République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Médéa
Faculté de Technologie
Département du Génie Électrique

THÈSE

En vue de l'obtention du diplôme de

Doctorat en Sciences

OPTION : *Génie Électrique*

THÈME :

ÉTUDE ET CONCEPTION D'UN CODEUR VIDÉO HAUTE DÉFINITION SUR UNE PLATEFORME MULTI-CORE

PRÉSENTÉ PAR :

MAAZOUZ MOHAMED

LE : 23/07/2019

Devant le jury composé de :

M. OULD ZMIRLI Mohamed	Prof.	Université de Médéa	Président
M. GUESSOUM Abderrezak	Prof.	Université de Blida	Examinateur
Mme. BENBLIDIA Nadjia	Prof.	Université de Blida	Examinatrice
M. SADOUDI Said	MCA.	EMP, Alger	Examinateur
M. BOUHEDDA Mounir	MCA.	Université de Médéa	Examinateur
M. BATEL Noureddine	Prof.	Université de Médéa	Directeur de Thèse

Année Universitaire : 2018/2019

Dédicace

Au nom de dieu le clément et le miséricordieux louange à

ALLAH *le tout puissant.*

Je dédie ce modeste travail en signe de respect, de reconnaissance et de remercièment :

A mes parents

A ma femme et enfants

A la mémoire de ma belle-mère

A mes amis, et à tous mes proches.

Remerciements

En tout premier lieu, je remercie Allah, tout puissant, de m'avoir donné la force pour survivre, ainsi que l'audace pour dépasser toutes les difficultés.

Le travail présenté dans cette thèse a été réalisé au laboratoire des systèmes électroniques avancés de la faculté de Technologie de l'Université de Médéa, sous la direction du Prof. Batel Noureddine. Ma plus grande gratitude va à mon encadreur, pour sa disponibilité et la confiance qu'il m'a accordée. J'ai profité pendant longtemps du savoir et du savoir-faire dont j'ai pu bénéficier au cours de nombreuses discussions. J'aimerais aussi le remercier pour l'autonomie qu'il m'a accordée, et ses précieux conseils qui m'ont permis de mener à bien ce travail.

Je voudrais aussi remercier mon co-directeur de thèse, Prof. Masmoudi Nouri, directeur du laboratoire LETI à l'ENIS, Sfax Tunisie. Il m'a transmis l'enthousiasme dans le travail quotidien en m'inondant, chaque fois, de nouvelles idées et de nouveaux défis. Je sens que je suis très chanceux et honoré d'avoir pu travailler avec lui et son équipe et j'espère que nos chemins se croiseront à nouveau au cours de notre vie professionnelle et personnelle.

Je souhaite aussi remercier mes parents, ainsi que ma femme, pour l'intérêt qu'ils ont toujours porté à ce que je réalise. Merci d'avoir toujours été là et de m'avoir tant aidé et encouragé.

Je suis très honoré de mon jury de thèse et je tiens à remercier :

Prof. Ouldzmirli Mohamed, directeur du laboratoire LSEA, pour l'honneur qu'il m'a fait en acceptant d'être président de mon jury de thèse. Je tiens à l'assurer de ma profonde reconnaissance pour l'intérêt qu'il porte à ce travail.

Prof. Guessoum Abderrezak et Prof. Benblidia Nadjia de l'université de Blida, pour l'honneur qu'ils m'ont fait pour leur participation à mon jury de thèse en qualité d'examinateurs de mon travail, pour le temps consacré à la lecture de cette thèse, et pour les suggestions et les remarques judicieuses qu'ils m'ont indiquées.

Monsieur Sadoudi Said, maître de conférences à l'École Militaire Polytechnique, et Monsieur Bouhedda Mounir, maître de conférences à l'université de Médéa, pour l'intérêt qu'ils ont manifesté en participant en qualité d'examinateurs dans ce jury.

Je souhaite exprimer ma gratitude à mon ami Nejmeddine Bahri, pour son soutien permanent et ses encouragements incessants.

Afin de n'oublier personne, mes vifs remerciements s'adressent à tous ceux qui m'ont aidée à la réalisation de ce modeste travail.

