Architectures matérielles pour le décodage des codes polaires

Camille LEROUX[1], Ido TAL[2], Alexander VARDY[2], Warren J. GROSS[1]

[1]Université McGill
3480 Rue de l'université, Montréal, H3A 2A7, Québec, Canada

[2]University of California San Diego
9500 Gilman Drive, La Jolla, 92093-0407, CA

camille.leroux@mcgill.ca, ital@mail.ucsd.edu
avardy@ucsd.edu, warren.gross@mcgill.ca

Résumé – Les codes polaires sont une famille de codes qui permettent, sous certaines conditions, d'atteindre la capacité d'un canal de transmission. De plus en plus de travaux traitent d'aspects théoriques liés à la définition de ces codes pour différents problèmes de la théorie de l'information. Cette étude porte sur la définition d'architectures de décodeurs à annulation successive pour le décodage des codes polaires. Nous montrons qu'un décodeur à annulation successive peut être implementé avec $O(n)$ noeuds de calculs et $O(n)$ éléments mémoires (au lieu de $O(n \log_2 n)$) tout en garantissant un débit constant. Enfin nous montrons que les noeuds de calculs peuvent être simplifiés en effectuant les calculs dans le domaine logarithmique et nous proposons une architecture en format signe et magnitude.

Abstract – Polar codes are recently-discovered capacity-achieving codes. Many recent works adress theoretical issues on the definition of codes for various information theory problems. This study focus on the practical aspect of hardware implementation of the associated decoders. We show that a successive cancellation decoder can be implemented with $O(n)$ processing nodes and $O(n)$ memory elements (instead of $O(n \log n)$) while maintaining the same throughput. We also show how processing nodes can be simplified in logarithmic domain and we propose an architecture in sign and magnitude format.

1 Introduction

Inventés par Arıkan en 2009, les codes polaires [1] sont des codes correcteurs d'erreurs qui atteignent asymptotiquement la capacité du canal de transmission si celui-ci est symétrique, sans-mémoire et à entrée binaire. De plus, leur construction est explicite et il est possible de les encoder et décoder avec une complexité $O(n \log_2 n)$ [1], n étant la longueur du code. Enfin, les codes polaires sont optimaux pour le codage de source [2] ou bien encore le codage sécurisé [3]. Etant les seuls codes à réunir toutes ces propriétés à la fois, les codes polaires sont perçus comme une avancée majeure en théorie de l'information. D'un point de vue pratique, les performances de ces codes ne sont intéressantes que pour une grande taille de bloc : $n > 2^{20}$. Pour des valeurs de n plus faibles, les performances des codes polaires sont inférieures à celles des codes LDPC [4] ou des turbo codes [5]. Il est néanmoins vraisemblable que l'on puisse améliorer les performances des codes polaires en les associant à des techniques existantes telles que la concaténation [6], le décodage par liste [7], ou bien encore le turbo-décodage [8].

Du point de vue architectural, leur structure récursive régulière et l'absence de propriété pseudo-aléatoire confèrent aux codes polaires une bonne prédisposition à l'implémentation matérielle.

1. Dans la suite, même si cela n'est pas spécifié tous les logarithmes sont en base 2

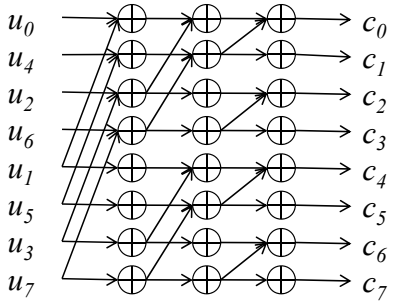

FIGURE 1 – Structure de l'encodeur pour $n = 8$

Dans [1], Arıkan propose une structure de codage et de décodage dont la complexité croît en $O(n \log n)$. Nous proposons de tirer profit du séquencement particulier des opérations dans le décodage des codes polaires pour définir une famille d'architectures dont la complexité matérielle est réduite à $O(n)$ sans perte de débit. Enfin nous proposons une approximation des calculs dans le domaine logarithmique qui permet de remplacer multiplieurs et diviseurs par de simples additionneurs et comparateurs.

