IMPLANTATION SUR DSP D'UNE METHODE RAPIDE D'INDEXAGE POUR LA QUANTIFICATION VECTORIELLE ALGEBRIQUE

J.M. Moureaux, P. Nus*, J.M. van Binneveld*, M. Antonini***

* CRAN - CNRS / Université Henri Poincaré, Nancy 1
11, Rue de l'Université, F-88100 Saint-Dié - FRANCE

** I3S - CNRS / Université de Nice-Sophia Antipolis
250, av. A. Einstein bât. 4 SPI, F-06560 Valbonne - FRANCE

RESUME

Le choix d'une méthode de compression est souvent contraint par son coût calcul et son coût mémoire. La quantification vectorielle algébrique présente l'avantage d'être très rapide et ne nécessite ni la construction ni le stockage d'un dictionnaire comme dans le cas des méthodes de classification. Ces caractéristiques la rendent particulièrement adaptée aux applications de codage bas débit. Cette méthode est cependant délicate à mettre en œuvre, en particulier en ce qui concerne l'indexage des vecteurs quantifiés. Nous avons proposé récemment un nouvel algorithme d'indexage basé sur un compromis efficace coût calcul - coût mémoire. Le but de cet article est de définir une architecture de type DSP adaptée à cet algorithme et permettant d'envisager des applications bas débit avec une transmission proche du temps réel.

ABSTRACT

In data compression applications, the efficiency of a method is strongly linked to its computational complexity and its storage cost. Lattice vector quantization (LVQ) is very fast and does not require the construction of the codebook like LBG-based algorithms. This particularity is very attractive for low bit rate applications. However, LVQ indexing is a key problem for transmission purposes. We recently proposed a new indexing algorithm that provides a good trade-off between computational complexity and storage requirements. This paper is concerned with the design of a specific DSP-based architecture for this new method in a real-time low bit rate coding context.

1. INTRODUCTION

La quantification vectorielle algébrique (QVA) est une méthode de quantification très rapide qui ne nécessite pas la construction et le stockage d'un dictionnaire comme dans le cas des méthodes de classification [2]. Cet avantage la rend particulièrement intéressante dans les applications de compression, puisqu'on peut ainsi utiliser des vecteurs de grande dimension n et des dictionnaires de grande taille L, et approcher la limite débit-distorsion. La QVA est cependant délicate à mettre en œuvre, en particulier en ce qui concerne l'indexage des vecteurs, du fait de la taille élevée des dictionnaires mis en jeu. Différentes méthodes purement analytiques ont été proposées pour résoudre ce problème (une courte description est donnée dans [4]). Si elles présentent l'avantage d'avoir un coût mémoire quasiment nul, elles ne fonctionnent souvent que dans des cas particuliers et sont également coûteuses en calcul donc parfois incompatibles avec le temps-réel et le codage bas débit.

Nous avons proposé récemment une nouvelle méthode d'indexage basée sur un compromis efficace coût calcul - coût mémoire, fonctionnant pour différents réseaux $(\mathbf{Z}^n, D_n)$ et différentes formes de dictionnaires (sphérique, pyramidale) et dimensions de vecteur [3], [4]. Le but de cet article est de proposer une architecture DSP adaptée à cette nouvelle méthode, tant d'un point de vue de la complexité calculatoire, que du coût mémoire, afin d'envisager des applications bas débits avec une transmission proche du temps réel [5]. D'autre part, nous nous focalisons ici uniquement sur l'implantation de l'indexage, le "desindexage" étant similaire et ne présentant pas de difficultés algorithmiques ou d'implantation supplémentaires.

