Les Turbo-Codes à Roulettes[1][2]

D. GNAEDIG[1], E. BOUTILLON[2], M. JEZEQUEL[3], V. C. GAUDET[4] et P. G. GULAK[5]

[1] TurboConcept, Technopôle Brest-Iroise, 115 rue Claude Chappe, 29280 Plouzané
[2] LESTER, Université de Bretagne Sud, BP 92116, 56321 Lorient Cedex
[3] ENST Bretagne, Technopôle Brest-Iroise, BP 832, 29285 Brest Cedex
[4] Dpt. of ECE, University of Toronto, 10 King's College Toronto, Ontario, Canada M5S 3G4
[5] Dpt. of ECE, University of Alberta, Edmonton, Alberta, Canada T6G 2V4

david.gnaedig@univ-ubs.fr, emmanuel.boutillon@univ-ubs.fr,
michel.jezequel@enst-bretagne.fr, vgaudet@ee.ualberta.ca, glenn.gulak@eecg.toronto.edu

Résumé – Le problème majeur dans l'implémentation matérielle d'un turbo-décodeur réside dans le manque de parallélisme des algorithmes de décodage MAP. Cet article propose un nouveau procédé de turbo-codage basé sur deux idées: le codage de chaque dimension par *P* codes convolutifs récursifs circulaires indépendants et des contraintes sur la structure de l'entrelaceur qui permet de décoder en parallèle les *P* codes convolutifs dans chaque dimension. La construction des codes constituants et de l'entrelaceur est décrite et analysée. Un haut degré de parallélisme est obtenu avec des performances équivalentes ou meilleures que les meilleurs turbo-codes connus. L'architecture parallèle du décodeur permet de réduire la complexité du turbo-décodeur pour des applications à très hauts débits.

Abstract – *The main problem with the hardware implementation of turbo codes is the lack of parallelism in the MAP-based decoding algorithm. This paper proposes to overcome this problem with a new family of turbo codes, called Slice Turbo Codes. This family is based on two ideas: the encoding of each dimension with P independent tail-biting codes and a constrained interleaver structure that allows parallel decoding of the P independent codewords in each dimension. The optimization of the interleaver is described. A high degree of parallelism is obtained with equivalent or better performance than the best known turbo codes. The parallel architecture allows reduced complexity turbo decoding for very high throughput applications.*

1. Introduction

Les architectures de turbo-décodeur généralement utilisées pour les applications haut débit effectuent une itération sur un mot de code de taille *N* en *N* cycles symboles. La réalisation de *I* itérations en temps réel nécessite alors la mise en série de *I* décodeurs. Ce schéma est relativement inefficace en terme de complexité matérielle car les mémoires des informations extrinsèques se trouvent ainsi dupliquées *I* fois [1]. Certains auteurs proposent de paralléliser le décodage des codes convolutifs de chaque dimension du turbo-code. Pour ce faire, ils divisent chaque trame en *P* segments de taille égale, afin de traiter indépendamment chaque segment par un décodeur MAP. Dans ces architectures, les effets de bord aux extrémités de chaque segment sont gérés de façon sous-optimale soit par l'utilisation d'un préambule [2], soit par le passage d'un pointeur donnant les états initiaux de chaque segment entre deux itérations [3]. Ces papiers abordent le décodage dans une seule dimension sans aborder le problème des conflits mémoires pouvant résulter de l'entrelacement.

Dans cet article, nous proposons une nouvelle famille de turbo-codes dans laquelle le codage dans les deux dimensions est réalisé par *P* codes Convolutifs Récursifs Systématiques Circulaires [4] (CRSC) indépendants que nous appelons des roulettes. Par une structure adaptée de l'entrelaceur, le décodage en parallèle de *P* roulettes devient réalisable sans conflits mémoire. Notons qu'une architecture parallèle similaire a été proposée indépendamment pas Dobkin *et al.*, mais associé à un turbo-code convolutif classique [5]. La structure de l'entrelaceur est similaire à celle présentée dans ce papier, mais son optimisation n'est pas décrite.

L'article est structuré en quatre parties. Après avoir présenté le principe des turbo-codes à roulettes, nous discuterons du choix de l'entrelaceur et de son influence sur les performances. Enfin, nous présenterons la réduction de complexité que permettent les turbo-codes à roulettes ainsi que les performances obtenues.

