GESTION DE LA MEMOIRE POUR L'ALGORITHME DU FORWARD-BACKWARD

E. Boutillon[1], **W.J. Gross**[2], **G. Gulak**[2]

[1]*Département COMELEC, Ecole Nationale Supérieure des Télécommunications (ENST)*
46 rue Barrault, 75634 Paris Cedex 13

[2]*Department of Electrical and Computer Engineering, University of Toronto*
10 king's College Road, Toronto, Ontario, M5S 3G4, Canada

RESUME- L'algorithme du Forward-Backward (FB) permet de décoder un code convolutif de façon optimale. Néanmoins, la réalisation matérielle de cet algorithme est complexe et des versions sous-optimales du FB sont généralement utilisées. Dans cet article, nous étudions l'un des aspects critiques de la réalisation matérielle du FB : la mémoire LIFO (Last In First Out) utilisée pour ré-ordonner les résultats de la récursion Backward. Nous proposons différentes organisations des calculs permettant de minimiser la taille de cette mémoire et de rendre sa réalisation par des registres avantageuses. Ces optimisations déplacent le compromis performances-complexité à l'avantage du FB.

ABSTRACT - The Forward-Backward algorithm (FB) performs the optimal decoding of convolutional codes. The drawback is that the hardware realization of this algorithm is complex and sub-optimal versions of the FB are usually used. In this paper, we focus on one of the critical problems of hardware realization of the FB algorithm : the LIFO (Last In First Out) memory that re-order data generated during the backward recursion. We propose several organizations of the computations that minimize the size of this memory. Thanks to this optimization, the implementation of this memory using registers is possible. The reduced complexity of the FB unit makes it much more attractive.

1. INTRODUCTION

L'algorithme du Forward-Backward (FB) — appelé aussi algorithme MAP (Maximum A Posteriori)— suscite actuellement beaucoup d'intérêt. En effet, cet algorithme permet de décoder un code convolutif en générant la fiabilité exacte des bits décodés et de nombreuses applications (Turbo-Code, en particulier [1]) peuvent en tirer profit. Néanmoins, la réalisation matérielle de cet algorithme est complexe et des versions sous-optimales du FB sont généralement utilisées [4]. Schématiquement, l'algorithme FB peut-être compris comme la fusion des informations (les métriques de nœuds) générées par deux décodeurs de Viterbi modifiés travaillant en sens inverse sur les données. Cette formulation simplifiée fait néanmoins apparaître une difficulté inhérente à cet algorithme : comment faire coïncider dans le temps des données générées à des instants différents ? Ce problème est proche de celui de la gestion de la mémoire survivante dans l'algorithme de Viterbi. Ainsi, dans [5], le même ordonnancement des calculs que celui de la méthode du "trace-back" de [8] est utilisé. Cet article présente différentes organisations des calculs qui permettent de réduire signifi-cativement la taille de la mémoire nécessaire pour stocker les métriques de nœuds. Cette réduction rend avantageuse l'implémentation de la mé-moire en utilisant de simples registres plutôt que des blocs de mémoire RAM. Outre une réduction de surface, la conception du bloc FB (et son transfert dans d'autres circuits), est facilitée.

Le principe du FB est expliqué à partir d'un exemple simple dans la section 2. Sa formulation dans le domaine des logarithmes est donnée dans la section 3. La section 4 présente l'organisation des calculs utilisée dans [5]. Enfin, dans la dernière section, différentes

architectures originales permettant (au prix d'un accroissement maîtrisé de la puissance de calcul) de réduire la taille de la mémoire sont proposées.

2. ALGORITHME FB SUR UN EXEMPLE

Considérons un codeur convolutif constitué d'une unique bascule initialisé à l'état 0 à l'instant k=0. Supposons que quatre bits $\{u_k\}_{0 \bullet k<4}$ entrent dans le codeur. Supposons, de plus, que la valeur du dernier bit de cette séquence, u_3, soit connue et égale à zéro. Les 4 symboles $\{x_k\}_{k=0..3}$ générés par le codeur convolutif sont transmis dans un canal bruité. Le récepteur reçoit alors les symboles $\{y_k\}_{0 \bullet k<4}$, avec $y_k = x_k + b_k$, ou b_k est le bruit ayant affecté le $k^{\text{ième}}$ symbole x_k transmis.