2. PRINCIPES DE LA METHODE

Par rapport aux méthodes existantes, la méthode proposée est une méthode hybride, en ce sens que le calcul de l'index s'effectue pour partie à l'aide d'un développement analytique mais également par la lecture dans une table de correspondance (ce qui permet d'accroître la rapidité). Le principe de cette méthode repose sur le fait qu'un vecteur quelconque $\boldsymbol{x}$ du dictionnaire est l'image par une permutation signée d'un unique vecteur $l(\boldsymbol{x})$ (appelé leader de $\boldsymbol{x}$). $l(\boldsymbol{x})$ appartient à une région fondamentale de l'espace qu'on appelera $\boldsymbol{R}$ englobant un sous-ensemble du dictionnaire [3], [4]. D'autre part, les normes L_1 et L_2 d'un vecteur $\boldsymbol{x}$ étant conservées par toute permutation signée de ses composantes, $\boldsymbol{x}$ et son antécédant $l(\boldsymbol{x})$ auront même énergie, quelle que soit la forme du dictionnaire (pyramidale ou sphérique). Ils pourront ainsi être indexés avec le même préfixe (énergie) mais des suffixes différents (position). Le calcul du préfixe étant trivial, nous nous attacherons dans la suite du papier à ne développer que le calcul du suffixe.

D'après ce qui précède, tous les vecteurs du dictionnaire peuvent être déduits de ceux de $\boldsymbol{R}$ par permutation signée. Ainsi, pour un dictionnaire de rayon R, la région fondamentale $\boldsymbol{R}$ est définie de la façon suivante:

$$\boldsymbol{R} = \left\{ (x_1,\ldots,x_n) \in \mathbf{Z}^n \;\middle|\; 0 \le x_1 \le \ldots \le x_n \le R \text{ and} \sum_i |x_i|^p \le R^p \right\} \quad (1)$$

où p=1 pour un dictionnaire pyramidal et p=2 pour un dictionnaire sphérique.

L'idée principale de la méthode consiste à déterminer l'index de $\boldsymbol{x}$ à partir de celui de son leader $l(\boldsymbol{x})$:

$$\text{Index}(\boldsymbol{x}) = \text{Index}(l(\boldsymbol{x})) + \text{Index}_{\boldsymbol{O}(\boldsymbol{x})}(\boldsymbol{x}) \quad (2)$$

où $\boldsymbol{O}(\boldsymbol{x})$ est l'orbite de $\boldsymbol{x}$, c'est à dire l'ensemble des vecteurs ayant même leader que $\boldsymbol{x}$.

2.1. Calcul de l'index orbital

D'après la définition de la région fondamentale, la transformation qui à tout vecteur $\boldsymbol{x}$ fait correspondre son leader $l(\boldsymbol{x}) \in \boldsymbol{R}$ peut être considérée comme la composition de deux fonctions, l'une assurant le passage à la valeur absolue et l'autre le tri des composantes en ordre croissant. Si chacune d'elles reçoit un index (respectivement $S_{\boldsymbol{x}}$ et $N(|\boldsymbol{x}|)$) et si de plus $n - n_0(\boldsymbol{x})$, est le nombre de coordonnées non nulles de $\boldsymbol{x}$ (où n est la dimension du vecteur et $n_0(\boldsymbol{x})$ le nombre de ses composantes nulles) on peut écrire:

$$\text{Index}_{\boldsymbol{O}(\boldsymbol{x})}(\boldsymbol{x}) = N(|\boldsymbol{x}|)\,2^{n-n_0(\boldsymbol{x})} + S_{\boldsymbol{x}} \quad (3)$$

- $2^{n-n_0(\boldsymbol{x})}$ est le nombre de vecteurs ayant même valeur absolue que $\boldsymbol{x}$.
- $S_{\boldsymbol{x}}$ est la valeur décimale d'un nombre binaire de longueur $n - n_0(\boldsymbol{x})$, dont les bits à 1 représentent les changements de signe et les bits à 0 la conservation du signe.
- $N(|\boldsymbol{x}|)$ représente le nombre de vecteurs précédant $|\boldsymbol{x}|$ dans l'ordre lexicographique. Il est tel que [4]:

$$0 \le N(|\boldsymbol{x}|) < N_{\max} = \frac{n!}{w_1!\ldots w_q!} \quad (4)$$

où q est le nombre de valeurs différentes prises par les coordonnées de $\boldsymbol{x}$ et w_i le nombre d'occurence de la $i$ème valeur (son poids).