ملـخــص

في الآونة الأخيرة ، ازدادت شعبية تطبيقات الوسائط المتعددة واستخدامها بشكل كبير بسبب اعتماد الأجهزة المحمولة من قبل السوق السائدة. ونتيجة لذلك ، يلعب ضغط الصور والفيديو دورًا حيويًا ، حيث يتيح تخزين ونقل كميات كبيرة من محتوى الوسائط المتعددة. ومع ذلك ، فإن معالجة هذه المعلومات تتطلب موارد حوسبة عالية ، ومن هنا تأتي أهمية تحسين الأداء الحسابي لخوارزميات الضغط هذه.

في ترميز الفيديو عالي الكفاءة (HEVC) ، يتم الحصول على الكفاءة في أداء الضغط بشكل أساسي من خلال دعم حجم أكبر لوحدة التشفير (CU) والمزيد من أنماط التنبؤ. ومع ذلك ، نظرًا لأن المشفر يجب أن يمر بجميع الخيارات الممكنة لإيجاد أفضل طريقة لتشفير البيانات ، فقد أدت هذه المرونة الكبيرة من حيث حجم وحدة التشفير وأنماط التنبؤ إلى زيادة كبيرة في وقت الترميز.

في هذه الأطروحة ، تم اقتراح مجموعة من الخوارزميات والتقنيات لتقليل تعقيد العمليات الحسابية في مشفر HEVC . إذ أنه يمكن الحد من التعقيد الحسابي إلى حد كبير مع خسارة ضئيلة من كفاءة الترميز. في البداية، اقترحنا تطبيقًا مخصصًا لمعالجات DSP المتخصصة ، حيث يعتمد تقليل التعقيد على لغة البرمجة اسسمبلر لبعض أجزاء البرنامج المرجعي. لم يتأثر أداء الضغط بهذا النوع من التحسين ، في حين كان التقليل وقت الترميز غير كاف.

في المرة الثانية ، اقترحنا تحسينا خوارزميا. يتعلق هذا التحسين بعملية تقسيم وحدات الترميز ، حيث يتم ترميز المناطق منخفضة التعقيد (متجانسة) بواسطة وحدات أكبر ، كما يتم ترميز المناطق الغنية بالنسيج بواسطة وحدات أصغر. عن طريق ضبط المتغيرات ، يمكن للخوارزمية تغيير المفاضلة بين وقت الترميز وكفاءة الترميز. حسب نتائج التجارب، يمكن أن يصل الحد الأقصى للتحسين في متوسط وقت الترميز إلى 41% بزيادة في التدفق قدرها 0.69% .

كما اقترحنا أيضًا تحقيقًا متوازيًا للمشفر HEVC موجه للبنى متعددة النواة. في اعدادات اللاينتر ، يتم تطبيق التوازي على مستوى الصورة باستخدام نموذج اوبنيع للبرمجة المتوازية. تظهر النتائج التجريبية تحسينات قريبة من المثالية للبنى المتجانسة. من ناحية أخرى ، فإن هذا النوع من التوازي مع الحمولة شبه المتوازنة لا يناسب المنصات غير المتجانسة (حالة بطاقة Cubieboard4 الخاصة بنا والمجهزة بمعالجين مختلفين رباعي النواة).

كلمات مفتاحية : مشفر الفيديو HEVC ، معالجات DSP ، تقسيم وحدات الترميز، منصات متعددة النوى، التوازي.

Résumé

Récemment, la popularité des applications multimédias et de leur utilisation a considérablement augmenté en raison de l'adoption des appareils mobiles par le marché grand public. En conséquence, la compression d'images et de vidéos joue un rôle capital, permettant le stockage et la transmission de gros volumes de contenu multimédia. Cependant, le traitement de ces informations nécessite des ressources de calcul élevées, d'où l'importance d'améliorer les performances de calcul des algorithmes de compression.

Dans le codage vidéo à haute efficacité (HEVC), le gain en performance de compression est obtenu principalement en prenant en charge une taille plus grande d'unité de codage (CU) et davantage de modes de prédiction. Cependant, comme l'encodeur doit parcourir tous les choix possibles pour trouver le meilleur moyen de coder les données, cette grande taille de bloc et de modes de prédiction a entraîné une augmentation considérable du temps de codage.

