Réduction du retard de calcul dans la méthode OLA

Márk FÉK[1,2], Annamária R. VÁRKONYI-KÓCZY[1], Jean-Marc BOUCHER[2]

[1]Université Technique de Budapest
Département Mesures et Systèmes d'Information
Budapest, H-1521, Hongrie

[2]ENST de Bretagne
Département Signal et Communications
Technopôle de Brest Iroise, BP 832, 29285 Brest Cedex, France

fek@mit.bme.hu, koczy@mit.bme.hu, JM.Boucher@enst-bretagne.fr

Résumé – L'algorithme OLA (Overlapp and Add) est classiquement utilisé pour le filtrage adaptatif dans le domaine des fréquences. Cette méthode introduit cependant un retard, qui peut nuire à son utilisation dans certaines applications temps réel. Dans cet article, nous proposons de réorganiser le calcul de la Transformation de Fourier Rapide permettant de réduire ce retard. Nous indiquons aussi une solution pour paralléliser la succession de deux opérations Transformation de Fourier Rapide et Transformation inverse. L'évaluation du gain fourni par ces modifications nouvelles est aussi donnée.

Abstract – The OLA (Overlapp and Add) algorithm is classicaly applied to frequency domain adaptive filtering. However, the delay caused by the calculation time can prevent the application of the algorithm in real-time cases. In this paper, we propose to reorganize the calculation of the Fast Fourier Transformation in order to reduce the delay of calculation. We also present a parallelization method for the succesion of an Inverse Fast Fourier Transformation followed by a Fast Fourier Transformation. The gain provided by these modifications is also given.

1 Introduction

L'algorithme OLA (Overlapp and Add) [1] permet de réaliser la convolution dans le domaine des fréquences, ce qui entraîne une réduction du nombre d'opérations. Cette technique est classiquement utilisée pour le filtrage adaptatif dans le domaine transformé [2]. La Figure 1 montre les opérations effectuées pendant le filtrage adaptatif réalisé par la méthode OLA, ce qui nécessite cinq transformations au total, dont le temps de calcul est assez important. Dans le cas des applications fonctionnant en temps réel, il est important de minimiser le retard introduit par le traitement. Il serait donc souhaitable de réduire le temps du calcul, ou bien le retard causé par ces opérations.

Nous proposons de modifier l'organisation de calcul des papillons dans l'algorithme de Transformation de Fourier Rapide (TFR) [3], ce qui permet de commencer les calculs avant le stockage complet d'un bloc de données, et ainsi de réduire le délai. De plus, en réorganisant la structure de la TFR, il devient possible de paralléliser la succession d'une TFR-inverse (TFRI) [3] et d'une TFR. De cette façon, nous pouvons réduire davantage le retard de calcul.

2 La TFR à retard réduit (TFR-RR)

Dans [5], l'auteur propose une structure de calcul de la Transformée de Fourier Discrète (TFD) adaptée aussi bien à un traitement par bloc, qui nécessite le stockage du dernier échantillon du bloc avant de débuter le calcul de la TFD, qu'à un traitement échantillon par échantillon, dans le cas d'une TFD à fenêtre glissante. Dans le cas d'une opération par bloc, une diminution du retard peut être obtenue en commençant les calculs avant le stockage du dernier échantillon.

Nous appliquerons ici la même idée de réduction de retard, mais dans le cas classique de la TFR partagée en fréquences (TFR-F) [3]. Les blocs fondamentaux sont les papillons connectés de la même manière que dans le cas de la TFR-F classique. La seule différence entre les deux algorithmes est l'ordre du calcul. La TFR-F stocke un bloc complet de données, et calcule les papillons colonne par colonne.

Le nouvel algorithme (Figure 2) effectue tous les calculs exécutables pour l'échantillon d'entrée courant, ce qui entraîne une organisation des calculs de papillons différente. Au lieu de l'organisation par colonnes, les calculs sont organisés par étapes. Une étape représente la collection des échantillons pour lesquels le même nombre de papillons doit être calculé.

