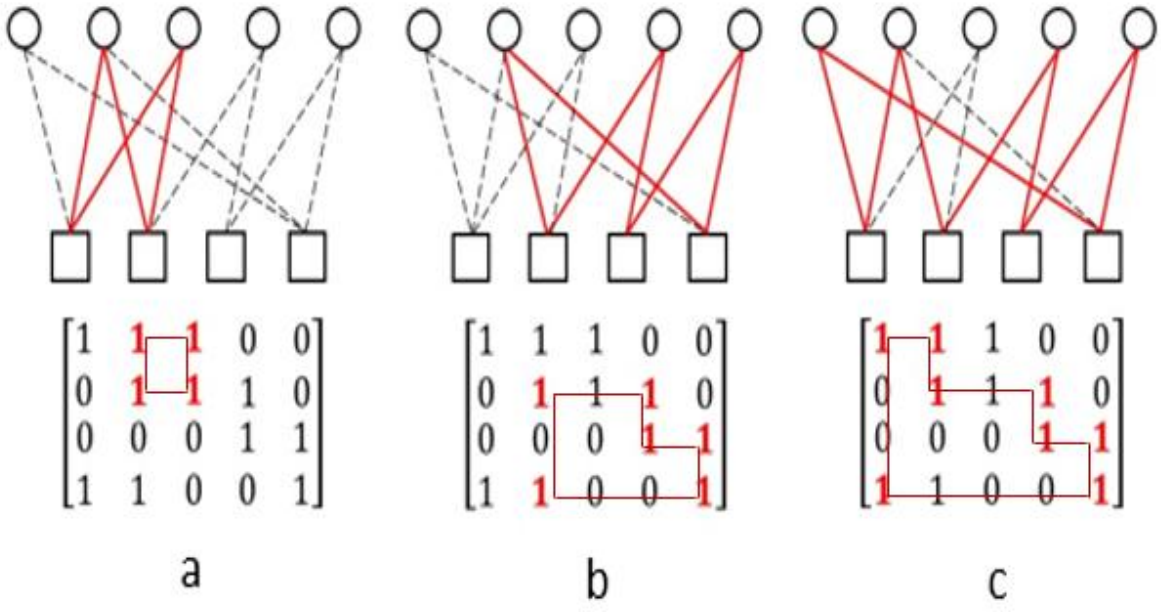


Figure 1. 8 : Cycles dans la matrice H. (a) d'ordre 4, (b) d'ordre 6 et (c) d'ordre 8

L'existence des cycles d'ordre inférieur par exemple d'ordre 4 dans un code LDPC crée une même dépendance entre deux bits du mot code au niveau de deux équations de parités différentes, ainsi même si les valeurs des deux bits changent, les deux équations de parités auxquelles ils participent restent satisfaites, ce qui empêche d'estimer le mot code reçu. Ce problème engendre une dégradation de performances en termes de BER [16].

2.1.2 Codes Quasi-Cycliques LDPC

Contrairement à la construction aléatoire, la construction structurée de la matrice H vise à réduire la complexité de l'encodage et du décodage, en choisissant des emplacements des éléments non nuls bien structurés.

Les codes LDPC quasi cycliques (QC-LDPC) sont une classe de codes structurés qui ont une bonne performance de correction d'erreur et permettent une implémentation matérielle efficace [17] [18]. La matrice H structurée du code QC-LDPC est composée de sous-matrices de dimension Z x Z, où Z est appelé facteur d'expansion et les sous-matrices Z x Z sont obtenues par la permutation circulaire à droite des colonnes de la matrice d'identité de dimension Z x Z.

Les nombres de permutation à réaliser sur chaque sous matrice d'identité de dimension Z x Z, sont appelés les coefficients de permutation, ces coefficients sont rassemblés dans des matrices d'expansion, notée H_{bm} de taille m_b x n_b, où $n_b = N / Z$ et $m_b = (N – K) / Z$.

La matrice H est construite à partir de la matrice d'expansion H_{bm}, en remplaçant chaque coefficients de permutation, notés $\delta_{i,j}$, avec i et j représentent respectivement les indexes des lignes et des colonnes de la matrice d'expansion H_{bm}, comme suit :

- Chaque coefficient de permutation $\delta_{i,j}$ de valeur strictement positive est remplacé par une matrice d'identité de taille Z x Z avec des colonnes permutées à droite $\delta_{i,j}$ fois.
- Chaque coefficient de permutation $\delta_{i,j}$ négative est remplacé par la matrice zéro de taille Z x Z.
- Chaque coefficient de permutation $\delta_{i,j}$ nul est remplacé par la matrice d'identité de taille Z x Z

La figure 1.9, représente la matrice d'expansion 12x24 qui permet la construction de la matrice de taille 2304x1152 de la figure 1.10, utilisé par la norme WIMAX (ieee 802.16) [19], le facteur d'expansion est fixé à 96 les sous matrices sont donc de taille 96×96. Chaque coefficient de permutation strictement positif est remplacé par une matrice d'identité 96×96 permutés circulairement à droite, chaque coefficient de permutation négatif par la matrice zéro de taille 96×96 et chaque coefficient de permutation nul par la matrice identité 96×96.

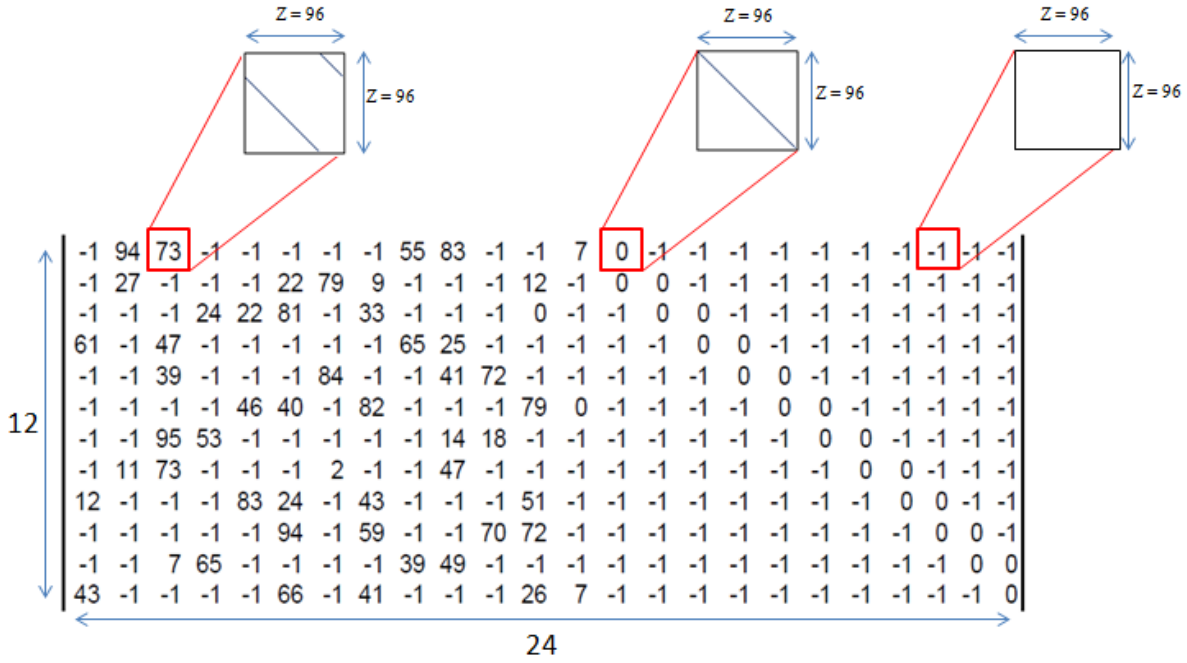


Figure 1. 9: Exemple de matrice d'expansion de dimension 12x96 de la norme Wimax

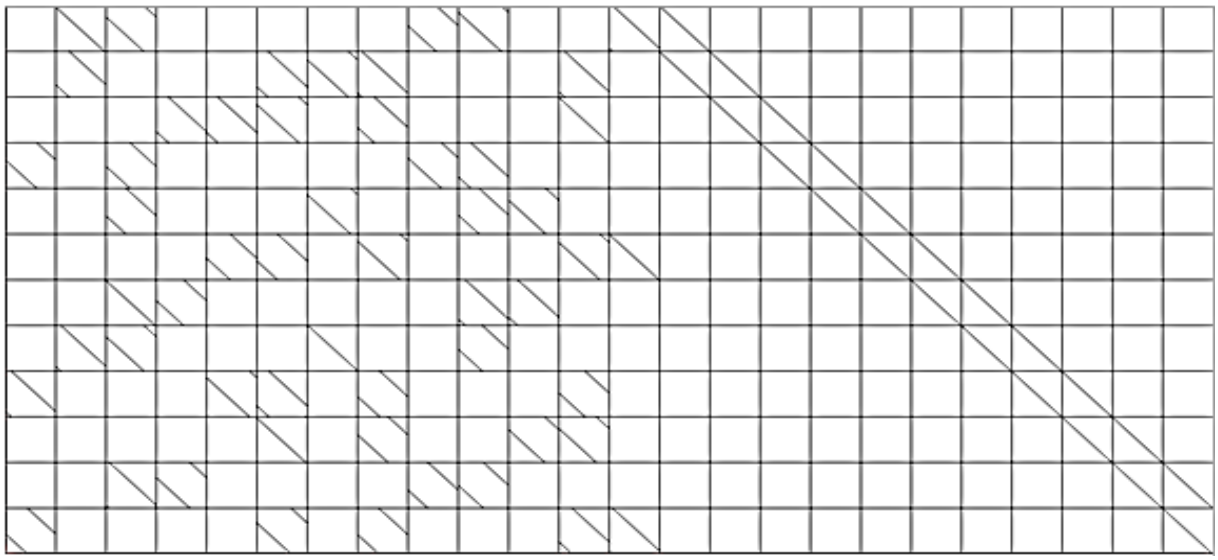


Figure 1. 10: Matrice 2304 x 1152 de la norme Wimax

2.2 Encodage des codes LDPC

De manière semblable aux autres codes correcteurs d'erreurs, l'encodage des codes LDPC consiste à générer un mot code *C* de taille N à partir d'un mot d'information *U* de taille K et d'un matrice génératrice $G_{K,N}$ en utilisant l'équation : $C = U \circ G$. Or les codes LDPC sont définit par leurs matrices de contrôle de parité H, Avant l'opération de codage, il faut

déterminer la matrice G à partir de la matrice H. Dans le cas où la matrice H est systématique, il suffit d'appliquer l'équation (1.12) pour trouver la matrice G.

Cependant pour les codes LDPC, la matrice H n'est jamais systématique car c'est contradictoire avec la deuxième condition sur laquelle se basent la construction des codes LDPC (chaque bit du mot code doit participer à au moins deux ensembles de contrôle de parité).

Richardson et al. [20], a proposé une technique efficace qui permet de réduire la complexité d'encodage. Cette technique est basée sur l'utilisation de matrice de contrôle de parité H sous la forme :

$$H_{N-K,N} = [A_{N-K,K} \quad B_{N-K,N-K}] \tag{1.24}$$

Où A est une sous matrice de taille (N-K x K) et B est sous une matrice carrée triangulaire inférieur de taille (N-K x N-K).

C'est le cas par exemple des matrices quasi-cyclique utilisées dans plusieurs standards tels que la 5ème génération (5G) du haut débit mobile [21], la deuxième génération de diffusion vidéo numérique par satellite (DVB-S2) [22] et les réseaux sans fil Wimax ieee 802.16 [19]. Dans ce cas le calcul de la matrice génératrice G dans sa forme systématique est donné par l'équation suivante :

$$G = [I_{K,K} \quad A^t_{K,N-K}\, o\, (B^t_{N-K,N-K})^{-1}] \tag{1.25}$$

Cette relation montre que la première contrainte sur le code est l'existence de la matrice inverse $(B^t_{N-K,N-K})^{-1}$, pour une matrice carrée triangulaire inférieur cette condition est vérifiée.

Pour une matrice de contrôle de parité H qui n'a pas cette forme, un réarrangement de la matrice de contrôle est effectué en utilisant la méthode d'élimination de Gauss-Jordan.

Les matrices utilisées dans le cadre de cette thèse sont des matrices quasi-cycliques afin de tirer profit de la flexibilité de ces matrices pour le processus de décodage et sont en même temps organisées en deux sous-matrices dont une est carrée bi-diagonale ce qui permet de réduire la complexité du codage.

3 Décodage des codes LDPC

Le décodage est le moyen qui permet de reconstruire le mot code C transmis à partir du message Y reçu en sortie du canal de transmission. La matrice de contrôle de parité H est utilisée par l'opération de décodage lors de l'estimation du message orignal et pour le calcul du syndrome $S = \hat{X} \circ H^t$ ($\hat{X}$ étant le message binaire estimé par le décodeur à partir du message reçu Y), Si le syndrome S est nul alors $\hat{X}$ est un mot code valide, sinon le message estimé $\hat{X}$ contient encore des erreurs de transmission.

Le décodage LDPC se fait à travers des algorithmes itératifs, ce qui permet en cas de présence des erreurs de transmission après une itération de ré-estimer le message original à partir du message reçu et de l'estimation de l'itération précédente et ainsi de suite jusqu'à obtention d'un mot code valide ou épuisement du nombre d'itération maximal défini au préalable. Les algorithmes de décodage LDPC sont divisés en deux familles :

Le décodage à entrées dures : le processus de décodage reçoit l'information démodulée en entrée, appelé information dure, L'estimation est réalisée directement par le démodulateur.

Le décodage à entrées souples : le processus de décodage se fait directement sur les valeurs réelles des données reçues avant la démodulation, ces valeurs sont quantifiées sur plusieurs bits. Ces données contiennent en plus de l'information indiquant la valeur du bit, un degré de fiabilité sur cette valeur de bit.

De point de vue performance en termes de correction des erreurs, les algorithmes de décodage à décision souple peuvent aboutir à des gains de décodage meilleurs que ceux obtenus avec les algorithmes de décodage à décision dure [23], en contrepartie d'une complexité calculatoire supérieure.

3.1 Décodage à décision dure

Les algorithmes de décodage à décision dure, simplifient la méthode de décodage en prenant en entrée la décision dure, noté x_n, prise par le démodulateur sur les valeurs du message reçu du canal de transmission (y_n).

Les algorithmes de décodage à décision dure sont basés sur la méthode de basculement de bit (bit flipping), qui utilise seulement les valeurs des syndromes et une fonction d'inversion pour repérer les bits erronés à inverser à la fin de chaque itération.

3.1.1 Algorithme Bit Flipping

L'algorithme Bit Flipping (BF) utilise l'estimation dure réalisé par le démodulateur, et repose sur le fait que le bit le moins fiable, à inverser, est celui qui participe le plus aux syndromes non nul.

Ainsi, le syndrome S_i relatif à chaque ligne *i* de la matrice H est calculé en utilisant l'équation du syndrome (1.14).

Après le calcul des syndromes par ligne, l'algorithme calcule la somme des syndromes $\sum S_i = S$, si cette somme est nulle, le décodage s'arrête, sinon il calcul la valeur de la fonction d'inversion pour chaque bit afin d'identifier les bits à inverser, cette fonction correspond au nombre de syndrome non nul auquel participe chaque bit du message reçu, elle estime par conséquence la fiabilité des bits, le bit le moins fiable est celui qui participe le plus aux syndromes non nul (correspond au maximum de la fonction d'inversion), ce bit est inversé pour refaire le décodage.

3.1.2 Algorithme Bit Flipping amélioré et dérivés

L'algorithme BF est le plus simple algorithme à implémenter parmi tous les algorithmes de décodage des codes LDPC, mais ses performances en termes de correction d'erreurs restent très réduites et suscite plusieurs itérations du fait qu'un seul bit est inversé à chaque itération. Pour avoir un décodage plus rapide, des algorithmes améliorés ont proposés d'inverser plusieurs bits dans la même itération, et ce, dès qu'un bit participe à un nombre de syndrome non nul définit au préalable, ce nombre est appelé seuil de la fonction d'inversion. Aussi, des bits inversés peuvent oscillés entre les valeurs 0 et 1 d'une itération à l'autre, et ce, en raison de présence de cycle court dans la matrice H.

Pour réduire l'impact des cycles courts et augmenter le taux de détection des bits erronés, d'autres algorithmes basés sur le principe de l'algorithme BF ont été largement étudiés dans la littérature et de nombreuses variantes ont été proposées. Parmi celles-ci, l'algorithme BF pondéré (**WBF**) [24], l'algorithme BF pondéré modifié (**MWBF**) [25], le Gradient Descent BF (**GDBF**) [26] et le Reliability Ratio Weighted GDBF (**RRWGDBF**) [27]. Ces travaux utilisent une pondération additive ou multiplicative dans la fonction d'inversion, comme métrique de fiabilité qui permet d'évaluer la fiabilité des syndromes et des bits reçus. Certes ces algorithmes dérivés permettent d'améliorer les performances en termes de correction d'erreur, mais au détriment de la complexité de décodage qui s'approche

de celle des algorithmes à décision souple sans autant atteindre les performances offertes par ces derniers.

3.2 Décodage à décision souple par propagation de croyances

3.2.1 L'algorithme LLR-BP :

L'algorithme de décodage itératif à décision souple est connu sous le nom propagation de croyance (belief propagation) BP, inventé *par* Judea Pearl [28] en 1982. Comme illustré sur la figure 1.11, cet algorithme peut être vu comme un processus, qui, via un ensemble de branches, échange des messages entre les différents types de nœuds du graphe de Tanner.

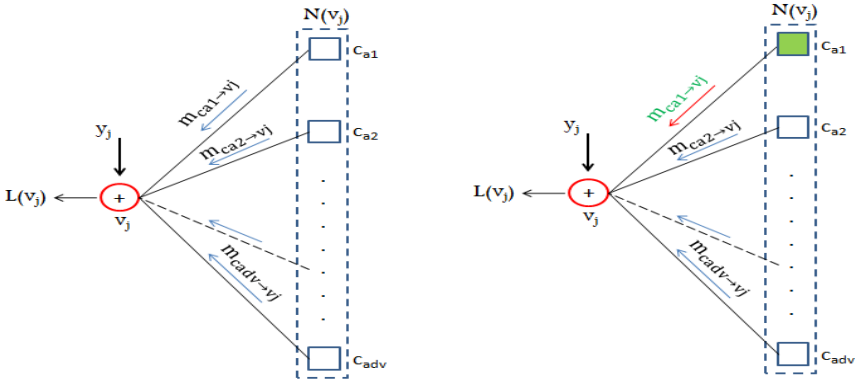


Figure 1. 11: Calcul des messages extrinsèques et des informations a posteriori

Les messages échangés entre les différents nœuds du graphe de Tanner ne sont que des rapports de probabilités appliquées sur chaque bit du message reçu, appelé rapport de vraisemblance LLR (Log-Likelihood-Ratio). Ce rapport permet d'évaluer le rapport des probabilités d'une variable binaire aléatoire U d'avoir la valeur 0 ou 1, celle-ci est définie par :

$$L(u) = \ln\left(\frac{P(u=0)}{P(u=1)}\right) \tag{1.26}$$

Où $P(u=0)$ et $P(u=1)$ dénotent respectivement les probabilités d'avoir 0 et 1 comme valeur de la variable aléatoire u.

Ainsi défini, le signe de la valeur du LLR représente la valeur binaire la plus probable de la variable aléatoire U, et son module représente le degré de vraisemblance (de fiabilité) de cette valeur.

- Si $L(u) > 0$ c.-à-d. $(u = 0) > P(u = 1)$, alors la valeur la plus probable de U est 0.
- Et si, $L(u) < 0$ c.-à-d. $(u = 0) < P(u = 1)$, alors la valeur la plus probable de U est 1.

Nous résumons ci-dessous les principales étapes d'un décodeur LDPC basé sur la propagation de croyances LLR-BP [29] :

Etape 0 (Initialisation) : Les messages LLR initiaux de chaque nœud de variable v_j, notés $L(0)_{v_j}$, sont calculés en utilisant l'information reçue du canal y_j, ils représentent les LLR des valeurs de chaque bit envoyé par l'émetteur v_j (1 ou 0) sachant les valeurs réelles y_j reçue depuis le canal. Les messages V2C passant des nœuds de variable v_j vers les nœuds de contrôle c_i, noté $m_{vj\to ci}$, sont initialisés en conséquence par les $L(0)_{v_j}$ comme suit :

$$m(0)_{vj\to ci} = L(0)_{v_j} = \ln\left(\frac{P(v_j = 0/y_j)}{P(v_j = 1/y_j)}\right) \tag{1.27}$$

Après l'initialisation, chaque itération consiste en deux phase, une phase horizontale (parcours par ligne de la matrice H) dans laquelle les nœuds de contrôle sont mises à jour, et une phase verticale (parcours par colonne de la matrice H) pour le calcul de la mise à jour des nœuds de variable.

Etape 1 (Phase horizontale : Mise à jour des nœuds de contrôle**)** : durant cette phase, l'algorithme met à jour les messages C2V passant des nœuds de contrôle c_i vers les nœuds de variable v_j, noté $m_{ci\to vj}$, il représente le LLR pour que l'équation de contrôle de parité associée au nœud c_i soit satisfaite en tenant compte des messages V2C arrivant des nœuds de variable qui lui sont adjacents. L'équation de mise à jour des messages C2V est donnée par :

$$m_{ci\to vj} = \prod_{vb\in N(ci)\setminus vj} sign(m_{vb\to ci})\, \emptyset\left(\sum_{vb\in N(ci)\setminus vj} \emptyset(|m_{vb\to ci}|)\right) \tag{1.28}$$

Où $\emptyset(x) = -\ln\left(\tanh\frac{x}{2}\right)$, sign( ) est la fonction qui donne le signe d'une valeur réel, et $N(ci)\setminus vj$, désigne l'ensemble des nœuds de variable connectés au nœud de contrôle c_i

en excluant le nœud de variable v_j. La démonstration de cette équation de mise à jour à partir de la définition de la valeur LLR est présentée en annexe B.

Etape 2 (Phase verticale : Mise à jour des nœuds de variable**)** : durant cette phase, l'algorithme mis à jour les messages V2C passant des nœuds de variable v_j vers les nœuds de contrôle c_i, noté $m_{vj\to ci}$, il représente le LLR pour que le bit associé au nœud v_j soit un 0 ou un 1, compte tenu des messages provenant du canal de transmission et les C2V provenant des nœuds de contrôle qui lui sont adjacents. L'équation de mise à jour des messages V2C est donnée par :

$$m_{\mathrm{v}j\to ci} = L(0)_{\mathrm{v}_j} + \sum_{ca \in N(\mathrm{v}j)\backslash ci} m_{ca\to \mathrm{v}j} \tag{1.29}$$

Où $N(vj)\backslash ci$, désigne l'ensemble des nœuds de contrôle connectés au nœud de variable v_j en excluant le nœud de contrôle c_i.

Durant cette même phase, les messages LLR des bits d'information reçus, noté $L(\mathrm{v}_j)$, sont mis à jour en prenant en compte leur valeur initiale $L(0)_{\mathrm{v}_j}$ et les valeurs provenant des nœuds de contrôle qui lui sont adjacents. L'équation de mise à jour des messages V2C est donnée par :

$$L(\mathrm{v}_j) = L(0)_{\mathrm{v}_j} + \sum_{ca \in N(\mathrm{v}j)} m_{ca\to \mathrm{v}j} \tag{1.30}$$

Où $N(vj)$, désigne l'ensemble des nœuds de contrôle connectés au nœud de variable v_j.

Etape 3 (Décision ferme) : Après la mise à jour des LLR de chaque bit d'information reçu, une décision, noté $\hat{\mathrm{X}}$, correspondant au message transmis, est prise selon le signe de $L(\mathrm{v}_j)$ de l'itération courante. $\hat{X} = [\hat{x}_n]$ tel que $\hat{x}_n = 0$ si $L(\mathrm{v}_j) \geq 0$ et $\hat{x}_n = 1$ si $L(\mathrm{v}_j) < 0$

Etape 4 (Vérification du syndrome) : on calcule le syndrome $S = \hat{\mathrm{X}}\ o\ \mathrm{H}^t$ si S est nul le message $\hat{\mathrm{X}}$ est un mot de code valide, sinon, la procédure de la mise à jour continue jusqu'avoir un syndrome nul ou atteindre le nombre d'itération maximal préalablement défini.

3.2.2 LLR pour un canal de transmission AWGN

D'après l'algorithme de décodage basé sur le LLR BP, on remarque que le calcul de toutes les valeurs de mise à jour des messages C2V et V2C de l'algorithme dépend de la valeur LLR initiale de l'équation (1.27). Cette valeur traduit la relation entre les valeurs à l'entrée du canal et les valeurs à sa sortie, par conséquence le calcul de cette valeur initiale dépendra du message reçu, canal utilisé, de la modulation adoptée et de la densité de bruit appliqué au message transmis.

Afin de décrire la perturbation due aux bruits appliqués au message X passant par le canal de transmission, différents modèles peuvent être utilisés. Cependant, dans la littérature, le modèle retenu est souvent le canal à bruit blanc gaussien additif (AWGN). En fait, d'après le théorème de la limite centrale [30], il est montré que la loi de probabilité d'un évènement qui représente la somme d'un grand nombre de variables aléatoires indépendantes est une loi normale. Par conséquent, le choix de la loi normale permet en particulier de modéliser la somme des différents phénomènes de bruits qui s'appliquent à un canal de transmission réel.

Dans les prochains chapitres du manuscrit, le canal AWGN sera principalement associé à une modulation par déplacement de phase binaire (BPSK). Avec cette modulation, chaque valeur binaire $x_i \in \{0, 1\}$ du mot code $X = [\, x_i \,]$ est transformée en vecteur de symbole S de valeur réelle $s_i \in \{1, -1\}$ par la relation $s_i = 1 - 2x_i$, c'est la séquence $S = [\, s_i \,]$ qui représente le signal modulé qui sera transmise via le canal. Ainsi la séquence reçue à la sortie du canal est $Y = W + S$, avec $W = [\, w_i \,]$ une variable indépendante et identiquement distribuée selon une loi normale (ou gaussienne) centrée sur zéro et de variance $\sigma^2 = {N_0}/{2}$, où N_0 est la densité spectrale de bruit.

Ainsi, Les messages LLR initiaux de chaque nœud de variable v_j sont donnés par l'équation suivante :

$$m(0)_{vi \to cj} = L(0)_{v_j} = LLR(y_i) = \frac{2y_i}{\sigma^2} \tag{1.31}$$

L'algorithme LLR-BP est reconnu pour ses performances en termes de correction d'erreurs. Mais, la complexité de calcul reliée à l'équation non linéaire (1.28) de mise à jour des messages C2V est très élevée, ce qui augmente la latence de décodage. Plusieurs versions modifiées de l'algorithme de décodage LLR-BP ont été développées et plus particulièrement

des simplifications de l'équation non linéaire de mise à jours des nœuds de contrôle. En contrepartie, ces simplifications engendrent des dégradations des performances en termes de correction des erreurs, une simplification est qualifiée de bonne si elle assure un bon compromis complexité de décodage versus performances en termes de correction des erreurs.

3.3 Revue de la littérature des différents algorithmes d'approximation de l'algorithme LLR-BP

Dans cette section nous présentons quelques principales approches pour l'approximation de l'algorithme LLR-BP.

3.3.1 Algorithme Min Sum

La complexité de calcul du décodeur sera fortement ou partiellement réduite si le calcul de l'équation de mise à jour des nœuds de contrôle (1.28) est simplifié, autrement dit si la fonction $\emptyset(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$ peut-être simplifiée, La représentation graphique de cette fonction est donnée à la figure 1.12.

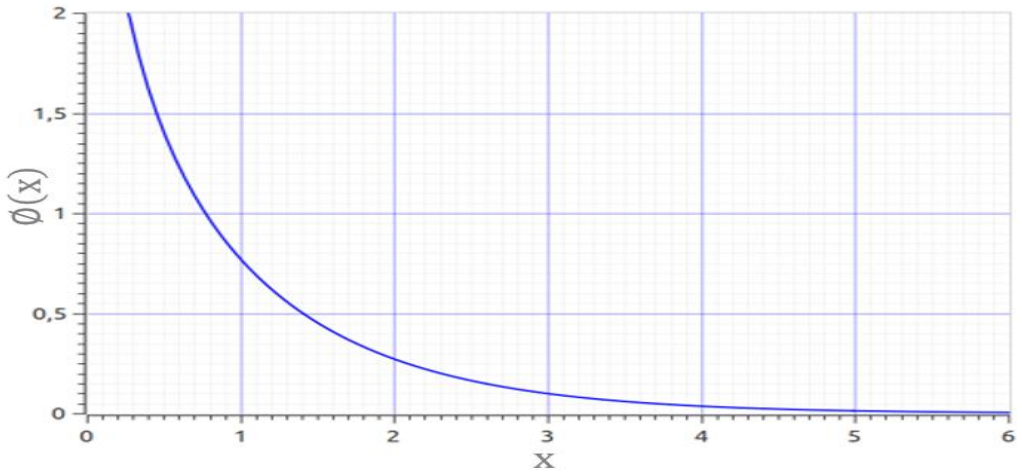


Figure 1. 12: Représentation graphique de la fonction involutive ∅(x)

Puisque $\emptyset(x)$ est une fonction positive alors :

$$\sum_{vb \in N(ci) \setminus vj} \emptyset(|m_{vb \to ci}|) > \emptyset\left(\min_{vb \in N(ci) \setminus vj}(|m_{vb \to ci}|)\right)$$

La fonction ∅(x) est monotone décroissante, donc :

$$\emptyset\left(\sum_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}\emptyset(|m_{\mathrm{v}b\to ci}|)\right) < \emptyset\left(\emptyset\left(\min_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}(|m_{\mathrm{v}b\to ci}|)\right)\right)$$

Et puisque la fonction est involutive c.-à-d. $\emptyset^{-1}(x) = \emptyset(x)$, on a donc :

$$\emptyset\left(\sum_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}\emptyset(|m_{\mathrm{v}b\to ci}|)\right) < \min_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}(|m_{\mathrm{v}b\to ci}|)$$

L'approximation Min-Sum consiste à surestimer le module du message *C2V* en acceptant que :

$$\emptyset\left(\sum_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}\emptyset(|m_{\mathrm{v}b\to ci}|)\right) \approx \min_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}(|m_{\mathrm{v}b\to ci}|)$$

A la fin, l'équation de mise à jour des nœuds de contrôles, $m_{ci\to \mathrm{v}j}$, devient sous la forme suivante :

$$m_{ci\to \mathrm{v}j} = \prod_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j} sign(m_{\mathrm{v}b\to ci}) \min_{\mathrm{v}b\in N(ci)\setminus \mathrm{v}j}(|m_{\mathrm{v}b\to ci}|) \qquad (1.32)$$

Dans cet algorithme appelé Min-Sum [31], le calcul des messages C2V, $m_{ci\to \mathrm{v}j}$, est réalisé par une simple fonction qui détermine la valeur minimale parmi les modules des messages V2C, $|m_{\mathrm{v}b\to ci}|$, appartenant à l'ensemble $N(ci)\setminus \mathrm{v}j$. En fait on aura deux minimums, le premier minimum $\min_1$ correspond à la valeur minimale sur tout l'ensemble $N(ci)$, et le deuxième minimum correspond à la valeur minimale sur l'ensemble $N(ci)\setminus \mathrm{v}j$, lorsque v_j correspond au premier minimum.

Les autres étapes de l'algorithme de décodage MS sont identiques à celles du décodage LLR BP.

Malgré, les avantages que présente cet algorithme par rapport à algorithme LLR-BP en termes de complexité, il introduit des dégradations importantes des performances de décodage en termes de BER. Ces pertes sont essentiellement dues à la surestimation au niveau du calcul du module $\left|m_{ci\to \mathrm{v}j}\right|$. Ces surestimations deviennent flagrantes surtout, lorsque les valeurs des modules $|m_{\mathrm{v}b\to ci}|$ sont très proches.

Exemple :

La *figure* 1.13, montre deux exemples, le premier exemple (en rouge) où l'approximation est bien vérifiée et la deuxième exemple (en vert) où la surestimation est très importante.

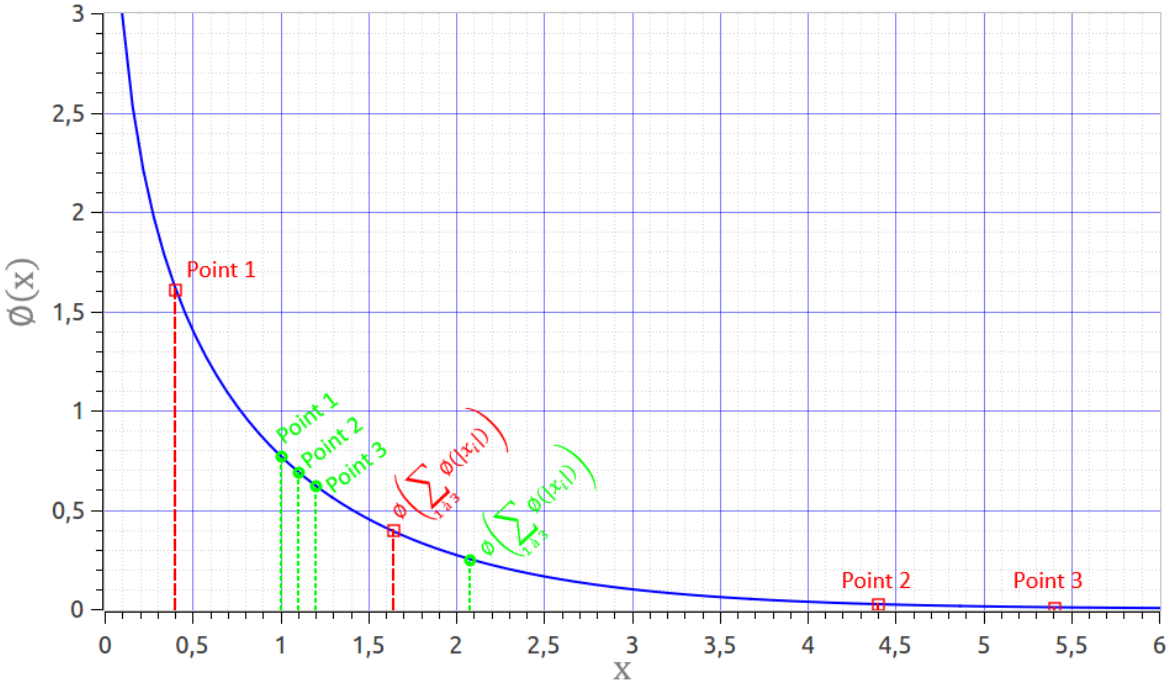


Figure 1. 13: Représentation des zones de surestimation de la fonction $\emptyset(x)$

Pour le premier exemple (en rouge), on a trois points : le point 1 correspond à $x = 0.4$, le point 2 à $x = 4.4$ et le point 3 à $x = 5.4$, la valeur minimale est donc 0.4.

On a $\emptyset(|0.4|) = 1.61$, $\emptyset(|4.4|) = 0.0245$ et $\emptyset(|5.4|) = 0.009$.

Donc : $\emptyset(\sum_{1 \text{ à } 3} \emptyset(|x_i|)) = \emptyset(1.61 + 0.0245 + 0.009) = \emptyset(1.6435) = 0.395$

Dans ce cas, l'estimation par la valeur minimale 0.395 est très proche au calcul exacte 0.4

Pour le deuxième exemple (en vert), on a trois points : le point 1 correspond à : $x = 1$, le point 2 à $x = 1.1$ et le point 3 à $x = 1.2$, la valeur minimal est donc 1 .

On a $\emptyset(|1|) = 0.77$, $\emptyset(|1.1|) = 0.69$ et $\emptyset(|1.2|) = 0.62$.

Donc : $\emptyset(\sum_{1 \text{ à } 3} \emptyset(|x_i|)) = \emptyset(0.77 + 0.69 + 0.62) = \emptyset(2.08) = 0.25$

Dans ce cas, la valeur minimale 1 surestime la valeur calculée de 0.25. On remarque que lorsque les deux valeurs de $min1$ et $min2$ sont espacés entre eux (cas de la 1ère zone $min1 = 0.4$ et $min2 = 4.4$), l'estimation est plus meilleur que lorsque les deux valeurs de $min1$ et $min2$ sont très proches entre eux (cas de la 2ème zone $min1 = 1$ et $min2 = 1.1$).

Afin d'atténuer cette surestimation, plusieurs méthodes de correction ont été proposées dans la littérature [32-38] pour améliorer l'algorithme MSA, mais avec une certaine augmentation de la complexité. Ces algorithmes sont détaillés dans les sections ci-dessous.

3.3.2 Algorithmes Normalized MS et Offset MS

Les algorithmes Normalized Min-Sum (NMS) et Offset Min-Sum (OMS) [32] sont des versions légèrement modifiées de l'algorithme MS qui reposent sur l'utilisation d'un facteur de normalisation ou d'un facteur d'offset pour compenser la surestimation des modules des messages $|m_{ci\to vj}|$. Le principe est de diminuer la différence entre la valeur surestimée et la vraie valeur du module de $|m_{ci\to vj}|$.

Le décodeur NMS compense cette surestimation en introduisant un facteur de normalisation positive $\alpha < 1$, dans l'équation de mise à jour des messages C2V. La bonne valeur du facteur α c'est la valeur qui donne des performances proches à celle de l'algorithme LLR BP en termes de BER, cette valeur peut être évaluée par simulation. L'équation de mise à jour des messages C2V devient :

$$m_{ci\to vj} = \alpha \left(\prod_{vb\in N(ci)\setminus vj} sign(m_{vb\to ci}) \min_{vb\in N(ci)\setminus vj} (|m_{vb\to ci}|) \right) \tag{1.33}$$

Le décodeur OMS introduit un facteur de correction β positif dans l'équation de la mise à jour des messages C2V afin de réduire l'effet de la surestimation. Donc l'équation devient :

$$m_{ci\to vj} = \prod_{vb\in N(ci)\setminus vj} sign(m_{vb\to ci}) \max\left(\left(\min_{vb\in N(ci)\setminus vj} (|m_{vb\to ci}|) - \beta \right), 0 \right) \tag{1.34}$$

Si $\left(\min_{vb\in N(ci)\setminus vj} (|m_{vb\to ci}|) - \beta \right) < 0$, le module du message $|m_{vb\to ci}|$ est mise à zéro car $|m_{vb\to ci}|$ est toujours supérieur ou égal à zéro.

En général, le même facteur de normalisation α (ou le facteur d'offset β) est utilisé durant la procédure de décodage, sa valeur peut être trouvée par simulation.