2. Principes des Turbo-Codes à roulettes

Les turbo-codes à roulettes sont construits de la façon suivante. La trame d'information de *N* symboles *m*-binaires est découpée en *P* blocs de *M* symboles chacun, avec $N = M \cdot P$. Le turbo-code est noté (N, M, P). Comme pour un turbo-code classique, l'opération de codage est d'abord effectuée dans l'ordre naturel afin de générer la redondance dans la première dimension. Chaque bloc est alors codé de manière indépendante par un code CRSC. Ces codes constituants sont appelés roulettes par analogie avec la technique de fermeture du treillis « tail-biting » ou « circulaire » utilisée. La trame d'information est ensuite permutée par un entrelaceur de taille *N* symboles. La trame entrelacée est également découpée en *P* blocs de longueur *M* et chaque bloc est codé indépendamment par un code CRSC pour produire la redondance de la deuxième dimension. Un poinçonnage est éventuellement appliqué pour obtenir le rendement de codage désiré.

[1] Brevet francais N°0204764, 16 avril 2002.

[2] Cette étude a été partiellement financée par la région Bretagne (Projet Virtual Turbo, PRIR n°A2C622).

L'entrelaceur est construit conjointement à l'organisation de la mémoire de façon à permettre le décodage en parallèle des *P* roulettes. En clair, sa structure permet de lire et écrire, à chaque cycle symbole *k*, les *P* données nécessaires aux *P* décodeurs des *P* bancs mémoires BM_1, BM_2,…, BM_P sans conflits. En effet, une seule lecture peut être effectuée au même instant dans une mémoire simple port : pour effectuer *P* accès en parallèle, *P* bancs mémoires sont alors nécessaires. Grâce à la solution proposée dans cet article, le degré de parallélisme peut être affiné suivant les besoins de l'application. Les avantages de ce parallélisme seront discutés dans la partie 4.

La section suivante présente la construction de l'entrelaceur, garantissant la contrainte de parallélisme et de bonnes performances.

3. Construction de l'entrelaceur

La méthodologie de conception de l'entrelaceur s'appuie sur la même méthode que celle proposée dans [6] : la structure de l'entrelaceur est associée à une architecture matérielle qui permet un décodage en parallèle.

3.1 Structure de l'entrelaceur

La figure 1 présente la structure d'entrelacement utilisée pour construire un code à roulettes garantissant le parallélisme de décodage.

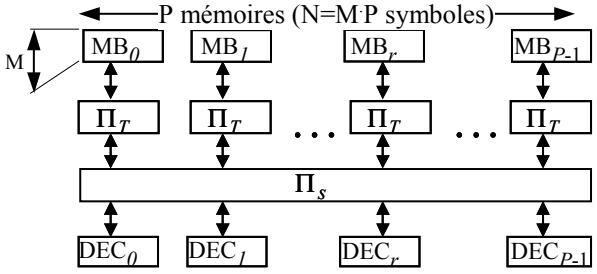


FIG. 1: Structure de l'entrelaceur

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

$$\Pi(k) = \Pi(t,r) = \Pi_S(t,r) \cdot M + \Pi_T(t,r) \qquad (1)$$

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

Afin de simplifier l'implémentation matérielle, la permutation temporelle est choisie identiquement pour tous les bancs mémoires. Ainsi $\Pi_T(t,r) = \Pi_T(t)$ dépend uniquement de l'indice temporel *t*. Cette solution présente l'avantage au niveau implémentation de ne nécessiter qu'un seul calcul d'adresse pour lire *P* données des *P* bancs mémoires. Les *P* bancs mémoires peuvent alors être fusionnés en une seule mémoire, afin de réduire encore la surface.

La permutation spatiale permet ensuite de transférer les *P* données lues à chacun des *P* décodeurs (nommés DEC dans la figure 1), le décodeur *r* recevant, à l'instant *t*, la donnée du banc mémoire $\Pi_S(t,r)$. Pour tout *t* fixé, la fonction $\Pi_S(t,r)$ est donc une bijection de l'indice $r \in \{0,...,P-1\}$ des décodeurs à l'ensemble $\{0,...,P-1\}$des indices des bancs mémoires.

Afin de maximiser le brassage des symboles entre les ordres entrelacé et naturel, nous contraignons la fonction $\Pi_S(t,r)$de façon à ce que *P* symboles consécutifs d'une même roulette proviennent de *P* bancs mémoires distincts. Ainsi, pour *r* fixé, la fonction $\Pi_S(t,r)$ est bijective et *P*-periodique. Elle est temporellement bijective sur l'ensemble $t \in \{0,..., P-1\}$ dans l'ensemble $\{0,..., P-1\}$des indices des bancs mémoires. Elle est *P*-périodique sur l'indice temporel, c'est-à-dire que $\forall t, \forall j$ tel que $t + j \cdot P < M$, $\Pi_S(t + j \cdot P, r) = \Pi_S(t,r)$.

