

Modélisation par Réseaux de Petri du codeur entropique de JPEG-2000

I.Aouadi

O.Hammami

ENSTA

32 Bd Victor 75739 Paris

{aouadi, hammami}@ensta.fr

Résumé

Bien que le nouveau standard de compression d'images JPEG-2000 propose une multitude d'améliorations par rapport à son prédécesseur JPEG, il présente une complexité algorithmique beaucoup plus élevée. La complexité du codeur entropique est la plus significative dans JPEG-2000. Dans le cadre de ce papier nous nous sommes intéressés aux modèles de calcul en particulier les Réseaux de Petri (RDP) pour l'extraction du parallélisme et l'optimisation de l'implémentation du standard.

Mots Clefs :

JPEG-2000, Réseaux de Petri, parallélisme, optimisation.

1. Introduction :

Le standard JPEG-2000 [1] de compression d'images fournit de nombreuses fonctionnalités grâce entre autres à son étape de codage entropique [2]. Le standard décrit le fonctionnement du traitement mais n'impose aucune implémentation particulière. Les implémentations du standard ou des parties du standard font donc l'objet d'études variées pour obtenir des solutions ayant un bon rapport cout-performances. Des implémentations à base de VLIW ou FPGA ont entre autres été publiées [3,4]. Néanmoins une analyse formelle de l'algorithme est souhaitable pour permettre l'extraction du parallélisme et des contraintes de ressources en vue d'une meilleure implémentation système sur puce. Les modèles de calcul sont des formalismes utilisés dans les méthodologies de conception pour implémentation sur puce [5,6]. Ils offrent un moyen de modélisation abstraite permettant d'explorer le parallélisme intrinsèque d'une application indépendamment de son implémentation. Parmi ces formalismes nous nous sommes intéressés aux réseaux de Petri afin de mener une étude sur le codeur entropique de JPEG-2000.

2. JPEG-2000

2.1 Chaîne de traitement

La chaîne de codage de JPEG-2000 commence par un prétraitement de l'image originale. Ce prétraitement permet de changer la représentation de l'image (facultatif), tel qu'une transformation de couleur RGB → YUV. Ensuite, on découpe l'image en « Tile ». Chaque « Tile » subit une transformée en ondelettes. Une quantification est par la suite appliquée sur chaque sous bande obtenue suite à la transformée en ondelettes (TO). Les coefficients obtenus

sont traités par le codeur entropique qui permet d'obtenir les données compressées. Ces données sont finalement mises en forme pour aboutir à un fichier JPEG-2000 (figure1).

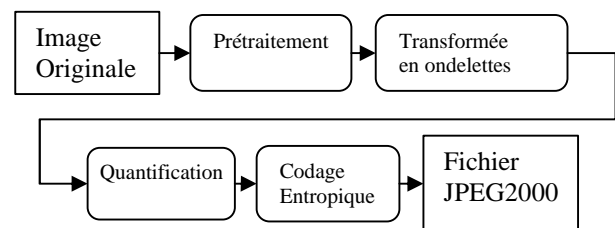


Figure 1 : Chaîne de codage JPEG-2000

2.2. Le codeur entropique

Cette partie de JPEG-2000 représente 70% à 80% du temps total d'exécution. Le principe de base du codage entropique est le EBCOT [2]. L'idée principale consiste à : (1) Décomposer chaque sous bande en des petits blocks (64x64 max), (2) Avoir un embedded bit-stream pour chaque code-block, (3) Le codestream à la sortie est composé d'un ensemble de couches (layers). Le codeur entropique est composé de deux blocs de traitement : le bloc de formation du contexte et le codeur arithmétique (figure 2).

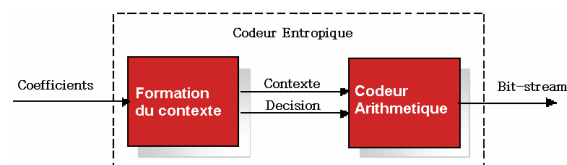


Figure 2 : schéma du codeur entropique

2.2.1. Formation du contexte

La formation du contexte est un processus au niveau des bit-planes qui balaye chaque bit-plane d'un code-block et génère un contexte et une décision. Pour chaque bit-plane (sauf le premier) on effectue trois passes de codage qui sont :

- (1) *significance pass*,
- (2) *Refinement pass*,
- (3) *clean-up pass*.