L'algorithme FB permet de décoder de façon "souple" chaque bit reçu. Pour cela, à partir de l'observation de l'ensemble des $\{y_k\}_{k=0..3}$ reçus, le FB calcule le logarithme du rapport de vraisemblance du bit décodé[1] (Log-Likelyhood Ratio (LLR), en anglais) :

$$LLR(u_k) = \ln\left(\frac{P(u_k = 1 / \{y_l\}_{0 \le l < 4})}{P(u_k = 0 / \{y_l\}_{0 \le l < 4})} \right) \qquad (1)$$

Le signe de $LLR(u_k)$ donne la valeur du bit décodé et sa valeur absolue indique la qualité de la décision.

L'algorithme FB utilise la représentation en treillis de l'évolution temporelle du codeur. Le treillis est représenté par un graphe dont l'axe des abscisses figure le temps et l'axe des ordonnées figure les différents états du registre à décalage (ici, deux états). Ainsi, à chaque instant k, une colonne constituée de 2 noeuds (s_0 et s_1) est associé aux 2 états (0 ou 1) que peut prendre le registre à décalage à cet instant. Les transitions possibles entre les nœuds de deux instants successifs sont représentées par des branches.

La première étape du décodage consiste à calculer les probabilités "a priori" de chaque transition du codeur à partir des symboles reçus. Le résultat de ce calcul, non détaillé dans cet article, est présenté dans la figure 1 : il s'agit du nombre placé sur chaque branche. Ainsi, par exemple, la réception du symbole y_0 nous informe que la probabilité "a priori" que le codeur emprunte la branche (s^0_0 ,s^1_0) est de $\gamma_0(s_0$,$s_0)$=4/11.

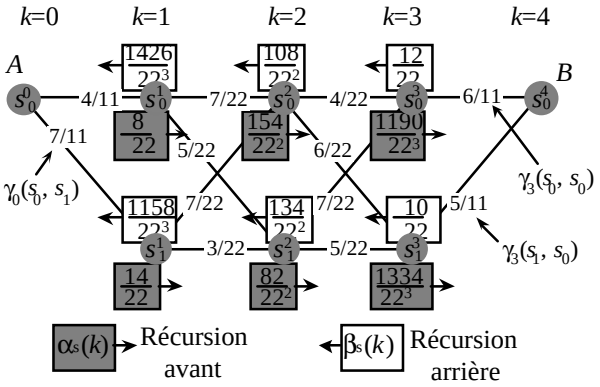

Figure 1 : Récursion "Forward" et "Backward"

L'algorithme FB procède en trois étapes : récursion avant (ou *Forward*), récursion arrière (ou *Backward*), et enfin, fusion des données. La récursion "avant" consiste à calculer les probabilités $\alpha_k(s)$ de passer par le nœud s à l'instant k à partir des informations obtenues en parcourant le treillis du point A (nœud s^0_0) vers le point B (nœud s^4_0). Ainsi, sachant que pour k=0, le point A est certain ($\alpha_0(s_0)$=1), la probabilité $\alpha_1(s_0)$ de passer par le nœud s^1_0 est de 4/11 (elle correspond à $\gamma_0(s_0, s_0)$). De la même façon, $\alpha_1(s_1)$ est égal à $\gamma_0(s_0, s_1)$=7/11. Etudions maintenant la probabilité $\alpha_2(s_0)$ de passer par le nœud s^2_0. La probabilité d'accéder à ce nœud en passant par le nœud s^1_0 est égale au produit de la probabilité associée du nœud s^1_0 par la probabilité de la branche $\gamma_1(s_0$,$s_0)$, soit $56/22^2$. De même la probabilité d'accéder le nœud s^2_0 par le nœud s^1_1 est égale au produit de la probabilité associée à s^1_1 par la probabilité de la branche $\gamma_1(s_1$,$s_0)$, soit $98/22^2$. La probabilité d'atteindre le nœud s^2_0 est donc la somme des probabilités des deux hypothèses, soit $\alpha_2(s_0)=154/22^2$. Un raisonnement analogue pour le nœud s^2_1 nous donne $\alpha_2(s_1)=82/22^2$. D'un point de vue formel, la récursion avant s'exprime par :

$$\alpha_{k+1}(s) = \alpha_k(s_0) \infty \gamma_k(s_0,s) + \alpha_k(s_1) \infty \gamma_k(s_1,s) \qquad (2)$$