Dans cette thèse, un ensemble d'algorithmes et de techniques est proposé pour réduire la complexité des calculs dans l'encodeur HEVC. Cependant, la réduction en complexité de calcul se fait au prix d'une perte négligeable d'efficacité de codage. Dans un premier temps, nous avons proposé une implémentation dédiée au processeur spécialisé DSP, où le gain en temps de calcul repose sur la programmation en assembleur de certaines parties du code de référence. Les performances de compression n'ont pas été affectée par ce type d'optimisation, de même que la réduction en temps d'encodage n'était pas non plus satisfaisante.

Dans un second temps, nous avons proposé une optimisation algorithmique. Cette optimisation concerne le processus de partitionnement des unités de codage, où les zones à faible texture (homogène) sont codées par des unités plus larges, tandis que les régions riches en texture sont codées davantage par des unités de taille plus petite. En ajustant les paramètres, l'algorithme peut changer le compromis entre le temps de codage et son efficacité. Dans les tests effectués, la réduction maximale en temps moyen d'encodage peut atteindre 41% avec une augmentation de 0.69% du débit.

Nous avons également proposé une réalisation parallèle de l'encodeur HEVC, avec les architectures multi-cœurs. Pour la configuration All-Intra, le parallélisme au niveau des images a été mis en œuvre, en utilisant le paradigme OpenMP de la programmation parallèle. Les résultats expérimentaux montrent un gain proche de l'idéal pour les architectures homogènes. Par contre, ce type de parallélisme à charge quasi-équilibré, n'est pas approprié pour les plateformes hétérogènes (cas de notre carte Cubieboard4, munie de deux processeurs quad-core différents).

Mots clés : Codage vidéo HEVC, Processeurs DSP, Partitionnement des unités de codage, Plateformes multi-cœurs, Parallélisme.

Abstract

Recently, the popularity of multimedia applications and their use has increased dramatically due to the adoption of mobile devices by the consumer market. As a result, image and video compression play a vital role, allowing the storage and transmission of large volumes of multimedia content. However, the processing of this information requires high computing resources, hence the importance of improving the computational performance of the compression algorithms.

In High-Efficiency Video Coding (HEVC), the gain in compression performance is achieved primarily by supporting a larger coding unit (CU) size and more prediction modes. However, as the encoder has to go through all the possible choices to find the best way to encode the data, this large block size and prediction modes has resulted in a considerable increase in coding time.

In this thesis, a set of algorithms and techniques is proposed to reduce the complexity of calculation in the HEVC encoder. However, the reduction in computational complexity is achieved at the cost of a negligible loss of coding efficiency. First, we proposed an implementation dedicated to the specialized DSP processor, where the gain in computation time is based on assembler programming of certain parts of the reference code. The compression performance was not affected by this type of optimization, and the reduction in encoding time was no longer satisfactory.

In a second time, we proposed an algorithmic optimization. This optimization concerns the partitioning process of coding units, where low-texture (homogeneous) areas are encoded by larger units, while texture-rich regions are further coded by smaller units. By adjusting the parameters, the algorithm can change the trade-off between coding time and coding efficiency. In the tests carried out, the maximum reduction in average encoding time can reach 41% with an increase of 0.69% of the bit rate.

We also proposed a parallel realization of the HEVC encoder, with multi-core architectures. For All-Intra configuration, image-level parallelism has been implemented, using the OpenMP paradigm of parallel programming. The experimental results show a gain close to ideal for homogeneous architectures. On the other hand, this type of quasi-balanced load parallelism is not appropriate for heterogeneous platforms (case of our Cubieboard4 card, equipped with two different quad-core processors).

Key words : HEVC video coding, DSP processors, Coding units partitioning , Multi-core platforms, Parallelism.

