Réduction de la complexité mémoire pour le décodage à Annulation Successive des codes polaires

Bertrand LE GAL[1], Camille LEROUX[1], Christophe JÉGO[1]

[1]Laboratoire IMS, Bordeaux-INP, Talence, France.
prenom.nom@ims-bordeaux.fr

Résumé – Les Codes Polaires sont une famille de code correcteurs d'erreurs récente. Le décodeur associé à ces codes est le décodeur à annulations successives (AS). Les performances architecturales des décodeurs AS sont limitées par l'encombrement de la mémoire (∼90% de la surface pour une technologie ASIC). Dans cet article, nous proposons deux méthodes pour réduire la complexité mémoire des décodeurs AS. Les dégradations induites sont négligeable sur les performances de décodage (<0.02dB). Les gains en surface obtenus sont en moyenne de ∼25%

Abstract – Polar codes are a new coding scheme that asymptotically achieves the capacity of various communication channels. The corresponding decoder is the successive cancellation (SC) decoder. Architectural performance of SC decoders is limited by the memory complexity. In this paper, two methods are proposed in order to reduce the memory conmplexity of SC decoders. The first method replace memorization by some recomputations. This optimization does not degrade decoding performance. The second method suggests to reduce the dynamic of internal data. This impacts the decoding performance in a rather neglectable manner (<0.02dB), as shown by performed simulations. The association of both methods allows a reduction of ∼25% of the memory complexity.

1 Introduction

Les codes polaires (CP) [1] sont des codes correcteurs d'erreurs récents qui permettent d'atteindre asymptotiquement la capacité de divers canaux de communication. L'algorithme de décodage qui permet d'atteindre ces performances est le décodage par annulation successive (AS). Depuis l'apparition des CPs, diverses architectures de décodeur AS ont été proposées pour améliorer le débit et réduire la complexité calculatoire résultante de l'implantation [2, 3, 4, 5]. Dans [5], un décodeur AS de taille $N = 2^{20}$ a été implémenté sur cible FPGA. Ce décodeur utilise 2% des ressources de calculs et 72% de la mémoire. En technologie ASIC, les résultats de [6] montrent que 90% de l'occupation en surface du circuit est consommée par la mémoire. Ces chiffres montrent clairement la dissymétrie existante entre les coûts de mémorisation et de traitement dans les décodeurs AS. Dans cet article, nous proposons deux méthodes qui permettent de réduire la complexité mémoire de 25 % en altérant de manière infime les performances de décodage.

La première méthode suggère de recalculer la moitié des Logarithmes du Rapport de Vraisemblance (LRV) plutôt que de les mémoriser. Cette approche permet de réduire la complexité nécessaire à la mémorisation des LRV du canal sans altérer les performances de décodage. La seconde méthode réduit la dynamique des LRV sur une partie de la mémoire interne du décodeur. Cela engendre une dégradation des négligeable performances (<0.02dB). La combinaison des deux méthodes permet de réduire la complexité mémoire de 25%. Cette méthode peut s'appliquer au décodage par listes qui souffre également du même symptôme [7].

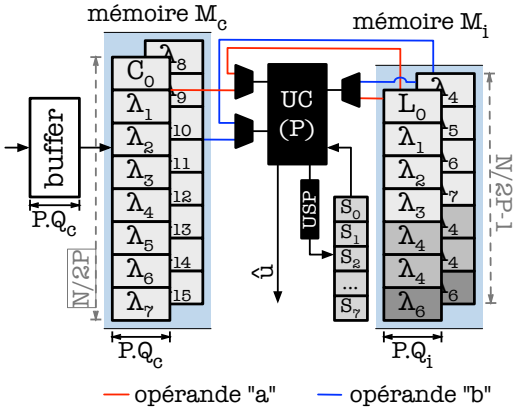

FIGURE 1 – Décodeur AS classique pour $N = 16$ et $P = 1$

Dans la suite de l'article, nous présentons d'abord l'architecture d'un décodeur AS classique. Puis, nous présentons successivement les deux méthodes de réduction mémoire. Les gains au niveau de la mémorisation sont ensuite estimés. Enfin des résultats de simulation de taux d'erreurs permettent d'attester de l'impact négligeable de ces deux méthodes sur les performances de décodage de l'algorithme de décodage AS.

