Dix-septième colloque GRETSI, Vannes, 13-17 septembre 1999

Organisation de la mémoire dans un turbo décodeur utilisant l'algorithme SUB-MAP

Adodo Dingninou, Fathi Raouafi, Claude Berrou

Département d'électronique - ENST-Bretagne
BP 832, 29285 Brest Cedex, France
Adodo.Dingninou@enst-bretagne.fr, Fathi.Raouafi@enst-bretagne.fr
Claude.Berrou@enst-bretagne.fr

Résumé – L'implémentation de l'algorithme MAP(ou BCJR, APP,...) requiert une mémoire de sauvegarde des probabilités qui peut être de taille importante. Des procédés de turbo décodage utilisant peu de mémoire, tout en gardant de bons pouvoirs de correction, ont été définis. Ces derniers ont été validés par simulation (C, VHDL). Les structures suggérées offrent au concepteur un panorama de solutions techniques pour implémenter un turbo décodeur à partir de l'algorithme SUB-MAP.

Abstract – The implementation of the algorithm MAP (or BCJR, APP,...) requires a backup memory of probabilities which can be of significant size. Processes of turbo decoding using little memory, while keeping good capacities of correction, were defined. These latters were validated by simulation (C, VHDL). The suggested structures offer to the designer a panorama of technical solutions to implement a turbo decoder from the algorithm SUB-MAP.

Introduction

Les turbo codes [1] sont construits soit à partir de la concaténation parallèle de deux codes convolutifs systématiques et récursifs, avec un entrelacement non uniforme, soit à partir de deux codes en blocs linéaires selon le principe des codes produits [2]. Les turbo codes convolutifs font l'objet de cet article. Le décodage de ces codes nécessite un algorithme fournissant des décisions pondérées. Un turbo décodeur à sorties pondérées utilisant le SOVA (Soft Output Viterbi Algorithm) a été déjà réalisé [3]. Il peut être également implémenté à partir de l'algorithme MAP (Maximum à Posteriori) [4] moyennant quelques simplifications. Pour diminuer la complexité de cet algorithme, une approximation dite SUB-MAP [5], un peu moins performante, peut lui être substituée. Des architectures de décodage minimisant la taille de la mémoire des métriques, indépendamment de la rapidité du décodage, sont détaillées dans ce document.

1 Description de l'algorithme SUB-MAP

L'algorithme SUB-MAP (ou encore Dual Viterbi Algorithm) met en oeuvre des additions, des comparaisons et des sélections en chacun des noeuds du treillis dans les deux sens de progression dénommés "aller" et "retour" (Fig.1). Les valeurs cumulées (métriques au sens de Viterbi) calculées en chacun de ces noeuds dans l'un des sens sont mémorisées afin qu'à l'étape suivante, dans l'autre sens, des décisions puissent être prises sur l'ensemble des traitements du bloc. Soit $M_k^a(m)$ la métrique calculée dans le sens "aller" pour un état m et un temps discret k et $M_k^r(m)$, la métrique calculée dans le sens "retour". Ces valeurs sont données par les relations de calcul suivantes :

$$M_k^a(m) = Min\left\{M_{k-1}^a(m') + \gamma_k\right\}_{m',i} et$$
$$M_k^r(m) = Min\left\{M_{k+1}^r(m') + \gamma_k\right\}_{m',i}$$

γ_k représente la métrique de transition au temps discret k, m l'état courant, m' l'état précédent, i le bit concerné (0 ou 1) et les grandeurs M les métriques cumulées.
Le logarithme du rapport de vraisemblance LLR (Logarithm Likelihood Ratio) relatif à la donnée d_k devient :

$$\Lambda(d_k) = \frac{1}{2}\left[Min\left\{M_{k,i=0}^a(m) + M_k^r(m)\right\}_m\right] - \frac{1}{2}\left[Min\left\{M_{k,i=1}^a(m) + M_k^r(m)\right\}_m\right]$$

Soit un bloc de longueur N à décoder selon l'algorithme SUB-MAP défini ci-dessus. Les métriques cumulées aux noeuds du treillis sont initialisées de manière équiprobable lorsque l'état de départ du treillis n'est pas connu (l'état de départ est connu au moins dans un sens). Pour prendre les décisions sur l'ensemble du bloc, on doit mémoriser les métriques cumulées minimales au cours du premier traitement choisi ("retour" ou "aller").