Table des matières

Table des figures VIII

Liste des tableaux X

Introduction générale 1

1 La norme de compression vidéo H.265/HEVC 4
- 1.1 Introduction . . . 5
- 1.2 Historique des standards/normes de codage vidéo . . . 6
- 1.3 Vue d'ensemble de la norme H.265/HEVC . . . 11
- 1.4 L'optimisation Débit-Distorsion (RDO : *Rate-Distorsion Optimization*) 42
- 1.5 Conclusion . . . 45

2 Optimisation et implémentation de l'encodeur HEVC sur DSP TMS320C6678 47
- 2.1 Introduction . . . 47
- 2.2 Description de la plateforme à DSP TMS320C6678 . . . 48
- 2.3 Environnement de développement . . . 52
- 2.4 Considérations pratiques pour l'implémentation du code HM sur DSP . 54
- 2.5 Optimisation de l'encodeur HEVC pour l'implémentation sur DSP . . . 56
- 2.6 Conclusion . . . 73

3 Optimisation du processus de partitionnement des unités de codage 75
- 3.1 Introduction . . . 75
- 3.2 Travaux connexes . . . 76
- 3.3 Optimisation du processus de partitionnement . . . 78
- 3.4 Résultats et discussion . . . 86
- 3.5 Conclusion . . . 92

4 Implémentation parallèle de l'encodeur HEVC en configuration AI 93
- 4.1 Introduction . . . 94
- 4.2 Paradigmes de programmation parallèle . . . 95
- 4.3 Niveaux de parallélisme pour l'encodeur HEVC . . . 98
- 4.4 Travaux connexes . . . 102
- 4.5 Description de l'algorithme proposé pour l'implémentation parallèle . . 104

4.6 Résultats d'implémentation sur une plateforme hétérogène de type Cubieboard4 107
4.7 Résultats d'implémentation sur une plateforme homogène de type HPC 113
4.8 Conclusion 114

Conclusions et perspectives **116**

Production Scientifique **120**

Bibliographie **121**

Table des figures

Figure 1 Chaîne conventionnelle de traitement du signal vidéo. 1

Figure 1.1 Évolution des standards de codage vidéo 10
Figure 1.2 Exemple de division d'une séquence vidéo en plusieurs GOPs . . . 14
Figure 1.3 Exemple de partitionnement d'une image en 8 Slices 14
Figure 1.4 Schéma de principe de l'encodeur HEVC [15] 15
Figure 1.5 Positions des échantillons chroma en codage vidéo 16
Figure 1.6 Exemple de partitionnement d'une CTU en CUs 17
Figure 1.7 Les tailles possibles pour les blocs de prédiction PBs en HEVC . . 18
Figure 1.8 Exemple de partitionnement d'un CTB de 64×64 [18] 19
Figure 1.9 Directions et modes de prédiction intra adopté par la norme HEVC 20
Figure 1.10 Pixels de référence utilisés par la norme HEVC en intra-prédiction 22
Figure 1.11 Pixels de référence utilisés par la norme HEVC en intra-prédiction 22
Figure 1.12 L'inter-prédiction d'un bloc à partir de plusieurs images de référence. Le vecteur de mouvement est défini par les déplacements Δx et Δy. 27
Figure 1.13 Positions entières et fractionnaires des pixels de Luminance 29
Figure 1.14 Positions des candidats pour le mode de fusion en HEVC : a) candidat temporel, b) candidats spatiaux 32
Figure 1.15 Les modes de balayage d'une partition TU 36
Figure 1.16 Représentation graphique de la configuration AI 40
Figure 1.17 Exemple de codage utilisant la configuration LD 41
Figure 1.18 Représentation graphique de la configuration RA 42
Figure 1.19 Exemple de courbes Distorsion-Débit 43

Figure 2.1 Vue d'ensemble du DSP TMS320C6678 49
Figure 2.2 Structure interne du C66x 50
Figure 2.3 L'environnement Code Composer Studio de Texas Instruments . . 53
Figure 2.4 Les étapes à suivre lors du développement d'un code. 55
Figure 2.5 Exemple de calcul dans un module SAD (xGetSad8). 65
Figure 2.6 Exemple de calcul dans le module SSE (xGetSSE). 66
Figure 2.7 L'algorithme de calcul dans la transformée Hadamard à huit points. 67
Figure 2.8 Temps d'encodage HEVC pour la configuration RA. 71