[1] Rapport des probabilités que le bit émis soit 1 et 0 connaissant l'observation de l'ensemble des symboles $\{y_i\}_{i=0..N-1}$ reçus.

avec $\alpha_{k-1}(s_0)$ et $\alpha_{k-1}(s_1)$ les probabilités des deux nœuds antécédents du nœud s et $\gamma_k(s_0,s)$et $\gamma_k(s_1,s)$ les probabilités "a priori", à l'instant k, des branches (s_0, s) et (s_1, s).

On peut remarquer que la somme des probabilités $\alpha_k(s_0)$ et $\alpha_k(s_1)$ est inférieure à 1. Sachant, à un instant k donné, que le chemin passe nécessairement par le nœud s_0 ou le nœud s_1, une normalisation des deux probabilités est théoriquement nécessaire. Dans la pratique, cette normalisation peut s'effectuer de façon globale durant la dernière étape de l'algorithme : elle n'est donc pas réalisée immédiatement. Le processus est itéré pour le calcul des probabilités des nœuds s^3_0 et s^3_1. Les résultats de la récursion avant sont indiqués en gris clair dans la figure 1.

La récursion arrière (ou Backward) est identique à la récursion avant, mais cette fois-ci, le treillis est parcouru dans le sens inverse, du point *B* vers le point *A*. Les résultats de la récursion arrière (les probabilités $\beta_S(k)$) sont aussi indiqués dans la figure 1.

L'étape suivante consiste à fusionner les résultats des deux récursions pour calculer la probabilité "a posteriori" des métriques de branches. Prenons le cas de la branche (s^1_0,s^2_0). La récursion avant (qui tient compte de toutes les métriques de branche entre *A* et s^1_0) donne $\alpha_1(s_0)$= 8/22. La récursion arrière (qui tient compte de toutes les métriques de branche entre s^2_0 et *B*) donne $\beta_2(s_0)$=108/22[2]. Sachant que la probabilité "a priori" de la branche (s^1_0,s^2_0) est de $\gamma_1(s_0,s_0)$=7/22, on calcule la probabilité "a posteriori" $\Gamma_1(s_0,s_0)$ de cette branche comme le produit de ces trois dernières probabilités, soit 25,8∞10^{-3}. Dans le cas général, on a :

$$\Gamma_k(s',s) = \alpha_k(s')\infty\gamma_k(s',s)\infty\beta_{k+1}(s) \qquad (3)$$

On applique (3) pour calculer $\Gamma_1(s_0,s_1)$, $\Gamma_1(s_1,s_0)$, et $\Gamma_1(s_1,s_1)$ (respectivement 22,9∞ 10^{-3}, 64,4∞10^{-3} et 24,0∞10^{-3}).

L'arrivée d'un bit 0 (respectivement 1) dans le codeur génère une transition menant à l'état s_0 (respectivement s_1). L'équation (1) se calcule donc par :

$$LLR(u_k) = \ln\left(\frac{\Gamma_k(s_0,s_0)+\Gamma_k(s_1,s_0)}{\Gamma_k(s_0,s_1)+\Gamma_k(s_1,s_1)}\right) \qquad (4)$$

L'utilisation du rapport de vraisemblance supprime la nécessité de normalisation des différentes probabilités. Dans l'exemple considéré, on obtient respectivement $\{LLR(u_k)\}_{k=0..3}$ = {0,35 ; -0,41 ; - 0,07 ; -inf}, ce qui indique que les bits les plus probablement émis sont 1, 0, 0 et 0, avec l'avant dernière décision très peu fiable, et la dernière certaine puisque l'on sait que l'on arrive au nœud s^4_0 à l'instant k=4 (point *B*).