Les autres étapes des algorithmes de décodage NMS et OMS sont identiques à celles du décodage LLR BP.

Une approche hybride offset et normalized MS, a été proposé par thran-thi et al.[33], ainsi l'équation de mise à jour de l'algorithme proposé est comme suit :

$$m_{ci\to vj} = \alpha \prod_{vb\in N(ci)\setminus vj} sign(m_{vb\to ci}) \max\left(\left(\min_{vb\in N(ci)\setminus vj}(|m_{vb\to ci}|) - \beta\right), 0\right) \quad (1.35)$$

Les performances en termes de correction d'erreur ont été augmentées de 0.1 dB par rapport aux algorithmes NMS et OMS initiaux, avec une légère augmentation de complexité des calculs dus à la multiplication par le facteur de normalisation dans le cas de l'algorithme NMSA et à l'addition du facteur d'offset dans le cas de l'algorithme OMSA [33].

3.3.3 Algorithme Self Adjustable Offset Min-Sum

Pour atténuer davantage l'effet de la surestimation, et se rapprocher des performances en termes de pouvoir de correction d'erreurs de l'algorithme LLR-BP, *Ji et al.[34]*, proposent l'algorithme SAOMS (Self Adjustable Offset Min-Sum) en calculant directement le facteur d'offset β de l'équation *(1.34)* en fonction de la différence entre les deux valeurs $min1$ et $min2$ comme suit [34] :

$$\beta = \gamma f(min1 - min2) \quad (1.36)$$

Avec γ est le facteur de normalisation optimal et f(x) représente le facteur de correction. L'expression de cette fonction est la suivante [31] :

$$f(x) = \ln(1 + e^{-|x|}) \quad (1.37)$$

Ji et al [34] ont montré que cette approche permet de récupérer les performances en termes de BER perdues par les algorithmes à complexités réduites, comme le MS, NMS et OMSA, mais au prix d'une légère augmentation de la complexité de calcul et de l'implémentation matérielle. Ceci est dû à la fonction non linéaire *f(x)*.

Une approximation ou une variante de la fonction *f(x)* a été implémentée soit par la table de correspondance (LUT Look Up Table) [35] ou, soit par des approximations en plusieurs fonctions linéaires ce qui en fait une fonction constituée de plusieurs morceaux PWL (PieceWise linear) sur 8 régions, sur 5 régions ou même sur 2 régions [36].

L'approximation par morceaux de $f(x)$ en 5 fonctions linéaires de type $f_i(x) = a_i x + b_i$ a été proposée dans notre équipe, et permet de limiter la puissance de calcul et d'optimiser les ressources hardware en comparaison avec l'algorithme LLR-BP [36]. Elle est représentée sur la *figure 1.14*, et les coefficients des différentes fonctions f_i sont donnés dans le tableau 1.3.

Tableau 1. 2: Approximation de la fonction non-linéaire $f(x) = ln(1 + e^{-|x|})$ par 5 fonctions linéaires

Intervalle	a_i	b_i	$f(x)$
[0, 0.875]	-0.4018	0.7	-0.4018x + 0.7
[0.875, 1.75]	-0.2151	0.5366	-0.2151x + 0.5366
[1.75, 2.75]	-0.0982	0.3321	-0.0982x + 0.3321
[2.75, 4]	-0.0351	0.1585	-0.0351x + 0.1585
[4, +∞[	0	0	0

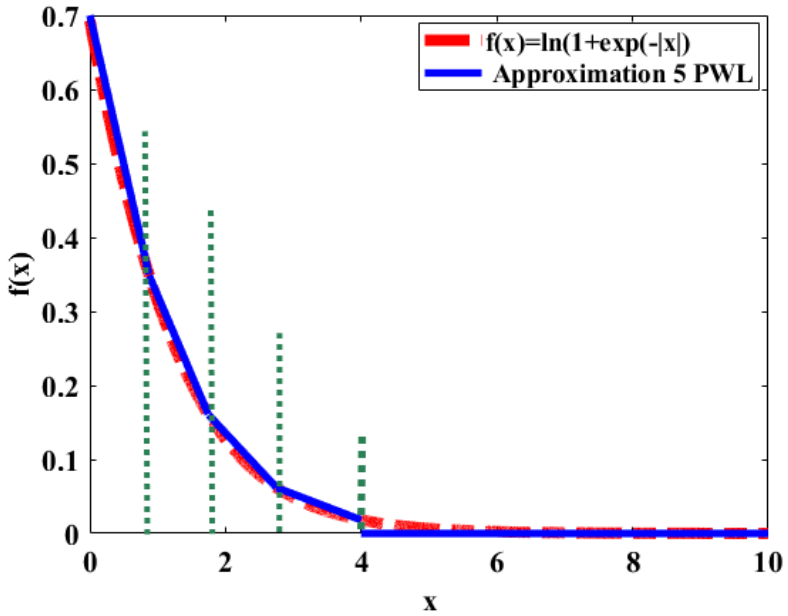


Figure 1. 14: Représentation graphique de la fonction non-linéaire $f(x) = ln(1 + e^{-|x|})$ et son interpolation sur 5 régions.

Donc, l'interpolation sur 5 régions s'avère être plus optimale en termes de BER et de ressources matérielles, les résultats trouvés sont plus proches de l'algorithme SAOMS qui utilise la fonction non linéaire *f(x)[36]*.

3.3.4 Algorithme MS premier min

Pour réduire encore la complexité des algorithmes MS, Darabiha et *al [37]* ont proposé une méthode, à un seul minimum, appelé smMS (second min MS) dans laquelle le deuxième minimum est estimé en appliquant un facteur de correction constant sur le premier minimum. Cette technique permet de réduire la complexité en calculant uniquement le premier minimum.

Ainsi, le signe des nœuds de contrôles, $m_{ci\to vj}$, reste le même que l'équation 1.28, la valeur absolue est calculé en se basant sur le premier minimum $min1$ comme suit :

$$|m_{ci\to vj}| = \begin{cases} min1 + 1 & si\ T(m) = 1\ et\ m_{vj\to ci} = min1 \\ min1 & sinon \end{cases} \quad (1.38)$$

Avec $1 \leq T(m) \leq dc$ est le nombre de fois où l'entrée $|m_{vj\to ci}|$ est égal à 1 .

Cependant, cet algorithme introduit une dégradation de performance non négligeable à cause de l'estimation du second minimum, cela se produit parce que la distance entre le premier minimum et le second minimum n'est pas constante, elle change en fonction des modules des massages V2C.

3.3.5 Algorithme variable weighted MS

Pour améliorer l'estimation du deuxième minimum, Angarita et *al* [38] ont remarqué que la différence entre les deux valeurs $min1$ et $min2$ augmente d'une itération à l'autre, par conséquence, il a proposé d'introduire sur l'algorithme smMS, un facteur de correction ω variable d'une itération à l'autre vwMS (variable weighted MS). Ainsi, le calcul du deuxième minimum est estimé en appliquant un facteur de correction variable sur la valeur de $min1$.

$$|m_{ci\to vj}| = \alpha.\begin{cases} min1 + \omega & si\ T(m) = 1\ et\ m_{vj\to ci} = min1 \\ min1 & sinon \end{cases} \quad (1.39)$$

La valeur du facteur de correction ω prend une valeur de l'intervalle $]0\ 4[$, et change de valeur d'une itération à l'autre avec un pas de 0.25 ce qui permet d'émuler l'algorithme MS où la différence entre le premier et le deuxième minimum augmente avec le nombre d'itérations.

Cette proposition améliore les performances de l'algorithme à un seul minimum, et les résultats trouvés par cette approche sont presque les mêmes que l'algorithme NMS.

Cependant cet algorithme introduit une augmentation de complexité, dû au calcul du facteur de correction à chaque itération, aussi les facteurs de l'itération courante dépendent de

ceux de l'itération précédente, ce qui nécessite plus de mémoires pour stocker les facteurs de correction.

3.4 Etude comparative des complexités de calcul des différentes approximations de l'algorithme LLR BP

Suite analyses et études, issues de la littérature, de la comparaison de la complexité de calcul associée aux mises à jour des messages C2V par les algorithmes de décodage LLR-BP , MSA , NMSA , OMSA et SAOMSA. Les résultats de cette comparaison sont illustrés sur le tableau 1.4 pour un seul nœud de contrôle et une seule itération, les calculs des signes et des valeurs absolues sont communs entre tous les algorithmes, par conséquent ils ne sont pas considérés dans le calcul des complexités.

Tableau 1. 3: Complexité calculatoire pour la mise à jour d'un seul nœud de contrôle

Algorithme	**Comparaisons**	**Additions**	**Multiplication**	$\emptyset(x)$	$f(x)$
LLR BP	0	$2 \times dc$	0	$2 \times dc$	_
MSA	$2 \times (dc-1)$	0	0	_	_
NMSA	$2 \times (dc-1)$	0	2	_	_
OMSA	$2 \times (dc-1)$	2	0	_	_
SAOMSA	$2 \times (dc-1)+1$	1	2	_	1

L'algorithme de décodage LLR-BP basé sur l'approche de Gallager reste le plus complexe, à cause du calcul d'une fonction non linéaire $\emptyset(x)$, présente dans l'équation de la mise à jour des nœuds de contrôle. En revanche, l'algorithme MSA est simple et moins coûteux en termes de complexité de calcul. Il ne nécessite que $2 \times (dc-1)$ comparaisons pour mettre à jour les messages C2V, mais introduit une surestimation et par conséquent, ce sont les performances de décodage qui vont subir des dégradations très significatives.

L'algorithme NMSA permet de réduire la surestimation de l'algorithme MSA en multipliant les valeurs $min1$ et $min2$ par un facteur < 1 , ce qui nécessite l'ajout de deux opérations de multiplication par nœud de contrôle. De même l'algorithme OMSA permet de réduire la surestimation de l'algorithme MSA en soustrayant les valeurs $min1$ et $min2$ par un facteur $\beta > 0$, ce qui nécessite l'ajout de deux opérations d'addition par nœud de contrôle, cet

algorithme présente l'inconvénient de perdre le signe initial d'un message C2V si l'un des deux valeur $min1$ et/ou à $min2$ sont inférieur au facteur β.

L'algorithme de décodage SAOMSA, permet de réduire la surestimation des deux valeurs $min1$ et $min2$ par un facteur de correction ajustable durant le processus de décodage itératif, cet méthode permet d'améliorer les performances, Cependant la complexité augmente puisque le calcul du facteur ajustable nécessite l'ajout d'une soustraction, d'une comparaison et le calcul de la fonction (x) , et ce, pour chaque nœud de contrôle.

La complexité globale par itération des différents algorithmes, dépend du nombre de fois où les nœuds de variable et les nœuds de contrôle sont mises à jours. Or, cela dépendra de l'ordonnancement adopté pour les échanges des messages de mise à jour entre les différents nœuds de variable et de contrôle, ces types d'ordonnancement seront présentés dans le paragraphe suivant, ensuite leurs complexités totales seront évaluées.

3.5 Ordonnancement des algorithmes à propagation de croyance

Lors de l'exécution du processus du décodage, l'ordonnancement définit la stratégie de sélection et l'ordre de mise à jour des nœuds ainsi que la manière de parcours du graphe de tanner. L'ordonnancement a un impact sur le pouvoir de correction, la vitesse de convergence, la complexité de décodage et le degré de parallélisme nécessaires pour le lancement de décodage sur des plateformes multi-cœurs.

Plusieurs stratégies d'ordonnancement à séquencement fixe et dynamique ont été proposées dans la littérature afin d'améliorer les performances du décodage en termes de correction d'erreurs, mais en contrepartie une complexité supplémentaire relative aux critères de choix de l'ordre de mise à jour des nœuds s'ajoute à la complexité des équations de mise à jour des algorithmes de propagation de croyance.

3.5.1 Ordonnancement par inondation

L'ordonnancement par inondation, en anglais : ″flooding scheduling″, est l'ordonnancement traditionnel avec lequel a été proposé l'algorithme de décodage LLR-BP [38], c'est un séquencement parallèle où tous les nœuds de contrôle sont mis à jour (f*igure 1.15 a*) en premier lieu, suivis par la mise à jour de tous les nœuds de variables (*figure 1.15 b*).

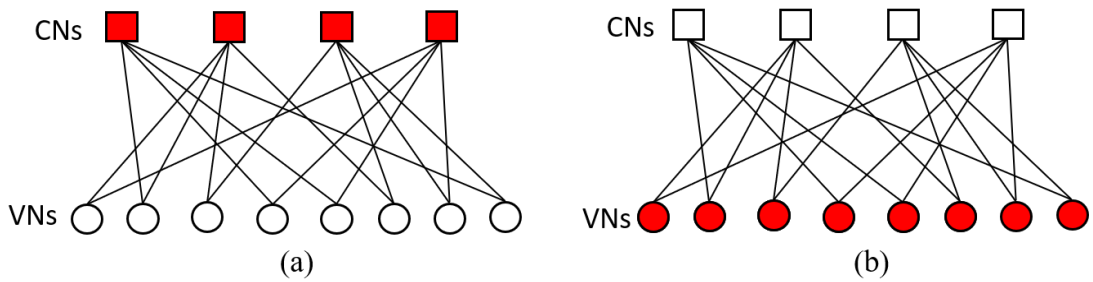


Figure 1. 15: Ordonnancement par inondation

L'ordonnancement par inondation est simple pour l'implémentation matérielle, mais la vitesse de convergence de l'algorithme est réduite, car les dernières informations mises à jour ne seront pas disponibles avant l'itération suivante et par conséquent le nombre d'itérations nécessaire pour décoder une information est élevé, et donc la latence de décodage est élevée.

Pour améliorer les performances en termes de correction des erreurs, d'autres stratégies d'ordonnancement ont été présentées. Les ordonnancements brassés (Shuffled) et dynamiques sont deux variantes de ces stratégies couramment utilisées [39, 43], dans lesquels les nœuds de variables ou de contrôle sont traités en série selon un ordre prédéterminé ou décidé dynamiquement lors du processus de décodage.

3.5.2 Ordonnancement brassé

Le principe de l'ordonnancement brassé consiste à mettre à jour chaque type de nœuds dans un ordre prédéterminé où, chaque nœud sera mis à jour, au cours d'une itération, autant de fois que son degré de connexion. Dès qu'une nouvelle information est disponible elle sera utilisée dans la mise à jour des autres nœuds dans la même itération [39].
Lorsque les nœuds de contrôle sont mis à jour séquentiellement, on parle d'Horizontal shuffle scheduling (HS), et lorsque les nœuds de variables sont mis à jour séquentiellement, on parle de Vertical shuffle scheduling (VS).

La *figure 1.16* illustre les étapes de l'ordonnancement Horizontal Shuffle, on commence par la mise à jour du premier nœud de contrôle, ensuite, tous les nœuds de variable connectés à ce nœud sont mis à jour, puis on passe séquentiellement au nœud de contrôle suivant et ainsi de suite jusqu'au dernier nœud de contrôle.

La *figure 1.17* illustre les étapes de l'ordonnancement vertical Shuffle. Dans ce cas, on commence par la mise à jour du premier nœud de variable, ensuite, tous les nœuds de contrôle connecté à ce nœud sont mis à jour, et on passe séquentiellement au nœud de variable suivant et ainsi de suite jusqu'au dernier nœud jusqu'au dernier nœud de variable.

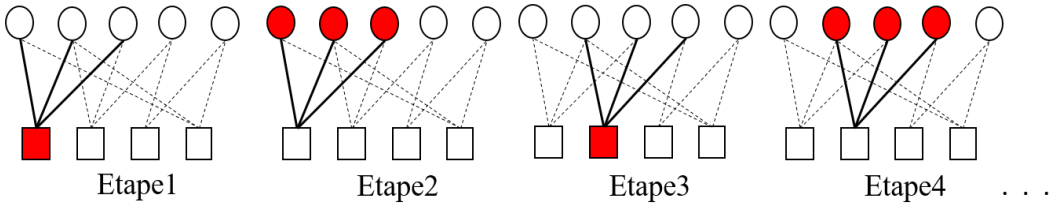


Figure 1. 16: Illustration de l'ordonnancement Horizontal shuffle

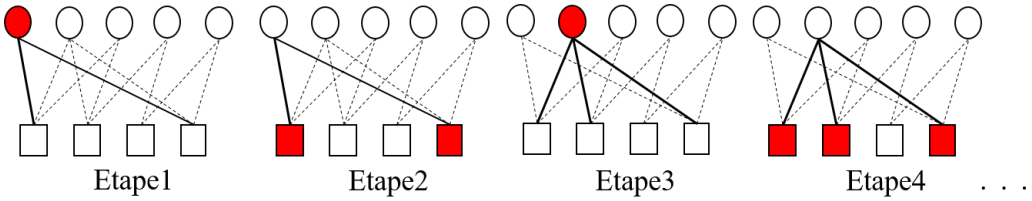


Figure 1. 17: Illustration de l'ordonnancement Vertical shuffle

Ce type d'ordonnancement permet une convergence plus rapide par rapport à l'ordonnancement par inondation, mais avec une augmentation du nombre d'opérations réalisées dans chaque itération.

3.5.3 Ordonnancement dynamique

Afin d'obtenir des performances de convergence plus rapides, des stratégies d'ordonnancement dynamique basées sur des informations calculées lors du processus de décodage ont été introduites. Parmi lesquelles le décodage par propagation de croyance résiduelle (RBP) [40], [41]. La valeur résiduelle est définie comme la différence entre les valeurs de mise à jour anciennes et récentes des messages C2V d'un nœud de contrôle c_i, elle permet ainsi d'évaluer l'apport de cette mise à jour sur l'ensemble des nœuds appartenant à la branche du graphe de racine c_i. Si la valeur résiduelle est importante, cela signifie que la nouvelle valeur de mise à jour aura un impact considérable sur les valeurs des autres nœuds de la même branche, et par conséquence la mise à jour de ces nœuds est priorisée. Dans le cas contraire, si la valeur résiduelle est faible ou nulle, aucune nouvelle information ne sera propagée aux nœuds de la branche y afférente et par conséquent la mise à jour de ces nœuds sera différée.

La valeur résiduelle du message C2V est définie comme suit :

$$r_{ci\to vj} = \left\| m^{pre}_{ci\to vj} - m_{ci\to vj} \right\| \tag{1.40}$$

Où $m^{pre}_{ci\to vj}$ est le pré-calcul du prochain message C2V mis à jour.

Cet algorithme d'ordonnancement ne met à jour que la branche du graphe de Tanner avec la valeur maximale des messages C2V résiduels. Par conséquent, au début du processus de décodage, tous les messages C2V sont calculés afin d'identifier la branche du graphe de Tanner à mettre à jour.

Une fois identifié, le message C2V pré-calculé est propagé aux nœuds variables de la même branche, puis les messages V2C sont mis à jour et propagés aux nœuds de contrôle voisin. Les nœuds de contrôle recevant les nouveaux messages V2C actualisent leurs messages C2V pré-calculés et leur valeur résiduelle. Enfin, la valeur résiduelle du message C2V qui vient d'être mise à jour est remise à 0 et le message C2V ayant le résidu le plus important est sélectionné pour la prochaine mise à jour.

Par rapport aux algorithmes séquentiels et d'inondation, la vitesse de convergence de l'algorithme RBP est plus rapide, mais après les premières itérations la capacité de correction s'affaiblit [41], car les principales ressources de traitement sont affectées et concentrées sur un petit nombre de nœuds et branches présentant la valeur résiduelle maximale.

Pour pallier à cette limitation, plusieurs algorithmes de décodage dynamique, basés sur différentes stratégies de sélection et de mise à jour des messages, ont été rapportés. L'algorithme Variable to Check (VC)-RBP [42] où la priorité des mises à jour des messages est donnée au message V2C ayant le résidu le plus élevé. Le VC-RBP a l'avantage d'avoir un calcul plus faible puisque l'équation de mise à jour des messages V2C est moins complexe que celle utilisée pour la mise à jour des messages C2V, mais la performance de correction d'erreur n'est pas aussi bonne que celle du RBP, car les nœuds de variables avec un résidu V2C minimal ne contribuent jamais au processus de décodage. Ces nœuds sont qualifiés de silencieux.

Ensuite, un algorithme RBP sans nœuds de variables silencieux (SVNF) [43] a été proposé. Dans cet algorithme tous les nœuds de variable participent au processus de décodage en échangeant leurs mises à jours avec le nœud de contrôle avec la valeur résiduelle C2V maximale. Pour ce faire, cette méthode utilise un tableau T de taille N pour marquer le statut de participation de chaque nœud de variable. Chaque fois qu'un nœud variable v_j est sélectionné, le terme correspondant dans le tableau T est mis à 1 ($T[j] = 1$). Ainsi, l'ordonnanceur favorise les nœuds de variable qui n'ont pas encore participé au processus de décodage, et ce, en vérifiant sur le tableau T leur participation à des mises à jour ultérieur. À

la fin de chaque itération, le vecteur T est remis à zéro pour réinitialiser l'état de participation. Le SVNF a l'avantage d'avoir une vitesse de convergence plus rapide que la RBP mais une complexité de calcul plus élevée.

Pour augmenter davantage les performances des algorithmes RBP, certains algorithmes dynamiques basés sur le message résiduel, utilisent d'autres critères de choix des nœuds à mettre à jour en premier [44-45]. L'algorithme Informed Variable-to-Check Residual Belief Propagation (IVC RBP) [44], la priorité de la mise à jour du message est donnée à la variable la plus instable. Un nœud variable est qualifié d'instable si son signe avant et après une mise à jour change, on dit qu'il oscille. Ainsi, les nœuds de variables sont divisés en deux groupes formés par les nœuds stables et ceux instables. Après avoir identifié les nœuds de variables les plus instables, la mise à jour des messages sélectionnés est effectuée sur la base de la procédure RBP. Le calcul de la métrique de stabilité augmente le nombre de calculs et par conséquence la complexité de décodage.

Dans l'algorithme Oscillating Variable Nodes Based Residual Belief Propagation OVC RBP [45], la priorité de la mise à jour est basée sur les nœuds de variable qui oscillent et qui participent à un grand nombre d'équations de contrôle de parité non satisfaites. Cette métrique est définie pour chaque nœud variable comme suit :

$$U(\mathrm{v}j) = \sum_{ci \in N(\mathrm{v}j)} T(c_i) \tag{1.41}$$

Où $T(c_i)$ désigne le résultat du contrôle de parité du nœud ci. Plus cette équation de contrôle n'est pas vérifiée par un nœud de variable donné, plus il est considéré comme moins fiable.

Par conséquent, chaque ensemble des nœuds de variables instables ou stables est divisé en deux autres ensembles selon que si U(v) est égal à sa valeur maximale ou non. Le nœud variable le plus instable avec le plus grand nombre d'équations de contrôle de parité non satisfaites est mis à jour en premier.

Ces nouveaux algorithmes de décodage dynamique, ont montré des performances significatives en termes de correction d'erreur et de convergence, par rapport aux ordonnancements fixes. Cependant, deux inconvénients majeurs subsistent. Premièrement, les nœuds de variables sont sélectionnés en fonction de leur valeur résiduelle et/ou des métriques

utilisées, et par conséquent tous les nœuds de variables n'ont pas la même chance d'être sélectionnés. Deuxièmement, la recherche dynamique des nœuds de variables les plus instables et/ou les moins fiables, et le calcul résiduel requis pour chaque itération du décodage, augmentent la complexité du décodage et rendent le processus de décodage plus séquentiel, ce qui ne permet pas de profiter des puissances de calcul offertes par les plateformes multi-cœurs.

3.6 Etude comparative des Complexités des algorithmes d'ordonnancement

La complexité des algorithmes d'ordonnancement est évaluée dans la littérature, en termes de nombre de fois requis pour la mise à jour des messages C2V et V2C au sein de chaque itération de décodage, où une itération désigne le processus de sélection et de mise à jour de toutes les branches du graphe de Tanner. Le nombre total de branches correspond au nombre d'éléments non nuls de la matrice H, $E = d_c \times M = d_v \times N$, où d_c et d_v désignent respectivement le degré des nœuds de contrôle et des nœuds variables et M et N désignent respectivement le nombre des nœuds de contrôle et le nombre des nœuds de variable.

Le tableau 1-4 présente la complexité de mise à jour des différents algorithmes de d'ordonnancement. Pour l'algorithme d'ordonnancement par inondation, les d_c messages C2V des M nœuds de contrôle sont mis à jour en premier, soit $d_c \times M = E$ mise à jour, ensuite les d_v messages V2C des N nœuds de variable sont mis à jour, soit $d_v \times N = E$ mise à jour.

Pour l'algorithme d'ordonnancement brassé horizontalement (HS), après la mise à jour d'un nœud de contrôle, tous les nœuds de variable y connectés sont mis à jour, ainsi pour chaque d_c messages C2V mis à jour, $d_c \times d_v$ messages V2C sont mis à jour. Par conséquent la mise à jour de l'ensemble des nœuds de contrôle nécessite $d_c \times M = E$ mises à jour des messages C2V et $d_c \times d_v \times M = dv \times E$ mise à jour des messages V2C. De même pour l'algorithme d'ordonnancement brassé verticalement, on montre qu'une seule itération nécessite $d_v \times N = E$ mise à jour des messages V2C et $d_c \times d_v \times N = d_c \times E$ mise à jour des messages C2V.

La complexité des algorithmes d'ordonnancement dynamique est plus importante en raison des calculs des valeurs résiduelles et des métriques nécessaires pour la recherche du nœud de variable qui satisfait les critères de choix de chaque algorithme, pour les deux algorithmes IVRBP et OVRBP la complexité reportée est quadratique $O(M^2)$ et $O(N^2)$ respectivement, malgré que les performances en termes de correction d'erreur soient améliorées.

Tableau 1. 4:Complexité équivalente des différents algorithmes d'ordonnancement

Algorithme d'ordonnancement	Calcul pour la recherche du nœud à mettre à jour	Calcul pour la mise à jour V2C	Calcul pour la mise à jour C2V
Flooding	0	E	E
HS	0	$E.d_v$	E
VS	0	E	$E.d_c$
IVRBP	M(M.dc - 1)	$E[(d_v-1)+(d_c-1)]$	$E[(d_v-1)(d_c-1)+1]$
OVRBP	N(N-1)	E	$E.d_c$

Cette thèse s'intéresse aux domaines applicatifs nécessitant une communication ultra fiable et à faible latence et implémentés sur des objets connectés avec des ressources matérielles très limitées. Ainsi, l'étude du compromis performances de correction et de complexité nécessite une analyse plus poussée pour permettre d'évaluer le pouvoir en terme de correction d'erreur et de quantifier la complexité calculatoire global requis par le processus de décodage en tenant compte du nombre d'itérations nécessaires pour converger vers un mot code valide, de la mémoire nécessaire pour le stockage de la matrice de contrôle de parité et des différents messages de mise à jour et du temps consommé par les accès mémoires.

Conclusion :

Les différents algorithmes et méthodes d'ordonnancement de la littérature ont des spécificités en terme de correction d'erreur, de rapidité de convergence d'une part, et d'une autre part de la difficulté d'implémentation et de la complexité des calculs. C'est en fonction du domaine d'application qu'il faut sélectionner un ou plusieurs schémas de décodage en fonction des différentes contraintes. En effet, selon les besoins en termes de latence, de ressources matérielles ou de pouvoir de correction, différents schémas de décodage LDPC présentant des compromis satisfaisant à ces besoins peuvent être retenus. Une analyse de complexité incluant le comportement dans le processeur et la mémoires des différents blocs de décodeurs LDPC sera l'objet du chapitre suivant, qui à travers une implémentation software des différents algorithmes de la littérature sur un environnement de simulation qui permettra à la fois d'estimer les performances en terme de correction d'erreur et de tracer le comportement du décodeur au niveau mémoire et au niveau processeur.

Chapitre II : Etude et analyse de la complexité d'implémentation du décodeur LDPC

En absence de modèle mathématique pour estimer le taux d'erreur des codes correcteurs d'erreur, la seule solution pratique consiste à effectuer des simulations de l'ensemble de la chaîne de transmission, afin de valider tous le système de codage et d'évaluer les performances de décodage en terme de BER/FER et de complexité des différentes configurations avant de les mettre en œuvre sur des systèmes réels. Cela signifie que certaines données sont générées, encodées, modulées, bruitées, décodées de manière aléatoire, et que les performances sont ensuite évaluées en mesurant le taux d'erreur binaire (BER) et le taux d'erreur paquet (FER) à la fin de la chaîne de communication. La complexité sera évaluée en terme de cycle CPU consommé, du nombre d'accès mémoire et d'autre caractéristiques en utilisant un outil de profilage qui permet de d'émuler le comportement du décodeur dans le processeur.

1 Architecture de l'environnement de test du décodeur LDPC

L'environnement de test et de simulation proposé est similaire à la chaine de communication numérique présentée dans la figure 1.1, avec ajout du module Moniteur qui permettra de collecter les messages sortants de la source du côté émetteur pour les comparer ensuite, avec les messages estimés du côté récepteur, pour évaluer les taux d'erreur des trames et des bits FER/BER. La figure 2.1, représente les différents blocs modélisés et développés en langage C, chaque bloc est validé avant de l'intégrer à l'environnement de test et de simulation.

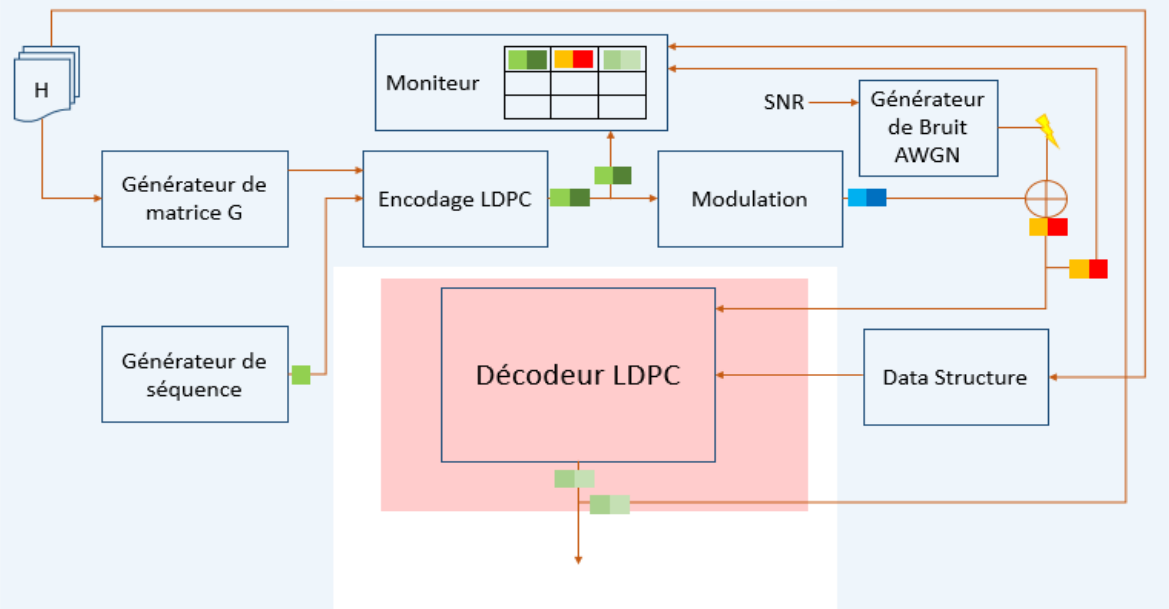


Figure 2. 1 : Architecture de l'environnement de test et de simulation

Les paramètres d'entrées de l'environnement de simulation sont les différents paramètres de codage et de décodage, les différents algorithmes de décodage, d'ordonnancement, ainsi que les différents types de matrices.

1.1 Bloc générateur de séquence et images tests

La partie source, ou émetteur pourrait être modélisée par un générateur pseudo-aléatoire de message de K valeurs binaires, ou en utilisant des données réelles comme du texte, de la voix, de l'image ou de la vidéo.

A partir d'une valeur d'entrée K, le générateur pseudo-aléatoire, génère K valeurs binaires aléatoires 0 ou 1. Afin de générer des valeurs pseudo-aléatoires sur des machines à système déterministes et pour ne pas avoir la même suite de valeurs à chaque simulation, on initialise le générateur avec trois valeurs : l'horloge système (time) de la machine, le numéro de la trame et le numéro de bit de la même trame.

Afin de pouvoir visualiser l'impact du bruit et de la correction des erreurs, des images tests sont utilisées. Un convertisseur des images d'extension ‘’.bmp’’ développé en langage C est utilisé. La figure 2.2 représente un tel générateur de séquence. Le convertisseur transforme une image vers un fichier décimal ‘’.txt’’ et vice-versa. Le fichier décimal ‘’.txt’’ obtenu contient les trois valeurs décimales des niveaux de couleur RVB (Rouge Vert Bleu) de chaque

pixel, ces valeurs sont comprises entre 0 et 255. Un convertisseur analogique numérique (CAN) est implémenté pour transformer ces valeurs décimales en binaire (encodé sur 8 bits), ensuite le fichier binaire produit est fragmenté en plusieurs messages de tailles K bits qui seront envoyés au bloc codage LDPC.

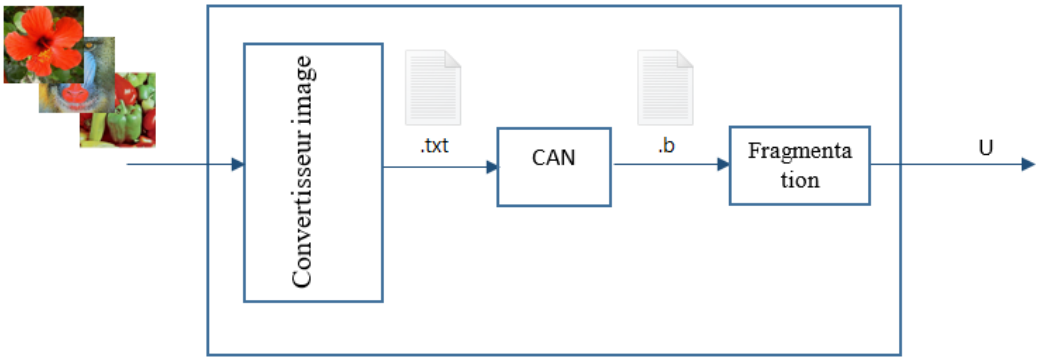


Figure 2. 2 Générateur de séquence à partir d'images tests

1.2 Bloc Codage LDPC

Etant donné un code LDPC de rendement R = K/N, le processus d'encodage systématique des messages à K bits produits par le générateur des séquences, consiste en l'ajout de N – K bits de parités, calculés à partir de la matrice G selon l'équation (1.8). Or, les codes LDPC sont définis par leurs matrices de parité H et non pas par leurs matrices génératrices G, il devient nécessaire avant d'entamer l'encodage de calculer la matrice génératrice G.

1.2.1 Calcul de la Matrice génératrice G

Etant donnée une matrice quasi cyclique composée de deux sous matrice A et B, où A est une sous matrice de taille (N-K x K) et B une matrice carrée triangulaire inférieure de taille (N-K x N-K), le calcul de la matrice génératrice G dans sa forme systématique G se fait à partir de la matrice H selon l'équation (1.25).

La figure 2.3, résume les étapes implémentées pour générer une matrice G, qui sont comme suit :

- A partir d'une matrice H on extrait les deux sous matrices A et B, la transposée A^t d'une matrice A est obtenue en échangeant les lignes et les colonnes de la matrice A.
- Le calcul de l'inverse de la matrice B^t noté $(B^t)^{-1}$ est réalisé en utilisant l'algorithme de pivot de Gauss qui découle de l'algèbre linéaire.

- Enfin, on réalise un produit matriciel des deux sous matrices obtenues et on ajoute à gauche la matrice identité conformément à l'équation (1.25).

Pour vérifier que la matrice ainsi obtenue est bien une matrice génératrice correspondante à la matrice H, il suffit de vérifier que le produit matriciel de l'équation (1.10) est égal à la matrice nulle, et ce, conformément à la définition de la matrice H.

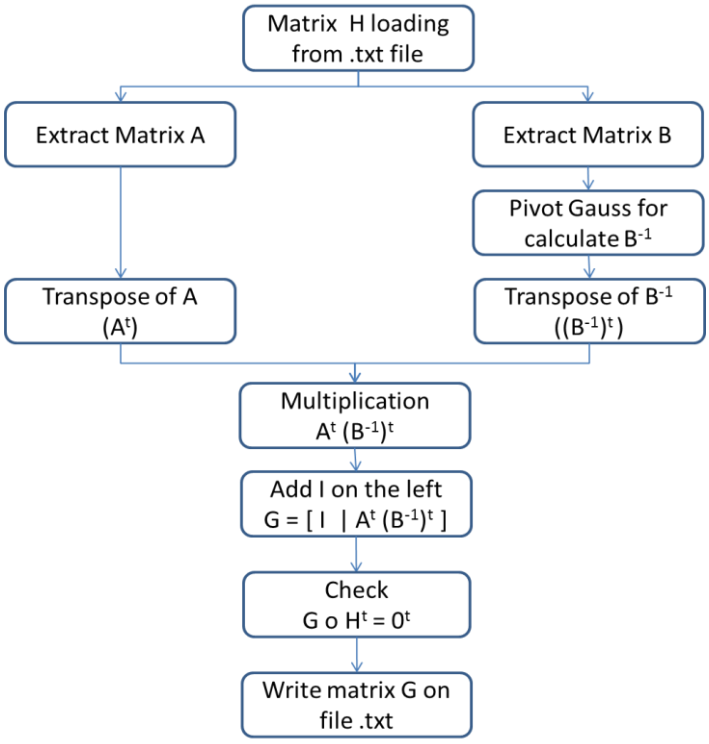


Figure 2. 3: Etapes de calcul de la matrice génératrice G

1.2.2 Modélisation de l'encodage LDPC

Une fois la matrice génératrice G calculée, l'encodage est réalisé en déterminant les (N – K) bits de redondances en utilisant l'équation (1.9), puis on les concatène au K bits du message d'origine pour avoir le mot code X de N bits.