2.2. Pre-calcul et stockage de l'index des leaders

Afin d'accroître la rapidité de calcul de la formule (2), nous proposons de stocker les index des leaders dans une table $\boldsymbol{T}$ (appelée table de codage) . Pour cela, les leaders sont générés dans l'ordre lexicographique (à énergie fixée) puis indexés suivant la loi [3], [4]:

$$\text{Index}(\boldsymbol{x}^{k+1}) = \text{Index}(\boldsymbol{x}^k) + \text{offset}(\boldsymbol{x}^k) \quad (5)$$

k étant l'itération et $\text{offset}(\boldsymbol{x}^k) = 2^{n-n_0(\boldsymbol{x})} N_{\max}$.

Cette opération est une opération indépendante de l'indexage en ligne. En d'autres termes, une fois $\boldsymbol{T}$ construite (pour une dimension et un dictionnaire fixés), l'indexage d'un vecteur quantifié quelconque de la source s'effectue selon la formule (2).

3. IMPLANTATION SUR DSP

Le séquencement des tâches liées à la formule (2) est représenté sur la figure 1. Le calcul de $N(|\boldsymbol{x}|)$ est l'opération la plus délicate à cause de la dynamique des nombres multipliés et divisés dans la formule (4). La méthode d'indexage devant évidemment être sans pertes, on ne peut tolérer aucune erreur (même d'un digit) sur l'index final. L'arithmétique classique flottante ne peut donc être utilisée ici à cause de l'erreur relative qu'elle génére rendant impossible la représentation exacte de tous les nombres entre 0 et n! (en particulier pour n=16). L'arithmétique en virgule fixe permet par

contre d'effectuer le calcul sans engendrer d'erreur. Le choix s'oriente donc naturellement vers un processeur travaillant en virgule fixe. La longueur (en bits) des données manipulées dans (4) conditionne également le choix du processeur. Ainsi, d'après (4) $N(|\boldsymbol{x}|)$ s'écrit sur 45 bits (pour $n=16$). Un processeur 24 bits en virgule fixe (type Motorola DSP56002)[1], [6] utilisé en simple ou double précision semble donc particulièrement approprié à la méthode.

4. PERFORMANCES

4.1. Coût stockage

Ce coût est uniquement lié à la table de codage $\boldsymbol{T}$. Pour des raisons informatiques, $\boldsymbol{T}$ est la structure rectangulaire minimale englobant $\boldsymbol{R}$. D'autre part, les composantes des vecteurs du réseau étant entières, elles peuvent être utilisées directement comme adresses de la table, chaque adresse contenant l'index du vecteur correspondant lorsqu'il s'agit d'un leader ou étant vide dans les autres cas. Cette construction rend la recherche de $\mathrm{Index}(l(\boldsymbol{x}))$ dans $\boldsymbol{T}$ particulièrement rapide (adressage direct par les composantes), il entraîne cependant le stockage d'adresses superflues (vecteurs non leaders). Une structure de base type DSP 56002 utilise une mémoire de 128 K mots de 24 bits et permet déjà de stocker des tables correspondant à des dictionnaires de plusieurs millions de vecteurs (par exemple un dictionnaire sphérique de 33 343 681 vecteurs dans le réseau $\mathbf{Z}^8$). Cette limite peut être encore facilement repoussée en paginant la mémoire. Ainsi deux pages mémoires supplémentaires permettent de faire passer le nombre de vecteurs du dictionnaire de 33 343 681 à 105 589 585.

Quand le coût de stockage devient néanmoins prohibitif, une autre solution consiste à changer la structure de $\boldsymbol{T}$ de façon à stocker uniquement les composantes des leaders et leur index associé. En relâchant la contrainte d'adressage très rapide, on diminue considérablement le coût de stockage dû à la table. De plus, le nombre de leaders demeure modeste comparé au nombre total de vecteurs du dictionnaire (même dans le cas de très grands dictionnaires), ainsi la recherche exhaustive de $\mathrm{Index}(l(\boldsymbol{x}))$ représente seulement une faible fraction du temps total d'indexage dû à la formule (2). Dans le cas d'un dictionnaire sphérique dans le réseau $\mathbf{Z}^8$, on a par exemple pour une taille de 4 905 416 097 vecteurs, seulement 10693 leaders, ce qui ne représente qu'un coût stockage de 94 Koctets !