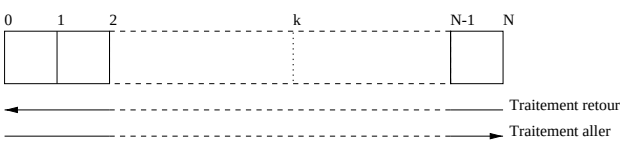

Fig. 1: Opérations sur une trame de taille N

Cette mémorisation est dans la suite effectuée lors du traitement "retour" afin de pouvoir prendre les décisions dans l'ordre normal des données (sens "aller"). Pour chaque échantillon k, on calcule $2*(2^\nu)$ métriques (2^ν métriques pour la transition correspondant à $d_k =' 0'$, idem pour la transition correspondant à $d_k =' 1'$). ν est la mémoire du code et la quantité 2^ν représente le nombre de noeuds ou encore la hauteur du treillis. Seules les valeurs minimales aux noeuds du treillis sont conservées à savoir 2^ν métriques. Les trames de données dans l'ordre naturel et dans l'ordre entrelacé sont concaténées en une nouvelle trame de longueur double sur laquelle sont effectués les traitements. L'état final du treillis lors du décodage de la trame de données dans l'ordre naturel correspond à celui de départ lors du décodage de la trame de données dans l'ordre entrelacé (auto-concaténation). La taille de mémoire nécessaire pour une trame de longueur N reçue est alors $C = N.2^{\nu+1}.q$, q étant le nombre de bits de quantification des métriques. Pour une hauteur de treillis $l = 2^\nu = 16$ et une longueur de trame $N = 2000$, on obtient une capacité mémoire de l'ordre de 640 Kbits pour des métriques quantifiées sur 10 bits. Dans la plupart des cas, il est nécessaire de réduire cette mémoire. Les paragraphes suivants fournissent quelques solutions de réduction de mémoire applicables quels que soient le nombre et l'organisation des unités de calcul ACS (Addition-Comparaison-Sélection) utilisées.

2 Réduction de la mémoire par pré-traitement sur le bloc entier

Un moyen pour réduire la taille de la mémoire consiste à faire précéder le premier traitement ("retour") d'un traitement d'initialisation R0 dans le même sens. La trame des données à décoder est constituée du bloc de données reçues de taille N suivie de celui de ces mêmes données dans l'ordre entrelacé également de taille N. Soit L un diviseur de $2*N$: $2*N = pL$. Les traitements "retour" et "aller" sont subdivisés en p sous-traitements respectivement R_k et A_k (k allant de 1 à p). Ces derniers (R_k ou A_k) étant effectués successivement sur des blocs de L données (indices $(k-1)L$ à $kL-1$), la capacité de mémoire nécessaire pour le calcul des décisions est alors $C_1 = L.2^\nu.q$. Les sous-traitements "aller" et "retour" sont réalisés soit alternativement avec un seul processeur ACS (R0 suivi de R1, de A1, ..., de R_k, de A_k, ..., de R_p et de A_p), soit simultanément avec deux processeurs ACS (R0 suivi de R1, de A1 et R2, ..., de A_{k-1} et R_k, ..., de A_{p-1} et R_p et de A_p). L'utilisation de deux unités ACS donne un décodeur plus rapide et réduit légèrement la capacité de mémoire nécessaire.

Pendant R0, on mémorise périodiquement (période L) les métriques cumulées minimales en chaque noeud du treillis (ceci est symbolisé par les petits rectangles sur le parcours de R0 sur Fig.2). Cela nécessite une mémoire de capacité $C_2 = (p-2).2^\nu.q$. Elles serviront de métriques d'initialisation pour les sous-traitements "retour" qui suivront au cours de la même itération. Ainsi, le traitement "retour" est effectué sur des fenêtres successives de