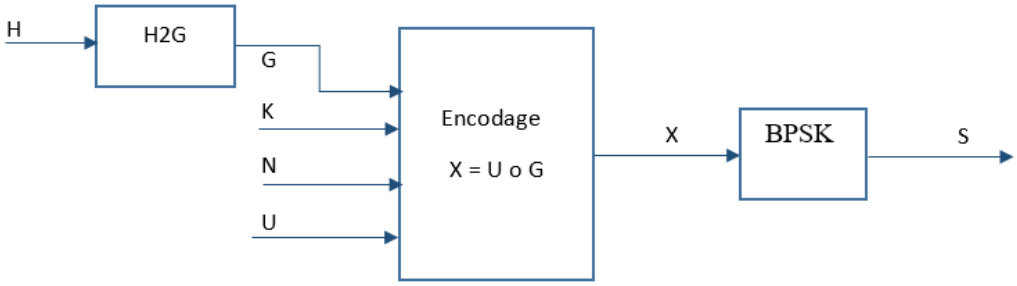


Figure 2. 4 : Bloc encodage LDPC et modulation BPSK

Pour vérifier qu'il s'agit bien d'un mot code, on vérifie que le produit du vecteur mot code et le transposé de la matrice de parité H est égal au vecteur nul conformément à l'équation (1.11).

La figure 2.4, représente les entrées et les sorties du bloc encodage, ainsi que le bloc modulation BPSK réalisé après l'encodage afin de transformer le flux binaire en signaux continus adaptés au canal de communication. Pour chaque valeur binaire $x_i \in \{0, 1\}$ du mot code X, une valeur $s_i \in \{1, -1\}$ du vecteur de symbole S est générée par la relation $s_i = 1 - 2x_i$. Ainsi, le canal de transmission le bit 0 sera représenté par le signal 1 et le bit 1 par le signal -1.

1.3 Bloc générateur de bruit AWNG (Box Muller)

Le canal de communication est modélisé par un canal à bruit blanc gaussien additif. Il additionne, aux éléments s_i du vecteur de symbole S obtenu après modulation, des échantillons de bruit blanc n_i de densité spectrale de bruit N_0, de répartition de probabilité gaussienne de moyenne 0 et d'écart type $\sigma = \sqrt{N_0/2}$, la sortie est exprimée par la formule suivante :

$$yi = si + ni \tag{2.1}$$

Pour générer des échantillons de bruit blanc, nous avons utilisé une fonction qui fournit des nombres pseudo aléatoires avec une répartition gaussienne de moyenne 0 et d'écart type 1. La transformée la plus connue pour passer d'une répartition uniformément distribuée à une répartition gaussienne est celle de Box Muller [60], qui stipule qu'une variable v a une répartition de probabilité gaussienne de moyenne 0 et d'écart type égal 1 si :

$$v = \sqrt{-2.\log(u_1)} \times \sin(2.\pi.u_2) \tag{2.2}$$

Avec u_1 et u_2 deux variables aléatoires indépendantes uniformément distribuées sur l'intervalle]0,1].

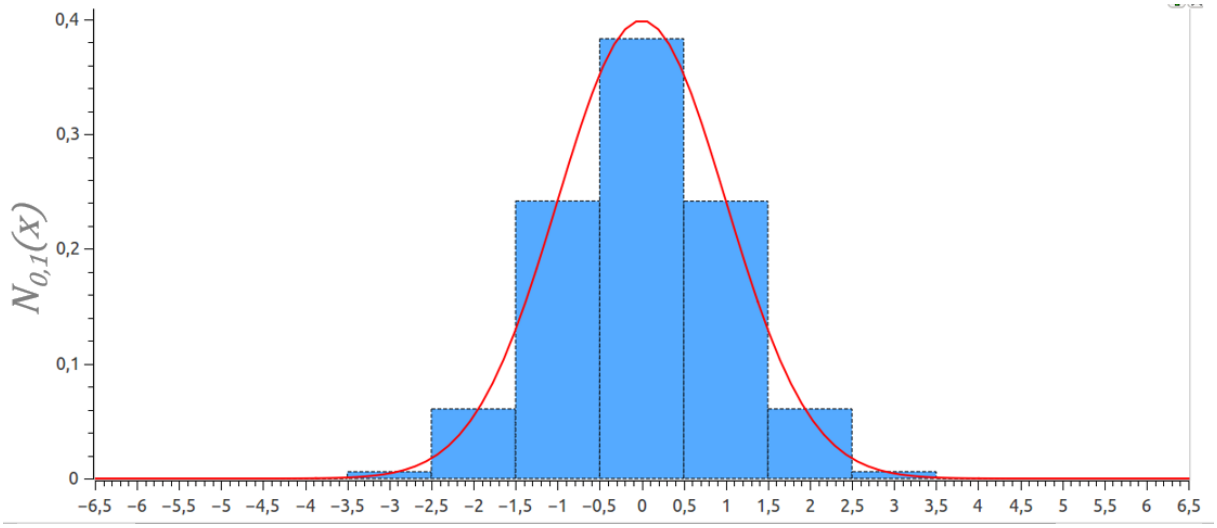


Figure 2. 5 : Comparaison des échantillons aléatoires générés par la fonction Box-Muller et la répartition théorique de la fonction gaussienne

Un examen préalable des données à l'aide de graphique ou de statistiques simples permet de visualiser si la distribution empirique suit une loi normale.

Un examen préalable des échantillons obtenus par notre générateur à l'aide de graphique, permet de visualiser si la distribution obtenue suit une loi normale. D'abord, nous avons simulé notre modèle avec 10^7 échantillons, ensuite on a classé les nombres aléatoires obtenus dans plusieurs intervalles compris entre -6.5 et 6.5 avec un pas de 1 pour chaque intervalle, chaque fois qu'on a un échantillon qui appartient à un intervalle on incrémente le compteur y afférent. Ainsi nous obtenons le nombre d'échantillons aléatoires pour chaque intervalle. La figure 2.5, représente les résultats sous forme d'histogramme ainsi que la répartition théorique de la fonction gaussienne de moyenne nulle et d'écart type 1, on remarque que les valeurs aléatoires obtenues par notre générateur s'ajustent parfaitement à une distribution normale.

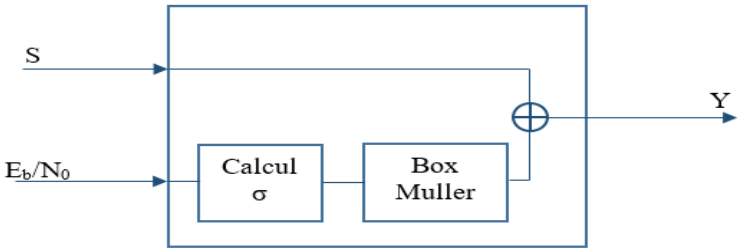


Figure 2. 6 : Bloc de modélisation du canal de communication

Une fois les échantillons gaussiens de moyenne 0 et d'écart type 1 sont obtenus par le module Box Muller, ils seront multipliés par l'écart type $\sigma = \sqrt{N_0/2}$ du bruit blanc gaussien de

densité spectrale N_0. La figure 2.6, représente les entrées et sorties du bloc modélisant le bruit du canal de transmission.

Le calcul de l'écart type du bruit blanc gaussien de densité spectrale N_0 est réalisé en calculant la densité spectrale, qui est reliée au rapport signal à bruit E_b/N_0 (en dB), pour une modulation BPSK, comme suit :

$$N_0 = \frac{1}{R \times 10^{\left(\frac{\left(\frac{E_b}{N_0}\right)_{dB}}{10}\right)}} \tag{2.3}$$

1.4 Bloc Moniteur

Le Bloc moniteur, permet de fournir les données nécessaires pour tracer les courbes BER et FER avec codage et sans codage canal, et de visualiser les images bruitées ainsi que les nombre d'itérations nécessaires pour le décodage. La figure 2.7, représente les entrées et sorties du bloc moniteur. En entrée, le bloc collecte les trois vecteurs à comparer :

- Les mots codes X avant injection du bruit blanc gaussien
- Les trames bruitées reçues depuis le canal Y auxquelles une décision dure est appliquée pour évaluer l'effet du bruit
- Les mots estimés $\hat{X}$ après décodage.

La fonction **Check()** du moniteur compare les différentes entrées pour calculer le nombre de bit et de trames erronés ce qui permet de calculer les BER et FER et de les enregistrer sur un fichier .txt sous format CSV utilisé pour tracer les courbes.

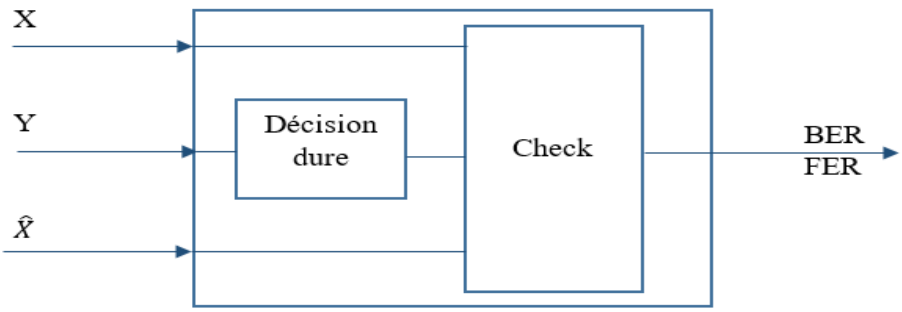


Figure 2. 7 : Modélisation du bloc moniteur

1.5 Architecture du bloc de décodage LDPC

Afin de pouvoir tester les différents algorithmes de décodage de la littérature et d'en proposer d'autres sans changer les autres blocs, une implémentation modulaire du décodeur LDPC est adoptée. Ainsi, les fonctions implémentant les équations de calcul des mises à jour des messages C2V, les équations de de mise à jour des messages V2C et les différentes fonctions implémentant le parcours des lignes et des colonnes de la matrice H conformément aux schémas d'ordonnancements adoptés sont codées séparément.

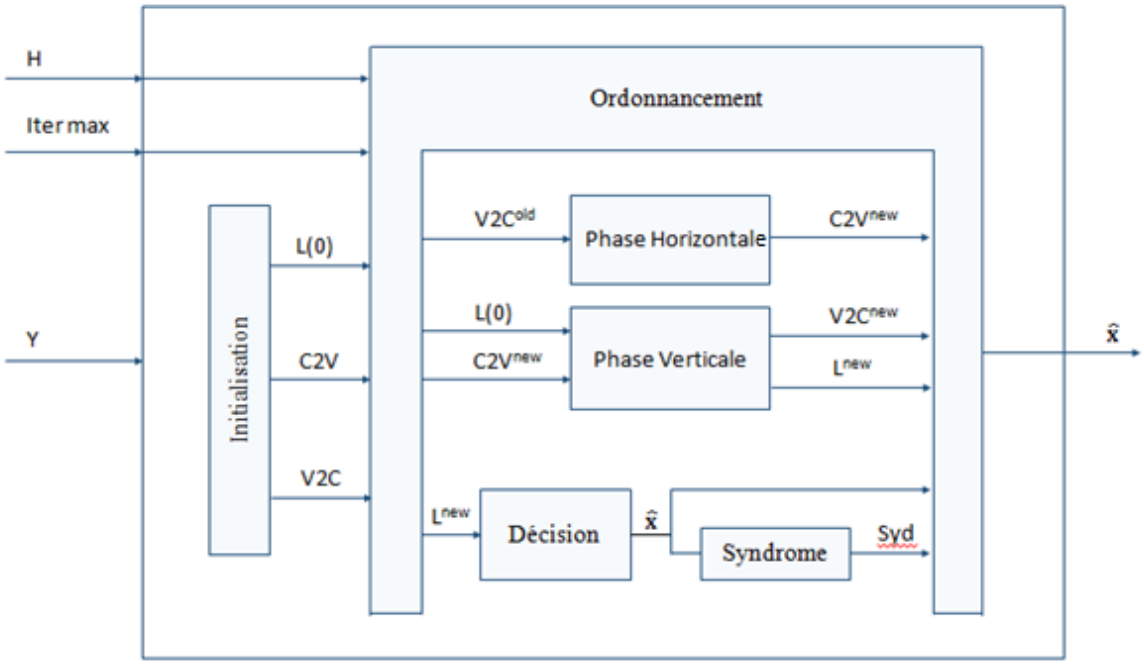


Figure 2. 8 : Modélisation du bloc décodeur LDPC

La figure 2.8, présente les différents modules du bloc décodeur LDPC :

Bloc initialisation :

Ce module permet d'initialiser les différentes variables nécessaires pour le décodage, notamment la valeur LLR des messages reçus Y conformément à l'équation (1.27), les messages C2V sont mis à zéro et les messages V2C sont initialisés par les LLR calculés, ce module est codé via une fonction C appelé initialization().

Bloc C2V update (Phase horizontal):

Ce module calcule les mises à jour des messages C2V conformément à l'équation (1.28). Ce module est codé via plusieurs fonctions C qui implémentent les équations des mises à jour selon l'algorithme adopté : les fonctions one_Cm_BP() et Box_Plus() implémentent l'algorithme LLR BP ; Les fonctions one_Cm_MSA() et MSA() implémentent l'algorithme MSA ; Les fonctions one_Cm_NMSA() et NMSA() implémentent l'algorithme NMSA et les fonctions one_Cm_SAOMSA() et SAOMSA() implémentent l'algorithme SAOMSA à 05 segments. D'autres algorithmes peuvent être ajoutés sans changer les autres parties du décodeur LDPC.

Bloc V2C update (Phase vertical):

Ce module calcule les mises à jour des messages V2C, ainsi que les mises à jour des LLR des nœuds de variable. Ce module est codé via la fonction C On_Vn() qui implémente les deux équations (1.29) et (1.30).

Bloc Décision :

Ce module réalise une décision dure sur les valeurs LLR mise à jour des nœuds de variable, afin d'estimer le mot code après décodage. Ce module est codé via la fonction decision() qui calcule la décision après la fin de chaque itération.

Bloc Syndrome :

Une fois la décision dure est faite, le module syndrome calcule le syndrome conformément à l'équation (1.15) moyennant la fonction syndrome().

Bloc Ordonnancement :

Ce module implémente la manière de parcours des différents nœuds de variables et de contrôles, le critère de choix du nœud à mette à jour le premier, ainsi toute les interactions entre les différentes fonctions du décodeur sont cadencés par le module Ordonnancement. Ce module est codé par plusieurs fonctions C qui implémentent la méthode d'ordonnancement adoptée : la fonction one_step_inondation() pour la méthode d'ordonnancement par inondation, les fonctions one_step_SH() et Zn_linkind2CM() pour l'ordonnancement brassé. D'autres méthodes d'ordonnancements peuvent être ajoutées sans changer les autres parties du décodeur LDPC.

Une fois les syndromes de toutes les lignes de la matrice de parité H sont nuls, ou le nombre d'itération maximal est atteint, le module ordonnancement arrête le processus de décodage et fournit en sortie le mot code estimé de N bits. Le message reçu correspond aux K premiers bits qui seront extraits en utilisant la fonction extract_recu().

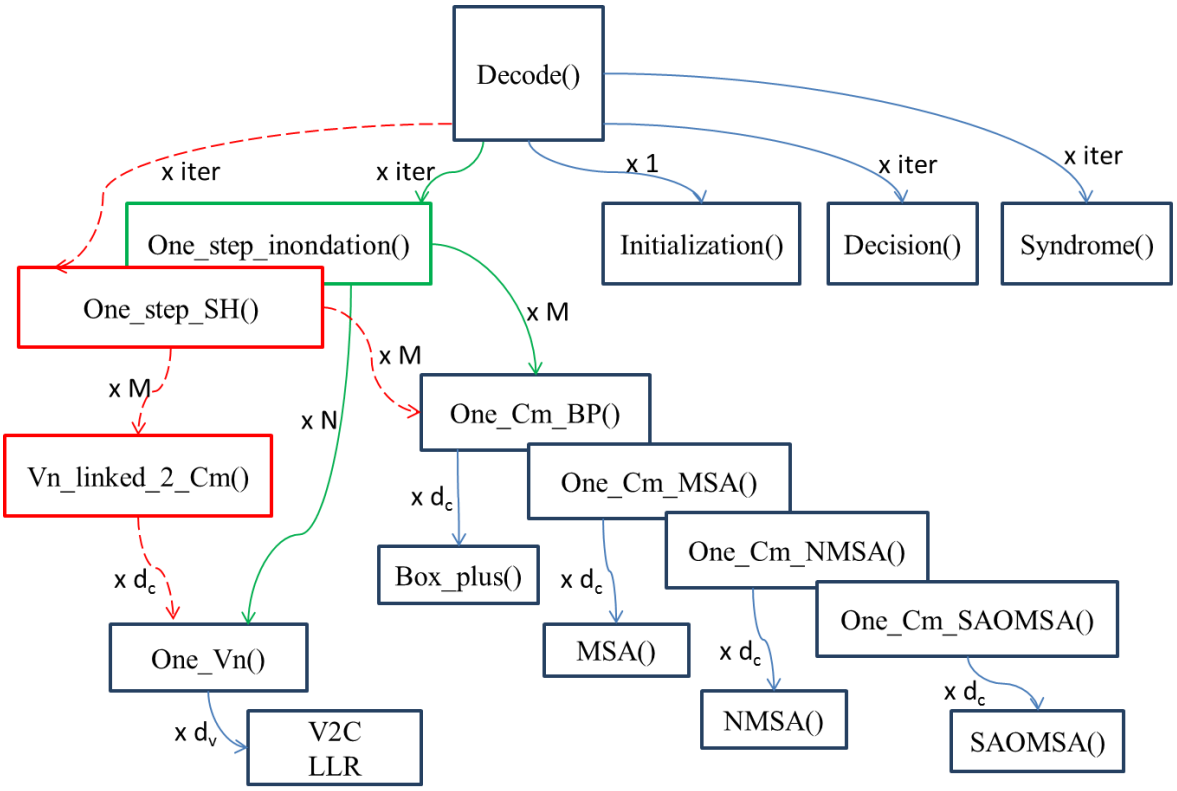


Figure 2. 9 : Diagramme des fonctions de l'implémentation du décodeur LDPC

La figure 2.9, résume les différentes fonctions constituant le décodeur LDPC ainsi que le nombre d'appel de chaque fonction pour les quatre types d'algorithmes de mises à jour des nœuds de contrôle (LLR BP, MSA, NMSA et SAOMSA) et deux types d'ordonnancement (par inondation et brassé HS). Le chemin en vert est emprunté par l'ordonnancement par inondation, et le chemin en rouge par l'ordonnancement brassé HS, les chemins en bleu sont communs pour les deux méthodes d'ordonnancement. Chaque flèche est étiquetée par le nombre de fois d'appel à la fonction. Les nombres Iter, M, N, d_c et d_v représentent respectivement le nombre d'itérations consommées lors du processus de décodage, le nombre des nœuds de contrôles, le nombre des nœuds de variables, le nombre d'éléments non nuls par ligne de la matrice et le nombre d'éléments non nuls par colonne de la matrice.

2 Structuration des données :

Les algorithmes de décodage LDPC se basent sur l'échange des mises à jour des messages C2V et V2C entre les nœuds de variable et les nœuds de contrôle conformément aux interconnexion fixées par la matrice de contrôle de parité H, ce qui nécessite la mémorisation de la matrice H, des messages de mise à jour, et des variables nécessaires pour le processus de décodage. Par conséquent, les variables nécessaires pour le bloc de décodage sont comme suit :

- La matrice de contrôle de parité H de taille $M \times N$
- Les N valeurs initiales des LLR des nœuds de variable, notées L_0, où N est le nombre de colonnes de la matrice de parité H ou bien le nombre des nœuds de variable dans le graphe de Tanner.
- Les E messages C2V, où E est le nombre d'éléments non nuls dans la matrice de parité H ou bien le nombre d'arcs dans le graphe de Tanner.
- Les E messages V2C.
- Les N valeurs de mise à jour des LLR des nœuds de variable, noté L.
- Les N valeurs binaires estimées après décodage et décision dure, notée $\hat{X}$

L'organisation et la structuration adoptée pour la mémorisation de ces variables a un effet sur le nombre de parcours ainsi que le nombre d'accès en mode lecture et écriture à la mémoire, et par conséquent sur la latence de décodage.

2.1 Organisation matricielle

L'organisation des messages C2V et V2C sous forme matricielle en adoptant la même structure de la matrice de parité H, demande une taille de mémoire importante exprimée par l'équation 2.4, et nécessite le parcours de toute la matrice pour le calcul des messages C2V ($M \times N$ parcours) et un deuxième parcours pour le calcul des messages V2C ($M \times N$ parcours) :

$$memory\ size = 2 \times N \times M \times sizeof(float) \tag{2.4}$$

Où *sizeof()* fait référence à l'opérateur qui donne la quantité de mémoire, en octets, nécessaire pour stocker un type de variable, ici *float* (un nombre réel).

Ces parcours sont pénalisants en termes de complexité et de nombres d'accès nécessaires en mode lecture/écriture à la mémoire, par conséquent cette méthode ne s'avère pas très pratique.

2.2 Organisation par méthode de stockage des matrices creuses (sparse matrix)

S'agissant de matrices creuses, les matrices de contrôle de parité H contiennent un nombre très réduit de valeurs non nulles, le tableau 2-1, présente le ratio d'éléments non nuls de quelques matrices utilisées dans le cadre de ces travaux, ainsi que les valeurs des degrés de connectivité des nœuds de variables qui varient de 2 à 6 et les valeurs des degrés de connectivité des nœuds de contrôles qui varient de 6 à 7. Ainsi, on peut réduire la taille de mémoire, nécessaire pour la mémorisation de la matrice, en ne conservant que les index (ligne, colonne) des éléments non nuls de la matrice. Ces index sont mémorisés dans deux tableaux, nommés ''Ligne'' et ''Colonne'', construits en balayant la matrice H par ligne et de gauche à droite. La taille de chacun de ces tableaux est de E éléments où E est le nombre des éléments non nuls dans la matrice H.

Tableau 2. 1 Dimensions des matrices creuses avec pourcentage des éléments non nuls

Dimension de la matrice (M , N)	Nombre d'éléments ($M \times N$)	(d_{vmin} ; d_{vmax} ; $d_{vmoy,}$)	(d_{cmin} ; d_{cmax} ; d_{cmoy})	Nombre des éléments non nuls (E)	Pourcentage des éléments non nuls
(4608, 9216)	42 467 328	(2 ; 6 ; 3,16)	(6 ; 7 ; 6,33)	29184	0,07%
(2304, 4608)	10 616 832	(2 ; 6 ; 3,16)	(6 ; 7 ; 6,33)	14592	0,14%
(1152, 2304)	2 654 208	(2 ; 6 ; 3,16)	(6 ; 7 ; 6,33)	7296	0,27%
(576, 1152)	663 552	(2 ; 6 ; 3,16)	(6 ; 7 ; 6,33)	3648	0,55%

En plus de ces deux tableaux, la mémorisation des messages C2V et V2C, nécessite aussi deux tableaux de taille E et qui suivent la même organisation des tableaux représentant la matrice H, par conséquent la taille mémoire nécessaire pour cette structuration des données est réduite à :

$$memory\ size = 2 \times E \times sizeof(integer) + 2 \times E \times sizeof(float) \quad (2.5)$$

Le premier terme de l'équation (2.5) concerne la mémorisation de la matrice et le deuxième concerne la mémorisation des messages C2V et V2C.

La figure 2.10, représente les quatre tableaux à mémoriser de la matrice (5, 10) ci-dessous

$$H = \begin{vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{vmatrix}$$

Ligne	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
Colonne	0	1	2	3	0	4	5	6	1	4	7	8	2	5	7	9	3	6	7	9
C2V	$m_{0\to0}$	$m_{0\to1}$	$m_{0\to2}$	$m_{0\to3}$	$m_{1\to0}$	$m_{1\to4}$	$m_{1\to5}$	$m_{1\to6}$	$m_{2\to1}$	$m_{2\to4}$	$m_{2\to7}$	$m_{2\to8}$	$m_{3\to2}$	$m_{3\to5}$	$m_{3\to7}$	$m_{3\to9}$	$m_{4\to3}$	$m_{4\to6}$	$m_{4\to7}$	$m_{4\to9}$
V2C	$m_{0\to0}$	$m_{1\to0}$	$m_{2\to0}$	$m_{3\to0}$	$m_{0\to1}$	$m_{4\to1}$	$m_{5\to1}$	$m_{6\to1}$	$m_{2\to1}$	$m_{4\to2}$	$m_{7\to2}$	$m_{8\to2}$	$m_{2\to3}$	$m_{5\to3}$	$m_{7\to3}$	$m_{9\to3}$	$m_{3\to4}$	$m_{6\to4}$	$m_{7\to4}$	$m_{9\to4}$

Figure 2. 10 Structuration des données par tableaux

Les parcours des différents tableaux se font comme suit :

- **Phase Horizontale :** pour chaque ligne *i*, les valeurs V2C sont lues, dans l'ordre d'organisation du tableau V2C, et sont ensuite utilisées dans le calcul des mises à jour des messages C2V qui sont enregistrés dans le même ordre du tableau C2V, ainsi le nombre de parcours est réduit à E parcours au lieu de $N \times M$ parcours dans le cas de la structuration par matrice.
- **Phase Verticale :** pour chaque colonne *j*, afin de lire les valeurs C2V nécessaires pour le calcul des mises à jour des messages V2C, il faut parcourir tout le tableau ''Colonne'' pour chercher les index des lignes, et ce, car les données sont organisées par ligne, ainsi le nombre de parcours est augmenté à $N\,x \times E$ parcours au lieu de $N \times M$ parcours.

Cette manière de mémoriser la matrice de contrôle de parité, a permis de réduire significativement la taille mémoire, cependant le nombre des parcours nécessaires lors du processus de décodage est pénalisant en termes de complexité et augmente considérablement le temps de décodage.

Afin de réduire le temps des simulations et d'avoir une flexibilité de parcours de la matrice nécessaire pour la simulation des différents types d'ordonnancement de profiter, une structure de donnée basé sur les pointeurs et les listes chainés et l'allocation dynamique de la mémoire a été adopté pour l'environnement de simulation.

2.3 Organisation en Graphe

La structuration des variables en graphe est réalisée en utilisant des tableaux de pointeur et de liste chainée comme suit :

- Les nœuds de variables sont représentés par un tableau de pointeur de taille N, chaque élément de ce tableau pointe sur le premier élément non nul de la colonne qu'il représente.
- Les nœuds de contrôles sont représentés par un tableau de pointeur de taille M, chaque élément de ce tableau pointe sur le premier élément non nul de la ligne qu'il représente
- Les éléments non nuls de la matrice sont représentés par la structure de donnée de la figure 2.11 ci-dessous :

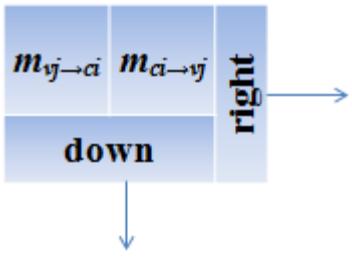


Figure 2. 11 : Structure de données utilisée par la méthode de Graphe

En plus des deux messages C2V et V2C de chaque élément non nul de la matrice H, cette structure mémorise l'adresse mémoire de la structure suivante sur la même colonne via la variable pointeur ''down'', et mémorise l'adresse mémoire de la structure suivante sur la même ligne via la variable pointeur ''right''. Lorsqu'il n'y a pas d'élément suivant par ligne ou par colonne les pointeurs ''down'' et ''right'' pointent sur NULL pour marquer la fin de la colonne ou de la ligne.

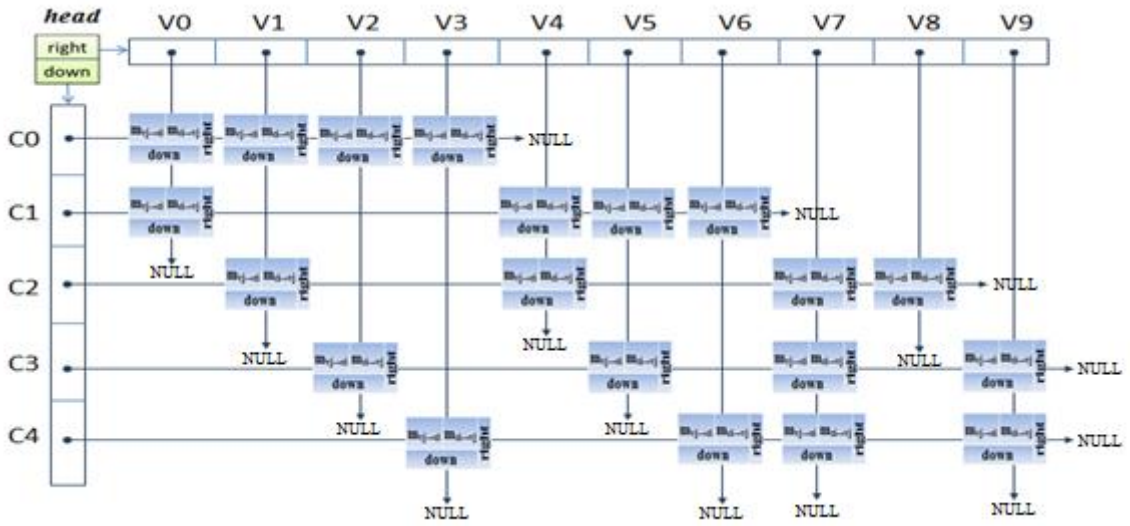


Figure 2. 12 : Exemple de structuration des données par Graphe de la matrice (5, 10)

La figure 2.12, présente le graphe proposé de la matrice (5, 10) du dernier exemple, un nœud ‘’*head*’’ pointe sur les deux tableaux de pointeurs est créé pour gérer tout le graphe, ainsi, avec l’adresse mémoire de ‘’head’’ on peut accéder à n’importe quel élément du graphe. Avec une telle organisation des données, le parcours est réalisé lors de la phase horizontale en suivant les pointeurs ‘’right’’ jusqu’à atteindre le pointeur ‘’NULL’’ et lors de la phase verticale en suivant les pointeurs ‘’down’’ jusqu’à atteindre le pointeur ‘NULL’’. La taille mémoire nécessaire dans ce cas est de :

$$memory\ size = (2 \times E + M + N + 2)\ \times\ sizeof(pointer) + \ 2 \times E\ \times\ sizeof(float) \qquad (2.6)$$

Le premier terme de l’équation 2.6 concerne la mémorisation de la matrice et le deuxième concerne la mémorisation des messages C2V et V2C. La taille mémoire nécessaire pour la mémorisation des variables C2V et V2C est la même que la méthode des tableaux, cependant, l’organisation en graphe augmente la taille mémoire, en raison de la taille mémoire nécessaire pour mémoriser une adresse mémoire $eof(pointer)$, qui est supérieure à celle nécessaire pour mémoriser une donnée de type entier $Sizeof(Integer\ Short)$. Le tableau 2-2 ci-dessous indique les tailles des types de données pour les deux systèmes d’adressage 32bit et 64 bit.

Tableau 2. 2 Taille des types de donnée

	Système d'adressage	
	32 bit	**64 bit**
$Sizeof(Integer\ Short)$	2 octets	2 octets
$Sizeof(pointer)$	4 octets	8 octets
$Sizeof(float)$	4 octets	4 octets

La méthode de structuration par graphe de Tanner, augmente légèrement la taille mémoire par rapport à la méthode des tableaux (sparse matrix), en contrepartie la complexité des parcours est réduite à E parcours pour la phase horizontale et E parcours pour la phase verticale, le tableau 3-2 récapitule la complexité de parcours en fonction de la méthode de structuration des données.

Tableau 2. 3 Comparaison des complexités des différentes méthodes de structuration des données.

		Complexité de parcours	
		Phase horizontale	Phase verticale
Forme matricielle		N x (N-K)	N x (N-K)
Tableaux	Par ligne	E	N x E
	Par colonne	M x E	E
Graphe de Tanner		E	E

3 Résultats des simulations

Afin d'estimer avec précision les performances en terme de BER/FER du décodeur LDPC, au moins 10 trames erronées sont simulées, ainsi la mesure d'un FER de 10^{-6} nécessite la simulation de la transmission de $10 \times 10^6 = 10^7$ trames. En supposant une trame de 1000 bits, le simulateur doit alors traiter la transmission de 10^{10} bits. En gardant à l'esprit que la complexité de calcul des algorithmes de décodage LDPC est importante, plusieurs semaines peuvent être nécessaires pour estimer avec précision les BER/FER d'un schéma de codage (en particulier si la densité de bruit est grande). Dans le cadre de cette thèse, la plateforme HPC MARWAN [47] a été utilisée pour évaluer les performances en termes de BER/FER, ce qui a permis de lancer plusieurs simulations en parallèle en profitant du nombre des nœuds disponibles ainsi que le nombre de cœurs par nœuds. Nous avons utilisé les codes (576, 288), (576, 144), (1152, 576) et (2304, 1152) qui sont des codes LDPC irréguliers quasi cycliques construits sur la base de la norme IEEE 802.16e [19]. Les courbes des BER et FER dépendent du SNR (E_b/N_0). Plus le SNR (E_b/N_0) est élevé, plus le bruit est faible et par conséquent le nombre d'erreurs est réduit.

3.1 Performance d'une communication avec codage et sans codage

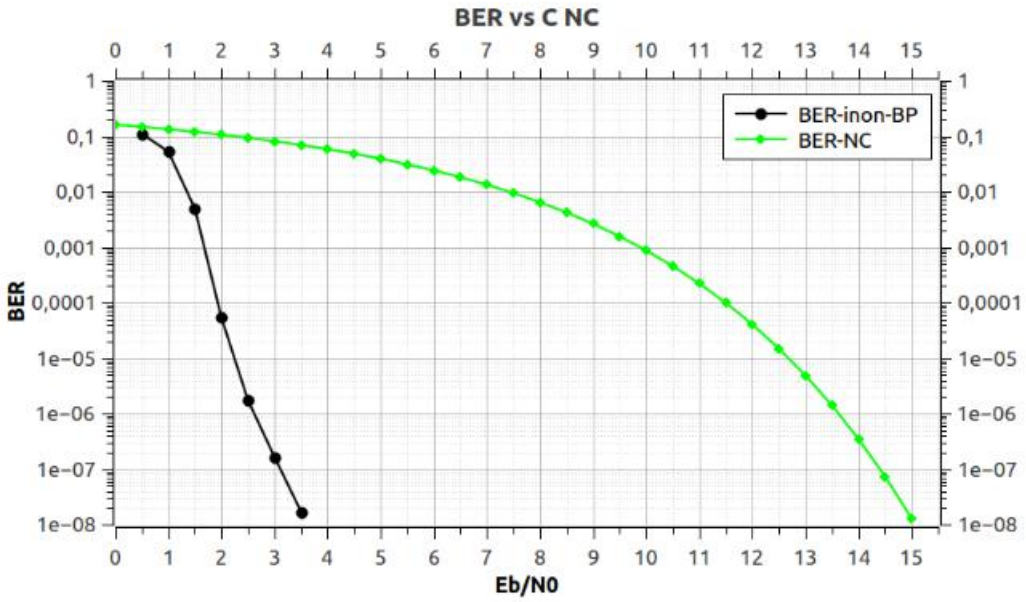


Figure 2. 13 : BER avec codage LDPC et sans codage vs SNR

La figure 2.13 illustre le BER obtenu par simulation d'une communication avec codage/décodage LDPC (courbe en noire) basé sur la propagation de la vraisemblance (BP) et la méthode d'ordonnancement par inondation et une communication sans codage (courbe en vert). La figure montre que pour un SNR de 3.5dB le BER est de $8x10^{-2}$ pour une communication sans codage et de 1.5 x 10^{-8} pour une communication avec codage LDPC, autrement dit, l'envoie de l'image 512 x 512 = 262144 pixels de l'exemple précédent (6291456 bits), sur un canal de transmission sans utilisé le codage canal génère 6291456 x $8x10^{-2}$ = 503 316 bits erronés, tandis qu'avec l'utilisation codage LDPC il ne génère aucun bit erroné, de telle performance ne peuvent être atteint sur un canal sans codage canal que pour un SNR de 15dB, ainsi le gain de codage/décodage est de 15dB – 3.5 dB = 11.5 dB.

3.2 Effet de nombre maximal d'itération sur les performances de décodage

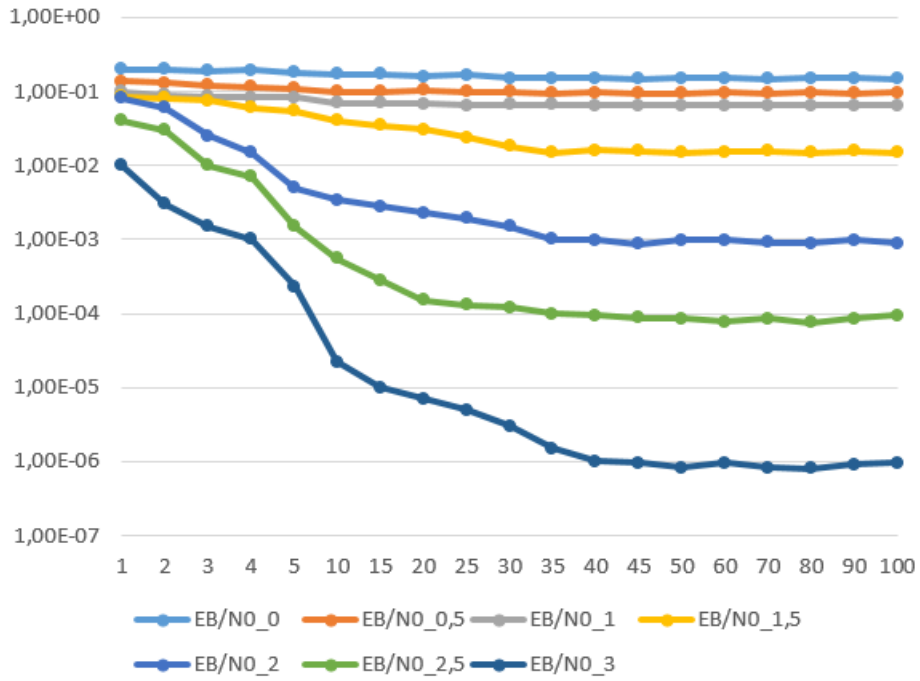


Figure 2. 14 :BER en fonction des itérations maximales

Au cours du processus de décodage, Le nombre d'itération maximum, permet à la fois d'exécuter suffisamment d'itérations pour trouver le mot code valide et de quitter la boucle de décodage dans le cas contraire, chaque itération induit une complexité de calcul et par conséquence une latence supplémentaire. Le choix optimal de ce critère est à mettre en balance avec les performances en terme de correction d'erreurs. La figure 2.14, trace les performances en termes de correction d'erreur pour différents SNR de 0dB à 3dB en fonction des itérations maximales qui varient de 1 à100 itérations. Comme illustré sur la figure pour des SNR très faibles (de 0 dB à 1,5 dB), l'ajout d'itération n'a pratiquement aucun impact sur le BER qui n'atteint même pas 10^{-2}. Pour des SNR supérieur à 2 dB, on remarque que l'ajout d'itération permet d'améliorer le BER, mais dès que le BER atteint une valeur limite, pour un SNR donné, l'ajout d'itération ne porte pas d'amélioration considérable.