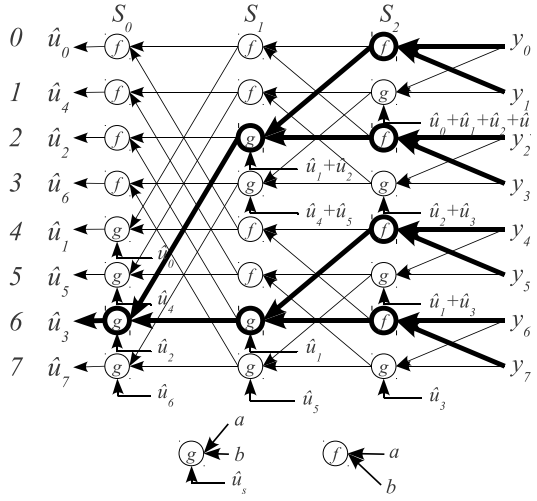

FIGURE 2 – Architecture d'un décodeur papillon pour $n = 8$.

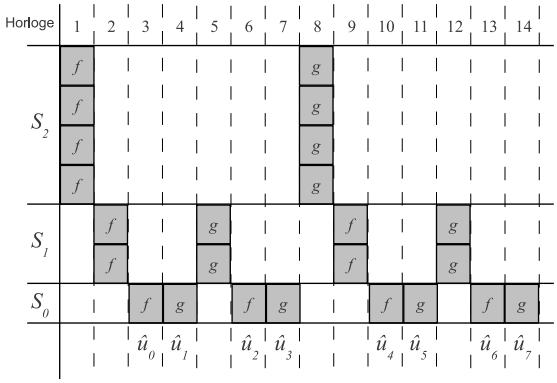

FIGURE 3 – Séquencement du décodeur papillon et en arbre pour $n = 8$.

2 Les codes polaires

Les codes polaires sont des codes en blocs linéaires construits à partir de la $m^{\text{ième}}$ puissance de Kronecker de la matrice $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. La matrice génératrice G_n est obtenue en appliquant un adressage *bit-reversed* aux lignes de $F^{\otimes m}$. La figure 1 représente G_8 sous forme graphique. Le mot de code résultant $\mathbf{c}$ est envoyé sur le canal de transmission. A la réception, en supposant que les bits u_i sont estimés en utilisant un décodage à annulation successive (AS), il est démontré dans [1] que chaque bit u_i est caractérisé par une probabilité d'erreur qui tend soit vers 0 (sans erreur) soit vers 0.5 (bruit pur) lorsque $n \longrightarrow \infty$. De plus la proportion de bits u_i ayant une faible probabilité d'erreur tend vers la capacité du canal de transmission. Les codes polaires tirent avantage de ce phénomène de *polarisation du canal* en envoyant k bits d'information sur les positions u_i les plus fiables et en fixant les $n - k$ bits restant à une valeur prédeterminée. Les codes polaires atteignent asymptotiquement la capacité du canal à la condition que le récépteur utilise un décodeur à annulation successive. Celui-ci estime la valeur des bits non-encodé $\hat{u}_i$ (successivement de $\hat{u}_0$ à $\hat{u}_{n-1}$) :

$$\hat{u}_i = \begin{cases} 0, \text{ if } \frac{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i = 0)}{\Pr(\mathbf{y}, \hat{u}_0^{i-1} | u_i = 1)} > 1, \\ 1, \text{ otherwise,} \end{cases} \tag{1}$$

$\mathbf{y}$ représente l'observation du canal. L'estimation d'un bit $\hat{u}_i$ est ensuite réutilisé pour l'estimation des bits suivants $\hat{u}_{i+1}^{n-1}$.