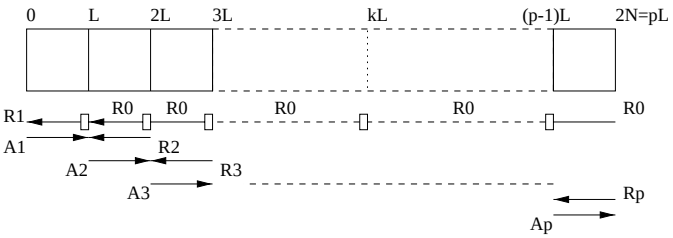

FIG. 2: Traitements sans modification du SUB-MAP

taille L tout en connaissant l'état de départ de chaque sous-traitement. L'utilisation d'un seul processeur ACS implique la sauvegarde successive des 2^ν métriques calculées sur la dernière donnée de l'intervalle de taille L, au cours des sous-traitements "aller", afin d'assurer la continuité de l'ensemble du traitement effectué dans ce sens. La capacité totale requise pour le cas ci-dessus est $C = (p+L-1).2^\nu.q$ contre $C = (p+L-2).2^\nu.q$ lorsque deux processeurs sont utilisés. Ces valeurs sont minimales pour $p = L = \sqrt{N}$, ce qui, comparées à la capacité trouvée au paragraphe 1, donne une réduction dans un rapport de l'ordre de $\sqrt{N}/2$ qui peut être très important.

3 Réduction de la mémoire par pré-traitement sur une fenêtre

Cette solution, fournie par [6], est basée sur la propriété de convergence du treillis [7]. Les traitements "retour" et "aller" sont, comme au paragraphe 2, effectués successivement sur des intervalles de L données. Les valeurs d'initialisation des sous-traitements "retour" ne sont plus issues d'un pré-traitement sur l'ensemble du bloc. Dans ce cas, on fait précéder chaque sous-traitement "retour" d'un pré-traitement sur un bloc de nL données. Ceci revient à faire des traitements "retour" successifs, initialisés de manière uniforme, sur une fenêtre de $(1+n)L$ données décalée à chaque fois de L données suivant le sens "aller". Les métriques calculées dans le sens "retour" sur les nL dernières données des intervalles servent uniquement à assurer une certaine convergence du treillis. Plus n est grand, plus les calculs effectués sur les L premières données sont précis. Les métriques cumulées minimales provenant de ces derniers calculs seront sauvegardées en vue de la prise de décision lors du sous-traitement "aller" sur le même bloc de L données. La capacité totale requise est alors $C = L.2^\nu.q$.

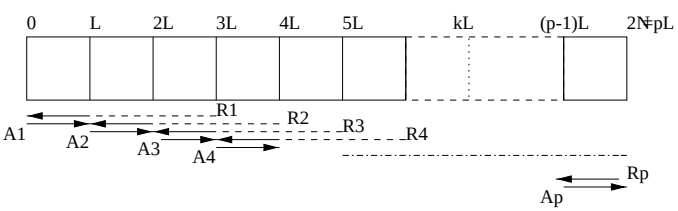

FIG. 3: Traitements sur des fenêtres ($n = 2$)

Les n derniers sous-traitements "retour" sont effectués sur des blocs proportionnels au nombre de données restant à traiter.

Cette solution réduit fortement la mémoire mais perd en rapidité. De bonnes performances impliquent des traitements sur une fenêtre de données assez importante. Sa mise en oeuvre avec un nombre de processeurs ACS inférieur à $(n+1)$ introduit une mémoire supplémentaire de $2^{\nu}.q$. L'utilisation de $(n+1)$ processeurs ACS, très coûteuse en surface, permet de contourner le problème de la rapidité du décodage.

4 Solution intermédiaire

Cette solution dérive de [6] et utilise la mémorisation périodique. Elle nécessite de ce fait plus de mémoire que la solution précédemment décrite, mais seulement deux processeurs ACS pour rendre le décodage plus rapide. La mémoire est organisée de la même manière qu'au paragraphe 2 ; la capacité totale nécessaire est $C = (p+L-1).2^{\nu}.q$ pour la solution avec un processeur ACS et $C = (p+L-2).2^{\nu}.q$ pour celle avec deux processeurs. Les sous-traitements "retour" sont effectués successivement sur des blocs de L données et les métriques cumulées qui y sont calculées sont par la suite mémorisées pour le calcul des décisions ($C_1 = L.2^{\nu}.q$). Chaque sous-traitement "retour" est suivi d'un traitement "aller" sur le même bloc de données.
Au cours de la première itération, ces sous-traitements "retour" sont initialisés de manière uniforme et arbitraire (l'état de départ n'étant pas connu).