Dans le cas général, le codeur convolutif contient ν registres (soit 2^ν états) et la taille *N* du mot à coder peut-être très grande. Ces changements de dimensions ne modifient pas les mécanismes du FB qui restent identiques.

3. ALGORITHME FB DANS LE DOMAINE DES LOGARITHMES

L'exemple précédent nous a montré que les récursions avant et arrière étaient identiques. Intéressons-nous à cette première. Le cœur opératif de la récursion, donnée dans l'équation (2), pose deux problèmes d'architecture : la complexité des opérateurs de multiplication et le problème de la dynamique des $\alpha_S(k)$. En passant dans le domaine des logarithmes, le cœur opératif devient très proche de celui de l'algorithme de Viterbi [9]. Posons :

$$A_S(k) = \ln(\alpha_k(s)) \qquad (5)$$

$$G_k(s',s) = \ln(\gamma_k(s',s)) \qquad (6)[2]$$

La récursion Forward de l'algorithme FB exprimée dans le domaine des logarithmiques devient alors :

$A_0(s)$=0, si s=0, $A_0(s)$= -• sinon.

Pour k=0..N-2

Pour tous les états s du treillis

$$A_{k+1}(s) = MAX^*((A_k(s'_0)+G_k(s'_0,s), A_k(s'_1)+G_k(s'_1,s)) \qquad (7)$$

---

[2]Dans le cas d'un canal Gaussien, $G_k(s,s')$ est la distance quadratique entre le symbole reçu y_k et le symbole émis par le codeur pour la transition entre l'état s et s'.

avec s'_0 et s'_1 les deux nœuds antécédents du nœud s dans le sens direct du treillis et avec l'opérateur *MAX** effectuant[3] :

$$MAX^*(X, Y)=MAX(X, Y)+\ln(1+e^{-|X-Y|}) \qquad (8)$$

La figure 2 représente le cœur opératif d'une récursion Forward exprimé dans le domaine des logarithmes.

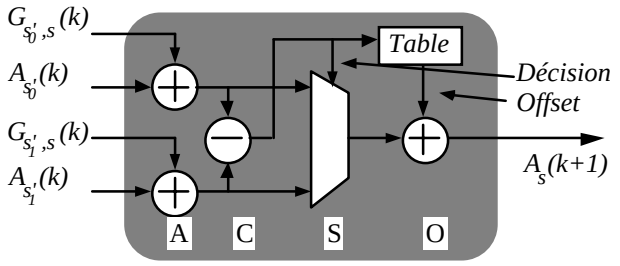

Figure 2 : Représentation de l'opération ACSO

Parmi les quatre opérations effectuées, on retrouve les opérations ACS ("Addition", , "Comparaison" et "Sélection") utilisées dans le décodeur de Viterbi. A ces trois opérations, est ajoutée l'opération "Offset". Cette dernière consiste à ajouter le deuxième terme de l'équation (8). Le facteur correctif à ajouter (l'Offset) est une fonction à décroissance très rapide et seules les faibles valeurs d'entrée donnent un résultat non nul. Il est donc peu coûteux de réaliser cette fonction directement par un simple accès dans une table pré-calculée [7].

La récursion Backward s'exprime de façon exactement symétrique, avec un cœur opératif identique.

Les équations (3) et (4) de l'étape finale se transforment en :

$$\boldsymbol{G}_k(s',s)= A_k(s') + G_k(s',s) + B_{k+1}(s) \qquad (9)$$

$$LLR(u_k)= MAX^*(\boldsymbol{G}_k(s'_1,s_1)) - MAX^*(\boldsymbol{G}_k(s'_0,s_0)) \qquad (10)$$

avec (s'_1,s_1) l'ensemble des couples d'états correspondant à l'arrivée d'un 1 dans le registre à décalage et (s'_0,s_0) l'ensemble des couples d'états correspondant à l'arrivée d'un 0 dans le registre à décalage.