Pour le premier bit-plane (MSB non nul) on applique seulement le *clean-up pass*.

Dans la figure suivante on représente la décomposition d'un code block en plusieurs bits plans. Chaque bit-plane est balayé à la manière montrée dans les figures 3 et 4.

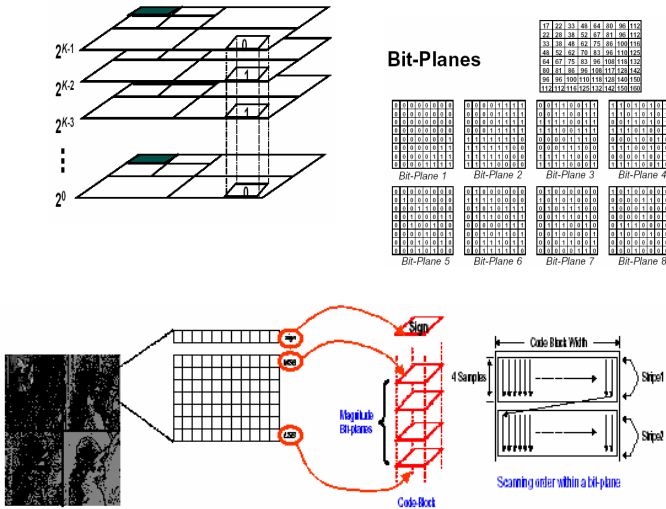


Figure 3 : Décomposition en bit planes

A chaque position du bit-plane une fenêtre de 3x3 voisins est utilisée pour la détermination du contexte.

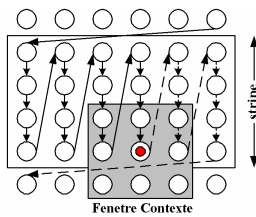


Figure 4 : Fenêtre Contexte

2.2.2. Le codeur arithmétique

Le codeur arithmétique compresse les coefficients de la TO en un codestream en utilisant les contextes générés par le bloc de formation de contexte. Le codeur arithmétique a à son entrée un contexte et un symbole (décision) et un Bitstream en sortie.

3. Modèles de Calcul et Réseaux de Pétri colorés

Les modèles de calcul [5, 6, 7] sont des formalismes permettant de représenter les propriétés des algorithmes et ce dans l'objectif d'une analyse orientée implémentation. Les modèles de calculs incluent: les modèles flot de données, (synchrones, dynamiques), les réseaux de processus, les modèles bases sur CSP, les modèles à base de machines d'états CFMSM, et enfin les réseaux de pétri (RDP) [7]. Les réseaux de pétri, plus particulièrement les RDP colorés présentent de nombreux avantages par rapport aux autres modèles pour le codeur entropique qui manipule des données suivant un flot. Le choix des RDP colorés est du à la compacité de leur représentation et leur niveau d'abstraction. Cette compacité nous permet de représenter des étapes de l'algorithme à des niveaux variés de granularité du parallélisme. Cette approche nous permet donc une méthode top-down avec éventuellement des

éclairages à une granularité plus fine pour des parties de l'algorithme présentant un parallélisme à grain fin. De plus différentes approches ont été proposées pour la synthèse/analyse de circuits à partir de RDP [8-10]. Les RDP colorés sont une extension des RDP. Il existe de nombreux outils permettant la conception et la simulation des RDP Colorés. Notre choix s'est porté sur Design/CPN [11].

4. Modélisation du Codeur Entropique :

La modélisation du codeur entropique part de l'algorithme de base :

Pour chaque Code-Block faire :

(1) Initialisation,

(2) Pour chaque bit-plane du code block faire :

Si bit-plane?premier bit-plane :

(a) Premier passe de codage,

(b) Deuxième passe de codage **Sinon**

(c) Troisième passe de codage.

Le modèle de cet algorithme est représenté dans la figure 5. En effet, une transition est attribuée à chaque passe de codage (P0, P1, P2). La condition pour la séparation entre les premiers bit-planes et le reste des bit-planes a été réalisée à l'aide de la transition « DistBitPlanes ». La place « Code Blocks » contient l'ensemble des code-blocks à traiter. Chaque code-block est défini par le triplet (cb(i), T(i), Nbp_i) ou i est l'indice du code-block cb(i), T(i) sa taille, et Nbp_i le nombre de bit-plane qui le constitue. Dans notre cas, seul trois code-blocks sont représentés avec Nbp1=Nbp2=Nbp3=2.