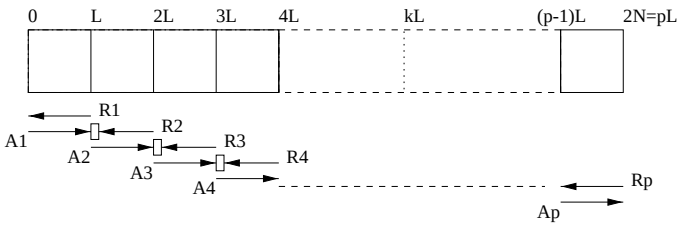

FIG. 4: Structure intermédiaire

Les 2^{ν} métriques minimales calculées sur la première donnée de l'intervalle de taille L dans chacun des sous-traitements "retour" sont conservées ; ceci nécessite une capacité $C_2 = (p-2).2^{\nu}.q$. Elles serviront de métriques d'initialisation pour ces traitements à partir de la seconde itération (les métriques calculées pendant le traitement R_k servant à l'initialisation de R_{k-1} à l'itération suivante).
Cette méthode est la plus efficace et aussi la plus rapide car elle ne fait intervenir ni de traitement d'initialisation ni de traitement forçant la convergence du treillis.

5 Simulations et résultats

Les figures 5 et 6 présentent les résultats des simulations pour un code à seize états utilisant les générateurs 37 (récursivité), 25 (parité). Ce code est poinçonné (une redondance sur deux est retenue) et de rendement $\frac{1}{2}$. La longueur de la trame de données est $N = 680$. Le décodage des données se fait dans l'ordre naturel et dans l'ordre entrelacé. La longueur effective de la trame décodée est alors $M = 1360$ avec $p = 85$ et $L = 16$. Les informations extrinsèques calculées sur les données dans l'ordre entrelacé sont utilisées lors des calculs sur les données dans l'ordre naturel. L'inverse est vrai.
La figure 5 donne les courbes de taux d'erreurs en fonction du rapport signal à bruit du MAP, du SOVA, du SUB-MAP, de la solution intermédiaire ("SI"), et de la solution avec pré-traitement sur une fenêtre ($n = 2$ pour "PF_48" et $n = 4$ pour "PF_80"). Le nombre d'itérations maximal est fixé à 7 et un critère d'arrêt basé sur la comparaison à un minimum des décisions pondérées est utilisé.