Dans le cas d'une TFR d'ordre N, la première étape consiste dans les premiers $N/2$ échantillons. La k-ième étape consiste dans $N/2^k$ échantillons suivants, et la dernière, $(log_2 N + 1)$-ième étape ne consiste que dans le dernier échantillon. Le nombre d'étapes donne le nombre de colonnes dans lesquelles il y a des papillons à calculer. Dans la première étape, les données d'entrée sont seulement stockées, sans effectuer de calculs. Dans la k-ième étape une partie des papillons des $k-1$ premières colonnes

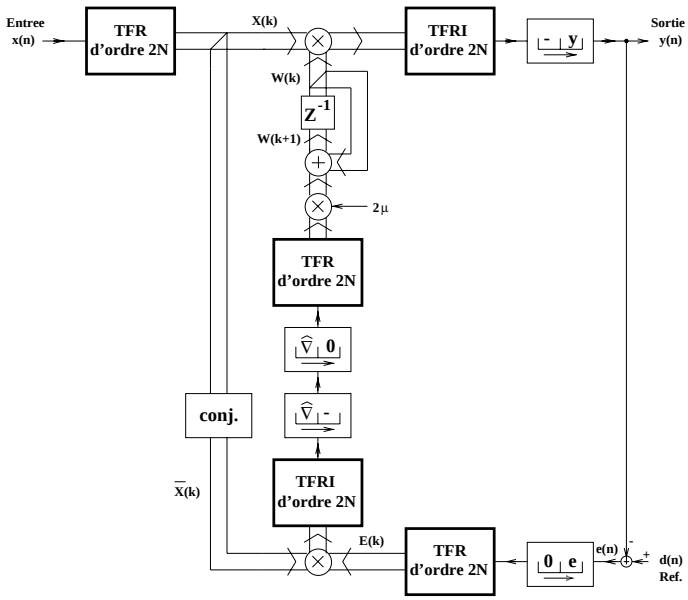

FIG. 1 – Filtrage adaptatif dans le domaine transformé par la méthode OLA

est calculée. Dans la première colonne, on ne calcule que le papillon correspondant à l'échantillon d'entrée courant. Dans la i-ième colonne 2^{i-1} papillons connectés aux papillons calculés dans la colonne précédente sont calculés. Les papillons dans la dernière colonne n'exigent pas une pondération, ainsi leur calcul est simplifié. Le résultat final, de la même manière que pour l'algorithme TFR-F, est fourni dans l'ordre de bits renversés.

Le Tableau 1 montre les organisations par étapes pour une transformation d'ordre 256.

La Figure 2 montre le fonctionnement de l'algorithme dans le cas d'une transformation d'ordre 8. Pour les quatre premiers échantillons les données d'entrée sont seulement stockées. Pour les quatrième et cinquième échantillons, on peut calculer les papillons de la première colonne. Pour le sixième échantillon, un papillon de la première et deux papillons de la deuxième colonne peuvent être calculés. Pour le septième échantillon, un papillon de la première, deux papillons de la deuxième et quatre papillons de la troisième colonne peuvent être calculés.

3 Complexité de calcul et réduction de retard

La complexité de calcul de l'algorithme reste proche de celle de la TFR-F classique, parce que le même nombre de papillons doit être calculé, et seule l'organisation des calculs est différente. Cette dernière comporte un petit surplus dans les calculs qui peut être minimisé par une programmation efficace.

L'intérêt majeur du nouvel algorithme réside dans la réduction de délai. Le retard d'ensemble consiste dans le délai de bloc et le retard introduit par le temps de calcul (Figure 3). Le délai de bloc est égal à la durée de N