De ce fait, la valeur optimale du nombre d'itération maximal est la valeur minimale qui permet d'atteindre cette valeur limite du BER. Aussi, on dit qu'un algorithme converge plus vite s'il nécessite un nombre d'itération maximal très réduit pour atteindre une limite de BER.

3.3 Effet du rendement du code R sur les performances de décodage

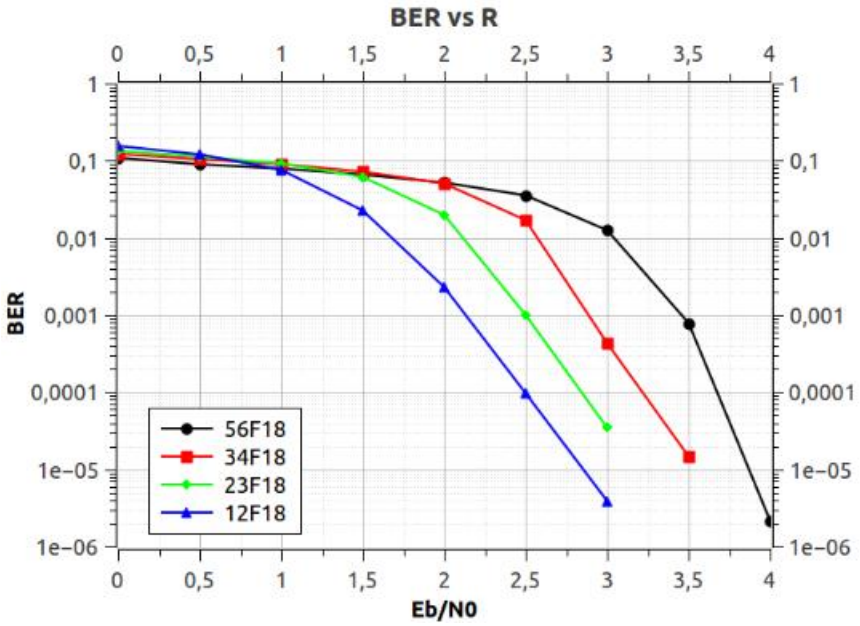


Figure 2. 15 : BER en fonction du rendement R du codage

La figure 2.15, montre les performances en termes de correction d'erreur de l'algorithme LLR BP, pour différentes valeurs de rendement de codage $R = \frac{5}{6}$; $\frac{3}{4}$; $\frac{2}{3}$; $\frac{1}{2}$. Plus le rendement d'un code est faible, plus son pouvoir de correction est élevé, par contre le nombre de bits d'information utile est faible. Comme illustré sur la figure 3.14, pour un SNR de 3dB l'utilisation d'un rendement R=1/2 (courbe bleu) est indispensable pour une communication sans erreurs avec un taux d'erreur binaire de BER à 3 x 10^{-6}, même si le débit utile dans ce cas représentera uniquement la moitié du débit réel transmis, tandis que pour un SNR de 4 dB l'utilisation d'un rendement R=5/6 (courbe noire) est suffisante pour avoir une communication sans erreur (BER à 2 x 10^{-6}), le débit utile dans ce cas représentera 5/6 du débit réel transmis. Par conséquent, le choix d'un rendement de codage/décodage est fait dynamiquement lors de la transmission en fonction de la qualité du canal de communication.

3.4 Effet de la taille du bloc d'information sur les performances de décodage

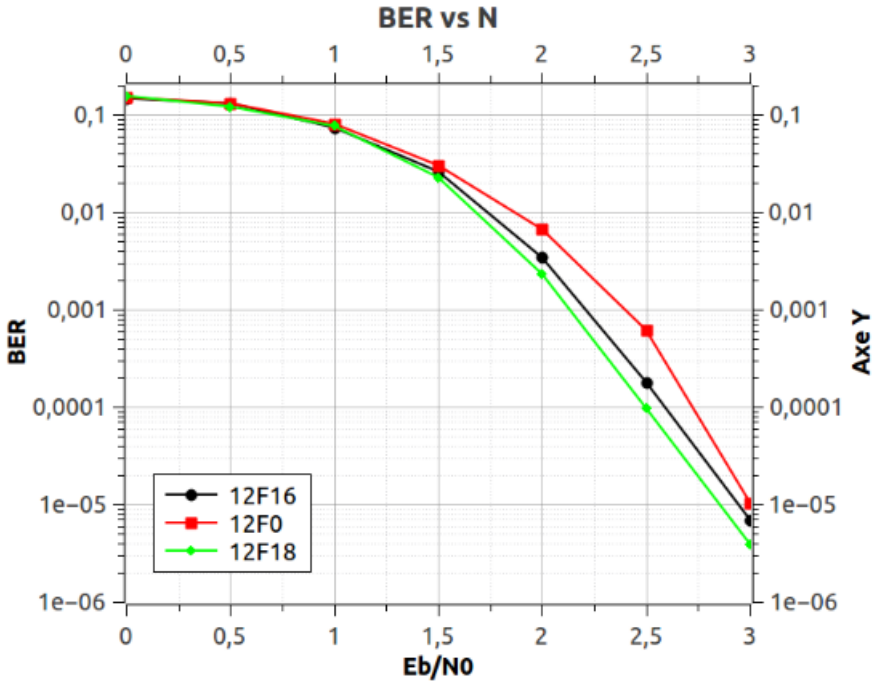


Figure 2. 16 : BER en fonction de la taille K du bloc d'information

La figure 2.16, trace les courbes de plusieurs tailles du bloc d'information K (1152 bits, noté sur la figure 12F18 ; 576 bits, noté sur la figure 12F16 ; 288 bits, noté sur la figure 12F0) en fonction du SNR, comme illustré sur la figure les performances en correction des erreurs augmentent avec la taille du message d'information. Cependant, le codage et le décodage d'un bloc d'information de grande taille requiert l'utilisation de matrice de parité H de grande taille avec plus des éléments non nuls E, ce qui augmente la complexité de codage et de décodage. Pour les codes quasi cycliques le nombre d'éléments non nuls E est proportionnellement relié à la taille de la matrice, et par conséquent à la taille du bloc d'information. Ainsi la complexité de de décodage d'une matrice H correspondante à un bloc d'information de taille K est similaire à celle relative au décodage de blocs d'information chacun de taille $K/2$. Par conséquent, le choix de grande taille K du bloc d'information permet d'avoir de bonnes performances en terme de convergence et de qualité, mais le stockage des variables correspondants dépendra de la taille mémoire disponible sur l'architecture cible.

3.5 Effet des algorithmes de mise à jour des messages C2V sur les performances de décodage

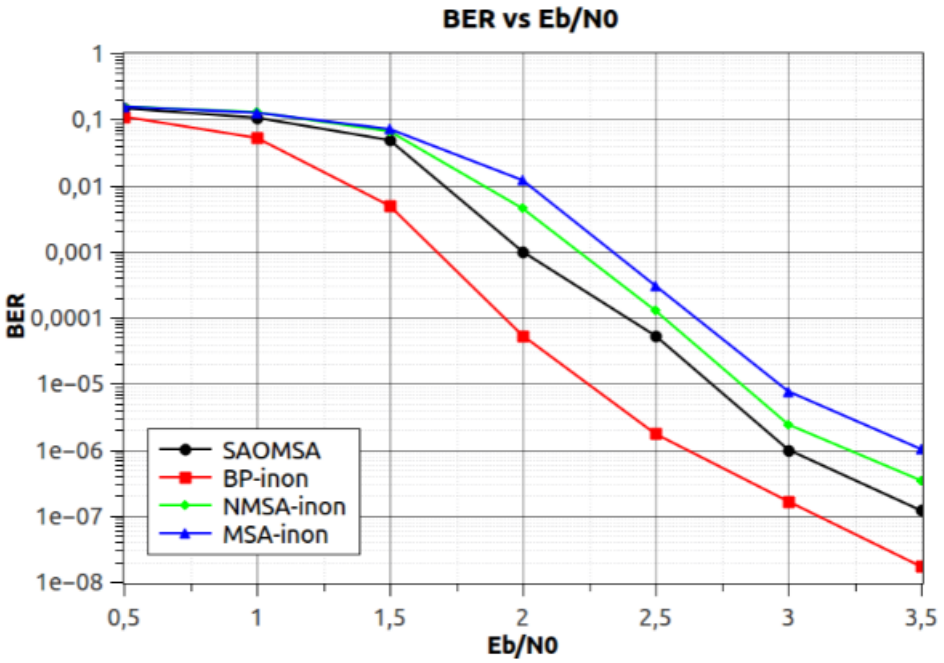


Figure 2. 17 : BER en fonction de l'algorithme de mise à jour des messages C2V

Comme évoqué dans le chapitre I, l'algorithme de base de décodage LDPC (LLR-BP) est reconnu pour ses performances en termes de correction d'erreurs. Mais, la complexité de calcul reliée à l'équation non linéaire (1.23) de mise à jour des messages C2V est très élevée, ce qui augmente la latence de décodage. La figure 2.17, met en évidence les performances en termes de correction des erreurs de l'algorithme de base ainsi que ses dérivés les plus utilisés. L'algorithme LLR BP permet d'atteindre des BER de 10^{-6} à un SNR de 2.5 dB, avec un gain de 1 dB en comparaison avec l'algorithme MSA le moins coûteux en termes de complexité de calcul et qui nécessite un SNR de 3.5 dB pour atteindre les performances de 10^{-6}. L'algorithme SAOMSA affiche les performances les plus proches de l'algorithme de base avec une perte de 0.3 dB mais sa complexité de calcul reste supérieure à l'algorithme NMSA à 0.2 dB de perte par rapport à l'algorithme SAOMSA et 0.5 dB de l'algorithme de base LLR BP. Les complexités de calcul seront détaillées dans les paragraphes suivants en utilisant l'outil de profilage.

3.6 Effet des méthodes d'ordonnancement sur la vitesse de convergence et les performances de décodage :

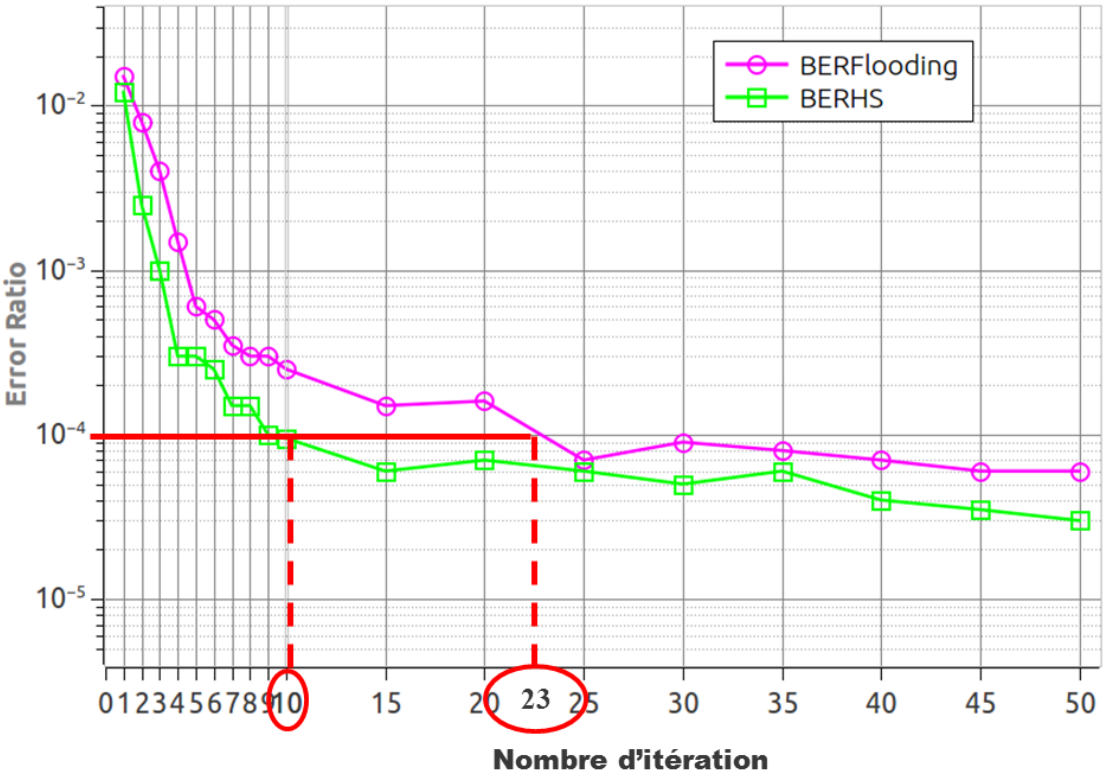


Figure 2. 18 : Convergence de décodage LDPC en fonction de la méthode d'ordonnancement

Lors de l'exécution du processus du décodage, l'ordonnancement définit la stratégie de sélection et l'ordre de mise à jour des nœuds ainsi que la manière de parcours du graphe de tanner. L'ordonnancement a un impact sur le pouvoir de correction, la vitesse de convergence et le degré de parallélisme nécessaires pour le lancement de décodage sur des plateformes multi-cœurs.

Pour mettre en évidence la vitesse de convergence des différentes méthodes d'ordonnancement, la figure 2.18, trace pour un SNR de 2,5dB le taux d'erreur BER en fonction du nombre d'itération maximale. On remarque que pour avoir par exemple un BER de 10^{-4}, l'ordonnancement par inondation nécessite 23 itérations, alors que pour la méthode d'ordonnancement brassé le même BER est atteinte uniquement après 10 itérations. Par conséquence, les méthodes d'ordonnancements brassés permettent d'améliorer les performances en termes de correction d'erreurs en dépassant la limite du BER et permettent aussi de converger rapidement en ne demandant que peu d'itération pour atteindre cette limite.

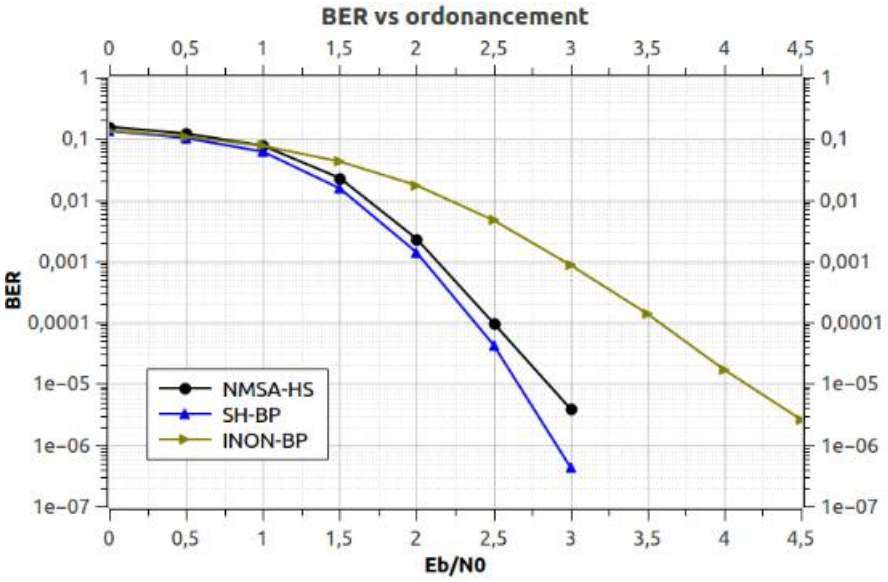


Figure 2. 19 : BER en fonction de la méthode d'ordonnancement

La figure 2.19, trace le pouvoir de correction en termes de BER des types d'ordonnancement les plus répondus dans la littérature. En fixant le même nombre d'itération maximale à une valeur de 5 itérations. On observe que selon le type d'ordonnancement adopté le BER est amélioré par rapport à la méthode d'ordonnancement par inondation dont le BER est de 2.10^{-6} à un SNR 4,5 dB, alors que pour la méthode d'ordonnancement brassé ''Horizontal Shuffle, le BER atteint les 2.10^{-7} à SNR de 3dB soit un gain de 1 dB entre les deux méthodes d'ordonnancements.

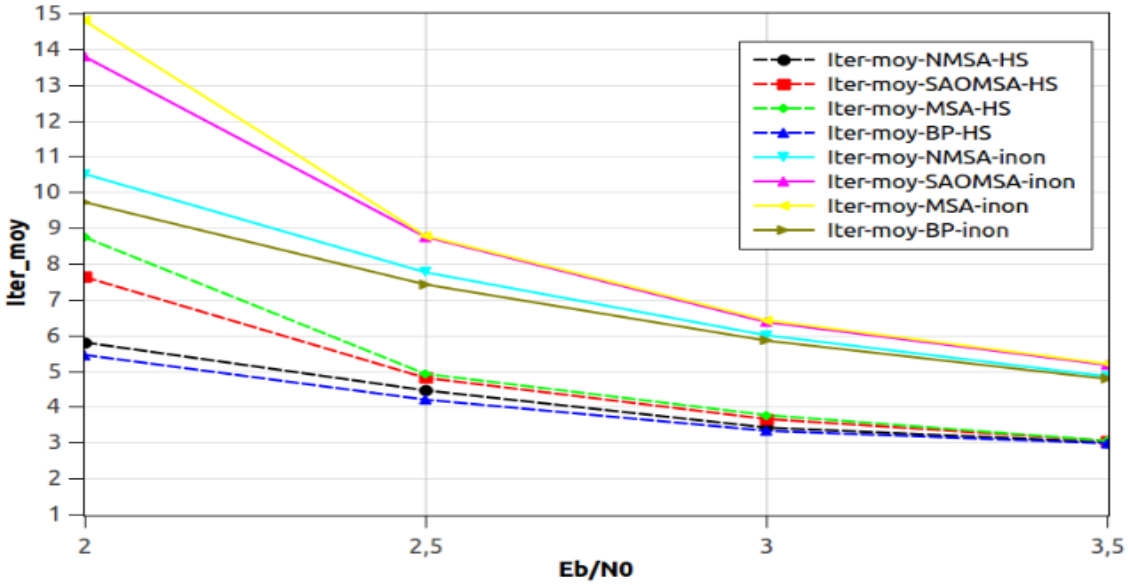


Figure 2. 20 : Itérations moyennes lors de décodage en fonctions des différents algorithmes de mise à jour et d'ordonnancement

La figure 2.20, illustre le nombre d'itération moyen consommé lors du processus de décodage, cette courbe montre une diminution significative du nombre d'itérations nécessaires pour corriger les erreurs et obtenir un mot code valide. En effet, l'ordonnancement par inondation a besoin d'une moyenne de 6 à 6,5 itérations pour converger à sa limite de BER pour un SNR de 3dB, alors que l'ordonnancement brassé permet de dépasser le BER obtenu par la méthode d'inondation pour le même SNR avec seulement 3,5 à 4 itérations

4 Outils et Résultats des profilages

4.1 Outils logiciels de profilage

Nous utiliserons dans les travaux de cette thèse, deux outils de profilage Gprof[58] et Valgrind[59], l'outil Gprof est une application de profilage qui collecte de l'information et compile des statistiques sur le code. Quand le programme est exécuté, un fichier de données brutes est créé et interprété par Gprof qui insère les temps de calcul et le nombre de fois d'appel de chaque fonction du programme.

L'outil Valgrind permet de détecter des problèmes de gestion mémoire dans des programmes écrits en C / C++, de simuler la mémoire cache et de comptabiliser le nombre d'appel des fonctions et les coûts associés en terme de nombre de cycle d'horloge CPU. Plus généralement, Valgrind repose sur une architecture générique permettant la supervision de programmes à l'aide de trois principales options :

- Memcheck : un détecteur de fuites mémoires, genre mémoire non nécessaire non libérée ou un objet stocké en mémoire mais n'est plus accessible par le code en cours d'exécution.
- Cachegrind : un simulateur des mémoires caches niveau 1 (L1), 2 (L2), et dernier niveau (Last level LL), qui permet d'identifier le nombre de fois que le programme cherche les données à l'extérieur de la mémoire cache.
- Callgrind : un profileur qui collecte le nombre d'appel des fonctions, les accès en lecture et en écriture de la mémoire et calcul le temps d'exécution en cycle d'horloge.

L'objectif du profilage est d'extraire des données sur le comportement au niveau processeur et au niveau de la mémoire cache lors de l'exécution du décodage LDPC, et ce afin de détecter les blocs codes qui consomment plus de cycle d'horloge CPU et qui peuvent être des bons candidats pour des éventuelles optimisations.

Lors d'exécution du décodage LDPC, l'accès à la mémoire cache plutôt que l'accès à la mémoire vive (RAM), permet de diminuer la latence de décodage car la mémoire cache est une mémoire plus rapide et plus proche du processeur, mais sa taille est très limitée et ne dépasse pas les 32Ko dans la plupart des architectures.

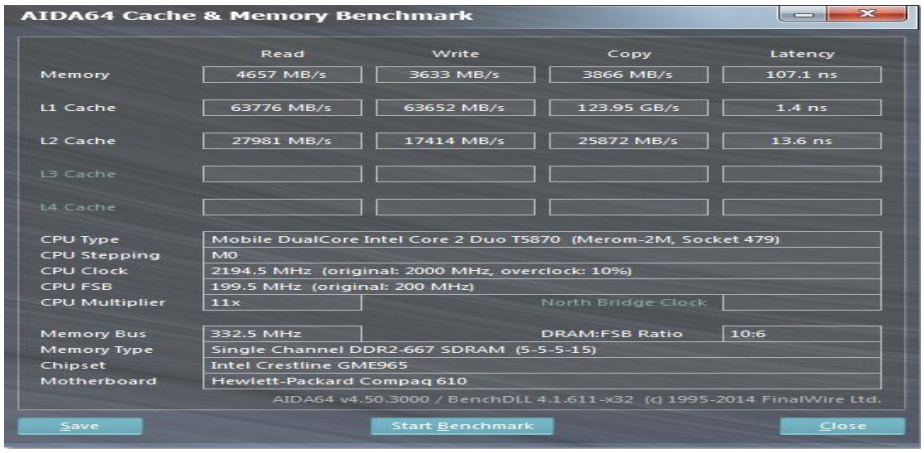


Figure 2. 21 : Comparaison de vitesse d'accès à différents types de mémoire

La figure 2.21, représente une comparaison de la latence nécessaire pour l'accès aux différents type de mémoire pour un PC core2 Duo, le temps d'accès à la mémoire RAM (107.1ns) est environ 100 fois plus grand que celui nécessaire pour l'accès au niveau L1 du cache (1.4 ns), aussi le temps d'accès au niveau L2 du cache (13.6 ns) est d'environ 10 fois plus grand.

4.2 Résultats de profilage

4.2.1 Effet de la structure de donnée sur le nombre de cycles CPU consommés

En utilisant l'outil de profilage ''Gprof'' qui permet de collecter le nombre de fois d'appel des fonctions et le temps en pourcentage consommé par chaque fonction, on commence par le profilage des différentes structurations de donnée présentées en haut pour la mémorisation de la matrice de contrôle de parité H.

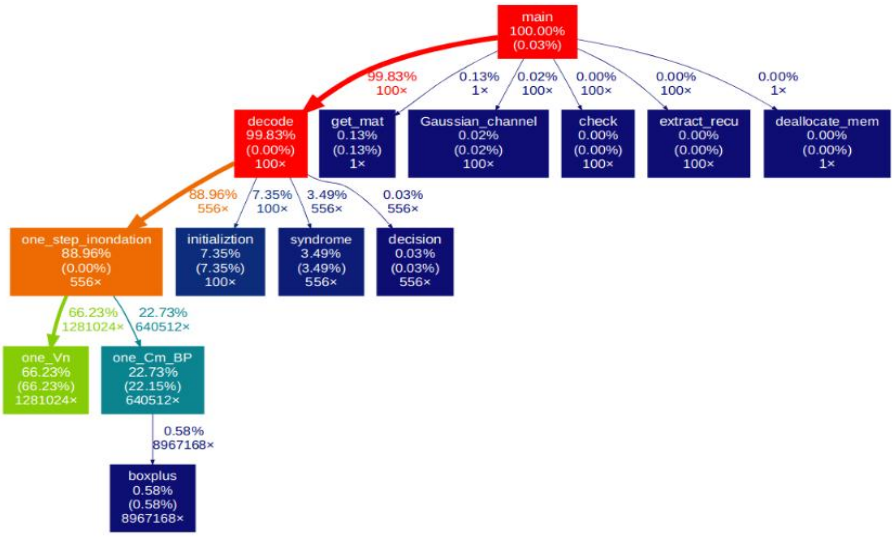


Figure 2. 22: Diagramme généré par l'outil Gprof relatif au appel et de temps CPU consommés par les différents fonctions du décodeur LDPC

La figure 2.22, montre les résultats de l'outil de profilage Gprof pour une structuration sous la forme matricielle.

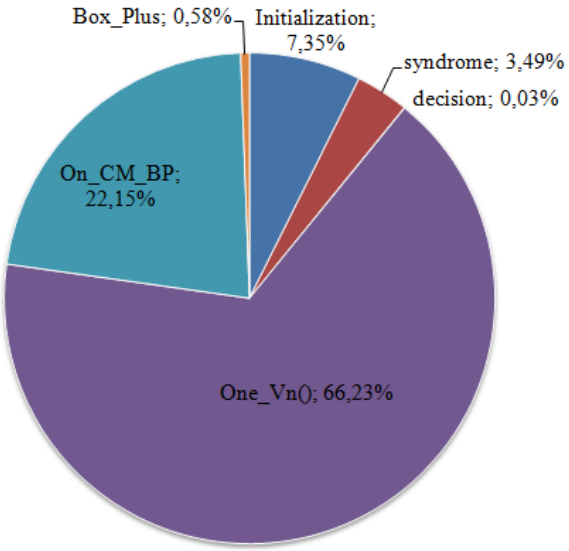


Figure 2. 23 : Diagramme en secteur de temps CPU consommés par les différents Bloc du décodeur LDPC

La représentation de ces résultats en diagramme de secteurs sur la figure 2.23 permet de constater que 88.38% du temps est consommé par dans des fonctions de parcours (66.23% de parcours en phase horizontale et 22.15% de parcours en phase verticale), alors que la fonction non linéaire de la mise à jour des messages C2V $boxplus$ () ne consomme que 0.58%. Ces résultats sont justifiés par le nombre excessif d'instructions nécessaires pour le parcours de la matrice, ainsi que le nombre important d'accès en mode lecture et écriture en dehors de la

mémoire cache, du fait que la taille de la matrice est très importante et ne peut pas être chargé dans la mémoire cache dont la taille est de l'ordre de 32 Ko.

e Detection | Relative to Parent | <> Shorten Templates | » | Cycle Estimation

decode.c

Types | Callers | All Callers | Callee Map | Source Code

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	-	140 382 675 531	Ir	
L1 Instr. Fetch Miss	-	485	I1mr	
LL Instr. Fetch Miss	-	485	ILmr	
Data Read Access	-	58 086 372 884	Dr	
L1 Data Read Miss	-	3 756 523 594	D1mr	
LL Data Read Miss	-	3 460 400 217	DLmr	
Data Write Access	-	788 281 992	Dw	
L1 Data Write Miss	-	67 813 392	D1mw	
LL Data Write Miss	-	67 519 251	DLmw	
L1 Miss Sum	-	3 824 337 471	L1m	= I1mr + D1mr + D1mw
Last-level Miss Sum	-	3 527 919 953	LLm	= ILmr + DLmr + DLmw
Cycle Estimation	-	531 418 045 541	CEst	= Ir + 10 L1m + 100 LLm

Figure 2. 21 : Résultats de simulation de la mémoire cache généré par l'outil valgrind

Pour mettre en évidence le comportement au niveau de la mémoire cache, la figure 2.24 illustre les résultats de l'outil de profilage ''Valgrind cache'' qui permet de simuler les niveaux L1, L2, et dernier niveau LL de la mémoire cache.

Le diagramme en secteur des résultats obtenus présenté sur la figure 2.25, montre que 66% des cycles sont consommés en cherchant des données en dehors de la mémoire cache, et ce, à cause de la mémorisation de la matrice H sous sa forme matricielle.

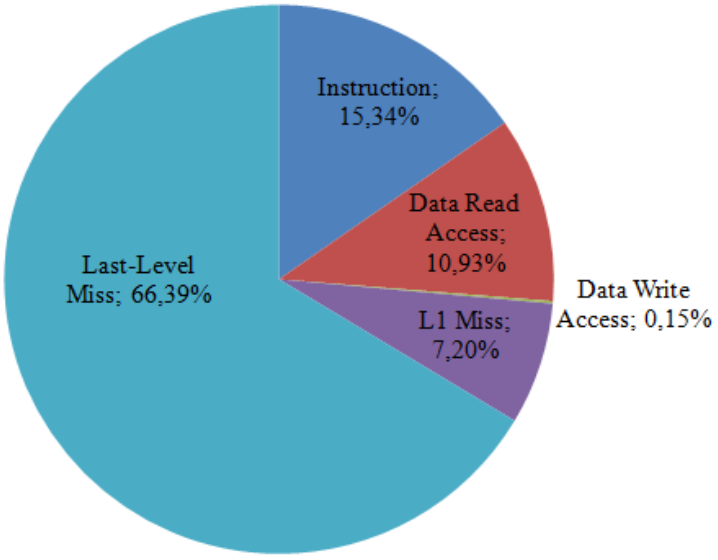


Figure 2. 22 : Diagramme en secteurs des Résultats obtenues par l'outil Valgrind

Les mêmes méthodologies de profilage sont appliquées pour la méthode de stockage de matrice creuse (sparse matrix) à l'aide de tableaux et en utilisant le graphe de Tanner, les résultats de profilage des trois méthodes de structuration des données sont récapitulés sur le tableau 2-4, les six lignes du tableau représentent le nombre totale de cycles CPU consommés pour le décodage (ligne Cycle), le nombre de cycles CPU consommés par les instructions du programme (ligne Instruction), le nombre de cycles CPU consommés par l'accès en lecture à la mémoire cache (ligne Data Read Access), le nombre de cycles CPU consommés par l'accès en écriture à la mémoire cache (ligne Data Write Access), le nombre de cycles CPU consommés par l'accès en dehors du premier niveau L1 de la mémoire cache (ligne L1 Miss) et le nombre de cycles CPU consommés par l'accès à la mémoire RAM c'est-à-dire en dehors du dernier niveau de la mémoire cache (ligne Last-Level Miss).

La représentation de ces résultats en diagramme en forme d'histogramme en échelle logarithmique sur la figure 2.26, permet de constater que la méthode du graphe permet de réduire considérablement le nombre total des cycles nécessaires pour le décodage et d'éliminer le problème d'accès en dehors du dernier niveau (Last Level LL) de la mémoire cache, appelé défaut de cache au niveau LL ou en anglais LL miss cache.

Tableau 2. 4 : Résultats de profilages des trois méthodes d'organisation des données

	Matrix	Tab 1	Graphe
Cycle	5 314 180 455	5 509 793 300	31 419 465
Instruction	815 080 268	2 398 445 702	14 309 658
Data Read Access	580 863 728	2 050 903 316	9 362 393
Data Write Access	7 882 819	5 976 087	6 392 733
L1 Miss	38 243 374	105 789 604	135 457
Last-Level Miss	35 279 199	6	1

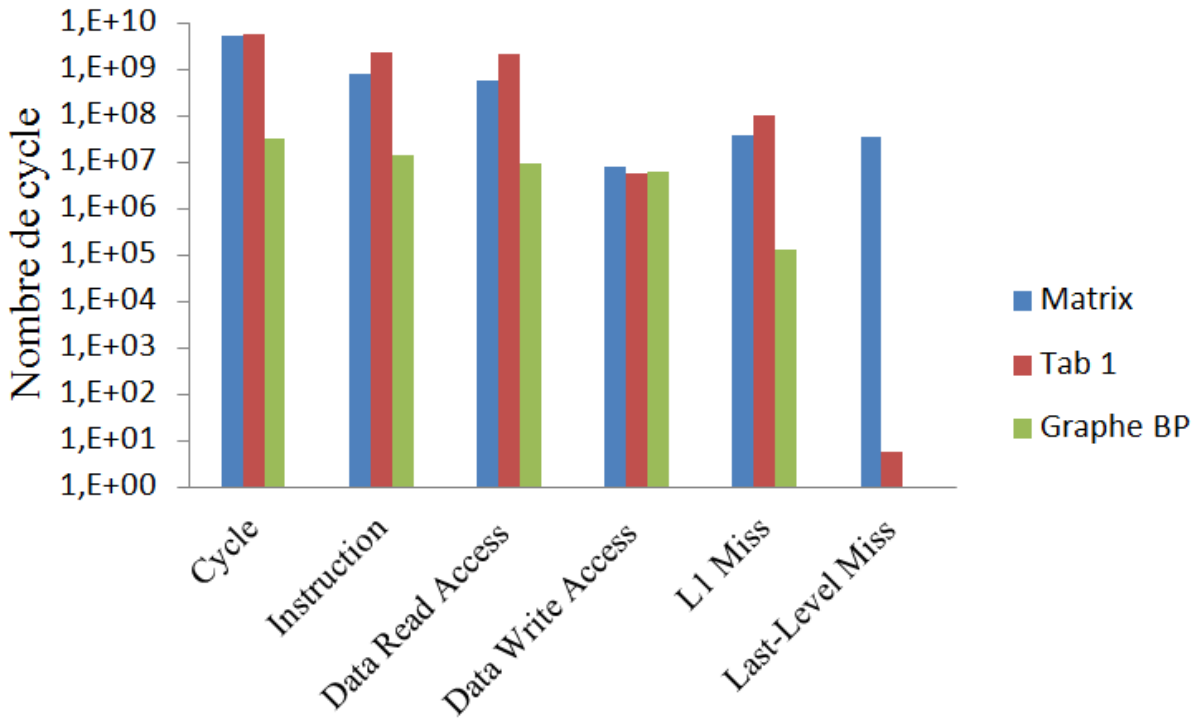


Figure 2. 23 : Histogramme des Résultats obtenues par l'outil Valgrind

4.2.2 Effet des algorithmes de mise à jour et de l'ordonnancement sur le nombre de cycles CPU consommés

De même en utilisant les outils de profilage ''Gprof'' et ''Valgrind cache'', on compare le nombre de cycles nécessaires pour les différents algorithmes de mise à jour des messages C2V, ainsi que les algorithmes d'ordonnancement.

Tableau 2. 5 : Résultats de profilages des algorithmes de décodage LDPC

	BP		MSA		NMSA		SAOMSA	
	Inon	HS	Inon	HS	Inon	HS	Inon	HS
Nombre de cycle par itération (en 10^6 cycles)	5.5	7.5	2.1	3.6	2.2	3.7	2.6	4.1
Nombre cycle de décodage (en 10^6 cycles)	31.4	25.9	14.4	13.7	12.5	11.7	16.7	15.6

Le tableau 2.5, montre le nombre de cycles CPU nécessaire pour une itération et celui nécessaire pour le décodage d'un mot code et ce pour quatre algorithmes de mise à jour des messages C2V : LLR BP, MSA, NMSA et SAOMSA chacun est lancé avec les deux méthodes d'ordonnancement par inondation et brassé HS.

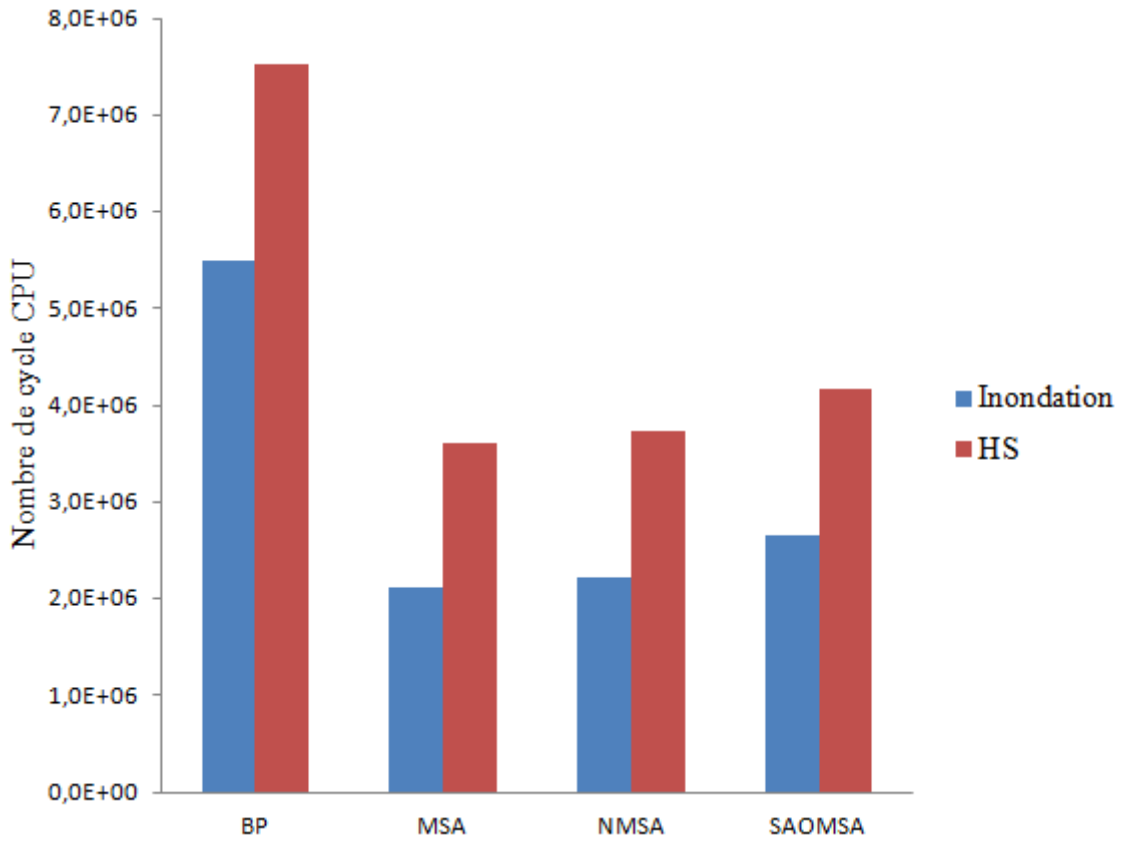


Figure 2. 24 : Histogramme du nombre de cycles CPU par itération des différents algorithmes de décodage LDPC

La figure 2.27 représente les résultats en histogramme, on remarque que le nombre de cycles par itération diminue de 61.5% en passant de l'algorithme LLR BR à l'algorithme MSA en raison de la simplification de l'équation de mise à jour des messages C2V.