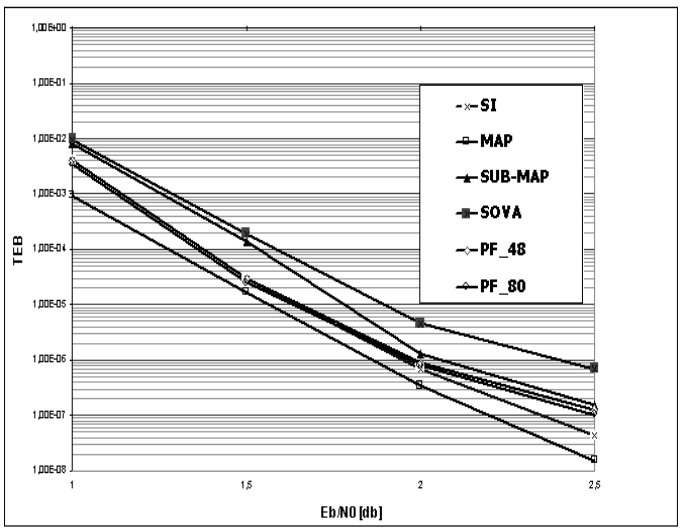

FIG. 5: Courbes de taux d'erreurs binaires en fonction du rapport signal à bruit

A partir de ces courbes, on peut noter que le SUB-MAP ainsi que les diverses solutions dérivées donnent de meilleures performances que le SOVA. Mais contrairement aux attentes, les deux dernières solutions dérivées du SUB-MAP donnent de meilleurs résultats que ce dernier. Ces performances se rapprochent de celles du MAP pour les forts rapports signal à bruit. La solution proposée au paragraphe 3 donne de bons résultats dès $n = 2$. L'amélioration de ces résultats est négligeable pour un n supérieur 2. Une taille de fenêtre de données égale $3L$ suffira pour réaliser cette solution.
La solution intermédiaire, quant à elle, donne des résultats assez proche de ceux de la solution précédente. Ses performances deviennent meilleures pour des rapports signal à bruit supérieurs à 2 db.
Pour mieux qualifier les écarts de performance entre le SUB-MAP et la solution intermédiaire, leur évolution au fur et à mesure des itérations a été étudiée. Les différentes courbes de taux d'erreurs suivant le nombre d'itérations effectué sont données par la figure 6 (Sx pour le SUB-MAP et Ix pour la solution intermédiaire, x étant le nombre d'itérations). Aucun critère d'arrêt n'est utilisé dans ce cas, le processus de turbo décodage est arrêté une fois le nombre d'itérations fixé atteint. On remarque qu'à la première itération les courbes des deux architectures sont pratiquement confondues. Les écarts sont plus nets dès la seconde itération.

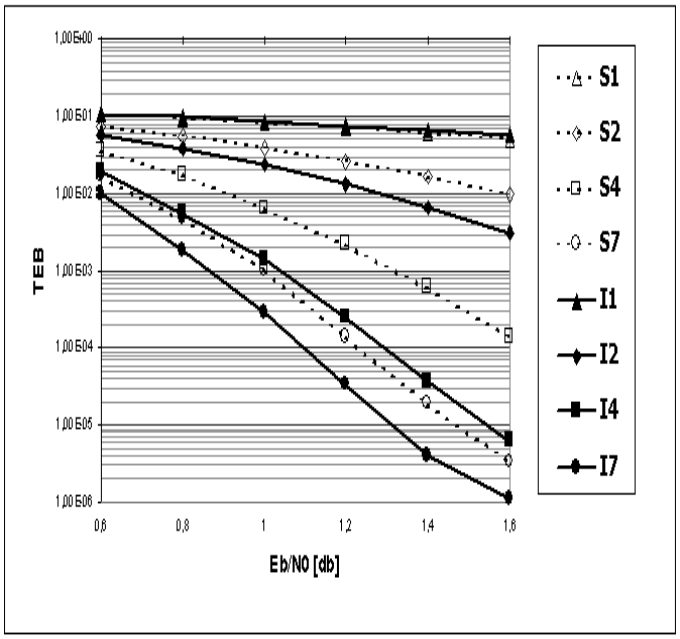

FIG. 6: Évolution de la courbe du taux d'erreurs de la solution intermédiaire suivant les itérations

En se rapportant à ces résultats, la solution décrite au paragraphe 4 donne tout au moins pour les caractéristiques de turbo code utilisé dans ces tests, de meilleures performances comparée au SUB-MAP (avec ou sans réorganisation de la mémoire n'influant pas sur les résultats). Elle semble être la meilleure alternative pour la réalisation de circuit turbo décodeur. Elle propose le meilleur compromis réduction de la mémoire, rapidité et encombrement, tout en garantissant de très bonnes performances.

Conclusion

Les méthodes de mémorisation qui ont été présentées dans ce document, conduisent à une forte réduction de la mémoire nécessaire à la mise en oeuvre de l'algorithme MAP ou SUB-MAP. Elles ont notamment permis la réalisation d'un turbo décodeur sur un circuit prédiffusé programmable.

Références

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error correcting coding and decoding : turbo-codes," in *IEEE ICC '93*, vol. 2/3, (Geneva), pp. 1064–1070, May 1993.

[2] C. Berrou, A. Glavieux, and R. Pyndiah, "Turbo-codes : Principes et applications," in *Quinzième Colloque sur le Traitement du Signal et des Images*, (Juan-les-Pins), pp. 1369–1376, Sep. 1995.

[3] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low complexity soft-output viterbi decoder architecture," in *IEEE ICC '93*, vol. 2/3, (Geneva), pp. 737–740, May 1993.

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Info. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[5] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. Telecommun.*, vol. 8, pp. 119–125, Mar-Apr. 1997.

[6] A. J. Viterbi, "An intuitive justification and simplified implementation of map decoder for convolutional codes," *IEEE Select. Areas in Commun.*, vol. 16, pp. 260–264, Feb. 1998.

[7] F. Hemmati, "Convolutional code structure and the performance of the viterbi algorithm," Master's thesis, Illinois Institute of Technology, 1977.