2 Architecture d'un décodeur AS

La Figure 1 représente l'architecture simplifiée d'un décodeur AS tel qu'il a été proposé dans [3], puis améliorer dans

[5]. Dans un souci de clarté, nous avons choisi une petite taille de code ($N = 16$) et un parallélisme unitaire ($P = 1$). La validité de la méthode pour d'autres valeurs de N et de P. Pour une taille de code N et un parallélisme P, le décodeur est ainsi composé :

- d'un buffer de canal qui mémorise P LRV codés sur Q_c bits et les transmet à la mémoire de canal M_c.
- d'un espace mémoire M_c pour les données du canal. Ce bloc est subdivisé en deux bancs mémoire contenant chacun $\frac{N}{2P}$ points mémoires. Chaque point mémoire contient P LRV codés sur Q_c bits,
- d'un espace mémoire M_i pour les données du canal. Ce bloc est subdivisé en deux bancs mémoire contenant chacun $\frac{N}{2P}$ points mémoires. Chaque point mémoire contient P LRV codés sur Q bits,
- d'une Unité de Calcul composée P noeuds de calcul.
- d'une Unité de mise à jour des Sommes Partielles (USP), dont la complexité est proportionnelle à P [5].

Il a été démontré dans [3], qu'avec une valeur de P relativement faible $P \approx 64$, le débit du décodeur est quasi maximal. D'autre part, les complexités du buffer de canal, de l'UC et de l'USP sont proportionnelles à P et donc indépendantes de N. Ceci implique que la complexité totale du décodeur est dominée par la complexité mémoire qui est elle proportionnelle à N. Ainsi, dans [5], un décodeur AS de taille $N = 2^{20} avec P = 64$ a été implémenté sur circuit FPGA. Ce décodeur utilise 2% des ressources de calcul et 72% de la mémoire disponible sur le circuit. Le constat est similaire en technologie ASIC [6]. Ceci montre à quel point les ressources de mémorisation sont un point critique des architectures de décodeurs AS.

En faisant l'hypothèse que $P << N$, l'occupation mémoire du décodeur AS peut être estimée par $A \approx Q_c N + QN + N$ (bits). $Q_c N$ représente le coût de la mémoire du canal, QN est le coût de la mémoire interne, N correspond au coût de la mémoire qui contient les bits gelés. Cette dernière n'apparaît pas sur les figures pour des raisons de lisibilité.

Le processus de décodage AS se déroule en 5 phases :

- Phase $\phi 0$: le décodeur bufferise les LRV L_c venant du canal par paquet de P. A l'issue de cette phase, les N LRV sont stockés dans la mémoire M_c.
- Phase $\phi 1$: le décodeur applique la fonction f sur les LRV du canal (lus dans M_c) et stocke les résultats dans la M_i.
- Phase $\phi 2$: le décodeur applique une succession de fonctions f et g sur les LRV de la mémoire M_i (L_i) et stocke les résultats dans cette même mémoire. Ensuite, les sommes partielles sont calculées et stockées dans la mémoire M_s. D'autre part, à ce stade, la moitié du mot de code ($\hat{u}_i$, $0 \leq i \leq N/2 - 1$) est décodée. Notons enfin que les informations s_i sont des bits et non des LRV.
- Phase $\phi 3$: le décodeur applique la fonction g sur les LRV du canal et les sommes partielles s. Les résultats sont stockés dans la mémoire M_c.
- Phase $\phi 4$: le décodeur applique une succession de fonctions f et g sur les LRV de la mémoire M_c et stocke les résultats dans cette même mémoire. A l'issue de cette dernière phase, l'autre moitié du mot de code ($\hat{u}_i$, $N/2 \leq i \leq N - 1$) est décodée.

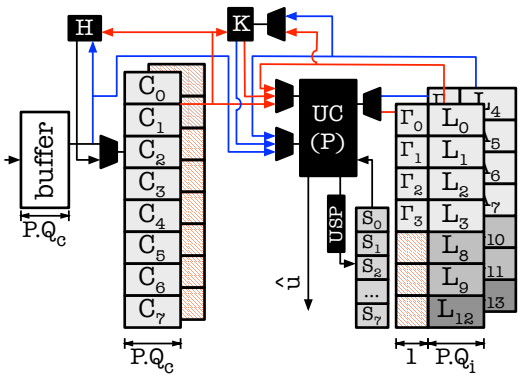

FIGURE 2 – Décodeur AS avec réévaluation des LRV

Les fonctions f et g utilisées sont détaillés dans [2] :