Des permutations spatiales bijectives et *P*-périodiques vont être utilisées dans la suite de l'article.

3.2 Un exemple simple

Construisons un simple code (12,6,3) afin de clarifier la construction de l'entrelaceur. Choisissons la permutation temporelle $\Pi_T(t) = \{1,4,3,2,5,0\}$ et pour la permutation spatiale un décalage circulaire d'amplitude $A(t \bmod 3)$, où la roulette d'indice *r* est associée au banc mémoire d'indice $\Pi_S(t,r) = (A(t \bmod 3) + r) \bmod 3$, avec $A(t \bmod 3) = \{2,0,1\}$. La permutation spatiale est alors bijective et 3-periodique.

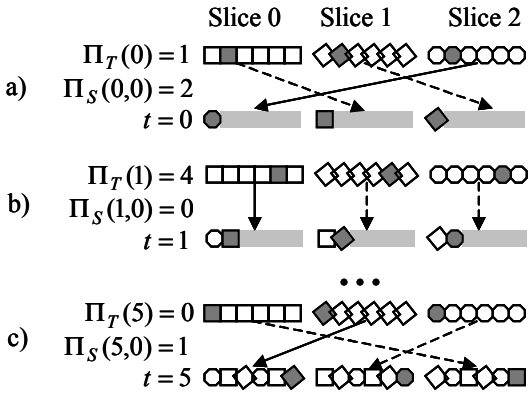


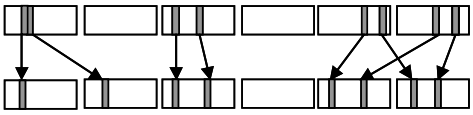
FIG. 2 : Un exemple simple d'un code (18,6,3).

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

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

3.3 Optimisation de l'entrelaceur

L'optimisation de l'entrelaceur vise à répondre à deux critères de performances. Le premier est la distance minimale pour les performances asymptotiques du code à haut rapport signal sur bruit. Le second, est une convergence rapide, c'est à dire obtenir la maximum de gain de codage en peu d'itérations à bas rapport signal sur bruit. La convergence est dégradée par les corrélations entre les informations extrinsèques, induites par la présence de cycles courts dans l'entrelaceur. Les cycles qui apparaissent avec la plus grande influence sont les cycles primaires et secondaires présentés sur la figure 3. Lorsque ces cycles correspondent à des combinaisons de séquences d'entrée de poids de Hamming faibles conduisant à des mots de codes du code CRSC de poids faible, ils sont appelés Motifs d'Erreur Primaire et Secondaire (MEP, MES).



a) pas de cycles b) cycle primaire c) cycle secondaire

FIG. 3 : Cycles primaire et secondaire

L'écart entre 2 symboles est défini par :

$$S(k_1,k_2)=\|k_1-k_2\|+\|\Pi(k_1)-\Pi(k_2)\| \qquad (2)$$

où $\|a-b\|$ est égal à $\min\left(|a-b|, M-|a-b|\right)$ dans le cas où les symboles sont dans la même roulette et est égal à l'infini dans le cas contraire. Cette définition provient de l'utilisation de codes convolutifs circulaires. L'écart minimal global est alors défini par $\min_{k_1,k_2}[S(k_1,k_2)]$.

Les MEPs de faible poids sont éliminés avec un écart minimal élevé. Comme la permutation spatiale est *P*-périodique et bijective, 2 symboles distants de moins de *P* symboles dans l'ordre entrelacés ne sont pas dans la même roulette dans l'ordre naturel: leur écart est alors infini. La permutation temporelle choisie est une permutation régulière qui maximise l'écart *S* entre les symboles distants de *P* dans l'ordre entrelacé.

Comme le poids des MESs n'augmente pas avec un écart minimal élevé, nous optimisons la permutation spatiale afin de maximiser le poids de ces motifs. Nous avons tout d'abord mis au point un outil permettant, pour un entrelaceur donné, de détecter tous les MEPs et MESs et leur poids de Hamming respectif. Cet outil permet d'exhiber de nombreux MESs de poids faible avec une permutation spatiale trop régulière. Afin de réduire le nombre de ces MESs, nous introduisons de l'irrégularité dans la permutation spatiale, ce qui par conséquent améliore la convergence du code.