D'après la définition de l'algorithme, un problème de latence de décodage se pose puisqu'il faut théoriquement attendre les N données du message avant de commencer la récursion arrière. Néanmoins, il est montré qu'en partant d'un vecteur B_k égal au vecteur nul B^0 (toutes les métriques de nœuds $\beta_k(s)$ à zéro), après $l > L$ récursions l'état obtenu B_{k-l} devient indépendant[4] de l'état de départ[5]. Cette propriété est mise à profit pour anticiper le décodage et résoudre ainsi le problème de la latence.

4. ARCHITECTURE CLASSIQUE DE FB

Le premier circuit VLSI "haut débit" publié pour l'algorithme FB [5] utilise 3 Unités de Récursions (UR) : deux effectuant la récursion "arrière" (notées RB_1 et RB_2), et une effectuant la récursion avant (noté RA). Chaque UR est constituée de 2^v opérateurs *MAX** travaillant en parallèle de façon à effectuer une étape de récursion en un cycle d'horloge. Les deux unités RB_1 et RB_2 jouent un rôle similaire à celui des deux unités de "trace-back" dans l'algorithme de Viterbi [8]. Utilisons la même représentation graphique que dans [2,5] pour expliquer l'ordonnancement des calculs. Dans la figure 3, l'axe horizontal représente le temps, avec le cycle symbole comme unité. L'axe vertical représente les indices des symboles reçus. Ainsi, la droite ($x=y$) indique qu'à l'instant k, le symbole y_k est reçu et peut donc être traité.

Décrivons, à partir de la figure 3, comment les L symboles $\{y_k\}_{L \bullet k<2L}$ sont décodés (segment I). A $k=3L$, en partant d'un vecteur $B_{3L-1}=B^0$, RB_1 permet, en L cycles de converger et d'atteindre le vecteur B_{2L} (segment II).

---

[3] Posons X=ln(x), Y=ln(y), z=x+y et cherchons à déterminer Z=ln(z) en fonction de X et Y. Supposons tout d'abord X>Y, on a :

$Z = \log(e^X+e^Y) = \log(e^X.(1+e^{Y-X}))= X + \ln(1 + e^{-|X-Y|})$.

En traitant de la même façon le cas ou Y• X, on obtient bien l'équation (8).

[4] à une constante près, dont la valeur s'annule dans le calcul du *LLR* (voir l'équation (8))

[5] L, la longueur de convergence, est de l'ordre de 6.ν.

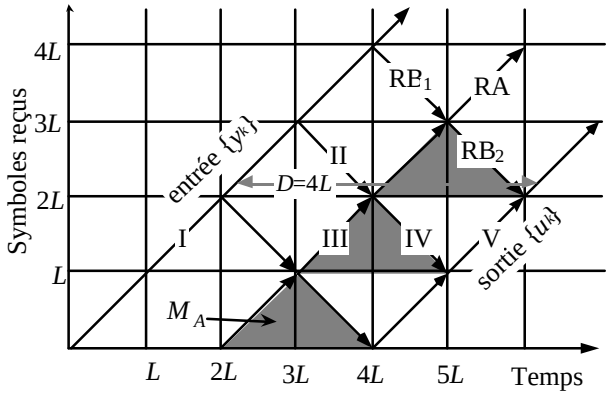

Figure 3 : Architecture (n_A=1, n_B=2, M_A).

Durant ces mêmes L cycles, RA génère les vecteurs $\{A_k\}_{L \bullet k<2L}$ (segment III). Ces L vecteurs sont alors mémorisés jusqu'à leur utilisation (zone en gris). Enfin, entre les instants 4L et 5L-1, RB_2 part de l'état B_{2L} pour calculer B_{2L-1} jusqu'à B_L (segment IV). A chaque cycle, le vecteur A_k correspondant au vecteur B_k calculé est extrait de la mémoire pour obtenir $LLR(u_k)$. Les résultats sont ensuite ré-ordonnés (segment V). Le même processus est ainsi réitéré tous les L cycles. Graphiquement, on obtient aussi :

(i) la latence de décodage : D=4L (distance horizontale entre les flèches "entrée y_k" et "sortie u_k" ;

(ii) le nombre de vecteurs à mémoriser : M=L (taille verticale maximale de la zone en gris) ;

(iii) le nombre de récursions effectuées par cycle symbole : P=3 (nombre de flèches coupées par une ligne verticale.