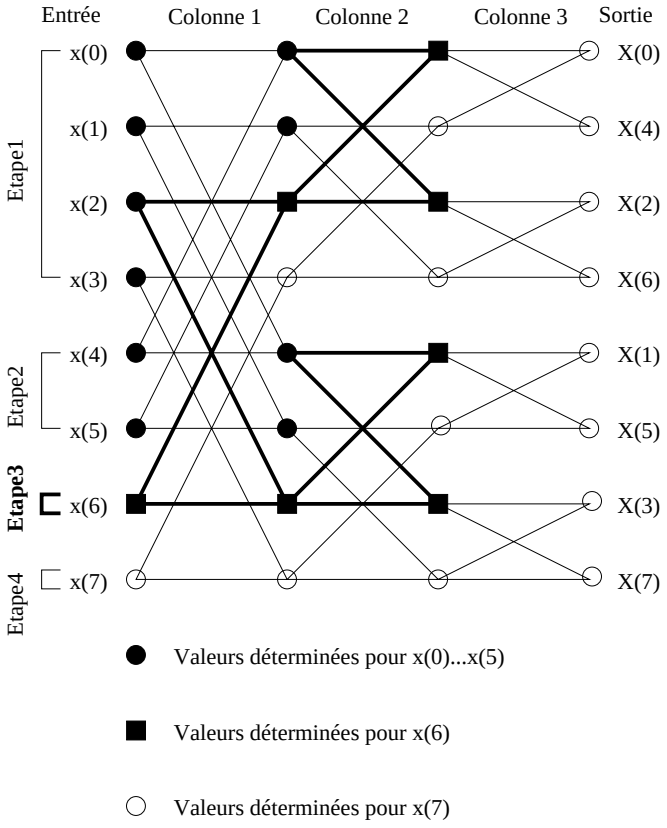

FIG. 2 – TFR à retard reduit d'ordre 8 partagée en fréquence

échantillons. Le retard introduit par le temps de calcul dépend essentiellement de la fréquence d'échantillonnage et de la puissance de calcul du processeur. Pour une fréquence d'échantillonnage donnée, en supposant que nous n'appliquons pas de papillons simplifiés (dont le calcul exige moins d'opérations arithmetiques que celui d'un papillon normal), nous pouvons approximativement caractériser la puissance de calcul du processeur par le nombre de papillons qui peuvent être calculés pendant la durée d'un échantillon. Une limite minimale de la puissance de calcul nécessaire peut être définie comme $\frac{1}{2}\log_2 N$ papillons par échantillon, pour permettre de calculer l'ensemble des $\frac{N}{2}\log_2 N$ papillons d'une TFD pendant la durée de N échantillons. Pour une architecture donnée, on peut préciser cette valeur en distinguant les papillons normaux des papillons simplifiés. Le gain de réduction de retard peut être défini comme la proportion du délai d'ensemble dans le cas de la TFR classique et du délai d'ensemble de l'algorithme TFR-RR. Si la puissance de calcul est assez grande pour calculer une transformation complète pendant la durée d'un échantillon, le gain devient minimal. Le Tableau 2 montre les valeurs de gain pour une TFD d'ordre 256.

4 Une possibilité de parallèlisation

Pendant le filtrage adaptatif réalisé par la méthode OLA, on calcule deux fois la succession d'une TFR-inverse et d'une TFR. Si nous utilisons des transformations classiques, le calcul de la deuxième transformation ne peut pas être commencée avant d'avoir fini la première. En ré-

TAB. 1 – Le nombre de papillons à calculer dans les étapes différentes pour une TFD d'ordre 256

Nombre d'étape	1	2	3	4	5	6	7	8	9
Nombre déchantillon	0..127	128..191	192..223	224..239	240..247	248..251	252..253	254	255
Papillons à calculer	0	1	3	7	15	31	63	127	255

TAB. 2 – Le gain de délai pour une TFD d'ordre 256, $f_s = 16$kHz

Puissance de calcul (papillons/échantillon)	4	5	6	7	8	15	31	1024
Délai d'ensemble - TFR classique (échantillon)	512	460	427	403	384	319	290	1
Délai d'ensemble - TFR-RR (échantillon)	480	400	368	350	336	291	269	1
Gain	1.07	1.15	1.16	1.15	1.14	1.10	1.08	1

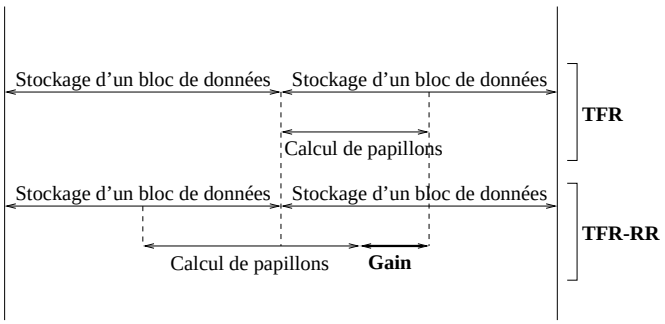

FIG. 3 – Réduction de retard de calcul par la TFR-RR

organisant l'ordre de calcul dans les deux transformations, on peut commencer à calculer la deuxième transformation en utilisant les résultats partiels de la première, qui est cependant toujours en train d'être calculée. Pour cela, nous avons évidemment besoin de deux unités de calcul (processeurs de signal) indépendantes, ce qui augmente la complexité et aussi le coût matériel.

Les connections entre les papillons de la transformation inverse doivent être organisées d'une manière similaire à la TFR partagée en fréquence, cependant les papillons de la transformation normale doivent être organisés comme dans une TFR partagée dans le temps (Figure 4). Pour le calcul d'un papillon dans la k-ième colonne, il est nécessaire d'avoir déjà calculé deux papillons correspondants de la colonne précédente. Dans le cas d'une TFD d'ordre N il faut donc évaluer $N-1$ papillons pour obtenir les deux premières valeurs transformées à la sortie. Après avoir obtenu ces deux valeurs, nous pouvons commencer la transformation inverse. Grâce à l'organisation des calculs dans les deux transformations, la deuxième peut être continuée parallèlement et d'une manière synchronisée avec la première, en supposant que les vitesses des deux processeurs se ressemblent.