$$f(a,b) = sgn(a).sgn(b).\min(|a|,|b|)$$

$$g(a,b,s) = (-1)^s a + b$$

3 Calculer plutôt que mémoriser

Dans l'architecture AS classique, on peut noter qu'une grande partie de la mémoire est consommée par M_c. Nous proposons de réduire la taille de cette mémoire en modifiant le processus de décodage. Cette approche ne change pas la fonctionnalité du décodeur. Cela garantie donc que les performances de décodage sont inaltérées. L'idée est la suivante : lorsque la deuxième moitié des LRV arrive du canal, plutôt que de stocker la valeur des LRV dans M_c, la fonction $f(a,b)$ est directement calculée et le résultat est stocké dans M_i. [1] Un bit supplémentaire (Γ) est également calculé et stocké dans M_i : $\Gamma = 0$ si $|f(a,b)| = |a|$, $\Gamma = 1$ sinon. D'autre part, pendant cette même phase, une fonction $h(a,b)$ est calculée et stockée dans M_c telle que $h(a,b) = b$ si $\Gamma = 0$, $h(a,b) = b$ sinon. Autrement dit, le bit Γ permet de localiser les valeurs absolues des opérandes ($|a|$ et $|b|$) dans les mémoires M_c et M_i. Ainsi, $\Gamma = 0$ signifie que $|a|$ se trouve dans M_i et $|b|$ dans M_c. $\Gamma = 1$ signifie exactement l'inverse. A partir des valeurs stockées dans M_i et M_c il est possible de reconstruire le couple de valeur (a,b) nécessaire au calcul de g. Pour reconstruire a :

$$k_a(L_c, L_i) = \begin{cases} L_c & \text{si } \Gamma = 1 \\ sign(L_c).L_i & \text{sinon} \end{cases} \quad (1)$$

$$k_b(L_c, L_i) = \begin{cases} L_c & \text{si } \Gamma = 0 \\ sign(L_c).L_i & \text{sinon} \end{cases} \quad (2)$$

Cette modification de l'architecture est présentée dans la Figure 2. Le décodage se déroule alors de la façon suivante :

1. Ce calcul direct est possible car la fonction $f(a,b)$ est appliquée sur les données d'indice j et $j+N/2$: $f(L_c(j), L_c(j+N/2)), 0 \leq j < N/2$. Pour alléger la notation, nous n'utiliserons plus les indices dans la suite de l'article.

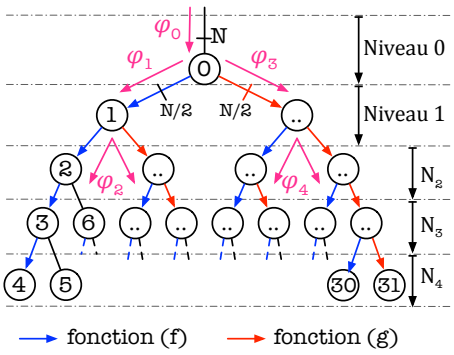

FIGURE 3 – Arbre de décodage AS

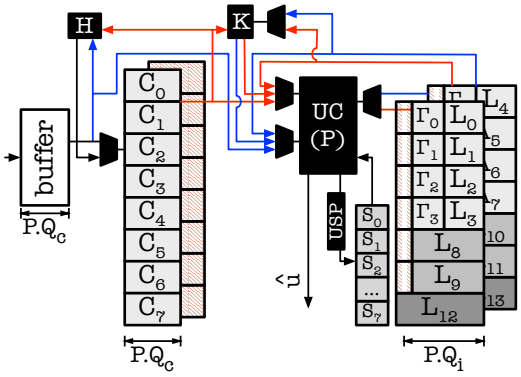

FIGURE 4 – Architecture avec réévaluation des LRV du canal et réduction de la dynamique des LRV internes.

— Phase $\phi 0$: $N/2$ LRV sont chargés dans la mémoire M_c par paquet de P.
— Phase $\phi 1$: le décodeur applique la fonction f, la fonction h et calcule le bit Γ au fil de l'eau sur les $N/2$ LRV restants qui proviennent du buffer de canal. Les résultats des fonctions f sont stockés dans la mémoire M_i. Les résultats de h sont, quant à eux, stockés dans M_c.
— Phase $\phi 2$: idem AS classique
— Phase $\phi 3$: le décodeur applique la fonction k_a et k_b sur les données C_i et L_i pour reconstruire les LRV nécessaires au calcul de g qui est également appliquée.
— Phase $\phi 4$: idem AS classique

Cette modification de l'architecture permet de s'affranchir de la mémorisation de $N/2$ LRV du canal, qui correspondent à $= Q_c \times N/2$ bits. Cette méthode requiert néanmoins l'ajout d'un bit (Γ) à la moitié des données mémorisées ($N/2$). Cela représente un surcoût mémoire de $N/2$ bits. Cette méthode permet donc de supprimer $N((1 - Q_c/2))$ bits. Pour garantir, les performances de décodage, il faut en général $Q_c \geq 4$. Cette méthode permet donc de réduire globalement la complexité mémoire du décodeur de 14% lorsque Qc=4. Des résultats numériques sont présentés et discutés dans la section 5 pour différentes configurations de décodeurs. Notons enfin qu'il est nécessaire d'ajouter des ressources de calcul. Cependant, en gardant à l'esprit que $P << N$, ce surcoût est a priori négligeable. D'autre part, les fonctions h, k_a et k_b peuvent partager des ressources de calcul (comparateurs et multiplexeurs) avec l'unité de calcul des fonctions f et g réduisant d'autant le surcoût.