Lorsque la taille du code est importante, l'étude des seuls MEPs et MESs n'est pas suffisant pour obtenir un entrelaceur efficace. En effet, d'autres motifs d'erreurs plus complexes apparaissent et ceux-ci conditionnent la distance minimale. En pratique, la modélisation et l'énumération exhaustive de ces motifs est trop complexe à mettre en oeuvre. Nous effectuons donc un post-traitement sur la permutation temporelle afin d'augmenter la distance minimale du code. Elle est évaluée par la méthode impulsionnelle d'évaluation de la distance minimale proposé par Berrou *et al.* [8].

4. Implémentation matérielle

Pour des implémentations haut-débit, une architecture série classique [1] nécessite pour chaque itération une duplication des mémoires et des parties combinatoires. Chaque roulette de chaque dimension est décodée par un décodeur à entrée souple / sortie souple (SISO) travaillant à D_S Mbits/s. Dans ce cas, pour atteindre un débit de D Mbits/s en effectuant I itérations il est nécessaire de dupliquer les SISOs $I \cdot D / D_S$ fois. Il en est de même pour les mémoires. Ainsi, la surface A_{TD}^S de l'architecture série est donnée par :

$$A_{TD}^S = I \cdot \frac{D}{D_S} \cdot 2 \cdot \left[A_{SISO} + Mem_E + Mem_I\right] \qquad (3)$$

où A_{SISO} représente la surface d'un SISO, Mem_I et Mem_E les surface des mémoires pour mémoriser respectivement les sorties du canal et les informations extrinsèques.

Dans notre architecture parallèle, au contraire, avec une organisation mémoire permettant d'effectuer $2 \cdot I \cdot D / D_S$ SISOs en parallèle, la surface relative aux mémoires peut être sortie de la parenthèse dans l'équation (3). Ainsi seule la partie combinatoire est dupliquée et chaque SISO décode indépendamment des autres une roulette dans chaque dimension. Comme les treillis des roulettes sont circulaires, après chaque itération, les états finaux des treillis deviennent les états initiaux de la même roulette. Les effets de bords comme dans [2][3] sont ainsi supprimés et l'algorithme de décodage SISO est optimal. Dans l'architecture parallèle, le même matériel (mémoires et SISOs) sont réutilisés 2 fois pour chaque itération, afin d'effectuer successivement le décodage des 2 dimensions du code. Un buffer additionnel est nécessaire pour permettre de mémoriser les sorties du canal de la trame pendant le décodage de la trame courante. La surface A_{TD}^P de l'architecture parallèle est alors donnée par :

$$A_{TD}^P = 2.Mem_I + Mem_E + I \cdot \frac{D}{D_S} \cdot 2 \cdot A_{SISO} \qquad (4)$$

Le tableau 1 compare les complexités des 2 architectures en utilisant les équations (3) et (4) et les surfaces d'un SISO et des mémoires données par la synthèse RTL en technologie 0.13µm. Le code duo-binaire (2048,256,8) (soit 4096 bits) utilisé est décodé avec 8 itérations de l'algorithme Log-MAP [9]. Le SISO fonctionne à un débit de 100 Msymboles/s (avec un horloge de 100 Mhz, ce qui est conservateur pour un

ASIC), correspondant au double du débit requis pour le décodeur (50 Msymboles/s, soit 100 Mbits/s). Comme dans l'architecture série classique la mémoire dupliquée représente 60% de la surface totale, notre architecture parallèle est 2 fois moins complexe que l'architecture série. L'écart est encore plus important avec des blocs plus longs ou avec plus d'itérations.

TAB. 1 : Évalutation de la complexité (en technologie 0.13µm) pour un code de 4 kbits de rendement 1/2, décodé avec 8 itérations et D/D_S=1/2.

	A_{SISO} 0.3 mm²	Mem_E 0.25 mm²	Mem_I 0.2 mm²	A_{TD} in mm²
Série	8→2.4mm²	8→2.0mm²	8→1.6mm²	6
Parallèle	8→2.4mm²	1→0.25mm²	2→0.4mm²	3.05

5. Résultats

En appliquant les différents méthodes d'optimisation développées dans la partie 3, nous construisons un code duo-binaire (2048,256,8). Une permutation intra-symbole est également appliquée pour augmenter la distance minimale [7]. Ces codes de rendement 1/2 sont comparés sur la figure 5 avec un code (2048,2048,1), construit avec une seule roulette en utilisant les équations du code DVB-RCS [7] avec les paramètres $P_0 = 45$ et $\{P_1, P_2, P_3\} = \{36,52,16\}$. Sa distance minimale est de 20. En optimisant uniquement la permutation temporelle (code TO sur la figure 5), la distance minimale de 14 provient de motifs d'erreurs secondaires et non de motifs primaires pour lesquels la distance minimale est supérieure à 30. En optimisant la permutation spatiale (SO), les motifs d'erreurs primaires et secondaires de faible poids sont éliminés et la distance minimale augmente à 18. En effectuant un post-traitement (PP) sur la permutation temporelle, la distance minimale est augmentée à 21.