Le nombre de cycle par itération de l'algorithme NSMA est légèrement supérieur à celui de l'algorithme MSA (environ 5%) ceci est dû principalement à la multiplication par le coefficient de normalisation α. Le dernier algorithme que nous avons simulé, SAOMSA, consomme plus de cycle CPU par rapport aux deux premiers algorithmes dérivés MSA et NMSA en raison de la fonction d'estimation de l'offset interpolée sur 05 segments. En comparant le nombre de cycle CPU consommés par tout le processus de décodage, on se rend compte que l'algorithme NMSA est le moins complexe, malgré que le nombre de cycles CPU consommés par itération par cet algorithme est supérieur à celui consommé par itération par l'algorithme MSA, et ce, comme déjà constaté lors de la comparaison de nombre d'itérations consommées par chaque algorithme, l'algorithme NMSA consomme moins d'itérations pour converger vers un mot code validé que l'algorithme MSA.

La comparaison de nombre de cycles par itération pour les deux méthodes d'ordonnancement montre que l'ordonnancement brassé HS consomme plus de cycle CPU par rapport à la méthode d'ordonnancement par inondation, cela est justifié par le fait que le nombre de mise à jour des messages V2C passe de E mise à jour à (d_v x E) mise à jour.

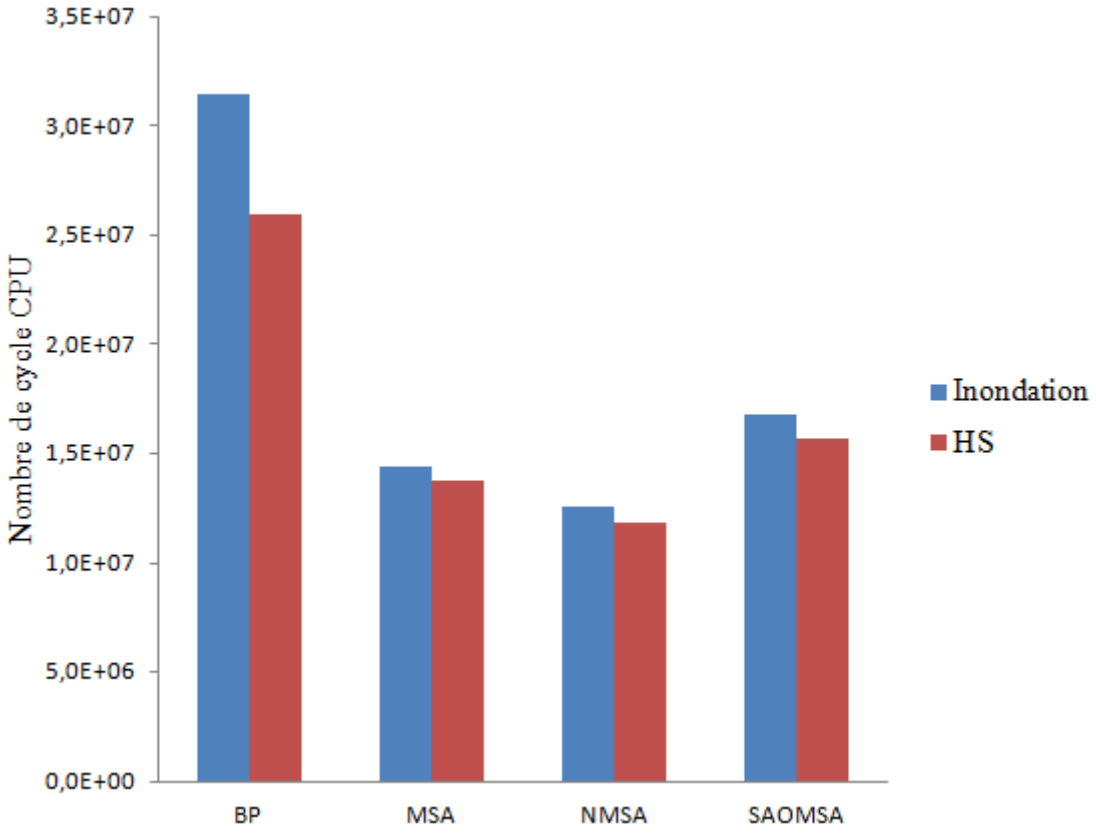


Figure 2. 25 : Histogramme du nombre de cycles CPU de décodage des différents algorithmes d'ordonnancement

De même, le nombre de cycle nécessaire pour le décodage d'un mot code avec l'ordonnancement brassé est inférieur à celui nécessaire pour le décodage d'un mot code avec l'ordonnancement par inondation, malgré qu'une seule itération de l'ordonnancement brassé HS nécessite plus de cycle CPU que l'ordonnancement par inondation, cela est expliqué par le fait que l'ordonnancement brassé HS converge plus rapidement que l'ordonnancement par inondation et par conséquence consomme moins d'itérations pour converger vers un mot code valide (Figure 2.25).

D'après les résultats de simulation (figure 2.19) et de profilage (figure 2.25), l'algorithme NMSA offre la simplification, de la fonction de mise à jour des messages C2V, la moins coûteuse en terme de cycles CPU et la plus proches de l'algorithme de base LLR BP en terme

de pouvoir de correction d'erreur. De ce fait, l'algorithme NMSA sera adopté dans le cadre des travaux de cette thèse.

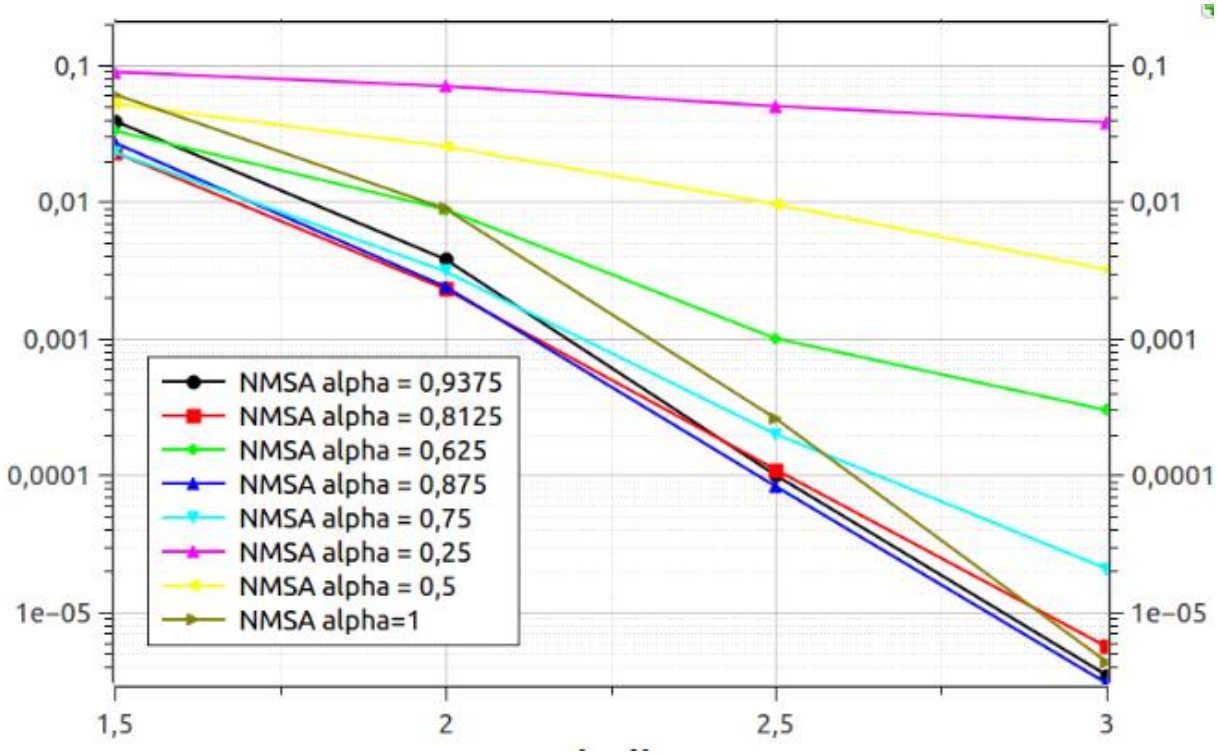


Figure 2. 26 : BER de l'algorithme NMSA en fonction de diverses valeurs α

Pour déterminer la valeur du facteur de normalisation α, qui donne les meilleures performances en terme de correction d'erreur, des simulations ont été lancés avec plusieurs valeurs α afin d'adopter la valeur qui offre les meilleures performances en termes de BER. La figure 2.26, montre les courbes BER des valeurs α simulées avec l'ordonnancement brassé horizontalement.

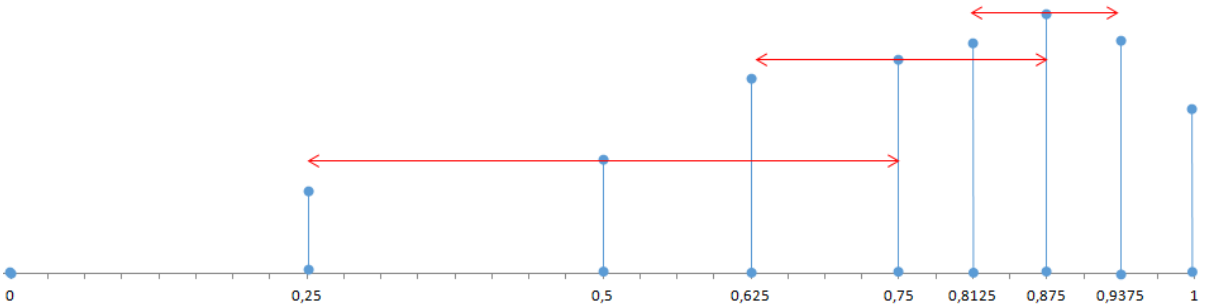


Figure 2. 27 : Méthode de dichotomie appliquée sur l'intervalle [0 1] pour déterminer la valeur optimale de α

Etant donné que le facteur de normalisation α est défini sur l'intervalle [0 1], nous avons identifié la valeur optimale en utilisant une sorte de dichotomie sur l'intervalle [0 1] ; La valeur α = 1 correspond à l'algorithme MSA. Comme présenté sur la figure 2.27, on commence par α = 0.5 le milieu de l'intervalle [0 1] (courbe jaune), puisque les performances de l'algorithme MSA (α = 1) sont meilleures que celles de α = 0.5, on évalue le centre de l'intervalle [0.5 1], on trouve que le BER est meilleur pour α = 0.75 (courbe bleu cyan), on passe alors au centre de l'intervalle [0.75 1], on trouve que le BER est meilleur pour α = 0.875 (courbe bleu), ensuit on passe au centre de l'intervalle [0.75 0.875], on trouve que la courbe BER de la valeur α = 0.8125 (courbe rouge) est moins performante que celle de la valeur α = 0.875. Par conséquence on s'arrête à cette précision (trois chiffres après la virgule) et on adopte la valeur α = 0.875 pour l'algorithme NMSA utilisé dans le cadre de cette thèse.

Conclusion :

D'après les résultats de simulation, d'études de complexité et de profilage on déduit ce qui suit :

- Une méthode d'ordonnancement qui converge rapidement peut améliorer la latence de décodage, malgré l'ajout de complexité par itération.
- L'algorithme NMSA offre la simplification, de la fonction de mise à jour des messages C2V, la moins coûteuse en terme de cycles CPU et la plus proches de l'algorithme de base LLR BP en terme de pouvoir de correction d'erreur.
- La manière de structurer les données dans la mémoire (matrice de parité H, les messages C2V, V2C et LLR), influence directement sur la latence du décodage.

Ainsi, Avant de passer à l'implémentation du décodeur LDPC sur des systèmes embarqués, nous avons proposé dans le cadre de cette thèse des améliorations et des optimisations qui concernent principalement la méthode d'ordonnancement, la structuration des données et l'optimisation de l'algorithme NMSA.

Chapitre III : Proposition d'un nouvel algorithme d'ordonnancement à faible complexité

Dans ce chapitre, nous proposons une nouvelle méthode d'ordonnancement à faible complexité, appelé Layered Vicinal Variable Nodes Scheduling LWNS. L'objectif de cette méthode est d'améliorer les performances de convergence et de correction d'erreur des méthodes d'ordonnancements dynamiques, mais avec un ordre de mise à jour des nœuds fixe déterminé au préalable en dehors du processus, et ce, afin de réduire la complexité de décodage.

La stratégie d'ordonnancement proposée est établie par une étape de prétraitement de la matrice de contrôle de parité H par conséquent n'aura pas d'impact sur la complexité du décodage. Au cours du décodage, chaque nœud de variable est mis à jour en échangeant des messages de vraisemblance avec tous les nœuds de contrôles et de variables avec lesquels il est relié sur le tout le graphe de Tanner avant de passer à la mise à jour du nœud variable suivant.

1 Ordonnancement par nœuds vicinaux

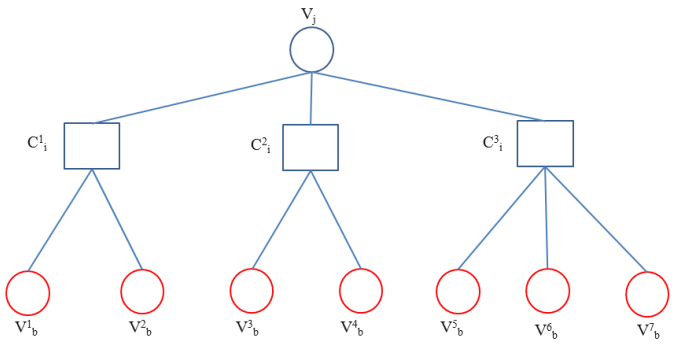


Figure 3. 1 : Sous-graphe du nœud de variable vj

La méthode d'ordonnancement proposé LWNS, est basée sur la mise à jour de chaque nœud variable v_j en échangeant les messages V2C et C2V d'une manière rapprochée avec tous les nœuds d'un sous graphe formé par ce nœud de variable v_j, et ce, avant de passer à la mise à

jour du nœud de variable suivant. Le sous graphe formé par un nœud de variable v_j contient les nœuds de contrôle c^k_i reliés avec le nœud v_j et les nœuds de variables v^k_b connectés à ces nœuds de contrôle c^k_i. Cet ensemble de nœuds variables est appelé nœuds de variables vicinaux de v_j et noté $v(v_j)$.

La figure 3.1, montre le sous-graphe sous lequel la mise à jour du nœud de variable v_j est effectuée; le but est d'éviter les cycles d'ordre 6, 8 et plus ; En plus, cela permet de traiter chaque nœud variable v_j, étroitement sous son sous-graphe avant de partager les mises à jour avec tout le graphe, de cette façon les dernières informations mises à jour directement liées à v_j sont utilisées pendant sa mise à jour ; permettant ainsi une convergence rapide du décodage. Autrement dit, un nœud de variable est mis à jour en échangeant les messages C2V et V2C entre les nœuds de variables appartenant aux équations de contrôle de parité auxquelles il participe avant de passer au nœud suivant.

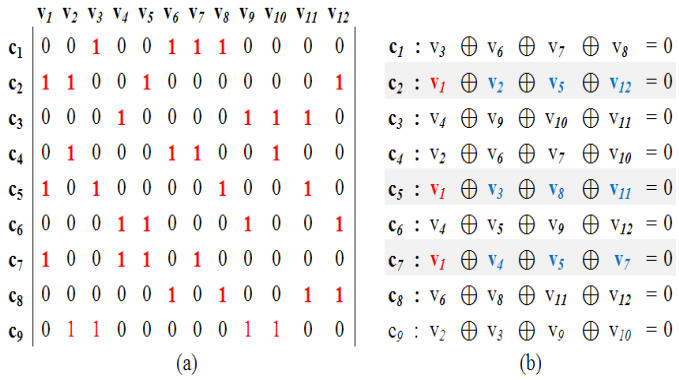


Figure 3. 2 Exemple des nœuds vacinaux du nœud de variable v_1 mis en évidence en utilisant les équations de parité

En prenant l'exemple de la matrice de contrôle de parité de la figure 3.2 (a) et de ses équations de contrôle de parité (b). Le nœud variable v_1 est mis à jour en échangeant les messages V2C et C2V avec les nœuds de variable vicinaux $v(v1) = \{v_2, v_5, v_{12}, v_3, v_8, v_{11}, v_4, v_7\}$, via les équations de contrôle auxquelles il appartient c_2, c_5 et c_7.

Les sous graphes ainsi définis, ne pourront pas contenir de cycle d'ordre 6 ou plus, ce qui permet d'éviter que les bits oscillent entre les valeurs 0 et 1 d'une itération de décodage à une autre sans recours aux calculs des métriques utilisées dans l'ordonnancement dynamique.

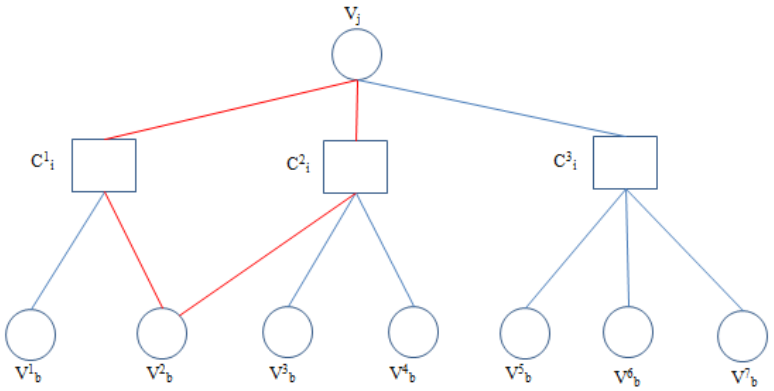


Figure 3. 3 : Cycle d'ordre 4 entre le nœud de variable vj et ces nœuds vicinaux

Cependant, un cycle court d'ordre 4 peut avoir lieu lorsqu'un nœud variable v_j est connecté à un nœud vicinal v_2^b par deux chemins différents à travers deux nœuds de contrôle distincts c^1_i et c^2_i, ce cas est illustré sur la figure 3.3.

Pour avoir des sous-graphes sans cycle d'ordre 4, un parcours de la matrice de parité H est réalisé afin de détecter et d'éliminer les cycles d'ordre 4.

2 Détection et élimination des cycles d'ordre 4 :

Un cycle d'ordre 4 est assimilé sur la matrice de parité H par un rectangle composé de deux nœuds de variables appartenant à une même ligne et leur projection sur une autre ligne, comme illustré sur la *figure 3.4.*

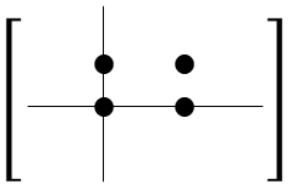


Figure 3. 4 : Représentation d'un cycle d'ordre 4 dans la matrice H

La présence de cycle d'ordre 4, ne permet pas au processus de décodage de prendre une décision sur les valeurs binaires des deux nœuds de variables participant au cycle, et ce, parce que chacune de ces deux variables admet deux solutions valides et par conséquent les valeurs binaires de ces deux nœuds de variable oscillent entre les valeurs 0 et 1.

L'algorithme 1, présente la méthode de détection de cycle d'ordre 4 qui se fait moyennant des parcours emboités de la matrice H, en se fixant sur deux nœuds de variable appartenant à une ligne, on vérifie si ces deux nœuds de variables appartiennent à une autre ligne, le cas échant

les indexe (ligne, colonne) de chaque élément participant à ce cycle sont reportés sur un fichier ‘’.txt’’

Algorithm 1 : Pseudocode de Détection des cycles d’ordre 4 dans une matrice de contrôle de parité H de taille (M, N):

```
1: for i = 1 to N do
2:    for j = i+1 to N do
3:        Initialiser à zéro le compteur, Cpt, qui confirme la présence de cycle d’ordre 4
4:        for l = 1 to M do
5:                calculer T = H[l][i] + H[l][j]
6:              if (T = 2 && Cpt = 0)
7:                      Incrémenter Cpt
8:                      Marquer la 1ère ligne l1 qui participe au cycle
9 :               end if line 6
10 :              if (T = 2 && Cpt = 1)
11:                     Incrémenter Cpt
12 :                    Marquer la 2ème ligne l2 qui participe au cycle
13 :              end if line 10
10 :              if (Cpt >= 2)
11:                     présence de cycle dans les nœuds (i, l1), (j, l1) , (i, l2), (j, l2)
12:             end if line 10
13:       end for line 4
14:  end for line 2
15:end for line 1
```

En cas de présence de cycle d’ordre 4, leur élimination se fait en éliminant en premier lieu l’élément non nul qui participe au maximum de nombres de cycles, et ce, afin de minimiser le nombre d’éléments non nul supprimés.

Aussi, plus un nœud de variable participe à un grand nombre d’équations de parité, plus d’information on aura pour corriger les erreurs reliées à ce nœud, par conséquent la valeur d_v est aussi à prendre en compte. Un nœud de variable qui ne participe à aucune équation de parité n’a aucune chance d’être corrigé par décodage. Ainsi l’élimination est réalisée pour les éléments non nuls de valeur d_v maximale et qui restera au moins égale à 2 après élimination

Aussi il faut prendre en compte que les lignes de la matrice H constituent une base du sous espace vectoriel F_2^{N-K} et par conséquence, lors de l’élimination des éléments non nuls, les lignes de la matrice doivent rester linéairement indépendantes entre elles, pour les matrices qui s’écrivent sous la forme [A B] avec B une sous matrice carrée diagonale inférieure ou supérieur. Cette condition est satisfaite en laissant intacts les éléments de la diagonale de la sous matrice B.

Exemple :

Soit une matrice H :

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

L'application de l'algorithme de détection de cycle d'ordre 4, permet de détecter six cycles d'ordre 4 et renvoie les index des éléments participant à ces cycles, la figure 3.5, illustre les différents cycles en utilisant des rectangles colorés :

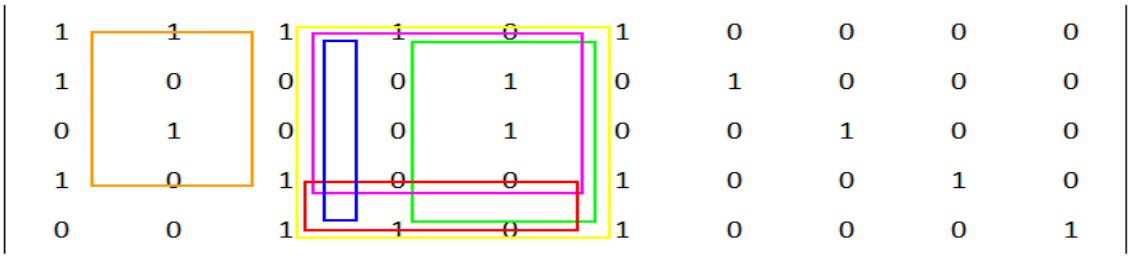


Figure 3. 5 : Cycles d'ordre 4 dans la matrice (5, 10)

Le tableau 3.1, représente les index des nœuds participant aux cycles d'ordre 4 au sein de la matrice (5, 10) détectés par l'algorithme 1 :

Tableau 3. 1 : index des nœuds participants aux cycles d'ordre 4 de la matrice (5, 10)

	Elément 1	Elément 2	Elément 3	Elément 4
Cycle 1 (orange)	(0, 0)	(0, 2)	(3, 0)	(3, 2)
Cycle 2 (jaune)	(0, 2)	(0, 5)	(4, 2)	(4, 5)
Cycle 3 (rose)	(0, 2)	(0, 5)	(3, 2)	(3, 5)
Cycle 4 (vert)	(0, 3)	(0, 5)	(4, 3)	(4, 5)
Cycle 5 (Bleu)	(0, 2)	(0, 3)	(4, 2)	(4, 3)
Cycle 6 (rouge)	(3, 2)	(3, 5)	(4, 2)	(4, 5)

Afin de minimiser le nombre d'éléments non nuls à éliminer, on calcule, pour chaque élément du tableau 3.1, le nombre total des cycles d'ordre 4 auxquels participe chaque élément, ainsi que la valeur d_v de chaque nœud. Le tableau 3.2 récapitule ces valeurs triées en ordre décroissant en terme de nombre de cycle et de valeur d_v, l'élément (0, 5) ne sera pas

représenté malgré qu'il participe à trois cycle d'ordre 4, et ce, car ce nœud est un élément de la diagonale de la sous matrice B :

Tableau 3. 2 : Nombre de cycle et degré de chaque nœud participant aux cycles d'ordre 4 dans la matrice (5, 10)

Elément	Nombre de cycle	d_v
(0, 2)	4	3
(3, 2)	3	3
(4, 2)	3	3
(4, 5)	3	3
(3, 5)	2	3
(0, 0)	1	3
(3, 0)	1	3
(0, 3)	1	2
(4, 3)	1	2

Le remplacement de la valeur 1 par la valeur 0 de l'élément (0, 2) permet d'éliminer quatre cycle d'ordre 4 (les cycles orange, Jaune, rose et Blue).

En répétant la même méthode pour les deux autres cycles on trouve que le remplacement de la valeur 1 par la valeur 0 de l'élément (4, 5) permet de les éliminer tous les deux.

Ainsi la matrice sans cycle d'ordre 4 obtenue est comme suit :

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3 Nœud de variable le plus influent :

Chaque nœud de variable génère son propre sous-graphe et par conséquent le nombre de sous-graphes est égal à N. Pour déterminer l'ordre de mise à jour des sous-graphes, nous classons les nœuds de variable en utilisant le concept du nœud de variables le plus influent. En effet, plus un nœud de variable est relié à un grand nombre de nœuds de variable et de contrôle, plus il est susceptible d'être plus influent.

La métrique ainsi définie correspond au nombre d'arcs de chaque sous-graphe, et sera calculée pour chaque nœud de variable v_j comme suit :

$$D(v_j) = dv(v_j) + \sum_{i=1}^{dv} (dc(c_i) - 1) \tag{3.1}$$

$dv(v_j)$ désigne le nombre total de nœuds de contrôle connectés au nœud de variable v_j, et $d_c(c_i)$ désigne le nombre total de noeuds de variables connectés aux nœuds de contrôle c_i. Le deuxième terme de l'équation de calcul correspond au nombre de nœuds de variable vicinaux du nœud de variable v_j. Le calcul de cette métrique est réalisé durant la phase de prétraitement de la matrice de parité H en dehors du processus de décodage. La mise à jour des nœuds de variable est réalisée dans l'ordre décroissant de la métrique d'influence.

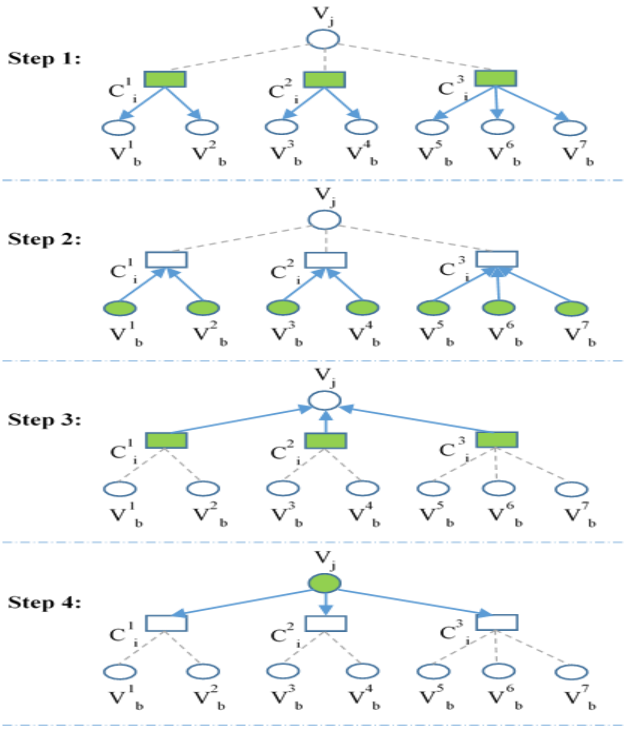


Figure 3. 6 : Etapes de l'algorithme proposé LWNS

La figure 3.6, montre le processus d'échange des messages C2V et V2C de chaque nœud de variable au sein de son sous-graphe. Dans la première étape, en utilisant l'équation (1.28), les messages $m_{ci \to vb}$ des nœuds de contrôle c_i connectés au nœud de variable v_j sont mis à jour en prenant en compte les messages $m_{vj \to ci}$ et sont propagés ensuite à tous les nœuds vicinaux $v_b \in v(v_j)$. Dans la deuxième étape, à l'aide de l'équation (1.29), les valeurs LLR des nœuds vicinaux v_b ainsi que les message V2C, $m_{vb \to ci}$, sont mis à jour et sont propagés vers les

nœuds de contrôle c_i. Dans ces deux premières étapes, les nœuds vicinaux sont mis à jour en tenant compte uniquement de la valeur LLR du nœud variable v_j.

Ensuite, à l'étape 3, en utilisant l'équation (1.28), les messages $m_{ci\rightarrow vj}$ sont générés en prenant en compte les nœuds vicinaux $v(v_j)$, et propagés au nœud de variables v_j. Enfin, les valeurs LLR des nœuds de variables v_j sont calculées à l'aide de l'équation (1.30). De cette façon, le nœud de variables v_j est mis à jour en tenant compte des valeurs LLR de ses nœuds vicinaux $v(v_j)$.

Le processus de décodage détaillé de l'algorithme LWNS avec un pseudocode est décrit dans l'algorithme 2.

Algorithme 2 : Pseudocode de l'ordonnancement proposé LWNS

1: Initialize all $m_{ci\rightarrow vj} = 0$, $m_{vj\rightarrow ci} = L(0)_{v_j}$ and Itermax
2: for i = 1 to Itermax do
3: for every v_j ordered based on the influence metric do
4: for every ci $\in N(vj)$ do
5: for every vb $\in N(ci)\backslash vj$ do
5: Generate and Propagate $m_{ci\rightarrow vb}$
6: end for line 5
7: end for line 4
8: for every vb $\in N(ci)\backslash vj$ do
7: Calculate $L(v_b)$
8: Generate and Propagate $m_{vb\rightarrow ci}$
9: enf for line 8
10: Calculate $L(v_j)$
11: end for line 3
12: if stopping rule is not satisfied then
13: Go back to line 3
14: end if
15:end for line 2

En outre, l'algorithme LWNS proposé peut fonctionner en parallèle, en mettant à jour les nœuds de variables par groupe (ou par couche ou layer). Chaque groupe contient l'ensemble des nœuds de variable, indépendants, qui ne partagent pas de nœuds vicinaux, et ce pour éviter le conflit d'accès mémoire à la même donnée lors du traitement parallèle des sous graphes. On dit que k nœuds de variable sont indépendants si et seulement si :

$$\bigcap_{k\ element} \boldsymbol{v}(\mathbf{v_{element}}) = \emptyset \tag{3.2}$$

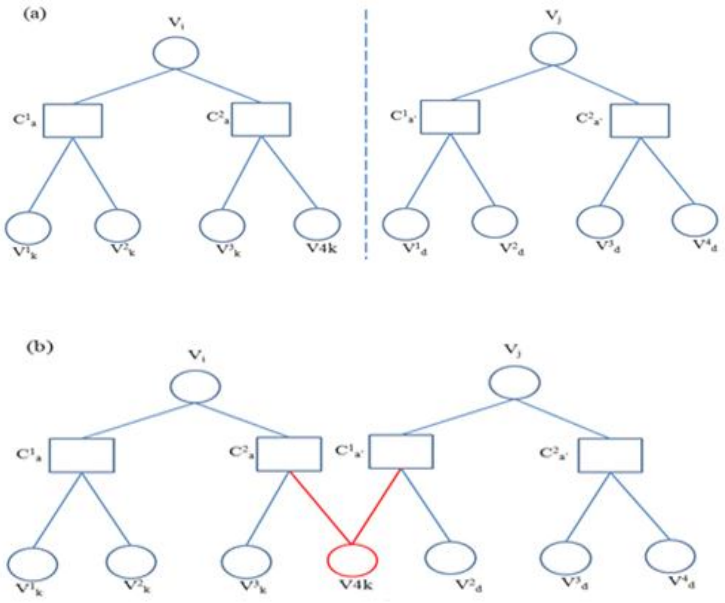


Figure 3. 7 : Ordonnancement parallèle en couches (a) les nœuds de variables v_j and v_i sont parallèles $v(v_i) \cap v(v_j) = \emptyset$. (b) les nœuds de variable v_j and v_i ne sont pas parallèles $v(v_i) \cap v(v_j) = v_k^4$.

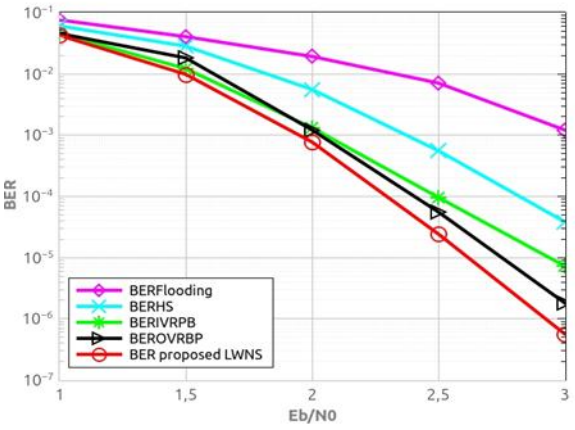
La figure 3.7, illustre les deux cas de figures deux nœuds de variable v_i et v_j dont les sous graphe sont complétement indépendants $\boldsymbol{v}(\mathbf{v_i}) \cap \boldsymbol{v}(\mathbf{v_j}) = \emptyset$ et deux nœuds de variable dépendants v_i et v_j dont les sous graphes partagent un même nœud $\boldsymbol{v}(\mathbf{v_i}) \cap \boldsymbol{v}(\mathbf{v_j}) = \mathbf{v_k^4}$.

Les nœuds de variables respectant l'équation (3.2) sont regroupés par couche, chaque couche sera lancée en parallèle, le passage d'une couche à une autre est séquentiel.

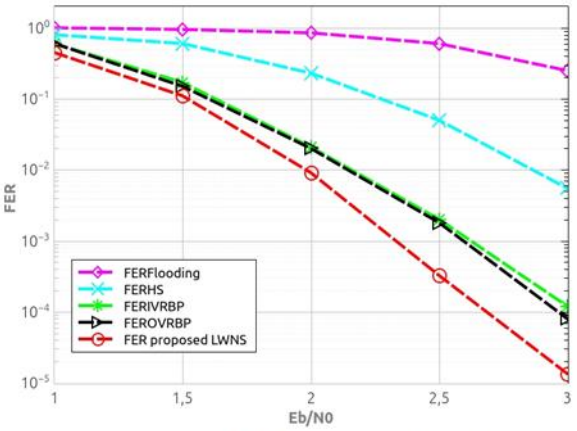
4 Evaluation des performances de l'algorithme proposé

4.1 Performances en correction d'erreurs

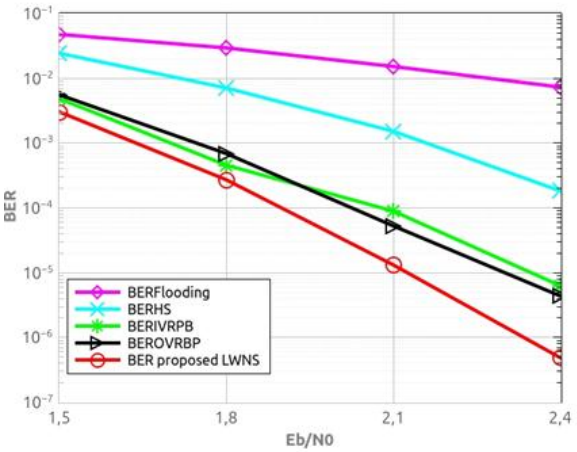
Afin d'évaluer les performances de décodage de l'algorithme LWNS proposé, une comparaison avec les algorithmes existants dans la littérature est effectuée, notamment les algorithmes basés sur un ordonnancement séquentiel par inondation et brassé HS, et les algorithmes basés sur un ordonnancement dynamique IVC-RBP et VO-RBP. Le nombre d'itération maximales est fixé à 5 itérations.



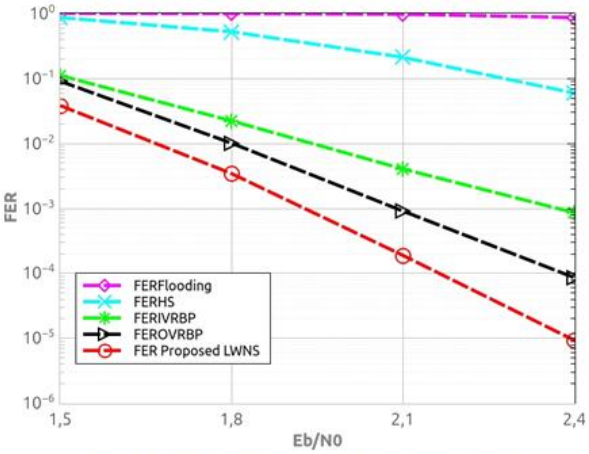
(a) : BER LDPC code (576, 288)



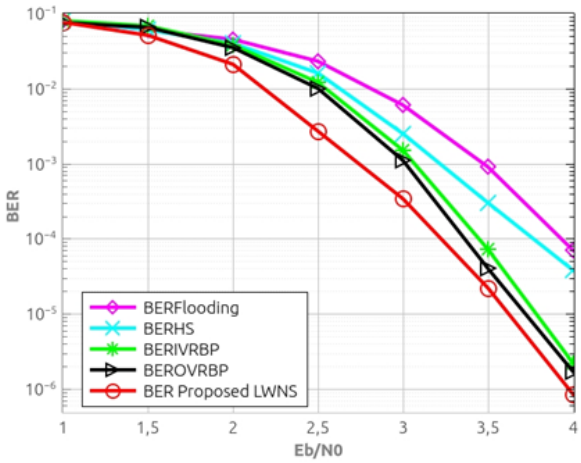
(b) : FER LDPC code (576, 288)



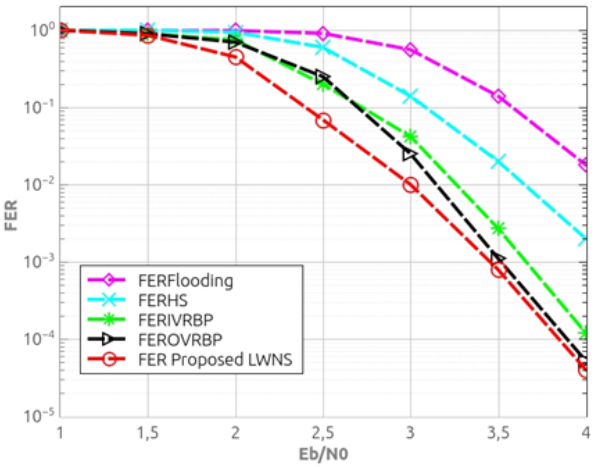
(c) : BER LDPC code (1152, 576)



(d) : FER LDPC code (1152, 576)



(e) : BER LDPC code (576,144)



(f) : FER LDPC code (576,144)

Figure 3. 8 : BER et FER vs Eb/N0 des algorithmes de décodage: inondation, Brassé HS, IVRBP, OVRBP et LWNS pour les codes LDPC irrégulier (576, 288), (1152, 576) et (576, 144).