4 Réduction de la dynamique des LRV

La méthode de réduction mémoire de la section 3 implique de mémoriser $N/2$ bits supplémentaire (Γ) pour localiser les opérandes dans les deux mémoires M_c et M_i. Nous proposons de compenser ce léger surcoût de mémorisation par une réduction de la dynamique des LRV internes du décodeur. Il a été mis en évidence, notamment dans [5], qu'une réduction trop importante de la dynamique des données internes introduisaient une perte de performance de décodage. Néanmoins, cette réduction était appliquée uniformément sur M_i. Nous proposons de réduire la dynamique uniquement sur une partie des données.

Le décodage AS peut être représenté conceptuellement par le parcours récursif d'un arbre binaire comme montré dans la Figure 3. La phase $\phi 0$ correspond à la branche supérieure de l'arbre, la phase $\phi 1$ et $\phi 3$ sont représentées par les branches gauche et droite du noeud racine. Les phases $\phi 2$ et $\phi 4$ correspondent respectivement au parcours des sous-arbres de gauche et de droite. Ainsi, dans cet arbre, la dynamique des données de la branche supérieure est de Q_c bits et celle des autres branches est de Q bits. La fonction f correspond à un calcul de minimum. Cette opération ne nécessite donc pas d'accroissement de la dynamique. La fonction g peut quant à elle générer des résultats avec une dynamique supérieure. De manière générale, plus on descend dans l'arbre plus les données ont tendance à saturer. Le corollaire de ce constat est qu'une réduction de la dynamique a un impact moins fort, si cette réduction est effectuée sur les niveaux proches du noeud racine.

Nous représentons l'évolution de la dynamique des données en fonction du niveau de l'arbre comme suit : $Q_c = x, Q = \{y, z, t, ...\}$. Cette notation signifie que nous utilisons x bits pour les LRV du canal (niveau 0 dans l'arbre), y bits pour les LRV internes du niveau 1, z bits pour le niveau 2, et t bits pour tous les niveaux restants. Dans une architecture classique, la dynamique utilisée est du type $Q_c = x, Q = \{y, ...\}$ puisque la même dynamique est utilisée pour tous les LRV internes. Nous proposons de réduire cette dynamique interne sur les deux premiers niveaux et de mesurer l'impact d'une part sur la complexité de mémorisation et d'autre part sur les performances de décodage. En effet, cette saturation des LRV induit nécessairement une perte d'information et donc une dégradation des performances de décodage. Au niveau du gain mémoire, une réduction de q bits pour le niveau N_i de l'arbre induit une suppression de $\frac{qN}{2^{N_i}}$ bits. En effet, plus on est proche du noeud racine dans l'arbre, plus la taille du banc mémoire associé est important. Ainsi, supprimer 1 bit sur l'étage 1 permet d'économiser $N/2$ bits. La Figure 4 représente l'architecture du décodeur AS pour laquelle a été appliquée les deux méthodes de réduction mémoire. La section suivante estime les gains en mémoire et les dégradations au niveau des performances de décodage

TABLE 1 – Estimation des gains en mémoires (Kbits) pour différentes configurations de décodeurs.

	$Q_c = 6, Q = 8$ ($Q_c = 6, Q = \{6, 7, 8, ...\}$)					$Q_c = 4, Q = 6$ ($Q_c = 4, Q = \{4, 5, 6, ...\}$)				
N	1k	2k	32k	128k	1M	1k	2k	32k	128k	1M
[scalable]	15	30	480	1920	15360	11	22	352	1408	11264
Optim. 1	12.5	25	400	1600	12800	9.5	19	304	1216	9728
Optim. 2	13.75	27.5	440	1760	14080	9.75	19.5	312	1248	9984
Optim. 1 et 2	11.25	22.5	360	1440	11520	8.25	16.5	264	1056	8448
Réduction totale	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%