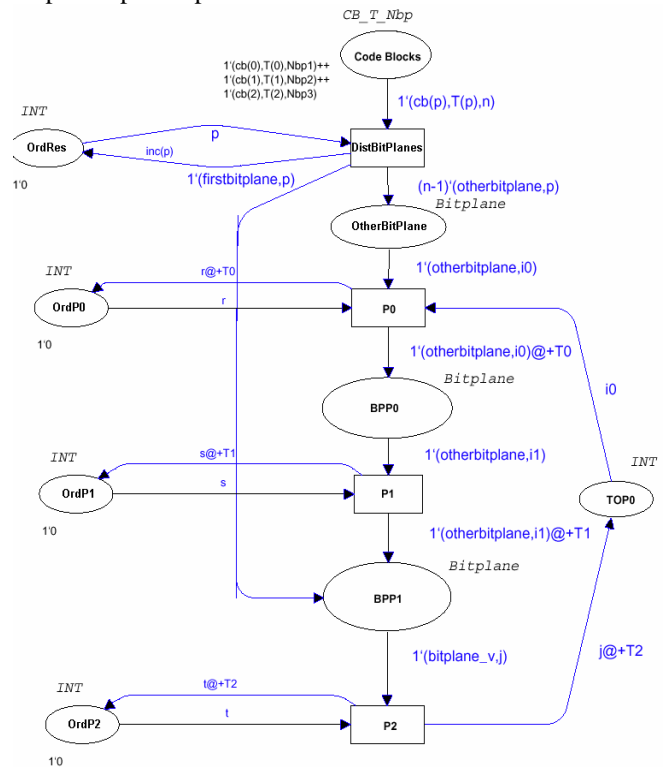


Figure 5 : modèle du codeur entropique.

Le choix du nombre des code-blocks ainsi que le nombre de bit-plane par code-block est totalement arbitraire. Les places « OtherBitPlane », « BPP0 », « BPP1 », représentent les ressources (les bit-planes) à traiter par chaque passe de codage. Pour chaque code-block le premier bit-plane est mis directement dans « BPP1 » et le reste des bit-planes dans « OtherBitplane ». Une place « OrdRes » a été prévue pour ordonnancer le traitement des code-blocks. Elle permet aussi de varier cet ordonnancement surtout lorsque les code-blocks ont des nombres de bit-planes différents. Les places : « OrdP0 », « OrdP1 », « OrdP2 » permettent d'avoir une simulation temporelle conforme à la réalité : ne pas traiter un nouveau bit-plane que si la tâche est libre. Ces dernières n'ont aucune influence sur le comportement du modèle en dehors de la simulation temporelle et peuvent être éliminées. La place « TOP0 » permet de marquer la fin du traitement d'un bit-plane et « P0 » peut commencer le traitement d'un nouveau bit-plane. Pour conserver le même esprit de la modélisation, c'est-à-dire, s'abstraire de toutes architectures, aucune contrainte n'a été imposée sur les ressources (les places ne sont pas bornées). De plus les ressources sont immédiatement disponibles pour chaque passe de codage. La taille et le format des données ne sont pas précisés dans ce modèle, car ils ne doivent pas influencer le comportement de base du modèle. Les premières constatations montrent qu'à ce niveau du système, seul un pipeline est envisageable, à condition de savoir assurer une totale indépendance entre les ressources de chaque code-block. En associant une durée d'exécution à chaque passe de codage (T0, T1, T2), plusieurs configurations sont donc possibles. Le cas idéal pour le fonctionnement du pipeline est $T_0=T_1=T_2$. Si cette condition n'est pas réalisable ou si elle est sous optimale, on peut avoir d'autres configurations qui peuvent influencer sur l'équilibrage de la charge des tâches. Pour tester l'impact de la durée de chaque passe de codage, on effectue l'ordonnancement des trois code-blocks. La simulation montre que le meilleur équilibrage des trois passe de codage est obtenu pour la configuration suivante : $T_2 < T_0$ et $T_2 < T_1$. Cependant, Vu la distribution des bits traités par chaque passe de codage (comme le montre la figure 6 tirée de [13]), cette distribution du temps d'exécution ne reflète pas la réalité, mais la condition reste nécessaire pour garder un meilleur fonctionnement du codeur. Pour l'implémentation du codeur, l'approche de [12] est insuffisante pour assurer le fonctionnement voulu suite à l'analyse réalisée.