4.2. Coût calcul

Le principal avantage de la méthode proposée est son faible coût calcul, lié seulement à *n* (la dimension des vecteurs) et non à la taille du dictionnaire. Le tableau 1 donne une estimation moyenne en fonction de *n* du coût calcul de chacune des opérations utilisées dans la détermination de l'index final. Il faut noter que cette estimation est donnée dans le cas du mode double précision (nécessaire pour *n* = 16), elle s'avère plus faible dans le cas simple précision (*n* = 4 ou 8) pour lequel la multiplication et la division sont plus rapides. D'après le tableau, le calcul d'un index en dimension 16 prend 2258 cycles instruction. Ainsi par exemple, dans le cas d'une horloge à 66 MHz, le temps total d'indexage vaut $4,2\ \mu s/pixel$. Ce temps peut être réduit en parallélisant la structure de base monoprocesseur définie plus haut. A titre indicatif (dans le cas *n* = 16), l'indexage d'images vidéo couleur au format UIT-T QCIF à 10 images/s $(144\times176\times3/2\times10\ pixels/s)$ nécessiterait seulement deux DSP en parallèle pour une transmission en temps réel.

5. CONCLUSION

En conclusion, les travaux présentés ici montrent que la nouvelle méthode d'indexage proposée est parfaitement implantable sur des calculateurs bas coût de type DSP et conduit à des durées de calcul compatibles avec le temps réel. Grâce à un compromis efficace entre coût de stockage et coût calcul, cette méthode permet une stratégie adaptative de codage en termes de débit binaire (en modifiant la taille du dictionnaire, sa forme, ou encore le type de réseau). L'utilisation du DSP permet aisément la modification de tous ces paramètres, ce qui représente un avantage considérable dans des applications telles que les applications embarquées, par exemple.

REFERENCES

[1] E. Carey, "Real-time Image Transform and Processing using a quad-TMS 320C44 Board", *ICSPAT*, Boston, USA, 7-10 October 1996, pp. 1453-1458.

[2] A. Gersho and R.M. Gray, "Vector Quantization and Signal Compression", Kluwer Academic Publishers, 1992.

[3] J.M. Moureaux, P. Loyer, M. Antonini, "Efficient Indexing Method for Lattice Quantization Applications", *IEEE International Conference on Image Processing (ICIP)*, Lausane, Switzerland, 16-19 September 1996, pp. 447-450.

[4] J.M. Moureaux, P. Loyer, M. Antonini, "Low Complexity Indexing Method for Z^n and D_n Lattice Quantizers", preprint I3S-CNRS-UNSA No. 96-51, submitted to *IEEE Transactions on Communications*, October 1996.

[5] W. Smith, J.M. Smith, "Handbook of real-time FFT algorithms to Product testing", *IEEE Press*, ISBN 0-7803-1091-8, 1995.

[6] DSP 56002 Digital Signal Processor, User's Manual, DSP 56002 UM/AD-REV1, Motorola-Inc, 1993.

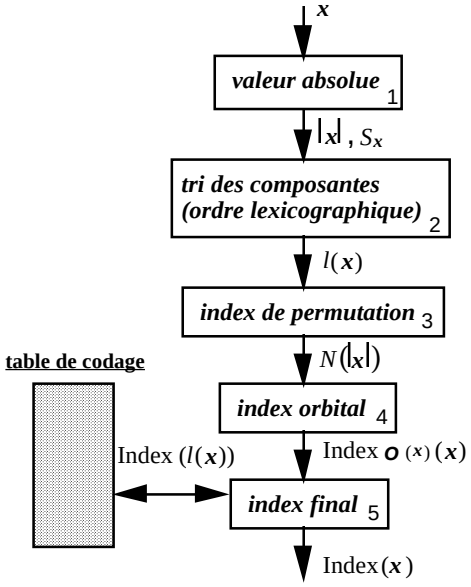

figure 1 : séquencement des tâches relatives à la formule (2)

tâches	cycles instruction / vecteur (estimation moyenne)
1 ->	$3.5n^2 + 57n + 7.5\lceil \log_2 n! \rceil + 81$
5	$n + 15$
total	$3.5n^2 + 58n + 7.5\lceil \log_2 n! \rceil + 96$

tableau 1 : coût calcul (DSP 56002)