Figure 3.1 Exemple de partitionnement d'une image en Slices et CTBs, et d'un bloc CB en PB et TB 79
Figure 3.2 Exemple structurel du concept quad-tree en HEVC avec CU = 64×64 et profondeur = 3. 79
Figure 3.3 Résultat de partitionnement de la première image de la séquence BasketballPass (taille : 416x240, QP=37) par le software de référence HM16.6 81
Figure 3.4 L'algorithme proposé pour le partitionnement rapide des CUs : version initiale 82
Figure 3.5 L'algorithme proposé pour le partitionnement rapide des CUs, 2^{eme} version 84
Figure 3.6 Voisinage d'un pixel 85
Figure 3.7 Résultats des algorithmes de détection de contour appliqués à la première image de la séquence BasketballPass (416x240). (a) filtre Sobel, (b) filtre de Prewitt, (c) filtre de Canny et (d) notre algorithme 86
Figure 3.8 Résultats de partitionnement de la première image de BasketballPass (taille : 416x240, QP = 37) en utilisant l'algorithme de partitionnement rapide proposé. 89

Figure 4.1 Système à mémoire partagée 96
Figure 4.2 Système à mémoire distribuée 97
Figure 4.3 Système hybrides : mémoire partagée et mémoire distribuée 98
Figure 4.4 Exemple de partitionnement d'image en segments de Slice, segments indépendants et leurs segments dépendants associés. . . 100
Figure 4.5 Partitionnement d'image en Tiles. 101
Figure 4.6 Dépendances de codage et parallélisation au niveau WPP. 102
Figure 4.7 L'algorithme proposé pour le codage AI HEVC parallèle. 105
Figure 4.8 Accélération de la réalisation parallèle avec Cubieboard4. 110
Figure 4.9 Représentation des threads d'exécution de la réalisation parallèle avec Cubieboard4. 111
Figure 4.10 Charge de travail des processeurs utilisant un, deux, quatre, six et huit threads (du haut vers le bas). 112
Figure 4.11 Accélération de la réalisation parallèle avec IBM® × 3550 M3. . 114

Liste des tableaux

Table 1.1 Correspondances entre modes de prédictions et paramètres d'angle 21
Table 1.2 L'angle inverse en fonction du paramètre d'angle (B=(256 * 32)/A) . . . 21
Table 1.3 Les tailles des blocs PU et modes de prédiction qui sont concernés par le filtre de lissage des pixels de référence en HEVC . . . 27
Table 1.4 Les tailles des blocs PU et modes de prédiction concernés par le filtre de lissage des pixels de référence en HEVC . . . 30

Table 2.1 L'impact des options d'optimisation sur les performances de calcul de SAD . . . 59
Table 2.2 Les instructions SIMD utilisées pour le calcul du SAD . . . 63
Table 2.3 Les instructions SIMD supplémentaires utilisées pour le calcul du SSE . . . 67
Table 2.4 Les instructions SIMD utilisées pour le calcul du HAD . . . 68
Table 2.5 Segmentation et usage de la mémoire . . . 69
Table 2.6 Le gain en temps d'exécution pour le calcul du SAD, SSE et HAD 70
Table 2.7 Mesure des performances des algorithmes proposés . . . 72
Table 2.8 Paramètres d'encodage considérés dans les tests . . . 73

Table 3.1 Choix du seuil pour chaque approche . . . 83
Table 3.2 Performances de l'algorithme N° 1 de partitionnement rapide proposé par rapport à l'HM16.6 de référence . . . 88
Table 3.3 Performances de l'algorithme de partitionnement rapide proposé par rapport à l'HM16.6 de référence . . . 90
Table 3.4 Comparaison des performances du partitionnement rapide proposé avec certains travaux connexes . . . 91

Table 4.1 Caractéristiques de la plateforme Cubieboard4 et de l'environnement de développement . . . 108
Table 4.2 Performances de l'implémentation parallèle de l'encodeur HEVC en configuration AI . . . 109
Table 4.3 Caractéristiques de la plateforme IBM® System x3550 M3 et de son environnement de développement . . . 113