Dans la suite de l'article, cette architecture sera nommée (n_A=1, n_B=2, M_A), avec n_A et n_B indiquant respectivement le nombre total d'UR avant et arrière utilisés et M_A (ou M_B) indiquant que les vecteurs A_k (ou B_k) sont mémorisés.

D'un point de vue "réalisation matérielle", cette organisation des calculs nécessite de mémoriser M=L vecteurs A_k (zone grise de la figure 1). Considérons, par exemple, une application avec un codeur à 16 états (ν=4), des métriques de nœud codées sur 10 bits et une longueur de convergence d'une valeur de L=6∞ν=24. Chaque vecteur A_k est constitué des 16 métriques de nœuds du treillis, il contient donc 16∞10=160 bits. Avec l'organisation des calculs présentée ci-dessus, une mémoire "24 mots de 160 bits" est nécessaire pour notre exemple. Ce facteur de forme très étiré ne permet pas d'utiliser efficacement des blocs RAMs.

Notons qu'il est aussi possible de mémoriser les vecteurs B à la place des vecteurs A en retardant la récursion RA de 4L cycles par rapport à l'arrivée des données (contre 2L cycles dans la figure 3). Cette organisation des calculs, notée (n_A=1, n_B=2, M_B), permet de générer les bits dans leur ordre naturel et de simplifier légèrement les accès mémoires des données $\{y_k\}$ dans les unités de récursion.

5. ARCHITECTURE PROPOSÉE POUR RÉDUIRE LA TAILLE DE LA MEMOIRE.

D'autres organisations des calculs, proposées par les auteurs, permettent de réduire le nombre M de vecteur à mémoriser. Elles sont présentées ci-dessous.

5.1 Architecture (n_A=1, n_B=2, $M_{(A+B)/2}$)

La récursion avant est retardée de 3L cycles par rapport à la réception des données (Fig. 3). Ainsi, L/2 vecteurs de type A et L/2 vecteurs de types B doivent être mémorisés durant les L/2 premiers cycles d'une itération. Les L données seront alors calculées durant les L/2 cycles restants. Cette architecture est efficace si deux algorithmes de FB doivent être effectués sur le même circuit (cas d'un turbo-decodeur, par exemple). Les deux FB peuvent alors se partager sans conflit les L mots mémoires en étant correctement synchronisés. Comme indiqué dans la figure 4, le deuxième FB (représenté par des flèches en gris et une zone mémoire hachurée), doit effectuer ces opérations avecc un déphasage de L/2 vecteurs par rapport au premier FB. Dans ce cas, le nombre de vecteurs à mémoriser, ramené à chaque unité de FB, n'est plus que de M=L/2.

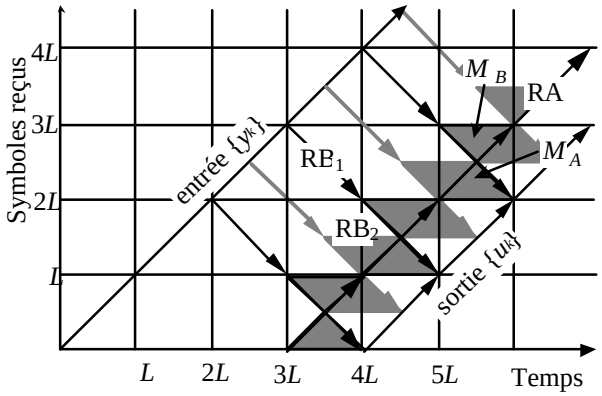

Figure 4 : Architecture (n_A=1, n_B=2, $M_{(A+B)/2}$).

5.2 Architecture (n_A=1, n_B=3, M_B).

Avec une unité de récursion arrière supplémentaire (RB_3), l'organisation des calculs présentée dans la figure 5 permet de diviser par deux la taille de la mémoire M . Elle permet aussi de réduire la latence de décodage de 4L cycles à 3L cycles.