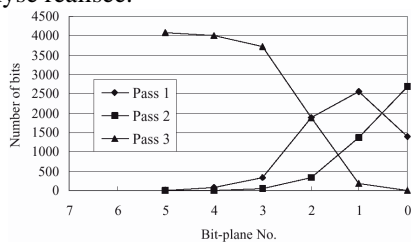


Figure 6 : Distribution des bits traités dans chaque passe de codage

5. Conclusion

L'analyse théorique d'une application dans un modèle de calcul en vue d'une implémentation système sur puce est prépondérante puisqu'elle sert de guide pour l'exploration de différentes implémentations logicielles et matérielles. Le codeur entropique a été ainsi modélisé par réseaux de pétri et a permis une extraction d'un parallélisme temporel entre les trois passes du codeur. Ce parallélisme temporel est difficile à détecter dans des modèles de calcul orientés contrôle plus que données comme le permet les réseaux de pétri. La modélisation par des RDP du codeur entropique est selon notre connaissance la première étude qui a été menée. Dans cette même démarche on a continué l'analyse des sous modules du codeur entropique.

Références :

- [1] JPEG 2000 Part I with Cor.1 Cor.2, Cor.3 and DCor.4, Amd.1, FPDAM.2, ISO/IEC JTC1/SC29/WG1 N2513R2, May 2002
- [2] D. Taubman. High performance scalable image compression with EBCOT. IEEE Transactions on Image Processing, 9(7):1158--70, July 2000.
- [3] O.Hammami, E.Zheng and I.Aouadi - Performance Evaluation of JPEG-2000 on VLIW Architectures, *Proc. of the 14th IEEE International Conference on Microelectronics*, pp.202-205, Beirut, Lebanon, Dec.11-13, 2002.
- [4] I.Aouadi, O.Hammami - A SOPC Oriented FPGA Implementation of the JPEG-2000 Entropy Coder', *IEEE Picture Coding Symposium PCS03*, Saint Malo, France, April 23-25, 2003.
- [5] E.A.Lee and A.Sangiovanni-Vincentelli, "Comparing Models of Computation", in *Proc. of the ICCAD'96*, 1996.
- [6] I.Jeukens and M.Strum, "On the Choice of Models of Computation for Writing Executable Specifications of System level Designs", in *proc. of the 13th Symp. On Integrated Circuits and System Design*, 2000.
- [7] Tadao Murata, *Petri Nets: Properties, Analysis and Applications*, *Proceeding of the IEEE*, Vol. 77 No. 4, April 1989
- [8] P.Rokyta, W.Fengler, T.Hummel, "Electronic System design Automation Using High-Level Petri nets", pp.193-204, in *Hardware Design and Petri Nets*, Kluwer Academic Publishers, 2000.
- [9] J.A. de Oliveira Filho and al. "Petri net Based Interface Analysis for Fast IP-Core Integration", *Proc. of the 1st ACM/IEEE MEMOCODE 2003*.
- [10] A.K.Deb, J.Oberg and A.Juntsch, "Simulation and Analysis of Embedded DSP Systems Using Petri nets", in *Proc. of the 14th IEEE Workshop on Rapid System Prototyping*, 2003.
- [11] S. Christensen, J.B. Jorgensen, and L.M. Kristensen. Design/CPN - A Computer Tool for Coloured Petri Nets. In E. Brinksma, editor, *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 209{223. Springer-Verlag, 1997.
- [12] Carsten Rust, Ac im Rettberg, and Kai Gossens, "From High-Level Petri Nets to SystemC", In *IEEE International Conference on Systems, Man & Cybernetics*, Hyatt Regency, Washington, D.C., USA, 5 - 8 October 2003.
- [13] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen, "Analysis and Architecture Design of Block Coding Engine for EBCOT in JPEG 2000", *IEEE Transaction on Circuits and Systems for video technology*, Vol. 13, No. 3, March 2003