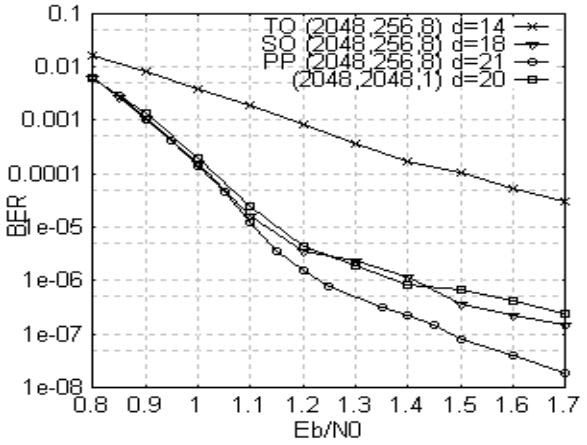


FIG. 4: Performance of the (2048,256,8) duo-binary codes and (2048,2048,1) code for 8 iterations.

Les résultats de simulation sont en accord avec les distances minimales des codes. Pour un taux d'erreur binaire de 10^{-7}, le code (2048,256,8) optimisé a un gain de 0,3 dB par rapport au code DVB-RCS. Les codes proposés dans ce standard sont de taille 48 à 1728 bits et l'entrelaceur n'a pas été construit pour des plus grandes tailles de trames. Pour des tailles de trames plus courtes, les turbo-codes à roulettes ont des performances identiques au code DVB-RCS aussi bien en distance minimale qu'en convergence.

6. Conclusion

Nous avons présenté une nouvelle famille de turbo code convolutif. L'entrelacement ainsi que les codes constituants proposés permettent un décodage en parallèle, adapté à des réalisations haut débit de faible complexité. L'architecture parallèle proposée permet une réduction de la complexité de 50% à 100 Mbits/s par rapport à une architecture série classique. Les simulations de performances montrent que l'introduction de parallélisme ne dégrade pas les performances, et un bon entrelaceur peut même les améliorer pour de grandes tailles de trames.

References

[1] C. Berrou, A. Glavieux and P. Thitimajshima. *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes*. Proc. ICC'93, Geneva, Switzerland, p. 1064-1070, Mai 1993.

[2] Z. Wang, Z. Chi and K.K. Parhi. *Area-Efficient High Speed Decoding Schemes for Turbo/MAP Decoders*. IEEE Trans. on VLSI Systems, 10(12), Dec. 2002.

[3] A. Worm, H.Lamm and N Wehn. *VLSI architectures for high speed MAP decoders*. Proc. 14th Int Conf VLSI Design, pp 446-453, 2001.

[4] C. Berrou, C. Douillard and M. Jézéquel. *Multiple parallel concatenation of circular recursive systematic codes*. Annales des Télécommunications, tome 54, n°3-4, pp 166-172, 1999.

[5] R. Dobkin, M Peleg and R. Ginosar. *Parallel VLSI Architecture for MAP Turbo Decoder*. PIMRC 2002, Lisboa, Portugal, Sep. 2002.

[6] E. Boutillon, J. Castura and F.R. Kschischang. *Decoder-first code design*. Proc. of the 2nd International Symposium on Turbo Codes and Related Topics, pp 459-462, Sept 2000.

[7] DVB-RCS Standard. *Interaction channel for satellite distribution systems*. ETSI EN 301 790, V1.2.2, pp. 21-24, Dec 2000.

[8] C.Berrou, S.Vaton, M.Jezequel and C.Douillard. *Computing the minimum distance of linear codes by the error impulse method*. GLOBECOM 2002, Taipei, Taiwan, Nov 2002.

[9] P. Robertson, E. Villebrun and P. Hoeher. *A Comparison of Optimal and Sub-optimal Decoding*

Algorithm in the Log Domain. *Proc ICC*, Seattle, WA,
pp. 1009-1013., June 1995