Les figures 3.8 (a), (b), (c) et (d), montrent les performances en terme de BER et de FER des algorithmes de décodage pour les codes LDPC de taux de codage R=1/2 et de longueurs de bloc respectives de 576 et 1152. Les performances de correction d'erreurs de l'algorithme proposé sont meilleures que celles des algorithmes de la littérature. Comme on peut le constater, l'avantage en termes de performances est préservé lorsque le rapport SNR augmente pour atteindre 5.6×10^{-7} lorsque le SNR est à 3dB pour les mots code de taille 576 et 4.8×10^{-7} à un SNR de 2,4dB pour les mots code de taille 1152.

Les figures 3.8 (e) et (f), représentent respectivement des performances en terme de BER et FER du code LDPC irrégulier de taille 576 avec un rendement R=3/4, l'algorithme LWNS proposé montre des performances comparées à l'algorithme IVRBP et un gain de 0.25 dB par rapport à l'algorithme OVRBP entre le SNR 2dB et 3.5dB. Avec un SNR de 4dB le BER atteint 8.10^{-7}. La complexité de ces différents algorithmes sera discutée dans le paragraphe suivant.

Tableau 3. 3 : Comparaison des performances de l'algorithme proposé LWNS et des algorithmes de la littérature :

Algorithmes	Taille du code	Rate	SNR [dB]	I_{avg}	FER	BER
OVRBP	576	0.5	3	1.3	8×10^{-5}	1.810^{-6}
LWNS	576	0.5	3	1.3	1.3×10^{-5}	5.6×10^{-7}
OVRBP	576	0.75	4	1	5.2×10^{-5}	1.7×10^{-6}
LWNS	576	0.75	4	1	4.1×10^{-5}	8.5×10^{-7}
OVRBP	1152	0.5	2.4	2	5.5×10^{-5}	4.3×10^{-6}
LWNS	1152	0.5	2.4	1.9	9.1×10^{-6}	4.8×10^{-7}

Le tableau 3.3, présente une comparaison entre les algorithmes OVRBP en termes de BER, FER et le nombre moyen d'itérations consommées lors du processus de décodage $Iter_{moy}$. On peut constater qu'avec avec le même nombre d'itérations de décodage les performances en terme de BER et FER de l'algorithme LWNS proposés sont meilleures que celles de l'algorithme dynamique OVRBP.

Pour évaluer les performances de l'algorithme LWNS proposé en fonction de la taille du mot code, les performances BER et FER pour les longueurs de bloc 576, 1152 et 2304 sont présentées sur la figure 3.9.

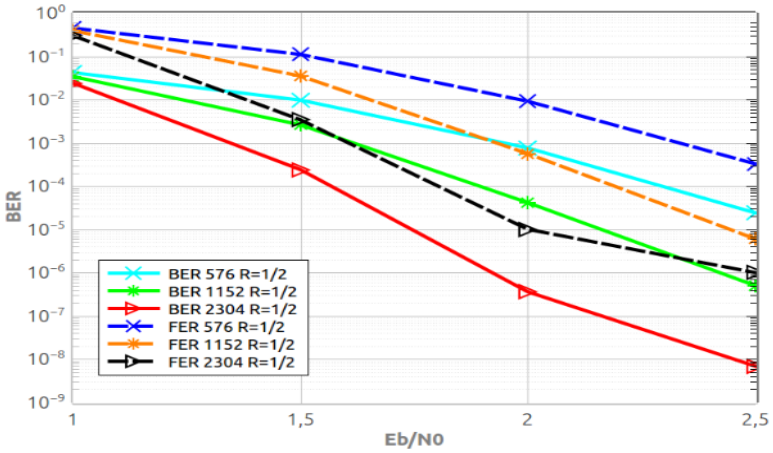


Figure 3. 9 : BER vs Eb/N0 de l'algorithme LWNS proposé pour différents taille de code (576, 288), (1152, 576) et (2304, 1152)

On peut clairement observer que les performances de l'algorithme de décodage LWNS sont de plus en plus importantes lorsque la taille du mot code augmente, le BER et le FER atteignent respectivement $6{,}9\times10^{-9}$ et 10^{-6} à un SNR de 2,4dB, autrement dit pour 138 Mbits envoyés un seul bit sera erroné.

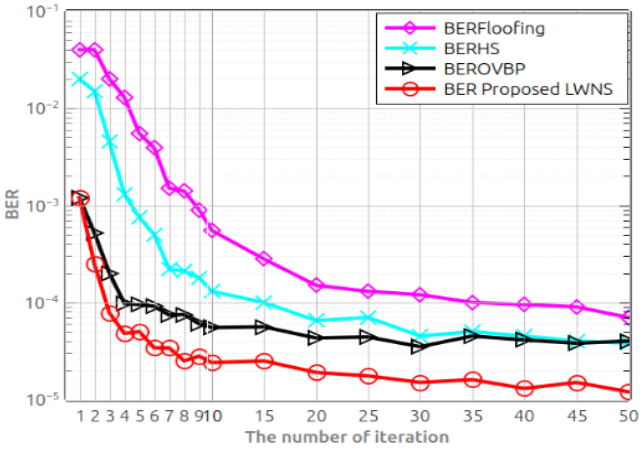


Figure 3. 10: BER vs le nombre maximal itérations pour les différentes méthodes d'ordonnancement du LDPC (576, 288) avec Eb/N0 = 2.5dB.

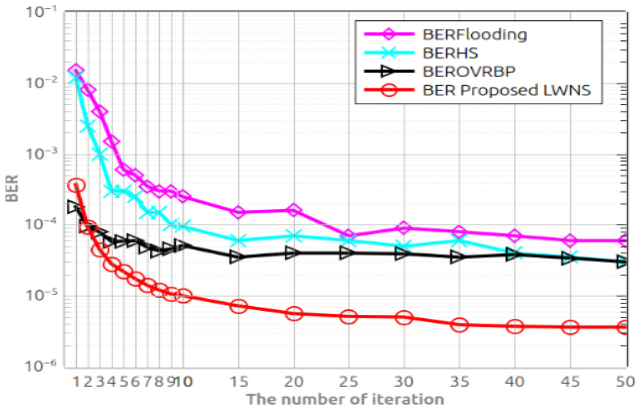


Figure 3. 11: BER vs le nombre maximal itérations pour les différentes méthodes d'ordonnancement du LDPC (576, 144) avec Eb/N0 = 3.5dB.

Les figures 3.10 et 3.11 montrent les performances en termes de BER en fonction du nombre d'itérations pour les codes LDPC (576, 288) à un SNR de 2,5 dB, et les codes LDPC (576, 144) à un SNR de 3,5 dB. Les résultats montrent que l'algorithme LWSN proposé converge plus rapidement que tous les autres algorithmes en 15 itérations. En d'autres termes, le LWSN présente non seulement de meilleures performances de correction d'erreurs mais aussi une vitesse de convergence plus rapide.

L'influence des cycles d'ordre 6 et 8 sur l'algorithme dynamique OVRBP est montrée dans les figures 9 et 10, les performances de correction d'erreur après les cinq premières itérations sont faibles. En revanche, le BER de l'algorithme LWNS augmente avec le nombre d'itérations, pour atteindre une valeur stable après 15 itérations, 3.10^{-6} à SNR de 3.5dB, au lieu de 3.10^{-5} pour l'algorithme OVRB.

4.2 Analyse de la complexité :

Dans ce paragraphe, nous évaluons la complexité de décodage de l'algorithme d'ordonnancement proposé en termes de nombre de mises à jour C2V et V2C requises pour chaque itération de décodage, où une itération signifie le processus de sélection et de mise à jour de tous les arcs du graphe de Tanner. Le nombre total d'arcs dans l'ensemble du graphe de Tanner correspond au nombre d'éléments non nuls de la matrice H : $E = dc \times M = dv \times N$, où d_c et d_v désignent respectivement le degré moyen des nœuds de contrôle et des nœuds variables.

Tableau 3. 4 Nombre de mise à jour des messages C2V et V2C de l'algorithme proposé

Etapes	Calcul pour la recherche du nœud à mettre à jour	Calcul pour la mise à jour V2C	Calcul pour la mise à jour C2V
Etape 1	0	0	dv. (dc - 1)
Etape 2	0	dv. (dc - 1)	0
Etape 3	0	0	dv
Etape 4	0	dv	0

Le tableau 3.4, présente la complexité en termes de nombre de mise à jour des messages V2C et C2V, nécessaires pour un seul sous graphe.

Pour chaque nœud de variable v_j, $d_v.(d_c - 1)$ messages C2V sont calculés et propagés vers les nœuds vicinaux. Ensuite, ces nœuds vicinaux calculent $d_v.(d_c - 1)$ messages V2C pour mettre à jour les nœuds de contrôle du sous graphe. Dans la 3ème étape, les nœuds de contrôle génèrent et propagent d_v C2V vers le noeud de variable v_j. Finalement lors de la 4ème l'étape, la valeur LLR du noeud variable v_j est mise à jour et d_v messages V2C sont propagés aux nœuds de contrôle connectés à v_j.

Par conséquent, pour chaque itération de l'algorithme LWSN, le nombre de mise à jour des messages C2V et V2C requis pour les N sous graphe est de $\{N \times [dv + dv \times (dc - 1)]\} = E \times dc$.

Tableau 3. 5 : Comparaison des complexités des différents algorithmes existants dans la littérature et celui l'algorithme proposé

Algorithme	Calcul pour la recherche du nœud à mettre à jour	Calcul pour la mise à jour V2C	Calcul pour la mise à jour C2V
Flooding	0	E	E
HS	0	E.dv	E
IVRBP	M(M.dc - 1)	E[(dv-1)+(dc-1)]	E(dv-1)(dc-1)
OVRBP	N(N-1)	E	E.dc
Algorithme proposé LWNS	0	E.dc	E.dc

Le tableau 3.5, présente la complexité de mise à jour des différents algorithmes de décodage. L'algorithme LWSN remplace par une étape de prétraitement non incluse dans le traitement de décodage.

La complexité des algorithmes IVRBP et OVRBP est quadratique $O(M^2)$ et $O(N^2)$ respectivement, et ce, en raison des calculs des valeurs résiduelles et des calculs des métriques nécessaires pour la recherche du nœud de variable le moins fiable, alors que la complexité de l'algorithme LWNS est d'environ O(E) car l'ordre de mise à jour des nœuds de variable est déterminé par une étape de prétraitement de la matrice H non incluse dans le processus de décodage.

Conclusion

En conclusion, les résultats de simulation et l'analyse de complexité montrent que l'algorithme proposé permet d'améliorer la vitesse de convergence et la complexité de décodage tout en assurant les mêmes performances en termes de correction d'erreurs sque les autres algorithmes rapportés dans la littérature.

Chapitre IV : Implémentation des décodeurs LDPC sur des plateformes embarquées multi-cœurs

Dans ce chapitre, nous proposons une nouvelle optimisation de l'algorithme de décodage NMSA qui permettra de réduire la latence de décodage. Dans un premier temps, en se basant sur les analyses des différents blocs du décodeur en tenant en compte de la complexité des calculs, de parcours et d'accès en mode lecture et écriture à la mémoire, nous proposons une optimisation visant à minimiser les nombres d'accès à la mémoire et une structuration de données compacte assurant une réduction de la mémoire requise et une implémentation parallèle minimisant la synchronisation entre les différents processeurs. Dans un deuxième temps, nous présentons l'architecture parallèle proposée du décodeur LDPC basé sur l'algorithme NMSA et nous détaillons l'architecture de l'implémentation logicielle parallèle du décodeur LDPC proposée sur des plateformes monocœur et multicœurs basé sur des processeur x86 d'Intel et multicoeurs basés sur des processeurs ARM embarqués afin d'évaluer la latence en ms du décodeur proposée.

1 Revue des implémentations des décodeurs LDPC :

Plusieurs travaux scientifiques se sont orientés vers une implémentation matérielle des décodeurs LDPC sous circuits intégrés spécifiques ou de circuits FPGA [48-51]. Cette approche permet d'obtenir une faible latence et un débit élevé, mais elle entraîne en contrepartie un coût de développement élevé, ce qui constitue une limitation pour les applications nécessitant une mise sur le marché rapide ou pour les technologies avec des normes multiples et en évolution rapide.

Une deuxième approche est basée sur l'implémentation logicielle du décodeur LDPC, cette approche réduit considérablement les ressources matérielles et le temps de développement nécessaires pour la mise sur le marché. Elle peut partager le même matériel utilisé par

d’autres algorithmes de traitement du signal et elle offre la possibilité de modifier ou de mettre à jour une version logicielle sans modifier le matériel.

Cette solution s’avère la mieux adapteé pour les communications ultra-fiables à faible latence (URLLC), dont les normes et les applications sont en développement perpétuel. Cette approche est également soutenue par l'émergence des processeurs multi-cœurs permettant un traitement parallèle de l’optimisation proposée, ce qui permet de s’approcher aux performances des solutions matérielles avec une réduction du coût et une adaptation aux différents contextes d'applications.

Plusieurs travaux de recherche se sont récemment concentrés sur les implémentations logicielles de décodeurs LDPC sur des plateformes multicœurs. Certains travaux manipulent des processeurs à usage général (General Purpose Processors GPP), comme les processeurs x86 d’Intel ou des processeur graphiques, en anglais Graphics Processing Unit (GPU) [52-53], d'autres utilisent l'architecture instruction unique données multiples, en Single Instruction Multiple Data anglais SIMD [54] pour accélérer le calcul et le troisième exploite les plateformes multicœurs sur carte pour profiter des avantages d'accélération matériel et de la flexibilité logicielle [55-56].

Dans [52], la comparaison des stratégies d’exécution parallèle de l'algorithme de décodage LDPC MSA a démontré que le GPU peut réaliser le décodage avec des débits élevés par rapport au processeur général. Cependant, les solutions basées sur les GPU ne fonctionnent bien que pour les codes de grande taille en raison du temps passé pour le transfert de données entre le processeur hôte et le GPU. Pour une longueur de code moyenne ou petite, le temps de transfert des données s'avère être plus élevé que le temps nécessaire au processus de décodage.

Pour que la plate-forme GPU soit adaptée à une petite longueur de code, Amiri et al. dans [54] ont proposé un décodeur LDPC multi-flux sur GPU, utilisant des données multiples à instruction unique pour décoder plusieurs mots code simultanément. Cette approche permet d'atteindre des débits plus élevés sur les appareils mobiles embarqués. Cependant, le temps passé pour le transfert de données entre le processeur hôte et le dispositif GPU, environ 4,8 ms, est encore plus élevé pour les applications à faible latence.

Dans [55], Kharin et al. ont réalisé une implémentation sur un système multicoeurs sur puce (SoC) disposant de 8 DSP- et 4 ARM-cores, cette solution profite de la flexibilité et de l'efficacité du DSP dans les tâches de traitement du signal, mais le temps pour la communication inter-processeur ARM-DSP est évalué à environ 2,75 ms pour le chargement des données du DSP, ce qui augmente la latence du décodage.

Ces travaux se sont concentrés uniquement sur les stratégies de mise en œuvre des décodeurs LDPC sur différentes plates-formes sans optimiser ni l'algorithme de décodage ni la structure de donnée adoptée pour la mémorisation de la matrice et des messages C2V et V2C.

Afin de répondre aux exigences des communications ultra fiables à faible latence, nous proposons dans cette thèse une implémentation logicielle efficace du décodeur LDPC sur des plateformes multicœurs embarqués pouvant être intégrés sur des dispositifs IoT.

Avant d'entamer l'implémentation, une optimisation de la partie du code la plus intensive en terme de consommation de cycle CPU est réalisée avec proposition d'une nouvelle structure de données compacte pour représenter la matrice de contrôle de parité LDPC visant à réduire la taille de la mémoire, les accès en mode lecture/écriture et de permettre un calcul parallèle sur les plateformes multicœurs. L'implémentation parallèle est réalisée sur une plateforme ARM multicoeurs embarqués avec un système d'exploitation embarqué et des outils de multithreading OpenMP afin de minimiser le temps passé dans la création et la synchronisation des threads. L'optimisation proposée a permis une nette réduction de la taille de la mémoire requise, d'environ 50%, la complexité calculatoire est réduite de 66% et la latence atteint 0,15ms pour 288 bits et 0,48ms pour 2304 bits avec un taux d'erreur paquet atteignant 10^{-6}.

2 Architecture proposée :

La mémoire cache joue un rôle très important pour réduire les accès mémoires en dehors du processeur et permet ainsi de réduire le temps de traitement. Autre que sa rapidité en raison de son emplacement près du processeur, les performances de la mémoire cache reposent sur le concept de la localité, nécessaire pour minimiser les accès hors cache. Il existe plusieurs types de localité. Deux d'entre eux sont essentiels pour la mémoire cache :

- La localité temporelle. Il s'agit de l'accès répété par le processeur aux mêmes ressources dans un court laps de temps.
- La localité spatiale. Il s'agit de l'accès par le processeur à diverses données ou ressources qui sont proches les unes des autres en terme d'adresse mémoire.

Pour tirer profit de la localité spatiale, il faut que les données soient consécutives et structurées dans l'ordre de leur utilisation, ainsi les données qui seront utilisées par

l'instruction suivante peuvent être chargées depuis la mémoire RAM et stockées dans la mémoire cache durant l'instruction courante, cela permet au processeur de minimiser le nombre de défauts de cache (Miss cache).

Certes la structuration en forme de graphe des messages C2V, V2C et de la matrice de parité H a permis d'éliminer le problème de défaut de cache au dernier niveau du cache LL (Last level), mais le défaut de cache au premier niveau L1 persiste, ce problème est dû principalement à l'allocation dynamique de la mémoire utilisée lors de la construction du graphe. En effet, les adresses attribuées dynamiquement ne sont pas consécutives, ce qui ne permet pas de tirer profit de la localité spatiale de la mémoire cache et augmente par conséquent le nombre de données non trouvées dans le premier niveau du cache L1 et par conséquent la latence de décodage.

2.1 Structuration des données dans la littérature

Les travaux de la littérature [52-56] ont utilisés une structure de donnée sous forme de deux matrices de valeurs entières pour mémoriser la matrice et deux matrices de valeurs réelles pour mémoriser les messages C2V et V2C. La matrice de contrôle de parité H est mémorisée comme suit :

- La première matrice ''colonne index'', destinée à être utilisée dans la phase horizontale, est de taille M lignes et d_{cmax} colonnes, où d_{cmax} est la valeur maximale des nombres d'éléments non nuls par ligne d_c. Chaque ligne de cette matrice contient les index des colonnes par ligne.
- La deuxième matrice ''ligne index'', destinée à être utilisée dans la phase verticale, est de taille N lignes et d_{vmax} colonnes, où d_{vmax} est la valeur maximale des nombres d'éléments non nuls par colonne d_v. Chaque ligne de cette matrice contient les index des lignes par colonne.

Pour les codes réguliers $d_{cmax} = d_c$ et $d_{vmax} = d_v$. Pour les codes irréguliers les poids des lignes et les poids des colonnes sont différents et par la suite certaines valeurs des deux matrices seront remplies par des valeurs d'index non valides (par exemple -1 ou null).

Exemple :

En prenant l'exemple de la matrice de parité H du code irrégulier (5, 10) ci-dessous :

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

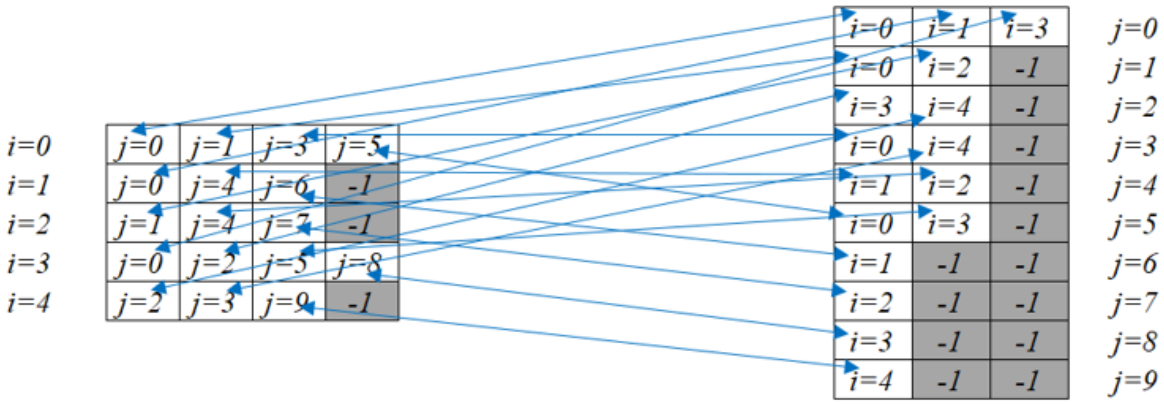


Figure 4. 1 Structuration des données proposée dans l'état de l'art

La figure 4.1, montre une structuration en deux matrices ‘’ligne index’’ et ‘’colonne index’’. De même, les messages C2V et V2C sont mémorisés par deux matrices qui ont la même forme respectivement de la matrice ‘’colonne index’’ et ‘’ligne index’’.

La mémoire nécessaire pour cette structuration est de :

$$memory\ size = Q \times\ sizeof(integer) +\ (Q + 2 \times N) \times\ sizeof(integer) \qquad (4.1)$$

Avec $Q = M \times d_{cmax} + N \times d_{vmax}$. On a $Q >\times 2\,E$ (car $d_{cmax} > d_c$ et $d_{vmax} > d_v$), le tableau du chapitre II (2.1), présente une comparaison entre les valeurs (d_{cmax}, d_{vmax}) et (d_{cmoy}, d_{vmoy}).

Les parcours dans ce cas de structuration de données se fait comme suit :

- **Phase horizontale :** on utilise tout d’abord la matrice ‘’colonne index’’ pour repérer les colonnes *j* connectés à chaque ligne *i* ; ensuite, les valeurs V2C sont lues depuis la matrice V2C[][] aux emplacements (*j*, *i*), ces valeurs V2C sont ensuite utilisées dans le calcul des mises à jour des messages C2V qui sont enregistrées ensuite dans la matrice C2V[][] à l’emplacement (*i*, *j*), ainsi le nombre de parcours est de : 2 x M x d_{cmax}.
- **Phase verticale :** on utilise tout d’abord la matrice ‘’ligne index’’ pour repérer les lignes *i* connectées à chaque colonne *j* ; ensuite, les valeurs C2V sont lues depuis la

matrice C2V[][] aux emplacements (*j*, *i*), ces valeurs C2V sont ensuite utilisées dans le calcul des mises à jour des messages V2C qui sont enregistrés ensuite dans la matrice V2C[][] à l'emplacement (*i*, *j*), ainsi le nombre de parcours est de : 2 x N x d_{vmax}.

Cette manière de structuration a l'avantage d'être consécutive ce qui permettra de profiter de la propriété de la localité spatiale de la mémoire cache, cependant elle présente les limitations suivantes :

- Les Messages C2V et V2C du même arc du graphe de Tanner sont mémorisés séparément ce qui ne permet pas de profiter pleinement de la localité spatiale de la mémoire cache.
- Le nombre des parcours a augmenté par rapport à la méthode de graphe, puisque (2 x M x d_{cmax}) > (2 x E) et (2 x N x d_{vmax}) > (2 x E).
- La taille mémoire n'est pas optimisée puisqu'il y a des éléments non valides qui sont mémorisés rien que pour organiser les données sous forme de matrice en cas d'irrégularité du code LDPC. Dans le cas de l'exemple de la matrice (5, 10), il y a 3 éléments non valides dans la matrice C2V[][] et 13 éléments non valides dans la matrice V2C[][], ces valeurs se manifestent comme des trous dans la mémoire et sont chargés dans la mémoire cache que pour indiquer la fin de parcours de la même ligne ou de le même colonne. Aussi, la présence de ces valeurs non valides rend nécessaire lors de la lecture des matrice ''colonne index'' et ''ligne index'' de vérifier la condition que les valeurs lues ne sont pas des valeurs non valides.

2.2 Optimisation de la structuration des données

Afin de pallier aux limitations ci-dessus, nous proposons une nouvelle méthode hybride de structuration des données dans la mémoire en utilisant des tableaux consécutifs pour assurer l'allocation successive des adresses des variables et en calculant les index d'accès dans l'ordre d'appel des données par le processus de décodage.

Les messages C2V et V2C sont structurés en utilisant un tableau dont chaque élément contient les deux messages C2V et V2C du même arc. L'ordre d'enregistrement des messages C2V et V2C du même arc est choisi selon l'ordre d'appel des données lors de la phase horizontale c'est-à-dire en balayant la matrice de contrôle de parité par ligne.

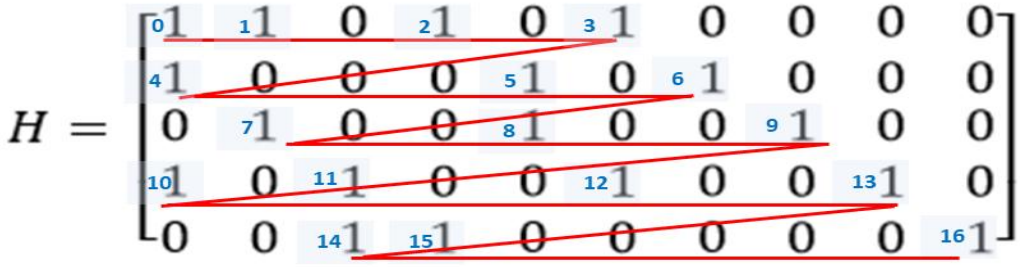


Figure 4. 2 Classement des éléments non nul par ligne

La figure 4.2, montre l'ordre par ligne des arcs (ou éléments non nul de la matrice de contrôle de parité H (5, 10) de l'exemple précédant).

Le tableau mémorisant les messages C2V et V2C, noté MEM, est de taille E, et aura la forme présentée par la figure 4.3.

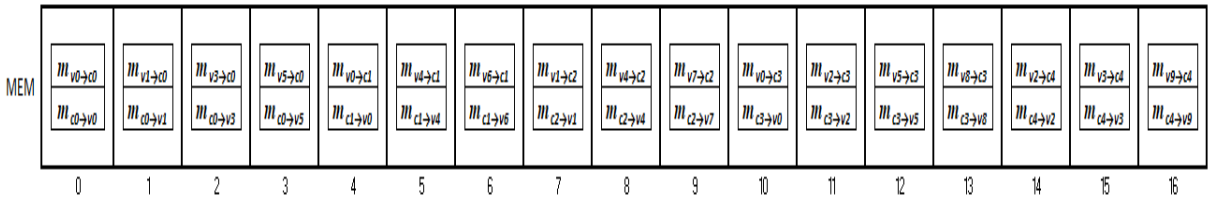


Figure 4. 3 Tableau de structure mémorisant les messages C2V et V2C triés par ligne

En conséquence, lors de la phase horizontale pour chaque ligne i, les messages V2C utilisés pour la mise à jour des messages C2V se trouvent sur la même région mémoire et l'accès se fait d'une manière directe sans besoin de parcours de recherche, il suffit d'indiquer pour chaque ligne *i* l'index de départ et le nombre d'élément à lire du tableau MEM.

Pour une ligne i, l'index de départ, noté $p(i)$, sur le tableau MEM correspond à l'ordre du premier élément non nul de la ligne i, (pour notre exemple on l'obtient à partir de la figure 4.2). Le nombre d'élément à lire du tableau MEM correspond à la valeur d_c de la ligne i, ces informations sont mémorisées sur un tableau de taille M, noté Cm, présenté par la figure 4.4.

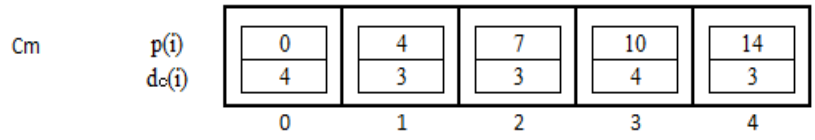


Figure 4. 4 Tableau de structure mémorisant pour chaque nœud de contrôle son degré d_c et le point de départ dans le tableau MEM

La figure 4.5, montre la méthode d'accès lors de la phase horizontale, pour la mise à jour des messages C2V de la ligne 0 par exemple, Tout d'abord, le tableau Cm donne l'index de départ sur le tableau MEM, dans ce cas p(0) = 0, ainsi que le nombre d'élément à lire du tableau MEM pour la ligne 0, ici c'est 4. Ensuite, on lit quatre messages V2C sur le tableau MEM (de l'index 0 à 3), puis On fait les calculs des mises à jour des messages C2V à partir des messages V2C lus et on écrit les mises à jour des messages C2V dans les mêmes index de MEM, mais dans la case destinée à mémoriser les messages C2V. Toutes ces données se trouvent sur la même zone mémoire ce qui permettra de profiter de la localité spatiale de la mémoire cache.

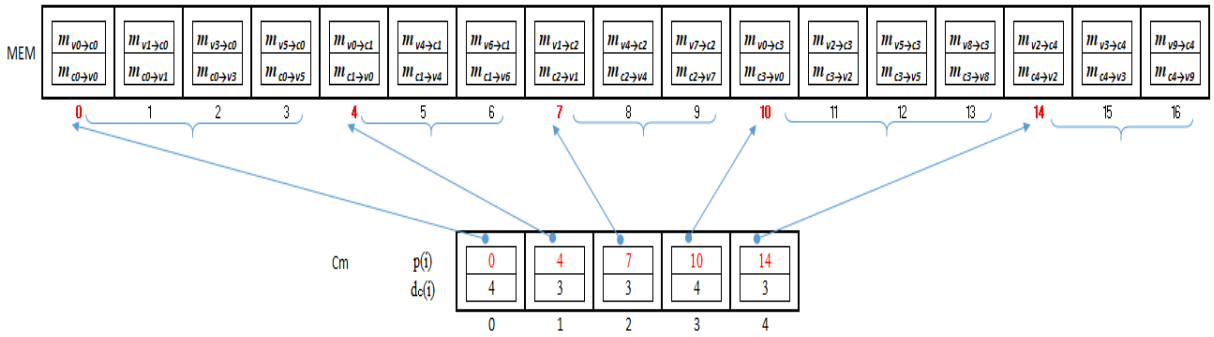


Figure 4. 5 Méthode d'accès pour le calcul des messages C2V en utilisant les deux tableau MEM et Cm

Pour la phase verticale, le parcours et les calculs sont réalisés par colonne alors que les messages ont été organisés dans le tableau MEM par ligne.

Afin d'éviter les parcours lors de la phase verticale du processus de décodage, le calcul des index du tableau MEM relatif à chaque colonne est calculé en dehors du processus de décodage par un prétraitement de la matrice de contrôle de parité H, ces index sont regroupés dans un tableau, noté Add, de taille E. ces index correspondants à l'ordre des éléments non balayés par ligne mais enregistrés par colonne (figure 4.2).

Dans l'exemple ci-dessous, les messages C2V de la colonne 0 (les messages $m_{ci \to v0}$) sont situés dans les index 0, 4, et 10 du tableau MEM (figure 4.3). La figure 4.6, présente le tableau Add de la matrice (5, 10), les valeurs du tableau sont prises de la figure 4.2 :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Add	0	4	10	1	7	11	14	2	15	5	8	3	12	6	9	13	16

Figure 4. 6 Tableau Add de la matrice (5, 10)

De même afin de déterminer pour chaque colonne *j*, l'index de départ p(j) et le nombre d'éléments à lire du tableau Add, on crée un tableau de taille N, noté Vn, qui contient, pour chaque colonne *j*, l'ordre du premier élément non nul dans la matrice H de la colonne *j* et son degré de connectivite d_v. La figure 4.7 présente le tableau Vn correspondant à la matrice (5, 10).

Vn		0	1	2	3	4	5	6	7	8	9
	$d_v(j)$	3	2	2	2	2	2	1	1	1	1
	p(j)	0	3	5	7	9	11	13	14	15	16

Figure 4. 7 tableau Vn de la matrice (5, 10)

La figure 4.8, présente la méthode de calcul, pour chaque colonne *j*, des index p(j) de départ de parcours du tableau Add. L'ordre des éléments est réalisé en balayant la matrice par colonne puis en prend que l'ordre des premiers éléments non nul par colonne.

$$H = \begin{bmatrix} 1_0 & 1_3 & 0 & 1_7 & 0 & 1_{11} & 0 & 0 & 0 & 0 \\ 1_1 & 0 & 0 & 0 & 1_9 & 0 & 1_{13} & 0 & 0 & 0 \\ 0 & 1_4 & 0 & 0 & 1_{10} & 0 & 0 & 1_{14} & 0 & 0 \\ 1_2 & 0 & 1_5 & 0 & 0 & 1_{12} & 0 & 0 & 1_{15} & 0 \\ 0 & 0 & 1_6 & 1_8 & 0 & 0 & 0 & 0 & 0 & 1_{16} \end{bmatrix}$$

Figure 4. 8 Classement des éléments non nul par colonne

La figure 4.9, montre la méthode d'accès lors de la phase verticale, pour la colonne 0 par exemple on lit depuis le tableau Vn l'index de départ sur le tableau Add, p(j), ici c'est 0 ainsi que le nombre d'élément à lire du tableau Add pour la colonne 0, ici c'est 3. On parcourt le tableau Add de 0 à 2 (3 éléments), les valeurs de ces éléments pour cet exemple sont 0, 4 et 10, ces valeurs représentent l'index dans le tableau MEM des messages C2V ($m_{c0\rightarrow v0}$, $m_{c1\rightarrow v0}$ et $m_{c3\rightarrow v0}$) et les messages V2C correspondant à la colonne 0 ($m_{v0\rightarrow c0}$, $m_{v0\rightarrow c1}$ et $m_{v0\rightarrow c3}$). L'accès à toutes ces données est réalisé directement en mémorisant les adresses des emplacements des messages C2V et V2C ce qui permettra d'éviter les parcours et les comparaisons pour trouver l'endroit des messages correspondant à un colonne *j*.

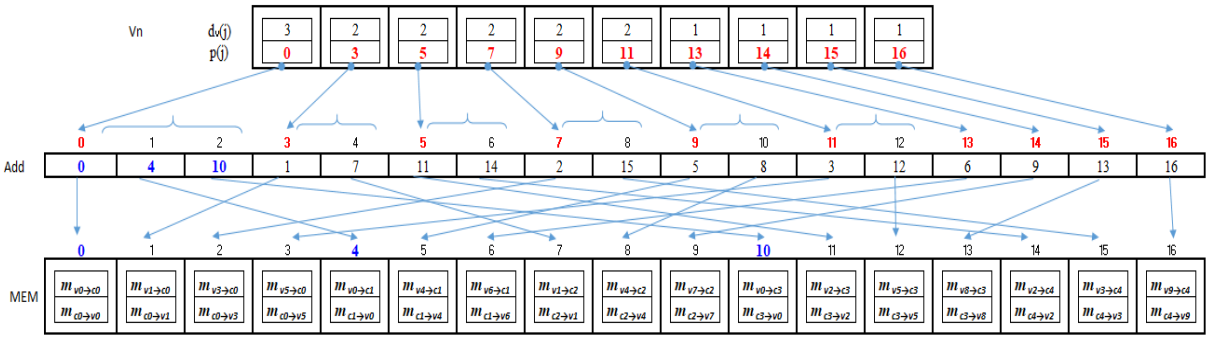


Figure 4. 9 Méthode d'accès pour le calcul des messages V2C en utilisant les tableaux MEM, Add et Vn

La taille de mémoire nécessaire pour la structure proposé est donnée par l'équation 4.1, elle est équivalente à celle de la structure de la littérature [52-56], mais les données sont organisées selon l'ordre de leur appel lors de la phase horizontale du décodage LDPC, ce qui permet un accès direct vers les nœuds concernés par les calculs sans parcours ou comparaisons :

$$memory\ size = (2M\ + 2N + \ E) \times \ sizeof(integer) + (\ 2N + 2E) \ \times \ sizeof(float) \qquad (4.2)$$

Ainsi, la taille mémoire nécessaire pour la méthode proposée, équation (4.2), est réduite par rapport à la méthode de la littérature, car : $M \times d_{cmax} + N \times d_{vmax} > 2 \times E$ et $2M + 2N < E$, voir tableau (2.1). Les résultats de profilage des différentes méthodes de structuration sont présentés à la fin de ce paragraphe.

2.3 Optimisation de l'algorithme par ordre d'appel dans le processeur

Contrairement aux travaux précédents [52-56], nous proposons une version optimisée de l'algorithme NMSA HS en fusionnant les étapes horizontale et verticale en une seule étape, permettant ainsi de diminuer les accès mémoire et de minimiser le nombre d'opérations arithmétiques.

La mise à jour séparée des messages C2V lors de la phase horizontale et des messages V2C et L(vj) lors de la phase verticale augmente l'accès à la mémoire pour les mêmes données.

Pour remédier à ce problème, l'optimisation proposée vise à calculer toutes les mises à jour de messages C2V, V2C et L(vj) correspondants aux données chargées dans le processeur CPU en une seule phase, et ce, avant de charger les données suivantes. Cela permet de profiter de la localité temporelle de la mémoire cache et d'effectuer tous les calculs sur les données chargées en CPU.