3 Du papillon à l'arbre

Dans [1] Arıkan a montré qu'il était possible de concevoir un décodeur AS en utilisant une structure similaire à une transformée de Fourier rapide (TFR). Cependant, bien que le décodeur utilise effectivement une topologie à base de *papillons*, les noeuds de calculs ainsi que le séquencement des opérations sont différents d'une TFR. Un *décodeur-papillon* pour un code polaire de taille $n = 8$ est décrit sur la Figure 2. Ce décodeur estime les bits non-encodés u_i à partir des rapports de vraisemblance (RV) reçus du canal. Ce décodeur est composé de $\log n$ étages S_l comportants chacun n noeuds de calcul. Chacun des noeuds calcule $f(a, b) = \frac{1+ab}{a+b}$ ou $g_{\hat{u}_s}(a, b) = a^{1-2\hat{u}_s} b$, a et b étant les RV d'entrée et $\hat{u}_s$ une somme partielle modulo-2 des bits précédemments décodés. L'estimation d'un bit $\hat{u}_i$ s'effectue en activant $n - 1$ noeuds de calcul suivant une structure en arbre (l'arbre utilisé pour l'estimation de $\hat{u}_3$ est indiqué en gras sur la Figure 2). Ainsi, à chaque instant du décodage, et dans chaque étage S_l, seulement 2^l noeuds sont actifs. Ceci suggère qu'il est possible d'effectuer le même traitement en implémentant un arbre de noeuds de calculs configurables [2] (NCC). De plus le décodeur-papillon peut être vu comme un ensemble d'arbres de calcul dont les noeuds se chevauchent. Par example l'estimation de $\hat{u}_2$ et $\hat{u}_3$ ne diffère que par un seul noeud. Il est par conséquent possible de réutiliser certains calculs à condition d'ajouter des resources mémoires entre les étages du décodeur. En analysant le séquencement des opérations on peut montrer qu'un seul élément mémoire par noeud de calcul est suffisant. Le *décodeur en arbre* résultant est ainsi constitué de $n-1$ NCC et $n-1$ éléments de mémoire. Dans les deux cas (décodeur-papillon ou en arbre), le schéma de séquencement est identique comme montré sur la Figure 3. Nous supposons ici qu'un seul étage est activé par cycle d'horloge. En supposant que la fréquence d'horloge est limitée par le temps de traversée d'un noeud de calcul t_{nc}, on peut montrer que le débit estimé résultant est

$$D = \frac{n}{(2n-2)t_{nc}} \approx \frac{1}{2t_{nc}}. \tag{2}$$

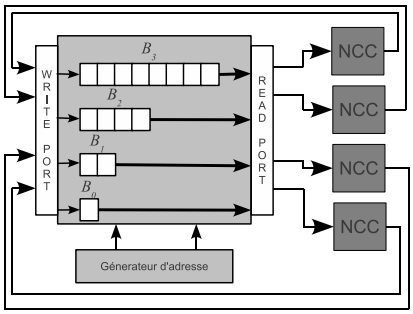

FIGURE 4 – Architecture du décodeur en ligne

4 La simple ligne

Il est possible d'aller encore plus loin dans la réduction de la complexité en notant qu'à chaque cycle d'horloge, un seul étage du décodeur en arbre est réellement activé comme montré sur la Figure 3. Dans le pire des cas $\frac{n}{2}$ calculs doivent être effectués en parallèle. $n-1$ éléments mémoires sont cependant toujours nécessaires pour les calculs intermédiaires. En d'autres termes, une ligne de $\frac{n}{2}$ NCC connectée à un arbre de $n-1$ éléments mémoire permet d'effectuer le même traitement que le décodeur en arbre tout en conservant le même débit puisque le schéma de séquencement reste le même. Un exemple de décodeur-ligne est détaillé sur la Figure 4. Des bancs mémoires sont accessibles par une ligne de NCC via un générateur d'adresse. Il est possible de réduire encore plus le nombre de NCC avec une faible perte de débit. Dans la figure 3, on constate que chaque étage l est activé 2^{m-l} fois. Par conséquent, les $\frac{n}{2}$ NCC du décodeur en ligne sont tous activés simultanéement à seulement deux reprises durant le décodage d'un vecteur. Ainsi, un décodeur semi-parallèle constitué de $\frac{n}{4}$ NCC ne nécessiterait que 2 cycles d'horloge supplémentaire pour décoder un vecteur. Un tel décodeur est en cours d'étude.