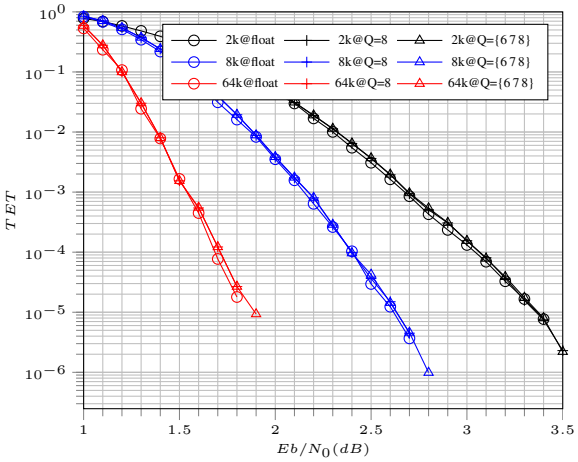

FIGURE 5 – TET pour une quantification du canal sur 6 bits.

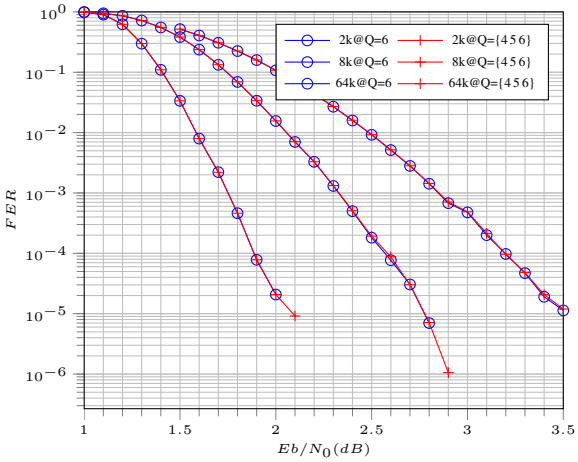

FIGURE 6 – TET pour une quantification du canal sur 4 bits.

pour les deux méthodes proposées.

5 Estimation des gains mémoire

Comme expliqué dans les sections 3 et 4, les deux méthodes proposées permettent de réduire le coût de la mémoire des décodeurs AS. La Figure 1 donne les estimations des gains en complexité mémoire pour chacune des méthodes et pour la combinaison des deux. Différentes tailles de code ($10 < log_2(N) < 20$) ont été testées ainsi que deux schémas de dynamique des données. Le schéma utilisé pour la réduction de la dynamique est précisé entre parenthèses.

De manière générale, la combinaison des deux méthodes permet de réduire de 25% la complexité de la mémoire. Du point de vue des performances de décodage, aucune dégradation notable (>0.02dB) n'est observable sur les courbes de taux d'erreurs (Figures 5 et 6). Une réduction supplémentaire de la dynamique est disponible pour les autres niveaux. Cependant, elle induit une dégradation notable des performances. De plus, un gain mémoire limité serait obtenu. En effet, nous agissons alors sur des niveaux de l'arbre éloigné du noeud racine, donc peu complexe.

6 Conclusion

Dans cet article, nous avons proposé deux méthodes de réduction mémoire pour le décodage AS des CPs. Les deux méthodes sont complémentaires. Elles permettent, sans dégradation notable pour les performances de décodage, de réduire la complexité de la mémoire. Les gains estimés sont de l'ordre de 25% quelque soit la taille du code considéré. Dans de futurs travaux, il est prévu une validation des deux méthodes proposées par une implantation matérielle. Il serait de plus, intéressant de projeter cette méthode sur l'algorithme de décodage par liste.

Références

[1] Arikan, E., "Channel polarization : a method for constructing capacity-achieving Codes for Symmetric Binary-Input Memoryless Channels," IEEE TIT, 2009

[2] Leroux, C. ; Tal, I. ; Vardy, A. ; Gross, W.J., "Hardware architectures for successive cancellation decoding of polar codes," IEEE ICASSP, May 2011

[3] Leroux, C. ; Raymond, A.J. ; Sarkis, G. ; Gross, W.J., "A Semi-Parallel Successive-Cancellation Decoder for Polar Codes," IEEE TSP, Jan 2013

[4] Sarkis, G. ; Giard, P. ; Vardy, A. ; Thibeault, C. ; Gross, W.J., "Fast Polar Decoders : Algorithm and Implementation," IEEE JSAC, May 2014

[5] Raymond, A.J. ; Gross, W.J., "Scalable successive-cancellation hardware decoder for polar codes," IEEE GlobalSIP, Dec. 2013

[6] YouZhe Fan, Chi-Ying Tsui, "An Efficient Partial-Sum Network Architecture for Semi-Parallel Polar Codes Decoder Implementation," IEEE TSP, June 2014

[7] Tal, I. ; Vardy, A., "List Decoding of Polar Codes," IEEE ISIT, July 2011.