D'après les équations (1.24) et (1.25) de l'algorithme LLR BP, nous pouvons observer que la valeur du message V2C de l'itération courante est incluse dans la valeur du LLR de la même itération, nous pouvons donc calculer la valeur du message V2C à partir de la différence entre la valeur du LLR du nœud de variable et le message C2V connecté selon l'équation ci-dessous

$$m_{\mathrm{v_j}\to\mathrm{c_i}} = L(\mathrm{v}_j) - m_{\mathrm{c_i}\to\mathrm{v_j}} \tag{4.3}$$

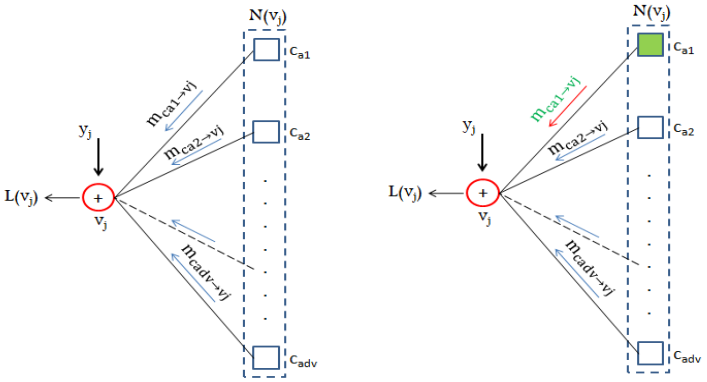


Figure 4. 10 : Mise à jour de la valeur LLR

De plus, comme illustré sur la figure 4.10, lorsqu'un message C2V est mis à jour, le calcul de la nouvelle valeur LLR du nœud de variable $\mathrm{v_j}$, $\mathrm{L(v_j)}$, peut être effectué directement en remplaçant l'ancienne valeur C2V par la nouvelle valeur mise à jour selon l'équation ci-dessous :

$$L(\mathrm{v}_j)^{new} = L(\mathrm{v}_j)^{old} - m_{\mathrm{c_a}\to\mathrm{v_j}}{}^{old} + m_{\mathrm{c_a}\to\mathrm{v_j}}{}^{new} \tag{4.4}$$

Ainsi, l'algorithme LLR BP, en utilisant l'algorithme NMSA devient :

Etape 0 (Initialisation) :

Les messages LLR initiaux de chaque nœud de variable v_j, notés $L(0)_{\mathrm{v}_j}$, sont initialisés directement par l'information reçue du canal y_j. Les messages C2V sont initiés à 0 :

$$\begin{cases} L(\mathrm{v}_j)^0 = y_j \\ m_{ci\to\mathrm{v}j} = 0 \end{cases} \tag{4.5}$$

Etape 1 (Calcul des mises à jour des messages C2V et L(v_j)) :

En une seule étape, les mises à jour des messages C2V et des valeurs LLR des nœuds de variables sont calculés comme suit :

$$\begin{cases} m_{ci\to vj} = \alpha \prod_{vb\in N(ci)\setminus vj} sign(L(v_b) - m_{c_i\to v_b}) \min_{vb\in N(ci)\setminus vj}(|L(v_b) - m_{c_i\to v_b}|) \\ L(v_j)^{new} = L(v_j)^{old} - m_{c_a\to v_j}{}^{old} + m_{c_a\to v_j}{}^{new} \end{cases} \quad (4.6)$$

L'algorithme 3 décrit le pseudocode de l'optimisation proposée. L'initialisation de C2V et L(v_j) est effectuée (ligne 1) selon l'équation (4.5), ensuite pour chaque nœud de contrôle c_i, le processeur charge les valeurs C2V et L(v_j) de chaque nœud de variable v_j connecté au nœud de contrôle c_i (lignes 3 et 4). Les valeurs min1 et min2 sont ensuite calculées à partir de la différence entre C2V et L(v_j) chargés au niveau du processeur (ligne 5 et 6), puis les mises à jour des massages C2V et L(v_j) sont calculées dans la même boucle (ligne 9,10,11) avant de passer au nœud de contrôle suivant. Les étapes de la décision ferme et de la vérification du syndrome resteront les mêmes que l'algorithme initial.

Algorithm 3 : Proposed Optimization NMSA
1 : Initialize all $m_{ci\to vj} = 0$, and L(v_j) = y_j Itermax
2 : for i = 1 to Itermax do
Horizontal processing :
3 : for each check node c_i
4 : for each variable node vj connected to ci
5 : Calculate $v2c = L(v_j) - m_{ci\to vj}$
6 : Calculate min_1 & min_2
7 : end for line4
8 : for each variable node v_j connected to c_i
9 : Calculate $c2v = \alpha \prod_{n'\in N(m)\setminus n} sign(m_{vb\to ci})\ min1\ or\ min2$
10 : Calculate $L(v_j) = v2c + c2v$
11 : $m_{ci\to vj} = c2v$
12 : end for line8
13 : end for line 3
Hard decision :
14 : for each variable node
15 : Make hard decision if L(v_j) > 0 then v_j = 1;else v_j = 0
16 : end for line 19
Parity check equations (Syndrome) and stopping criteria :
17 : if $H \bullet v^T = 0$ then break else i = i + 1
18 : end for line 2

Côté structuration des données, l'optimisation proposée ne nécessite que la mémorisation des messages C2V et L(v_j), alors que l'algorithme initial nécessite la mémorisation des messages

C2V, V2C, $L(0)_{vj}$ et $L(v_j)$, ce qui permet de réduire la taille mémoire nécessaires pour emmagasiner les messages échangés lors du processus de décodage.

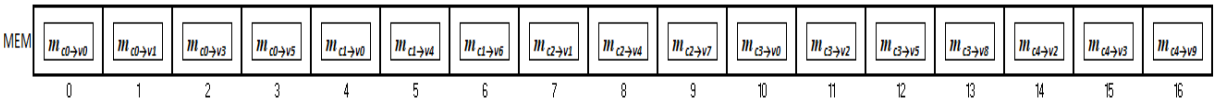


Figure 4. 11 : Tableau MEM après l'optimisation proposée

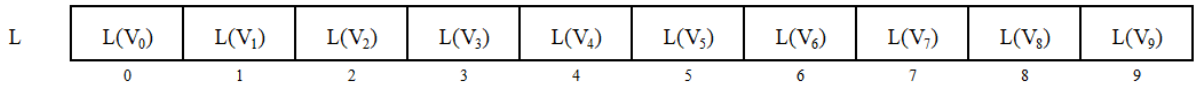


Figure 4. 12 : Tableau L mémorisant les LLR des nœuds de variable

En plus, l'optimisation proposée fusionne les deux étapes horizontale et verticale en une seule étape avec un parcours par nœud de contrôle (autrement dit par ligne de la matrice de parité H). A cet effet, les messages C2V sont également ''mappés'' en mémoire dans l'ordre de ligne dans des positions de mémoire consécutives et ce afin de profiter de la localité spatiale de la mémoire cache. Les figures 4.11 et 4.12, montrent respectivement le tableau MEM de taille E qui mémorise les messages C2V et le tableau L de taille N qui mémorise les valeurs $L(v_j)$ de la matrice (5, 10) de l'exemple précédant.

Pour les données relatives à la mémorisation de la matrice de contrôle de parité H, avec l'élimination de la phase verticale, on n'a pas besoin des deux tableaux Add et Vn utilisées lors de la phase verticale. L'optimisation proposée ne nécessite que la mémorisation des index de colonnes de chaque ligne, et ce, puisqu'après chaque mise à jour d'une ligne, les colonnes y connectées via un élément non nul sont mises à jour. La figure 4.13, illustre le tableau de taille E qui mémorise les colonnes de chaque ligne de la matrice (5, 10).

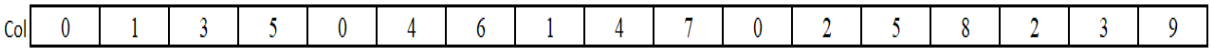


Figure 4. 13 : Table Col des index des colonnes par ligne de la matrice (5, 10)

Comme illustré sur la Figure 4.14, les différentes interactions réalisées lors du processus de décodage en utilisant l'optimisation sont très réduites.

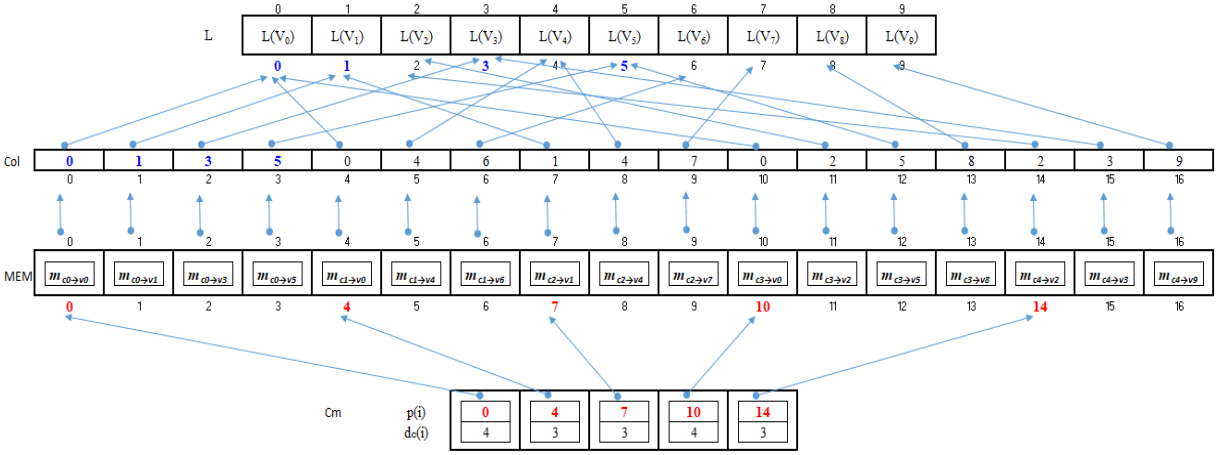


Figure 4. 14 : Interconnexions de différentes données lors du processus de décodage en utilisant l'optimisation proposée

L'accès mémoire aux tableau C2V et Col se fait directement car les messages C2V et les index des colonnes sont stockés dans l'ordre d'exécution en CPU (rangés par ordre de ligne) ; L'accès mémoire au message $L(v_j)$ se fait moyennant la table Col, par exemple le nœud de contrôle c_0 contient 4 éléments non nuls, le premier est connecté au nœud de variable v_0 (Col[0] = 0), le deuxième est connecté à v_1 (Col[1] = 1), le troisième est connecté à v_3 (Col[2] = 3) et le quatrième est connecté au nœud variable v_5 (Col[4] = 5).

La taille de mémoire de l'ensemble des données nécessaires pour le processus de décodage devient :

$$memory\ size = (2M + E) \times\ sizeof(integer) + (N + E) \times\ sizeof(float) \tag{4.7}$$

La première partie de l'équation concerne le stockage de la matrice H (tableau Col, et Cm) et la seconde partie concerne le stockage des messages (C2V et $L(v_j)$). Il ressort clairement des équations (4.1) et (4.7) que la taille de la mémoire requise pour la structure de données proposée est inférieure de 50% à celle de la structure de données proposée dans [52-56] et, par conséquent, le temps d'exécution requis pour l'accès à la mémoire est fortement réduit.

2.4 Structuration des données par mémorisation de la matrice d'expansion

Pour réduire davantage la taille mémoire nécessaire pour emmagasiner la matrice de parité H, on a proposé dans le cadre de cette thèse, de mémoriser uniquement certains paramètres de la matrice d'expansion utilisés dans le cas des codes LDPC quasi-cycliques. En effet, dans ce cas et comme expliqué dans le chapitre I, la matrice de parité H de taille (N x

M) peut être compressé en une matrice d'expansion H_{bm} de taille n_b x m_b qui contient des coefficients de permutation δ , avec $n_b = N/Z$ et $m_b = M/Z$ et chaque coefficient de permutation génère une sous matrice de taille Z x Z selon sa valeur :

- Si δ < 0, une sous matrice zéro de taille Z x Z est généré.
- Si δ = 0, une sous matrice identité de taille Z x Z est généré.
- Si δ > 0, une sous matrice identité de taille Z x Z est généré, avec une permutation circulaire à gauche de ces colonnes δ fois.

La méthode proposée se base sur la mémorisation par ligne des indexes des colonnes des éléments supérieur ou égal à zéro de la matrice H_{bm}. Etant donné l'index de la ligne i de la matrice H, la colonne j correspondante de la matrice de contrôle de parité H peut être calculé à partir de l'index de la colonne c de la matrice H_{bm} et le coefficient de permutation δ comme suit :

$$j = (c \times Z) + (\delta + i)\ mod\ Z \tag{4.8}$$

Exemple :

Soit la matrice d'expansion H_{bm} de taille 2 x 4 avec Z = 3 :

$$H_{bm} = \begin{bmatrix} 1 & 2 & 0 & -1 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Ainsi, la matrice de parité H générée de la matrice H_{bm} est :

$$H = \left[\begin{array}{|ccc|ccc|ccc|ccc|} \hline 0&1&0&0&0&1&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0&1&0&0&0 \\ \hline 0&0&1&0&1&0&1&0&0&1&0&0 \\ 1&0&0&0&0&1&0&1&0&0&1&0 \\ 0&1&0&1&0&0&0&0&1&0&0&1 \\ \hline \end{array}\right]$$

La figure 4.15 représente, la structuration proposée de la matrice H en tableaux Col et C_m proposé par le chapitre précédent.

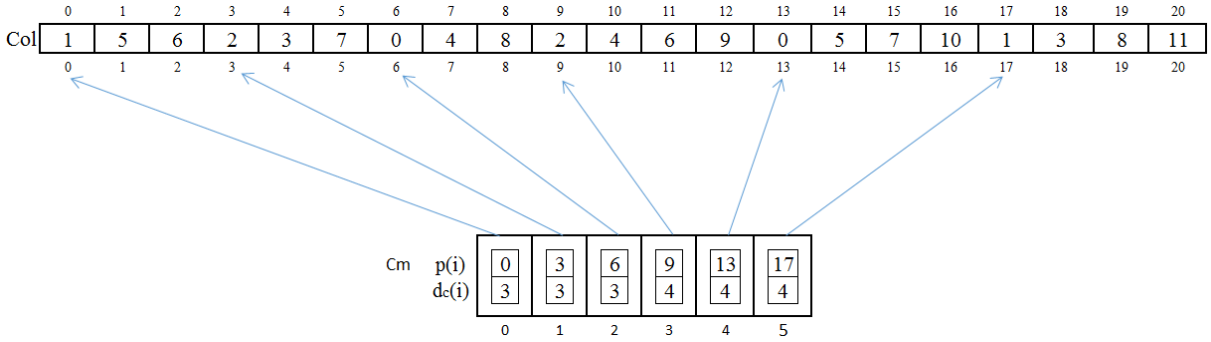


Figure 4. 15 Tableau Col et Cm nécessaire pour la mémorisation de la matrice (12, 6)

En appliquant la même structuration de donnée sur la matrice d'expansion H_{bm}, les deux tableaux Col_{bm} et C_{bm} qui représentent la matrice d'expansion H_{bm} de l'exemple auront la forme présentée par la figure 4.16.

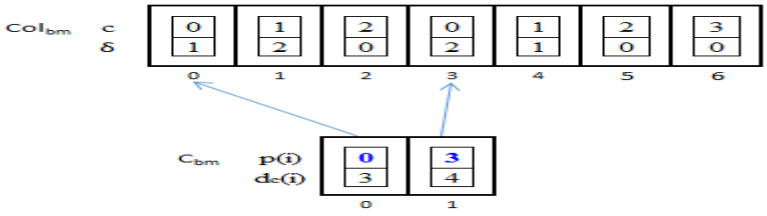


Figure 4. 16 : Représentation de la matrice d'expansion H_{bm}

Le tableau Col_{bm} contient pour chaque ligne de la matrice H_{bm}, l'index de la colonne des éléments positifs ainsi que le coefficient de permutation δ correspondant. Le tableau C_{bm} contient pour chaque ligne de la matrice H_{bm}, la valeur d_c et l'index de départ p(i) sur le tableau Col_{bm}. Une seule ligne de H_{bm} représente Z lignes de la matrice H (dans cet exemple Z = 3). Le calcul des colonnes reliées à chacune des Z lignes, est réalisé selon l'équation (4.8).

Tableau 4. 1 : Méthode de calcul des lignes et colonnes de la matrice H (12, 6) à partir des coefficient de permutation de la matrice d'expansion H_{bm} (4, 2)

Ligne H_{bm}	Ligne H i	Colonne H_{bm} c	Coefficient de permutation Δ	Colonne H $j = (c \times 3) + (\delta + i) \bmod 3$
0	0	0	1	1
		1	2	5
		2	0	6
	1	0	1	2
		1	2	3
		2	0	7
	2	0	1	0
		1	2	4
		2	0	8

1	0	0	2	2
		1	1	4
		2	0	6
		3	0	9
	1	0	2	0
		1	1	5
		2	0	7
		3	0	10
	2	0	2	1
		1	1	3
		2	0	8
		3	0	11

Le tableau 4.1, représente les calculs réalisés, durant le processus du codage, des index des colonnes de la matrice H à partir des paramètres mémorisés de la matrice H_{bm}, et ce, en utilisant l'équation (4.8). Ainsi, on obtient les mêmes colonnes du tableau Col de la matrice (12, 6) de la figure 4.15.

La taille mémoire nécessaire pour mémoriser les messages C2V et $L(v_j)$ reste la même. Cependant, la taille mémoire nécessaire pour la mémorisation des paramètres de la matrice H_{bm} est réduite à :

$$memory\ size = (2\ m_b + 2\ e_b) \times sizeof(integer) \tag{4.9}$$

Avec $m_b = M/Z$ est le nombre de ligne de la matrice H_{bm}, et $e_b = E\ /Z$ est le nombre d'éléments positifs de la matrice H_{bm}. Ainsi, la taille mémoire nécessaire pour mémoriser la matrice est presque divisée par Z par rapport à la dernière structuration proposée ce qui permettra de réduire les accès à la mémoire, en contrepartie les calculs des index des colonnes selon l'équation (4.8) sont réalisés durant le processus de décodage LDPC, ce qui augmente la complexité calculatoire du décodage. La comparaison des résultats de profilage des deux structures est nécessaire pour évaluer l'effet sur la latence de décodage.

2.5 Evaluation des cycles d'exécution des différentes structurations des données proposées

Afin de comparer les trois optimisations proposées en termes de nombre de cycles CPU consommés, des profilages sont lancés pour différents taille de code (9216, 4608), (2304, 1152) et (576,288). Les figures 17, 18, et 19, présentent l'histogramme du nombre de cycles

CPU consommés par les instructions de décodage, noté IR, et celui consommé par les accès mémoire en mode lecture et écriture notés respectivement DR, DW.

L'optimisation proposée a permis de réduire le nombre total de cycle CPU de $1{,}05x10^7$ à $3{,}45x10^6$ pour le code (9216, 4608), de $2{,}6x10^6$ à $8{,}5x10^5$ pour le code (2304, 1152) et de $6{,}5x10^5$ à $2{,}1x10^5$ pour le code (576,288).

La méthode de structuration en utilisant les paramètres de la matrice d'expansion H_{bm} a permis de réduire les accès à la mémoire, cependant le nombre d'instruction a augmenté en raison de calcul des index des colonnes lors du processus de décodage en utilisant l'équation (4.8).

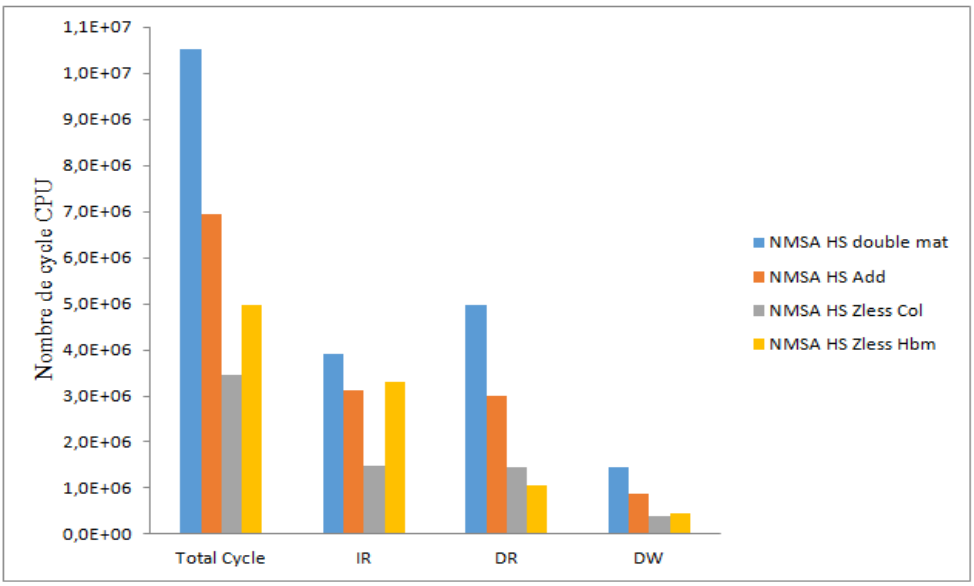


Figure 4. 17 Résultats de profilage des différentes structurations de la matrice (9216, 4608)

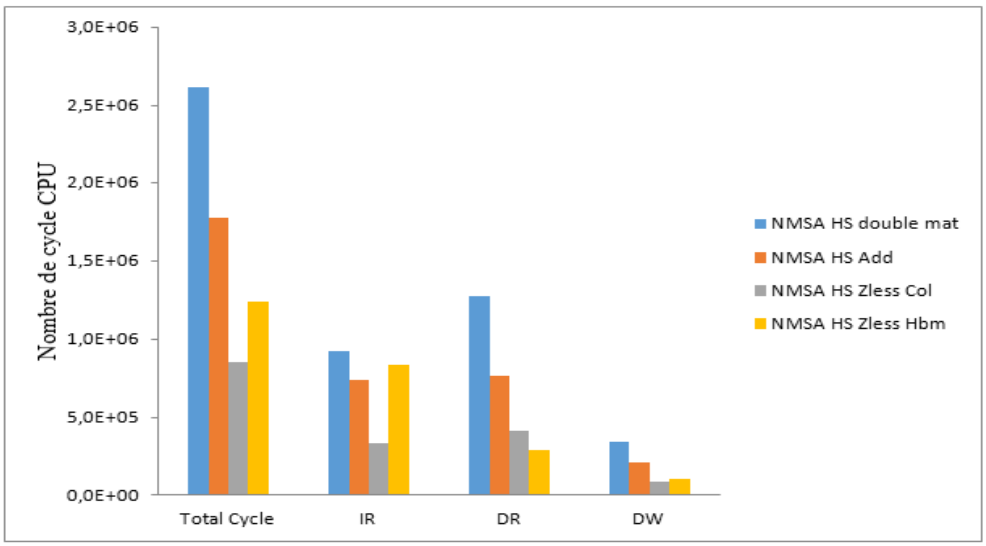


Figure 4. 18 Résultats de profilage des différentes structurations de la matrice (2304, 1152)

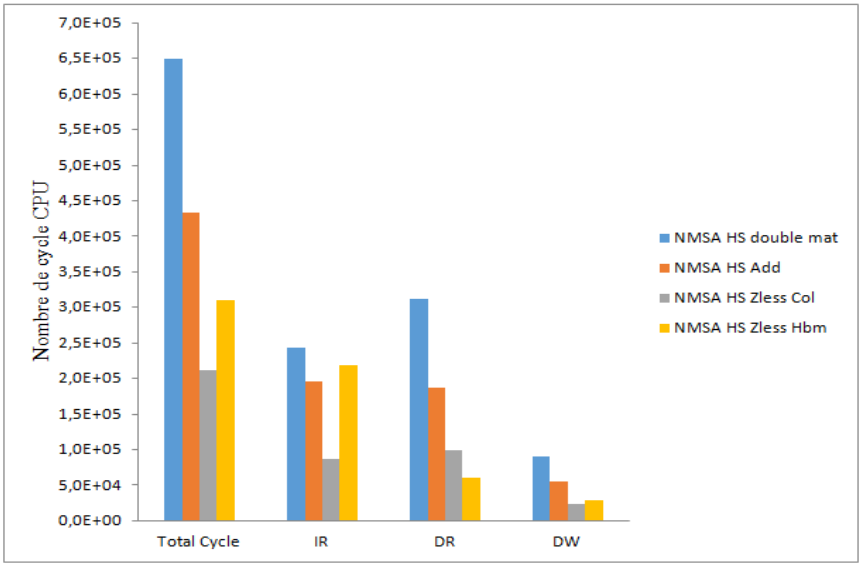


Figure 4. 19 Résultats de profilage des différentes structurations de la matrice (576, 288)

Le pourcentage de réduction du nombre de cycle CPU nécessaire pour le décodage, ainsi que la taille mémoire de l'optimisation basée sur l'union des deux phases de décodage Horizontale et verticale en une seule phase est présenté sur le tableau 4.2. La taille mémoire est réduite de 62% et le nombre de cycle requis pour le décodage est réduit de 66%, ce qui correspond à une accélération de 3 fois en termes de temps de traitement par rapport au décodeur LDPC de la littérature utilisant deux étapes séparées horizontale et verticale.

Les accès à la mémoire en lecture (DR) et en écriture (DW) sont plus faibles par rapport aux travaux de la littérature, confirment ainsi la réduction de la taille mémoire nécessaire et que l'optimisation proposée permet profiter de la localité spatiale et temporelle de la mémoire cache.

Tableau 4. 2 Gain en terme du nombre de cycle CPU et de taille mémoire par rapport aux travaux précédents

Code size	**CPU cycles état de l'art**	**CPU cycles optimisation proposée**	**% de reduction**	**Taille mémoire état de l'art (Ko)**	**Taille mémoire optimisation proposée (Ko)**	**% de réduction**
(9216, 4608)	$1.05x10^7$	$3.45x10^6$	66,67%	599	230	61,6%
(2304, 1152)	$2.6x10^6$	$8.5x10^5$	66,00%	149	57	61,7%
(576, 288)	$6.5x10^5$	$2.1x10^5$	67,7%	37	14	62,2%

3 Modèle de calcul parallèle proposé

La latence de décodage peut également être réduite en fonction de la plateforme multicœur et du niveau de parallélisme de l'algorithme, qui est corrélé aux dépendances des données pendant le processus de décodage.

La structure de données proposée permet une exécution parallèle ; En effet, chaque nœud de contrôle a sa partie indépendante des tableaux Col et MEM consécutif et trié dans l'ordre de l'exécution du processus de décodage. Cependant, le tableau $L(v_j)$ est partagé entre tous les nœuds de contrôle et plusieurs processeurs peuvent lire ou écrire la mise à jour de la valeur $L(v_j)$ du même nœud de variable v_j au même temps, ce qui génère un conflit d'accès et nécessite une synchronisation entre les différents processeurs lors d'accès simultané à la même adresse mémoire, cette synchronisation engendre un temps d'exécution supplémentaire et influence par conséquent la latence de décodage.

Pour éviter le temps supplémentaire de synchronisation, seul les nœuds de contrôle connectés à des nœuds de variables distincts seront lancés en parallèle, ainsi, le traitement parallèle est autorisé entre k nœuds de contrôle qui ne partagent aucun nœud de variable, autrement dit :

$$\cap_{k\ check\ nodes} N(c_k) = \emptyset \tag{4.10}$$

Avec $N(c_k)$ l'ensemble des nœuds de variables connectés au nœud de contrôle c_k.

La valeur k représente le niveau de parallélisme de l'algorithme, les k nœuds de contrôle satisfaisant l'équation (4.10) sont groupés dans une seule couche (Layer) et sont lancés en parallèle, le passage d'une couche à l'autre se fait d'une manière séquentielle.

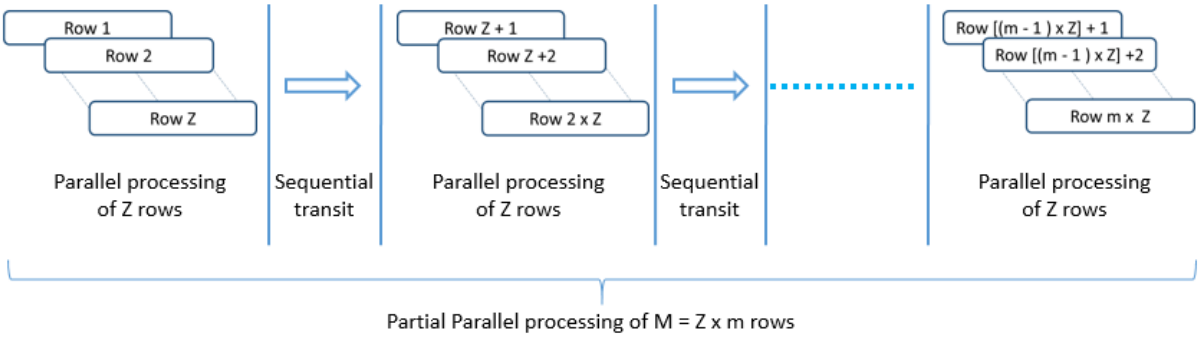


Figure 4. 20 : Modèles de calcul parallèle pour LDPC Quasi-cyclique

Pour les codes LDPC quasi-cycliques, chaque ligne de la matrice d'expansion H_{bm} est étendue à Z lignes en remplaçant chaque coefficient de décalage par une matrice d'identité Z x Z ou une matrice d'identité Z x Z décalée circulairement; ainsi, les Z lignes générées par une ligne de H_{bm} répondent à l'équation (4.10), car les lignes d'une matrice identité ne partagent aucune colonne. Par conséquent, comme le montre la figure 4.20, Z threads sont lancés en parallèle ; chaque thread traite une ligne de la matrice H, la transition vers les Z lignes suivantes est effectuée séquentiellement, et ainsi de suite jusqu'à ce que toutes les M lignes soient traitées. Puisque toutes les lignes générées à partir d'une ligne de la matrice d'expansion H_{bm} ont la même valeur dc, les Z threads qui traitent la même ligne H_{bm} ont exactement le même nombre de données et de parcours à traiter et par la suite ont le même temps d'exécution et ne nécessitent par conséquent aucune synchronisation ou de temps d'attente entre les Z threads traités en parallèle.

4 Implémentation logicielle du décodeur LDPC sur des plateformes multicoeurs

L'implémentation parallèle proposée est lancée sur deux plateformes de traitement parallèle différentes en utilisant l'outil OpenMP[57]. La première plate-forme est une plate-forme HPC Marwan avec Intel Xeon gold 6130 avec 08 CPU à 2.10 Ghz [58] sous Lunix. La deuxième plate-forme est une carte à base du processeur quad-core Cortex-A72 (ARM v8) @ 1.5GHz avec une distribution Linux.

4.1 Bibliothèque d'implémentation parallèle Open MP (Open Multi-Processing)

OpenMP (Open Multi-Processing) [57] est un outil d'aide à la programmation parallèle des codes écrits en langage C/C++ ou Fortran. Il est basé sur un modèle multithreading pour des architectures à mémoire partagée. Il se présente sous la forme de directives de compilation insérées dans le code source (des pragmas), des variables d'environnement et d'une bibliothèque logicielle permettant de développer rapidement des applications parallèles.
Un programme OpenMP est fait de régions séquentielles ainsi que de régions parallèles qui peuvent être exécutées par plusieurs threads en parallèle sur différents processeurs. Le compilateur se charge de la création des threads, leurs synchronisations et leurs échanges, ce qui permet de réduire le temps d'implémentation parallèle.

4.2 Implémentation sur la plateforme HPC MARWAN

Le Centre National pour la Recherche Scientifique et Technique CNRST a mis à disposition des chercheurs Marocains un Centre Data universitaire (Cluster) de calcul haute performance HPC MARWAN[58] accessible à distance, pour permettre d'effectuer leurs calculs scientifiques et ce dans différents domaines. La plateforme compte 38 nœud avec les capacités ci-dessous :

- 1672 cœurs CPU
- 10.4 TB de mémoire
- 396 TB de Stockage.
- 4 GPUs

Depuis une machine cliente, une connexion réseau avec le serveur hpc-login.marwan.ma est établie via le protocole hautement sécurisé SSH (ou Secure SHell), l'allocation d'un nœud de calcul libre avec les détails de la commande tels que le nombre de nœuds, le temps alloué, la partition des nœuds à utiliser, le nombre de processeur et le nom du fichier de sortie qui contient les résultats de simulation sont spécifiés via un script qui englobe les différentes commandes appropriées.

4.3 Implémentation sur processeur ARM quad-core Cortex-A72

La carte Raspberry Pi 4 fait partie de la famille des Nano-Ordinateurs, monocarte à processeur ARM, elle permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux, notamment Debian.

Il existe plusieurs versions du Raspberry PI Zéro, A, Pi B, Pi, qui se distinguent par leurs tailles, les éléments de connectivités, la taille mémoire RAM et le processeur, allant de la version zéro, l'ultra réduite au format d'un clé USB Pour des applications embarquées nécessitant un maximum de gain de place à la version Raspberry PI 4 au format d'une carte bancaire avec une augmentation de la vitesse du processeur, des performances multimédia, de la mémoire et de la connectivité par rapport aux générations précédentes.

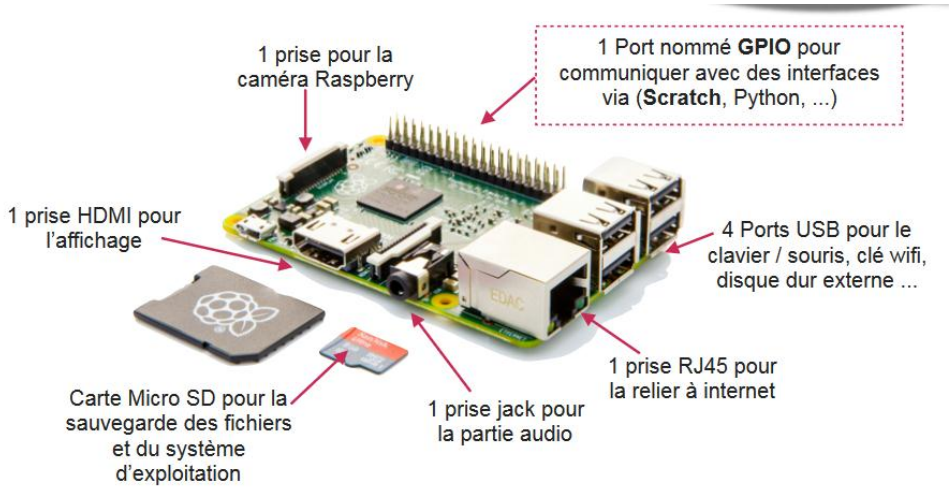


Figure 4. 21 : Carte raspberry Pi 4

La figure 4.24, présente une photo de la carte Raspberry Pi 4, avec ses différents fonctionnalités clés, notamment un processeur quad-core 64 bits Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GH et une mémoire RAM allant à 8 Go.

Après avoir installé un système d'exploitation qui contient un serveur SSH, sur une carte Micro SD, qui est insérée sur la carte Raspberry, depuis une machine cliente, une connexion réseau avec la carte est établie via le protocole SSH (en attribuant une adresse IP à la carte Raspberry), la compilation des algorithmes de décodage LDPC est réalisé directement sur la carte Raspberry pour créer un exécutable compatible avec les processeurs ARM.

4.4 Résultat de latence sur les plateformes multicœurs

Afin d'évaluer, en milliseconde, la latence les performances du décodeur LDPC proposé sur les deux plateformes, Nous avons utilisé différents taille de codes LDPC irréguliers quasi cycliques construits sur la base de la norme IEEE 802.16e [19]. Le décodage LDPC est lancé sur un seul processeur, puis sur deux processeurs et ainsi de suite jusqu'à l'utilisation du nombre maximal de processeurs disponibles.

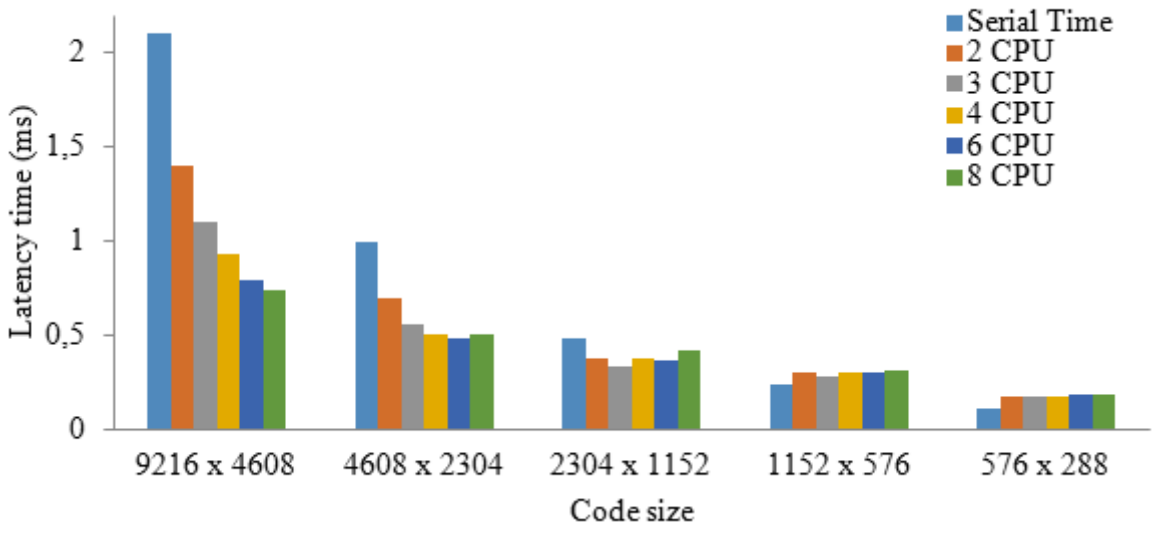


Figure 4. 22 : Comparaison de la latence en (ms) entre le traitement en série et le traitement en parallèle sur plateforme MARWAN pour les différentes tailles de code.

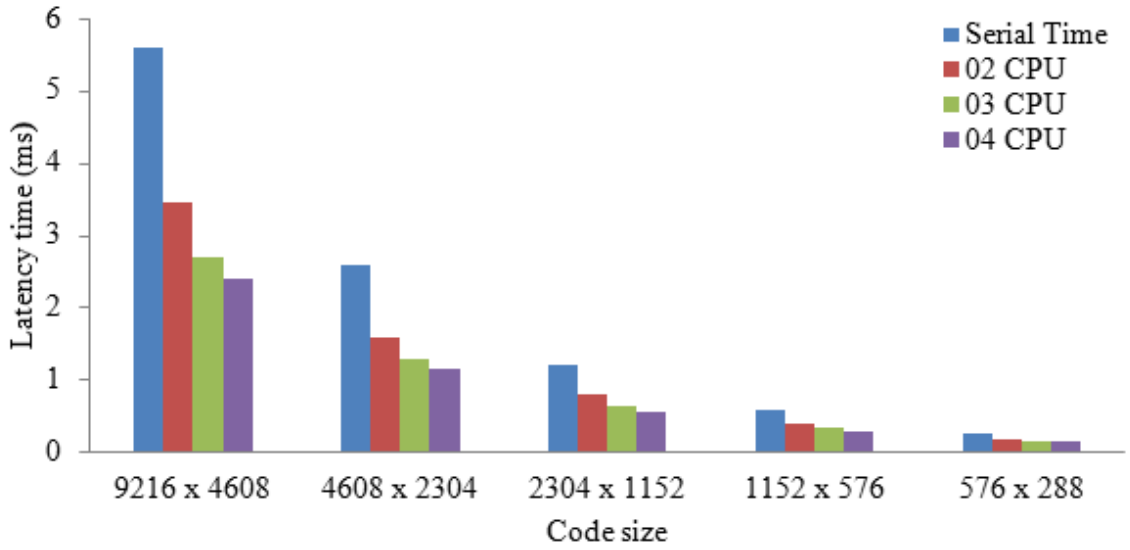


Figure 4. 23 : Comparaison de la latence en (ms) entre le traitement en série et le traitement en parallèle sur la plate-forme Raspberry Pi 4 pour les différentes tailles de code.