5 L'arbre entrelacé

Le décodeur AS en ligne tire profit des étages inactivés du décodeur en arbre pour réduire la complexité. Une alternative est d'utiliser ces étages inactivés pour décoder d'autres mots de code en parallèle en ainsi augmenter le débit global de traitement. L'entrelacement n'est possible qu'à condition de dupliquer une partie des resources de calcul. Un décodeur entrelacé est donc un décodeur en arbre dans lequel on duplique un ou plusieurs étages de NCC.
La table 1 donne un exemple de séquencement pour un décodeur dans lequel seul S_0 est dupliqué pour $n=8$. On constate que $P=3$ mots de codes sont traités en même temps alors qu'un seul NCC est ajouté. Les resources mémoires doivent néanmoins être dupliquées de manières à fournir les données intermédiaires pour chaque mot de code. On peut montrer que

2. capable de calculer alternativement f ou g

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S_2	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$					$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$						
S_1		$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$		$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$		
S_0			$\mathbf{y}_1$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_1$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_1$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_1$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$
S_{0d}				$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_3$	$\mathbf{y}_2$	$\mathbf{y}_3$			$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_3$	$\mathbf{y}_2$	$\mathbf{y}_3$	

TABLE 1 – Séquencement du décodage de $P=3$ mots de codes dans le décodeur en arbre entrelacé pour $n=8$.

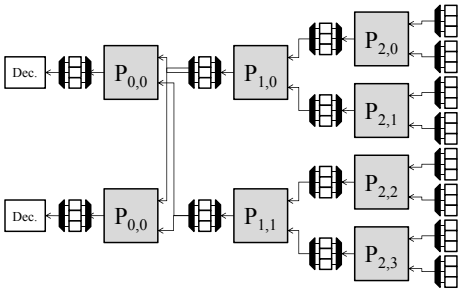

FIGURE 5 – Décodeur en arbre entrelacé pour $n=8$ et $P=3$

le nombre de NCC croît en $O(n+log(P))$, P étant le nombre de mots de code traités en parallèle. Pour sa part, la complexité mémoire croît de manière linéaire avec le parallélisme : $O(nP)$. Cette architecture augmente le débit de traitement qui croît également linéairement avec P.

6 Approximation dans le domaine algorithmique

Dans le décodage AS proposé dans [1] les données sont représentées sous forme de rapports de vraisemblance. Les fonctions f et g nécessitent des multiplications et des divisions qui sont complexes à implémenter sur du silicium. De manière similaire à ce qui se fait dans le décodage des codes LDPC et des turbo codes, nous proposons de transposer les calculs dans le domaine des logarithmes de rapport de vraisemblance (LRV) :

$$\begin{cases} f(\lambda_a,\lambda_b) = 2\tanh^{-1}\left(\tanh\left(\frac{\lambda_a}{2}\right)\tanh\left(\frac{\lambda_b}{2}\right)\right) \\ g_{\hat{u}_s}(\lambda_a,\lambda_b) = \lambda_a(-1)^{\hat{u}_s}+\lambda_b, \end{cases} \quad (3)$$

λ_a en λ_b sont les LRVs. Bien que le passage dans le domaine logarithmique permette de réduire le calcul de la fonction g à une simple addition/soustraction, la fonction f reste complexe à estimer (produit de $\tanh$). Il a été montré dans [9] qu'un simple calcul de minimum permettait d'obtenir une bonne approximation de f. Les calculs effectués au sein du décodeur deviennent alors :

$$\begin{cases} f(\lambda_a,\lambda_b) \approx \ \text{sign}(\lambda_a)\ \text{sign}(\lambda_b)\min(|\lambda_a|,|\lambda_b|). \\ g_{\hat{u}_s}(\lambda_a,\lambda_b) = \lambda_a(-1)^{\hat{u}_s}+\lambda_b, \end{cases} \quad (4)$$

La figure 6 compare les performances de décodage pour deux codes avec et sans approximation. La dégradation des performances induite est négligeable pour $n=1024$ et se limite à 0.1dB pour $n=16384$.

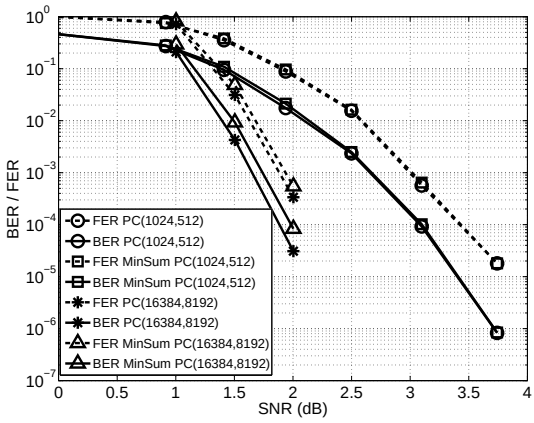

FIGURE 6 – Performance de décodage AS avec et sans approximation.