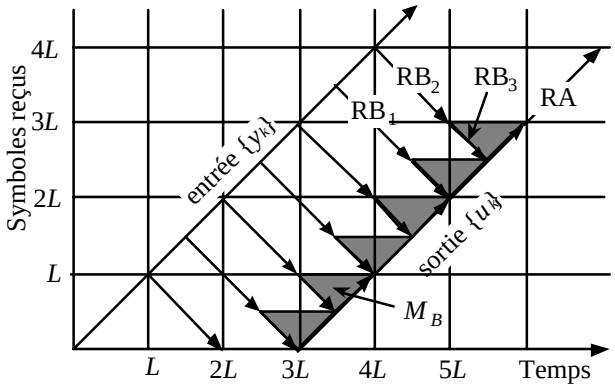

Figure 5 : Architecture (n_A=1, n_B=3, M_B)

Il s'agit ici d'un compromis entre "puissance de calcul" et "puissance de mémorisation".

5.3 Architecture (n_A=2, n_B=2, M_B)

De façon symétrique, il est possible de rajouter une unité de récursion avant RA_2 afin d'obtenir le séquencement indiqué dans la figure 6.

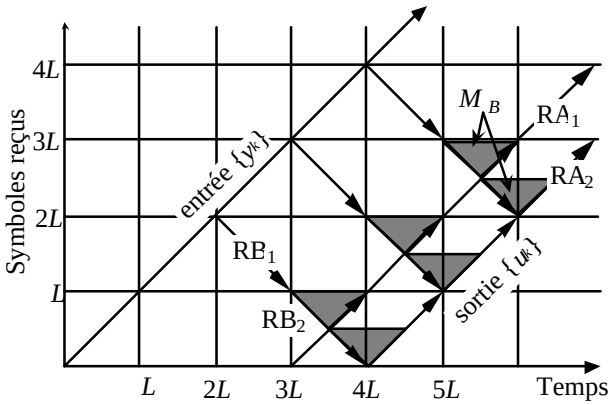

Figure 6 : Architecture (n_A=2, n_B=2, M_B)

La taille M de la mémoire est de nouveau divisée par 2 mais la latence de décodage D reste de 4L cycles. Nous pouvons remarquer que RA_2 ne fait que recalculer, avec L cycles de retard, les résultats de RA_1. En sauvegardant des informations générées par RA_1 (décisions et offsets), il est possible de simplifier d'un facteur 2, la complexité de RA_2 (voir la figure 7).

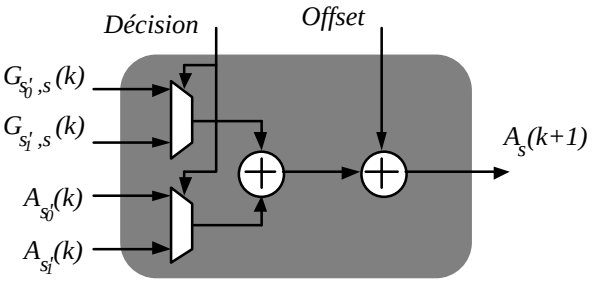

Figure 7: Architecture simplifiée de l'opérateur MAX pour le bloc RA_2.*

De nouveau, il y a un échange entre puissance de calcul et puissance de mémorisation.

5.4 *Architecture (n_A=1, n_B=3, M_B, Pt_{pB}).*

Le paramètre Pt_{pB} de cet algorithme signifie, que sur une période de L cycles, p vecteurs B_k sont mémorisés pour former p pointeurs. L'idée est d'utiliser ces pointeurs comme valeurs initiales d'une troisième unité RB_3 qui recalcule, décalés dans le temps, les mêmes vecteurs B_k que RB_2. La figure 8 illustre le processus avec p=4.