Les figures 4.22 et 4.23 présentent une comparaison des latences en millisecondes entre le traitement série et le traitement parallèle pour les différentes tailles de code. Dans la première plate-forme, la rapidité du traitement en parallèle montre des résultats significatifs par rapport au traitement en série, elle est d'environ x2,8 pour la taille du code (9216 x 4608) et x2 pour la taille du code (4608 x 2304).

Cependant, les latences du décodage parallèle pour les matrices de taille inférieure à (1152 x 576) affichent des latences inférieures à celles du traitement en série. Par conséquent, l'impact

du parallélisme dépend du degré de parallélisme Z de chaque matrice, de la complexité de décodage exprimé en termes d'éléments non nul dans la matrice, de la vitesse d'horloge des processeurs utilisés et du temps nécessaire pour la création et la synchronisation des threads.

Pour la plateforme Raspberry Pi 4 avec un processeur de 1.5 Ghz de vitesse d'horloge, le traitement parallèle montre un gain de rapidité par rapport au traitement en série. Contrairement à la première plateforme, même pour la matrice 1152, nous pouvons noter que la rapidité a doublé en utilisant le traitement parallèle, ceci est principalement dû à la vitesse d'horloge du Raspberry de 1,5 GHz inférieure à celle de la première plateforme (2,1 GHz), le temps de calcul sur un seul processeur de la plateforme Raspberry Pi 4 est supérieur au temps nécessaire pour la création et la synchronisation des threads.

Tableau 4. 3 : Performances de décodage sur les deux plateformes

			Reliability			Intel Xeon gold 6130		Rasp berry Pi4	
Matrix Size	Edge	Z	SNR (dB)	FER	BER	Latency (in ms)	CPU Number	Latency (in ms)	CPU Number
9216 x 4608	29 184	384	4,5	10^{-6}	$7x10^{-10}$	0,74	8	2,4	4
4608 x 2304	14 592	192	4,5	$4x10^{-6}$	$8,7x10^{-10}$	0,48	6	1,16	4
2304 x 1152	7 296	96	4,5	$2,2x10^{-6}$	$2,2x10^{-9}$	0,33	3	0,56	4
1152 x 576	3 648	48	4,5	$6x10^{-6}$	$5,2x10^{-9}$	0,24	1	0,29	4
576 x 288	1 824	24	5	$2x10^{-6}$	$3,4x10^{-9}$	0,11	1	0,15	4

Le tableau 4.3 présente les résultats de la latence pour plusieurs tailles de code obtenues par un choix dynamique entre le traitement en série ou en parallèle à l'aide de la clause "if" d'OpenMP appliqué sur le degré de parallélisme Z en utilisant les résultats reportés sur les figures 4.25 et 4.26. La latence est de 0,11 ms pour le code (576, 144) et de 0,74 ms pour la plus grande taille du code avec un taux d'erreur de trame compris entre entre 10^{-6} et $6x10^{-6}$, et un taux d'erreur de bit entre $3,4x10^{-9}$ et $7x10^{-10}$. Sur la plate-forme Raspberry, la latence est de 0,15 ms pour le code (576 x 288) (équivalent à 32 octets) avec une erreur de trame de $2x10^{-6}$, ce qui montre clairement que les résultats obtenus en matière de latence répondent largement à la latence et à la fiabilité requises pour les communications ultra-fiables à faible latence.

Conclusion :

Dans ce chapitre, nous avons présenté une nouvelle optimisation de l'algorithme NMSA avec un ordonnancement brassé de type HS, et ce afin de réduire la latence de décodage avec les mêmes performances en terme de correction d'erreurs. La version optimisée de l'algorithme, remplace les étapes horizontales et verticales par une seule étape sans stockage des mises à jour des messages V2C et avec une structure de donnée compacte permettant une réduction de la taille de la mémoire requise de 62% et une mise en œuvre sur une plate-forme multicœur. L'algorithme proposé permet d'atteindre un taux d'erreur de trame FER de 10^{-6} et la complexité est réduite de 66%. De plus, la latence a atteint 0,15ms pour 288 bits et 0,48ms pour 2304 bits avec un parallélisme dynamique sur les plateformes multicoeurs. Nos résultats de simulation et d'analyse montrent que l'algorithme LDPC embarqué proposé est aligné avec les objectifs de latence et de fiabilité des communications ultra-fiable à faible latence URLLC, rendant possible son utilisation pour ces types de communications sur divers types de matériels IoT.

Conclusion Générale & Perspectives

Les travaux de recherche effectués au cours de cette thèse se sont focalisés sur l'étude et la proposition d'une nouvelle méthode d'ordonnancement, ainsi que l'implantation embarqués de décodeurs LDPC pour les communications ultra fiables à faibles latence (URLLC).

Les codes correcteurs d'erreurs permettent la correction des erreurs de transmission sur des canaux de communication peu fiables. Différentes familles de codes correcteurs ont été proposées dans la littérature dont les plus utilisées de nos jours sont présentées dans le chapitre I. Ce rappel permet de préciser les concepts nécessaires à la bonne compréhension des différentes contributions faites dans le cadre de la thèse. Ce premier chapitre aborde aussi les codages LDPC qui fournissent des performances en correction des erreurs de transmission proches de la limite théorique de Shannon. Ces algorithmes usuels employés pour le décodage de codes LDPC sont décrits dans le chapitre I. Les différents ordonnancements des calculs et les différentes architectures de décodeurs associées sont détaillés et analysés. En effet, ces dernières présentent différents niveaux de parallélisme et différents niveaux de performances. Afin de réduire la latence de décodage tout en conservant les performances en correction des erreurs de transmission, l'utilisation d'algorithmes sous-optimaux s'avère nécessaire. De plus, l'impact de la structure de donnée adopté pour la mémorisation des différents paramètres de décodeur LDPC est mis en évidence. À partir de l'ensemble de ces éléments, une méthode de conception spécifique est proposée afin de permettre l'implantation logicielle de décodeurs LDPC sur des systèmes embarqués en fonction des contraintes applicatives.

Une première étude est détaillée dans le chapitre II. Il s'agit d'une étude conjointe des codes LDPC et des algorithmes de décodage à entrées souples afin d'identifier un couple code/algorithme fournissant un compromis efficace entre performances et complexité calculatoire, sur lequel des optimisations sont proposées pour réduire davantage la latence de décodage. Cette étude est réalisée sur un environnement de simulation et de profilage que nous avons développé dans le cadre de cette thèse. Cet environnement organisé d'une façon modulable est capable de simuler n'importe quel code LDPC afin d'évaluer ses performances en terme de correction des erreurs de transmission et aussi de simuler leur comportement dans le CPU et la mémoire cache. Les performances simulées montrent que l'algorithme NMSA

associé à un ordonnancement dynamique, constitue le meilleur compromis entre les performances en termes de correction d'erreurs et les performances en termes de complexité. Ce choix de couple algorithme/ordonnancement est le point de départ pour développer les contributions incluses dans cette thèse visant l'implémentation d'un décodeur LDPC permettant d'atteindre une latence et un pouvoir de correction satisfaisant pour le type de communication ciblé. Quatre objectifs spécifiques ont été établis pour atteindre cet objectif principal.

Objectif 1 : Réduire la complexité d'ordonnancement

L'ordonnancement dynamique IDS, permet d'accélérer la vitesse de convergence du processus de décodage vers un mot code valide, cependant les calculs des différentes métriques définissant le choix du nœud de contrôle ou de variable à mettre à jour en premier, augmente la complexité calculatoire et par conséquence la latence de décodage. Dans le cadre de cette thèse on a proposé une méthode d'ordonnancement fixe basé sur la mise à jour des nœuds de variable au sien des équations de parité auxquels ils participent (nœud vaccinaux). Aussi, un prétraitement de la matrice de contrôle de parité H est réalisé afin de détecter et d'éliminer les cycles d'ordre 4 et afin de calculer le critère défini pour l'ordre de mise à jour des nœuds. Ces prétraitements sont réalisés en dehors du processus de décodage ce qui n'impacte par la complexité de décodage. Les résultats obtenus par simulation montrent que la méthode proposée a de bonnes performances en termes de correction d'erreurs par rapport aux travaux de l'état et avec une complexité linéaire au lieu de la complexité quadratique des méthodes d'ordonnancement dynamiques.

Objectif 2 : Réduire la complexité de parcours de la structure de donnée représentant les différents paramètres du code LDPC.

Lors du processus de décodage, les accès aux différents messages échangés conformément à la structure de la matrice de contrôle de parité ne permettent pas un accès consécutif à la mémoire. Par conséquence, la mémorisation de la matrice de contrôle de parité est indispensable. Plusieurs types de structure de donnée ont été étudiés en prenant en compte la taille mémoire qui engendre, ainsi que la complexité calculatoire relative à leur parcours. La méthode de structuration proposée dans le cadre de cette thèse vise à réduire la taille mémoire et à éviter les parcours de ces structures de données lors du processus de décodage. La structure de sonnée proposé consiste à organiser les données à mémoriser dans l'ordre de traitement de la phase horizontal en structurant les messages échangés par ligne, les adresses

mémoires nécessaires durant la phase verticale sont ensuite calculé hors processus de décodage par un prétraitement de la matrice H et enregistré sur un tableau Add utilisé pour accès directe aux messages en évitant le coût calculatoire engendré par les parcours des structures de données. Cette méthode a permis de réduire la complexité de 66% et la taille mémoire de 61%.

Objectif 3 : Réduire le nombre de message échangés lors du processus de décodage

L'émulation du comportement du processus de décodage au niveau processeur et mémoire cache, réalisé en utilisant les outils de profilage Gprof et Valgrind, a permet de constater que la séparation des phases horizontal et vertical ne permet pas de tirer profit de la localité spatial et temporel de la mémoire cache, puisque lors des deux phase espacées dans le temps les mêmes données sont chargés dans la mémoire cache puis enregistré dans la mémoire après traitement, ce qui augmente le nombre des accès mémoires et par conséquence la latence du décodage. Pour réduire davantage le nombre des accès à la mémoire, une optimisation basée sur l'union des deux phases horizontales et verticales en une seule phase est proposée, cette optimisation permet de faire toutes les mises à jour basées sur la valeur de la donnée chargée en processeur avant de chargé la donnée suivante, ce qui permet de réduire le nombre des accès mémoire. Cette optimisation a permis aussi de réduire les données à mémoriser puisque les données mémorisées pour la phase verticale ne sont plus nécessaire. Cette méthode a permis de réduire la complexité de 66% et la taille mémoire de 62% par rapport aux travaux de la littérature.

Objectif 4 : Implémentation embarquée sur une architecture multi processeurs dont le temps induit par le traitement parallèle n'affecte pas la latence visée.

L'émergence des processeurs multi-cœurs embarqué sur des systèmes sur puce, SoC, permet de s'approcher aux performances des solutions matérielles avec une réduction du coût de développement et une adaptation aux différents contextes d'applications. Malgré que les interconnexions entre les nœuds de contrôle et les nœuds de variable, définit par la matrice de contrôle de parité ne permettent pas de lancer des calculs totalement parallèles, la structure de donnée proposée permet le lancement des calculs partiellement parallèle en groupant les nœuds de contrôle indépendant entre eux dans des couches, chaque couche est lancée d'une manière totalement parallèle et le passage d'une couche à une autre se fait séquentiellement.

Les optimisations proposée dans le cadre de cette thèse sont implémentées sur deux plateformes multicoeurs afin d'évaluer la latence sur des systèmes parallèles, la première

plateforme est un HPC Marwan avec Intel Xeon gold 6130 avec 08 CPU à 2.10 Ghz sous unix, la deuxième plate-forme est une carte Raspberry Pi 4 quad-core Cortex-A72 (ARM v8) @ 1.5GHz avec une distribution Linux Raspbien.
L'impact de la répartition des calculs sur plusieurs processeurs a été évalué en comparant la latence obtenue avec un seul processeur et celle obtenue sur plusieurs processeurs, La rapidité du traitement parallèle est 2 fois à 2.8 fois plus importante que le traitement sur un seul processeur. Cependant, lorsque la vitesse d'horloge du processeur est importante et le traitement à lancer en parallèle n'est complexe (cas de la matrice 576 x 288 sur la première plateforme CPU à 2,1 GHz), le temps nécessaire à la création et à la gestion des tâches parallèles est plus importante que les calculs eux-mêmes. Ainsi, l'efficacité du traitement parallèle dépend du degré de parallélisme Z de chaque matrice, de la complexité de décodage exprimé en termes d'éléments non nul dans la matrice, de la vitesse d'horloge des processeurs utilisés et du temps nécessaire pour la création et la synchronisation des threads.

Les optimisations proposées dans le cadre de cette thèse ont permis d'implémenter un décodeur LDPC à petite latence et avec un pouvoir de correction élevé sur un système embarqué utilisé dans plusieurs projets IoT. Par exemple, le code (576, 288) a atteint une latence de 0,15 ms soit avec un taux d'erreur trame de $2x10^{-6}$, ce qui correspond parfaitement aux communications URLLC dont la latence exigé est de 1ms avec un taux d'erreur de 10^{-5} pour un paquet de 32Octets.

D'après les travaux réalisés dans cette thèse ainsi que les résultats obtenus, nous pouvons dégager les perspectives suivantes :

- Déployer une méthode de l'IA, qui permet de prédire les positions du mot code les plus susceptible d'être erronées, avec un auto-apprentissage implémenté en dehors du processus de décodage LDPC.
- Faire appel aux méthodes d'analytique Big data pour la détermination des facteurs de normalisation α qui changent d'une itération à une autre.
- Augmenter le coefficient d'expansion par matrice de contrôle de parité, afin de rendre la section parallèle plus large et diminuer le nombre des passages entre les sections parallèles qui se fait en créant des threads à chaque entrée de la section parallèle, ce qui augmente le temps nécessaire pour la création des threads.
- Utiliser la méthodologie Co-design en implémentant la partie de l'algorithme NMSA optimisé sur matériel (un circuit FPGA), et la partie ordonnancement brassé ainsi que l'organisation en mémoire des variables en software.

Annexe A : ***Liste des contributions***

1 Publications indexées

- ***Mhammed Benhayoun****, Mouhcine Razi, Anas Mansouri, Ali Ahaitouf, "Embedded Parallel Implementation of LDPC Decoder for Ultra-Reliable Low-Latency Communications", Applied Computational Intelligence and Soft Computing, vol. 2023, Article ID 5573438, 12 pages, 2023*
- ***Mhammed Benhayoun****, Mouhcine Razi, Anas Mansouri, and Ali Ahaitouf, Low-Complexity LDPC Decoding Algorithm Based on Layered Vicinal Variable* Node Scheduling, Modelling and Simulation in Engineering 2022, pp. 1-12.
- ***Benhayoun, M.****, Razi, M., Mansouri, A., Ahaitouf, A.,* Efficient memory parity check matrix optimization for low latency quasi cyclic LDPC decoder. Lecture Notes in Electrical Engineering , 2021, 681, pp. 71–79
- *Razi, M.,* ***Benhayoun, M.****, Mansouri, A., Ahaitouf, A.,* An improvement and a fast DSP implementation of the bit flipping algorithms for low density parity check decoder. International Journal of Electrical and Computer Engineering 2021, 11(6), pp. 4774–4784

2 Conférences internationales

- ***Benhayoun, M.****, Razi, M., Mansouri, A., Ahaitouf, A.* Efficient memory parity check matrix optimization for low latency quasi cyclic LDPC decoder, International Conference on Electronic Engineering and Renewable Energy (ICEERE'20)
- ***Benhayoun, M.****, Razi, M., Mansouri, A., Ahaitouf, A.* New memory load optimization approach for software implementation of irregular LDPC Encoder/Decoder.
  2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems, WITS 2019, 2019, 8723841.
- *Razi, M., Mansouri, A.,* ***Benhayoun, M.****, Madi, A.A., Ahaitouf, A.* Fast DSP implementation of a low complexity LDPC decoder.
  2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems, WITS 2019, 2019, 8723838.

3 Workshop internationaux

- ***Mhammed Benhayoun****, Mouhcine Razi, Anas Mansouri, Ali Ahaitouf,* Optimisation de l'implémentation parallèle de l'algorithme à décision souple de décodeur LDPC, International Workshop on **E**mbedded **S**ystem, and **A**pplications (**ESA**'19)
  SIGER Laboratory, USMBA University, Fez, Morocco.
- *Mouhcine Razi,* ***Mhammed Benhayoun****, Anas Mansouri, Ali Ahaitouf,* Comparison and DSP Implementation of the hard decision and soft decision algorithms for LDPC Decoder, International Workshop on **E**mbedded **S**ystem, and **A**pplications (**ESA**'19) SIGER Laboratory, USMBA University, Fez, Morocco.

Annexe B : ***Démonstration de l'équation de mise à jour des messages C2V***

Cette partie de preuve, vise à démontrer que le LLR relatif à la satisfaction d'une équation de contrôle de parité s'écrit en fonction de la fonction $\emptyset(x) = -\ln\left(\tanh\frac{x}{2}\right)$.

La valeur du LLR d'une variable binaire aléatoire *U* est définie par :

$$LLR(u) = L_U(u) = \ln\left(\frac{P_U(u=0)}{P_U(u=1)}\right) \quad (1)$$

Où $P_U(u)$ dénote la probabilité d'avoir *u* comme valeur de *U*.
On a :

$$\frac{P_U(u=0)}{P_U(u=1)} = e^{L_U(u)} \quad (2)$$

$$\frac{P_U(u=0)}{P_U(u=1)} + 1 = e^{L_U(u)} + 1 \quad (3)$$

$$\frac{P_U(u=0) + P_U(u=1)}{P_U(u=1)} = e^{L_U(u)} + 1 \quad (4)$$

Puisque $P_U(u=0) + P_U(u=1) = 1$, on aura :

$$\frac{1}{P_U(u=1)} = e^{L_U(u)} + 1 \quad (5)$$

Donc, la probabilité d'avoir $u = 1$ comme valeur de U, s'exprime en fonction de LLR comme suit :

$$P_U(u=1) = \frac{1}{e^{L_U(u)}+1} \quad (6)$$

Ainsi, en déduit la probabilité d'avoir $u = 0$ comme valeur de U, en fonction de LLR :

$$P_U(u=0) = 1 - \frac{1}{e^{L_U(u)}+1} \quad (7)$$

$$P_U(u=0) = \frac{e^{L_U(u)}}{e^{L_U(u)}+1} \tag{8}$$

Pour deux variables binaires aléatoires statiquement indépendantes U_1 et U_2 on peut écrire :

$$P(u_1 \oplus u_2 = 0) = P(u_1 = 0)\,P(u_2 = 0) + P(u_1 = 1)\,P(u_2 = 1) \tag{9}$$

En remplaçant chaque probabilité avec son expression on obtient alors :

$$P(u_1 \oplus u_2 = 0) = \frac{e^{L(u1)}}{e^{L(u1)}+1}\,\frac{e^{L(u2)}}{e^{L(u2)}+1} + \frac{1}{e^{L(u1)}+1}\,\frac{1}{e^{L(u2)}+1} \tag{10}$$

En faisant les calculs on trouve :

$$P(u_1 \oplus u_2 = 0) = \frac{1+ e^{L(u1)}e^{L(u2)}}{1+ e^{L(u1)}+e^{L(u2)} + e^{L(u1)}e^{L(u2)}} \tag{11}$$

De même, la probabilité que la somme modulo 2 de deux variables soit égal à 1 est donnée par :

$$P(u_1 \oplus u_2 = 1) = 1 - P(u_1 \oplus u_2 = 0) = \frac{e^{L(u1)}+e^{L(u2)}}{1+ e^{L(u1)}+e^{L(u2)} + e^{L(u1)}e^{L(u2)}} \tag{12}$$

Ainsi, le LLR de la somme modulo 2 de deux variables est :

$$L(u_1 \oplus u_2) = \ln\left(\frac{P(u_1 \oplus u_2=0)}{P(u_1 \oplus u_2=1)}\right) = \ln\left(\frac{1+ e^{L(u1)}e^{L(u2)}}{e^{L(u1)}+e^{L(u2)}}\right) \tag{13}$$

Cette équation peut être écrite sous la forme :

$$L(u_1 \oplus u_2) = \ln\left(\frac{(e^{L(u1)}+1)(e^{L(u2)}+1)+(e^{L(u1)}-1)(e^{L(u2)}-1)}{(e^{L(u1)}+1)(e^{L(u2)}+1)-(e^{L(u1)}-1)(e^{L(u2)}-1)}\right) \tag{14}$$

Et puis sous cette forme :

$$L(u_1 \oplus u_2) = \ln\left(\frac{1+\frac{(e^{L(u1)}-1)(e^{L(u2)}-1)}{(e^{L(u1)}+1)(e^{L(u2)}+1)}}{1-\frac{(e^{L(u1)}-1)(e^{L(u2)}-1)}{(e^{L(u1)}+1)(e^{L(u2)}+1)}}\right) \tag{15}$$

Par induction on peut démontrer que pour J variables binaires aléatoires statiquement indépendantes, le LLR de la somme modulo 2 des J variables est donné par :

$$L\left(\sum_{j=1}^{J} \oplus u_j\right) = \ln\left(\frac{1+\frac{\prod_{j=1}^{J}(e^{L(uj)}-1)}{\prod_{j=1}^{J}(e^{L(uj)}+1)}}{1-\frac{\prod_{j=1}^{J}(e^{L(uj)}-1)}{\prod_{j=1}^{J}(e^{L(uj)}+1)}}\right) \quad (16)$$

Cette formule peut être écrite sous la forme :

$$L\left(\sum_{j=1}^{J} \oplus u_j\right) = \ln\left(\frac{1+\prod_{j=1}^{J}\left(\frac{(e^{L(uj)}-1)}{(e^{L(uj)}+1)}\right)}{1-\prod_{j=1}^{J}\left(\frac{(e^{L(uj)}-1)}{(e^{L(uj)}+1)}\right)}\right) \quad (17)$$

Or,

$$\frac{(e^{L(u)}-1)}{(e^{L(u)}+1)} = \tanh\left(\frac{L(u)}{2}\right) \quad (18)$$

L'équation (17) devient :

$$L\left(\sum_{j=1}^{J} \oplus u_j\right) = \ln\left(\frac{1+\prod_{j=1}^{J}\tanh\left(\frac{L(uj)}{2}\right)}{1-\prod_{j=1}^{J}\tanh\left(\frac{L(uj)}{2}\right)}\right) \quad (19)$$

Puisque,

$$\tanh^{-1}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right) \quad (20)$$

L'équation (19) devient :

$$L\left(\sum_{j=1}^{J} \oplus u_j\right) = 2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{L(uj)}{2}\right)\right) \quad (21)$$

Les deux fonctions $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ et $\tanh^{-1}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$ sont des fonctions symétriques impaires, monotones et croissantes alors l'équation peut être réécrite de façon à ce qu'on fasse apparaitre à la fois le signe $sign(L(uj))$ et le module $|L(uj)|$.

$$L\left(\sum_{j=1}^{J} \oplus u_j\right) = \prod_{j=1}^{J}\ sign(L(uj))\ 2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right) \quad (22)$$

Maintenant on va montrer que :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right) = \emptyset\left(\sum_{j=1}^{J}\emptyset(|L(uj)|)\right) \qquad \text{Avec } \emptyset(x) = -\ln\left(\tanh\frac{x}{2}\right)$$

D'après les deux équations (19) et (21) on a :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ \ln\left(\frac{1+\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)}{1-\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)}\right) \quad (23)$$

Cette égalité peut être écrite sous forme :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ln\left(\frac{\frac{1}{\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)}+1}{\frac{1}{\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)}-\ 1}\right) \quad (24)$$

Puis sous cette forme :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ln\left(\frac{e^{-\ln\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)}+1}{e^{-\ln\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)}-1}\right) \quad (25)$$

On a :

$$-\ln\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)=\ -\ \sum_{j=1}^{J}\ln\left(\tanh\left(\frac{|L(uj)|}{2}\right)\right) \quad (26)$$

Donc :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ln\left(\frac{e^{-\ \sum_{j=1}^{J}\ln\left(\tanh\left(\frac{|L(uj)|}{2}\right)\right)}+1}{e^{-\ \sum_{j=1}^{J}\ln\left(\tanh\left(\frac{|L(uj)|}{2}\right)\right)}-1}\right) \quad (27)$$

Soit la fonction $\emptyset(x)=\ -\ln\left(\tanh\left(\frac{x}{2}\right)\right)=\ln\frac{e^{x}+1}{e^{x}-1}$. Cette fonction est décroissante, non linéaire et involutive (car elle est sa propre réciproque $\emptyset(\mathrm{x})^{-1}=\ \emptyset(x)$).

L'égalité (v) devient :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ \ln\left(\frac{e^{\sum_{j=1}^{J}\emptyset(|L(uj)|)}+1}{e^{\sum_{j=1}^{J}\emptyset(|L(uj)|)}-1}\right) \quad (28)$$

Or,

$$\frac{(e^{\sum_{j=1}^{J}\emptyset(|L(uj)|)}-1)}{(e^{\sum_{j=1}^{J}\emptyset(|L(uj)|)}+1)}\ =\ \tanh\left(\frac{\sum_{j=1}^{J}\emptyset(|L(uj)|)}{2}\right) \quad (29)$$

L'égalité (28) peut s'écrire sous la forme :

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right)\ =\ -\ln\left(\tanh\left(\frac{\sum_{j=1}^{J}\emptyset(|L(uj)|)}{2}\right)\right) \quad (30)$$

Or,

$$\emptyset(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right) \tag{31}$$

Donc,

$$2\ \tanh^{-1}\left(\prod_{j=1}^{J}\ \tanh\left(\frac{|L(uj)|}{2}\right)\right) = \emptyset\left(\sum_{j=1}^{J}\emptyset(|L(uj)|)\right) \tag{32}$$

Finalement de (22) et (32) On a :

$$L\left(\sum_{j=1}^{J}\oplus u_j\right) = \prod_{j=1}^{J}\ sign(L(uj))\emptyset\left(\sum_{j=1}^{J}\emptyset(|L(uj)|)\right) \tag{33}$$

Références bibliographiques

[1] 3GPP TR 38.913 V17.0.0 (March 2022) requirements page 24 https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996

[2] C. E. Shannon. “A Mathematical Theory of Communication”. In: The Bell System Technical Journal 27.4 (Oct. 1948), pp. 623–656. doi: 10.1002/j.1538-7305.1948.

[3] Richard Hamming Error-detecting and error-correcting codes Bell Syst. Tech. J. 29 pp 147 à 160 1950

[4] Peterson, W. W.; Brown, D. T. (January 1961). "Cyclic Codes for Error Detection". Proceedings of the IRE. 49 (1): 228–235.

[5] R.C. Bose and D.K. Ray-Chaudhuri, On a class of error correcting binary group codes, In.Control 3 (1960), 68–79. 15

[6] G. Reed, I.S. Solomonn, Polynomial codes over certain finite fields, Journal of the society for industrial Applied Mathematics 8 (1960), 300–304.

[7] P. Elias, « Coding for Noisy Channels », IRE conv. Rec., vol. 3, pt. 4, pp. 37–46, 1955.

[8] A.J. Viterbi. Error bound for convutional codes and an asymptotically optimum algorithm. Transactions on Information Theory, 13(2) :260–269, 1967.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima, «Near shannon limit error-correcting coding and decoding: Turbo-codes,» Proceedings of the IEEE International Conference on Communications (ICC), pp. 1064–1070, 1993.

[10] D. J. C. MacKay and R. M. Neal. “Good Codes Based on Very Sparse Matrices”. In: IMA International Conference on Cryptography and Coding (IMA-CCC). UK: Springer, Dec. 1995, pp. 100–111. doi: 10.1007/3-540-60693-9_13 (cit. on pp. 8, 12).

[11] R. Gallager. “Low-Density Parity-Check Codes”. In: IRE Transactions on Information Theory 8.1 (Jan. 1962), pp. 21–28. doi: 10.1109/TIT.1962.1057683 (cit. on pp. 8, 12).

[12] R. Tanner, “A Recursive Approach to Low Complexity Codes,”IEEE Trans. Information Theory, vol. 27, no. 5, pp. 533-547, Sept.1981

[13] A. Shokrollahi. “Raptor Codes”. In: International Symposium on Information Theory (ISIT). IEEE, June 2004, pp. 36–. doi: 10.1109/ISIT.2004.1365073 (cit. on p. 9).

[14] E. Arikan, «Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,» IEEE Transactions on Information Theory, vol. 55, n. 7, pp. 3051–3073, 2009.

[15] C. -L. Ho, "Analysis of threshold of regular and irregular LDPC codes using Gaussian approximation," 2009 Wireless Telecommunications Symposium, Prague, Czech

Republic, 2009, pp. 1-7, doi: 10.1109/WTS.2009.5068932.

[16] G. Lechner, "The effect of cycles on binary message-passing decoding of LDPC codes," 2010 Australian Communications Theory Workshop (AusCTW), Canberra, ACT, Australia, 2010, pp. 43-47, doi: 10.1109/AUSCTW.2010.5426759.

[17] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low density parity check codes," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 51, no. 6, pp. 1106–1113, Jun. 2004.

[18] Y. Dai and Z. Yan, "Optimal overlapped message passing decoding for quasi-cyclic low-density parity-check codes," in Proc. Global Telecomun. Conf. (Globecom), Dec. 2005, vol. 4, pp. 2395–2399.

[19] IEEE. "IEEE Std. 802.16e-2005 and IEEE Std.802.16-2004/Cor1-2005",2006.

[20] R.L. Richardson, T.G. Urbanke, Efficient encoding of low-density parity-check codes, IEEE Transactions on information theory 47 (2001), no. 2, 638–656.

[21] Zhu, Yufei & Xing, Zuocheng & Li, Zerun & Zhang, Yang & Hu, Yifan. (2021). High Area-Efficient Parallel Encoder with Compatible Architecture for 5G LDPC Codes. Symmetry. 13. 700. 10.3390/sym13040700.

[22] Digital Video Broadcasting DVB, Second Generation Framing structure, Channel Coding and Modulation systems for Broadcasting, Interactive Services News Gathering and other broadband satellite applications, jun 2004.

[23] hard et soft decision performances

[24] Erl-Huei Lu, Tso-Cho Chen and Pen-Yao Lu, "Theoretic approach to BP-based WBF decoding algorithm of LDPC codes," International Symposium on Wireless and pervasive Computing (ISWPC), Taipei, Taiwan, 2013, pp. 1-4, doi: 10.1109/ISWPC.2013.6707424.

[25] J. Zhang, and M. P. C. Fossorier, «A modified weighted bit-flipping decoding of low-density parity-check codes,» IEEE Communications Letters, vol. 8, n. 3, pp. 165-167, 2004.

[26] T. Wadayama, et al., «Gradient descent bit flipping algorithms for decoding LDPC codes,» IEEE transactions on communications, vol. 58, n. 6, pp. 1610-1614, 2010.

[27] T. Phromsa-ard, et al., «Improved gradient descent bit flipping algorithms for LDPC decoding,» chez Digital Information and Communication Technology and it's Applications (DICTAP), IEEE Xplore, Bangkok, Thailand, 2012.

[28] J. PEARL, «Reverend Bayes on inference engines: a distributed hierarchical approach,» Proceedings of the National Conference on Artificial Intelligence, 1982.

[29] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke. "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit, IEEE

Commun. Letters, vol. 5, pp. 58-60, February 2001.

[30] https://www.bibmath.net/dico/index.php?action=affiche&quoi=./t/tcl.html

[31] M. P. C. Fossorier, M. Mihaljevic and H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation", IEEE Trans. Commun., vol. 47, no. 5, pp. 673-680, May 1999.

[32] J. Chen and M.P.C. Fossorier, «Density Evolution for Two improved BP-Based Decoding Algorithms of LDPC Codes,» IEEE Communications Letters, vol. 6, n. 5, p. 208–210, 2002.

[33] Tran-Thi, Bich & Thien Truong, Nguyen & Hong, Hoa & Hoang, Trang. (2021). An Improved Offset Min-Sum LDPC Decoding Algorithm for 5G New Radio. 106-109. 10.1109/ISEE51682.2021.9418782.

[34] W. Ji, M. Hamaminato, H. Nakyama, and S. Goto, «Self-Adjustable Offset Min-Sum Algorithm For ISDB-S2 LDPC Decoder,» IEICE Electronis Express, vol. 7, n. 17, p. 1283–1289, 2010.

[35] Hu, X.Y., Eleftheriou, E., Arnold, D.M., Dholakia, A., «Efficient implementations of the sum-product algorithm for decoding ldpc codes,» GLOBECOM'01. IEEE Global Telecommunications Conference, San Antonio, TX, USA, 2001.

[36] A. Ait Madi, A. Mansouri, A. Ahaitouf, «A New PWL Approximation for the Self-Adjustable Offset Min-Sum Decoding With a Highly Reduced-Complexity» International Journal of Computer Applications, vol. 61, n. 19, pp. 1-6, 2003.

[37] A. Darabiha, A. Carusone, F. Kschischang, «A bit-serial approximate min-sum LDPC decoder and FPGA implementation» chez IEEE International Symposium on Circuits and Systems, ISCAS, 2006.

[38] F. Angarita, J. Valls, V. Almenar, V. Torres, «Reduced-complexity min-sum algorithm for decoding LDPC codes with low error-floor» IEEE Tran. Circuits Syst. I Regul. vol. 61, n. 7, p. 2150–2158, 2014.

[39] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," IEEE Trans. Commun., vol. 53, no. 2, pp. 209–213, Feb 2005.

[40] A. I. V. Casado, M. Griot, and R. D.Wesel, ``Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in Proc. IEEE Int. Conf. Commun., Glasgow, U.K., Jun. 2007, pp. 932_937.

[41] A. I. V. Casado, M. Griot, and R. D. Wesel, ``LDPC decoders with informed dynamic scheduling," IEEE Trans. Commun., vol. 58, no. 12, pp. 3470_3479, Dec. 2010.

[42] J. Kim, M. Nam, and H. Song, "Variable-to-check residual belief propagation for LDPC codes," IET Electronics Letters, vol. 45, no. 2, pp. 117–118, Jan. 2009.

[43] C. Aslam, Y. Guan, K. Cai, and G. Han, “Low-complexity belief propagation decoding via dynamic silent- variable- node- free scheduling,” IEEE Commun. Lett., vol. 21, no. 1, pp. 28–31, Jan. 2017.

[44] Y. Gong, X. Liu, W. Ye and G. Han, “Effective informed dynamic scheduling for belief propagation decoding of LDPC codes,”IEEE Trans. Commun., vol. 59, no. 10, pp. 2683-2691, Oct. 2011.

[45] X. Liu, Y. Zhang and R. Cui, “Variable-node-based dynamic scheduling strategy for belief-propagation decoding of LDPC codes,”IEEE Commun. Lett., vol. 19, no. 2, pp. 147-150, Feb. 2015.

[46] OX, G.E.P, M.E. MULLER, A Note on the Generation of Random Normal Deviates, Annals Math, 58.

[47] https://hpc.marwan.ma/index.php/fr/

[48] C. Kun, S. Qi, L. Shengkai and P. Chengzhi, "Implementation of encoder and decoder for LDPC codes based on FPGA," in Journal of Systems Engineering and Electronics, vol. 30, no. 4, pp. 642-650, Aug. 2019

[49] Chen, W.; Zhao, W.; Li, H.; Dai, S.; Han, C.; Yang, J. Iterative Decoding of LDPC-Based Product Codes and FPGA-Based Performance Evaluation. Electronics 2020, 9, 122.

[50] Tamkeen, S.A., Hamad, A.A. (2021). FPGA Implementation of Scaled “Quasi-Cyclic LDPC” Decoder Using High-Level Synthesis. In: Peng, SL., Hao, RX., Pal, S. (eds) Proceedings of First International Conference on Mathematical Modeling and Computational Science. Advances in Intelligent Systems and Computing, vol 1292. Springer, Singapore.

[51] Verma, A.; Shrestha, R. Low Computational-Complexity SOMS-Algorithm and High-Throughput Decoder Architecture for QC-LDPC Codes. IEEE Trans. Veh. Technol. 2022, 1–14.

[52] Ling, J.; Cautereels, P. Fast LDPC GPU Decoder for Cloud RAN. IEEE Embed. Syst. Lett. 2021, 13, 170–173.

[53] Li, A., Jiang, Q., Xie, K., Wang, M., Li, L. and Luo, W. (2020), Low latency LDPC hard-decision algorithm for 5G NR. IET Circuits Devices Syst., 14: 229-234.

[54] Dai J, Yin H, Lv Y, Xu W, Yang Z. Multi-Gbps LDPC Decoder on GPU Devices. Electronics. 2022; 11(21):3447.

[55] A. Kharin, S. Vityazev, E. Likhobabin and V. Vityazev, "LDPC decoder implementation on DSP+ARM platform with OpenCL" 2018 7th Mediterranean Conference on Embedded Computing (MECO), 2018, pp. 1-4.

[56] Razi, Mouhcine & Benhayoun, Mhammed & Mansouri, Anass & Ali, Ahaitouf. (2021). An improvement and a fast DSP implementation of the bit flipping algorithms for low

density parity check decoder. International Journal of Electrical and Computer Engineering (IJECE). 11. 4774. 10.11591/ijece.v11i6.pp4774-4784.

[57] Introduction to Open MP tutorial https://www.openmp.org/wp - content/uploads/Intro_To_OpenMP_Mattson.pdf

[58] https://www.cs.tufts.edu/comp/150PAT/tools/gprof/gprof.pdf

[59] https://valgrind.org/docs/manual/valgrind_manual.pdf

[60] Scott, David. (2011). Box–Muller transformation. Wiley Interdisciplinary Reviews: Computational Statistics. 3. 10.1002/wics.148.