7 Architecture d'un noeud de calcul configurable

L'implémentation matérielle de f et g dans le domaine logarithmique permet de remplacer l'utilisation de multiplications et divisions par de simples additions et comparaisons. De plus, au sein d'un même NCC, les fonctions f et g ne sont jamais executées en même temps, il est donc envisageable de partager des resources de calculs. La fonction f effectue simulatanément des calculs sur le signe et la valeur absolue des LRVs. Nous proposons par conséquent d'effectuer les calculs en format signe et magnitude. L'architecture d'un NCC est détaillée sur la figure 8. Une simple porte XOR permet de calculer le signe de f. Le signe de g correspond au signe de l'entrée dont la magnitude est la plus forte. La magnitude de f est la magnitude minimale des entrées. En format signe et magnitude, $|g|$ dépend de s_a, s_b, de la somme partielle $\hat{u}_s$ et des valeurs relatives de L_a et L_b. La magnitude de $|g|$ s'exprime alors

$$|g| = \max(|\lambda_a|, |\lambda_b|) + (-1)^{\hat{u}_s \oplus (\mathrm{s}(\lambda_a) \oplus \mathrm{s}(\lambda_b))} \min(|\lambda_a|, |\lambda_b|)$$

Un NCC comporte seulement un additionneur et un comparateur qui est par ailleurs utilisé dans le calcul de f et g.

8 Conclusion

Nous proposons de tirer profit du séquencement particulier du décodage à annulation successive pour réduire la complexité des resources de calcul. Dans le tableau 2, pour chaque architecture mentionnée ici, nous comparons les complexités en termes de resources de calcul et de resources mémoires. Nous donnons également le débit estimé de fonctionnement. Nous proposons enfin une architecture de noeud de calcul qui fonctionne dans le domaine logarithmique avec un format signe et magnitude.

Arch.	Nb de NCC	Nb d'éléments mém.	Débit
Papillon [1]	$n \log n$	$n \log n$	$1/2t_{nc}$
Arbre	$n-1$	$n-1$	$1/2t_{nc}$
Ligne	$\frac{n}{2}$	$n-1$	$1/2t_{nc}$

TABLE 2 – Comparaison des architectures de décodeurs AS.

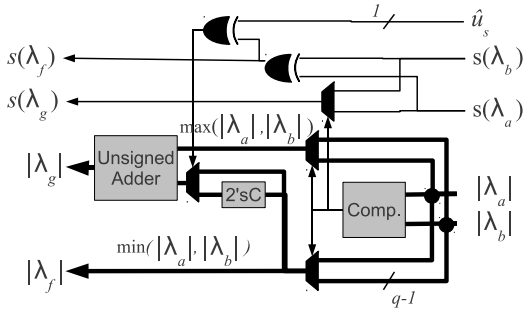

FIGURE 7 – Architecture d'un NCC.

Références

[1] E. Arıkan : Channel Polarization : A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE Trans. on Inform. Theory, 2009.*

[2] N. Hussami, R Urbanke et S.B. Korada. : Performance of polar codes for channel and source coding. *IEEE Intern. Symp. on Inform. Theory, 2009.*

[3] H. Mahdavifar. et A. Vardy : Achieving the secrecy capacity of wiretap channels using Polar codes. *IEEE Intern. Symp. on Inform. Theory, 2010.*

[4] R. G. Gallager. : Low-Density Parity-Check Codes. *IRE Trans. on Inform. Theory, vol. IT-8, 21-28, 1962.*

[5] C. Berrou, A. Glavieux and P. Thitimajshima. : Near Shannon limit error-correcting coding and decoding : Turbo-codes. *IEEE Intern. Conf. on Comm., Geneva, Vol. 2, pp. 1064-1070, 1993.*

[6] M. Bakshi, S. Jaggi et M. Effros : Concatenated Polar codes. *IEEE Intern. Symp. on Inform. Theory, 2010*

[7] I. Tal and A. Vardy. : List decoding of polar codes. *IEEE Intern. Symp. on Inform. Theory, 2011*

[8] R.M. Pyndiah : Near-optimum decoding of product codes : block turbo codes. *IEEE Transactions on Communications*

[9] 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. on Comm., vol. 47, no. 5, pp. 673 –680, May. 1999.*