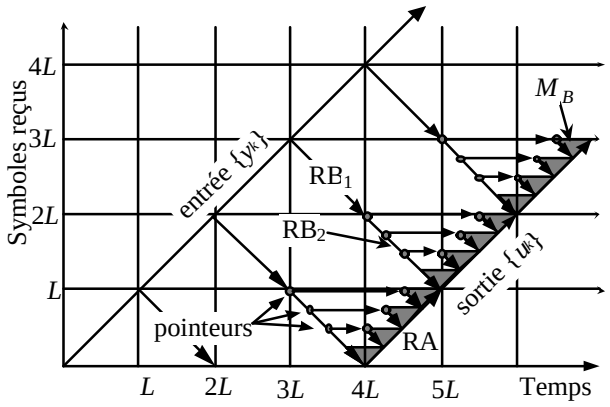

Figure 8 : Architecture (n_A=1, n_B=3, M_B, Pt_{4B})

Dans le cas général, la taille de la mémoire des vecteurs *B* est réduite d'un facteur *p* au prix du coût d'une mémorisation de *p*-1 vecteurs B (les pointeurs) et d'une UR supplémentaire RB_3. Notons que la complexité de ce dernier peut être réduite d'un facteur 2 en utilisant une architecture simplifiée pour l'opérateur MAX^* (figure 6).

6. CONCLUSIONS ET PERSPECTIVES

La table 1 permet de résumer différents compromis "puissance de calcul"[6] *P*, taille de la mémoire *M* et latence de décodage *D* en fonction de la longueur de convergence *L*. Bien sûr, toutes combinaisons des architectures décrites dans cet article sont envisageables. La dernière ligne du tableau propose une architecture (n_A=1, n_B=3, $M_{(A+B)/2}$, Pt_{4B}) qui permet de réduire la taille *M* de la mémoire de *M*=24 (=*L*) à *M*=6 vecteurs. La réalisation de cette mémoire par des bancs de registres devient alors plus avantageuse en surface, en rapidité de conception et en portabilité du bloc (pas de bloc RAM interne). Ainsi, une architecture (n_A=1, n_B=3, M_A, Pt_{4A}) a été choisie pour concevoir [6].

Dans le cas où les UR peuvent réaliser *a* itérations (a•1) par cycle symbole, les méthodes proposée peuvent encore être appliquées pour trouver une solution efficace. Plus de détails sur les architectures du FB peuvent se trouver dans [3].

n_A	n_B	Type mémoire	***P***	***M***	***D***	Fig.[1]
1	n•2	M_A, M_B	1+n	$\frac{L}{n\text{-}1}$	$\frac{2nL}{n\text{-}1}$	3, 5[2]
n•1	2	M_A, M_B	$2{,}5+\frac{n}{2}$	$\frac{L}{n}$	4*L*	6
1	2	$M_{(A+B)/2}$	3	$\frac{L}{2}$	4*L*	4
1	3	M_B,Pt_{pB}	3,5	$\frac{L}{p}+p$	4*L*	8
1	3	$M_{(A+B)/2}$, Pt_{4B}	3,5	$\frac{L}{8}+3$	4*L*	-

[1]Numéro de la figure associée à la configuration.

[2]Figure 3 pour n=2, figure 5 pour n=3.

Table 1 : Résumé des différentes configurations

REFERENCES

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding : Turbo Codes", Proc. ICC'93, Geneva, pp. 1064-70, May 1993.

[2] E. Boutillon, N. Demassieux, "A generalized precompiling scheme for surviving path memory management in Viterbi decoders", ISCAS'93, New-Orleans,. vol. 3, pp. 1579-82, May 1993.

[3] E. Boutillon, W.J. Gross, G. Gulak, "VLSI Architectures for the Forward-Backward algorithm", Submitted as a regular paper to IEEE transactions on communications, Aug.1999.

[4] CAS 5093 Data Sheet, Rev 4.1, Comatlas SA, May 1995.

[5] H. Dawid, H. Meyr, "Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding", PIMRC'95, Vol. 1, pp. 193 -7, New York, 1995.

[6] W.J. Gross et al. "VLSI realization of a turbo-decoder" not published.

[7] W.J. Gross, P. G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders", Electronics Letters, vol.34 n° 16, pp 1577-8, Aug. 1998.

[8] Qualcomm INC, "Q-1401 Single-chip Viterbi decoder", Production Information, Qualcomm Inc, 1987.

[9] A.J. Viterbi, " An intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes ", IEEE Journal of Selected Area in Communications, vol. 16, N°2, pp. 260-264, Feb. 1998.

[6] En nombre d'UR effectués par cycle, avec, une complexité de moitié pour les unités de récursions de complexité réduite.