Pour estimer le gain accessible par cette parallélisation, nous considérons le cas d'une TFRI d'ordre N suivie par une TFR de même ordre. Pour évaluer une transformation d'ordre N, il faut calculer $\frac{N}{2}\log_2 N$ papillons. Sans paralléliser les deux transformations, il faut donc calculer $Nlog_2N$ papillons au total. A l'aide de la parallélisation le nombre de papillons à calculer reste le même, mais le temps de calcul se réduit à $N-1+\frac{N}{2}\log_2 N$ (Figure 5). Dans le cas de $N = 1024$, cela entraîne une réduction de

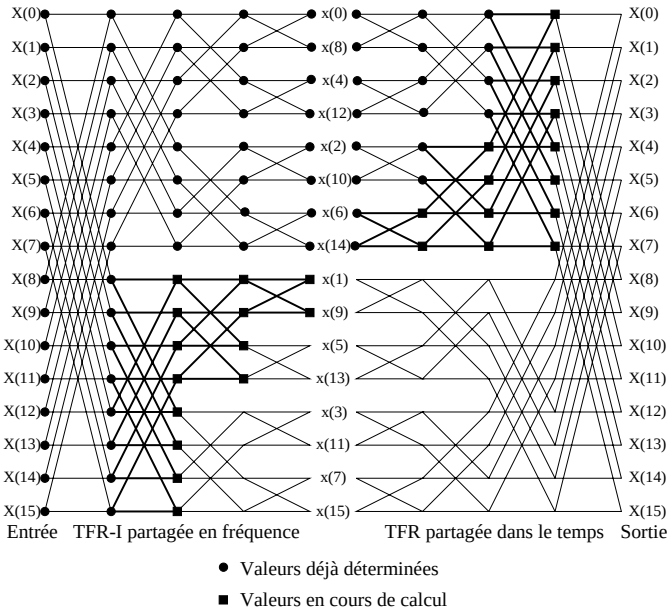

FIG. 4 – Réalisation parallèle de la succession d'une TFR-Inverse et d'une TFR. Les papillons en gras représentent une phase de calcul et ils peuvent être calculés parallèlement dans les deux transformations.

60% du temps de calcul.

5 Conclusion

Dans cet article, deux méthodes de calcul de la Transformation de Fourier ont été proposées pour réduire le retard de calcul dans l'algorithme OLA. L'une de ces méthodes permet de finir le calcul plus tôt que la TFR classique, puisque les calculs sont déjà commencés pendant le stockage d'un bloc de données. L'autre méthode parallélise la succession d'une TFR-inverse et d'une TFR, réduisant le temps de calcul de 60% dans le cas d'une TFR d'ordre 1024. Cependant, cette parallélisation nécessite deux unités de calcul, augmentant ainsi la complexité du hardware.

Références

[1] R.E. Crochiere, L.R. Rabiner. *Multirate Digital Signal Processing*. Englewood Cliffs, NJ : Prentice-Hall, 1983.

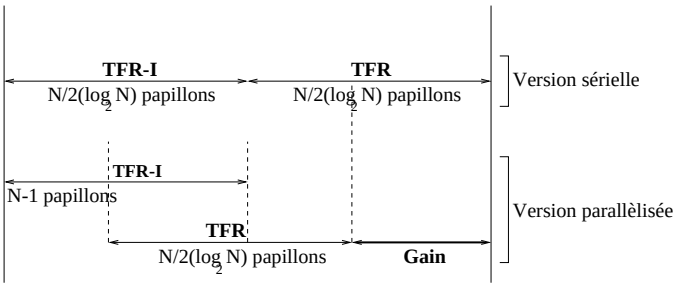

FIG. 5 – Gain de réduction de temps de calcul réalisable par la parallélisation de la succession d'une TFR-Inverse et d'une TFR

[2] J.J. Shynk. *Frequency Domain and Multirate Adaptive Filtering.* IEEE SP MAG, Jan. 1992, pp. 14-37.

[3] S.K. Mitra, J.F. Kaiser. *Handbook for Digital Signal Processing.* John Wiley & Sons, 1993.

[4] L.R. Rabiner, B. Gold. *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ : Prentice-Hall, 1975.

[5] A.R. Várkonyi-Kóczy. *A Recursive Fast Fourier Transformation Algorithm.* IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 42, No. 9, Sep. 1995, pp. 614-616.