Introduction générale

Contexte

Depuis plus d'un siècle, le secteur de la production vidéo s'est considérablement développé en réponse à l'introduction de nouvelles technologies. Il est évident, que la vidéo est devenue un produit très utilisé dans notre vie quotidienne, grâce à l'évolution du Web et des outils de partage tels que Youtube, Vimeo ou encore Facebook. La lecture est de plus en plus remplacée par la consultation de vidéos courtes de 30 à 60 secondes, dans le cas de promotion d'un produit, d'un service ou d'une entreprise. Techniquement, une séquence vidéo peut provenir de diverses sources. Celles-ci incluent la capture de scènes naturelles, la numérisation de films, l'enregistrement analogique de vidéos, ou les vidéos synthétiques, telles que des animations générées par ordinateur. En outre, des mélanges ou combinaisons des sources susmentionnées sont possibles. Néanmoins, les données brutes de la vidéo ne peuvent être directement manipulées ainsi dans cette forme. La configuration d'un système complet de traitement vidéo peut être organisée selon le schéma fonctionnel de la Figure 1.

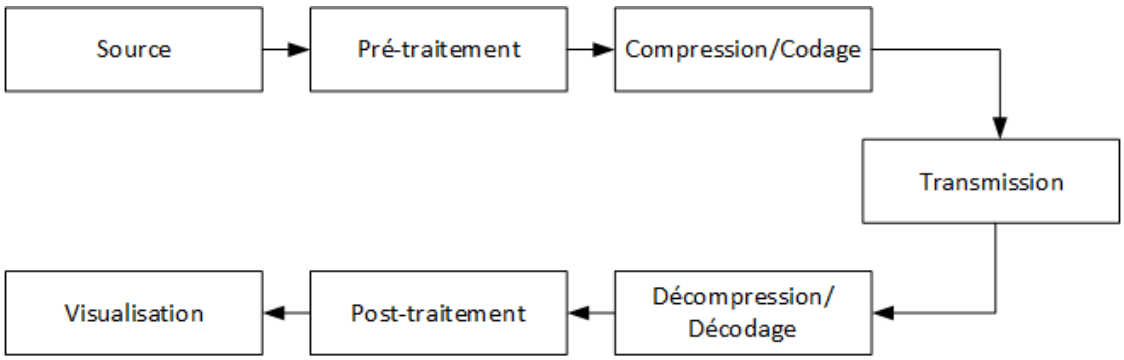


Figure 1. Chaîne conventionnelle de traitement du signal vidéo.

Malgré la limitation des besoins en compression due aux progrès que connaissent les périphériques de stockage et des réseaux, la compression reste une étape incontournable dans la chaîne de traitement du signal vidéo. En effet, l'amélioration des dispositifs d'acquisition et d'affichage et les besoins accrus des utilisateurs, sollicitent la proposition d'algorithmes de codage de plus en plus performants. Cette tâche fût l'objet, depuis longtemps, des organismes de normalisation tels que le groupe d'experts (MPEG) sous l'autorité de l'organisation internationale de normalisation (ISO), la commission électrotechnique internationale (IEC), et le groupe d'experts du codage vidéo (VCEG), appartenant à l'union internationale des télécommunications (IUT-T).

En 2003, la norme H.264/MPEG-4 AVC a été définie, assurant un taux de compression d'environ 50% par rapport à la norme MPEG-2, définie en 1993. En janvier 2013, la norme HEVC a été publiée, pour remplacer la norme H.264/MPEG-4 AVC, offrant par là, 50% de gain en débit pour la même qualité perceptible. Le travail dans cette thèse, a débuté en 2013 avec la norme HEVC qui n'était pas entièrement achevée. Par conséquent, l'accent a été mis sur l'optimisation de la norme HEVC avec les nouvelles techniques qui pourraient être adoptées dans les variantes à venir.

Contributions

Différentes propositions basées sur le critère débit-distorsion (RD) et visant à optimiser l'encodeur HEVC, ont été développées au cours de cette thèse. Elles peuvent être divisées en trois catégories. La première catégorie comprend l'optimisation du software de référence (HM), en vue d'une implémentation utilisant la dernière technologie de processeurs DSP de Texas Instruments. Sur la base des caractéristiques architecturales du DSP, différentes optimisations SIMD (Single-Instruction-Multiple-Data) sont adoptées pour minimiser le temps d'encodage des fonctions les plus lourdes, dans le software de référence. Cette technique conduit à une optimisation sans altération de l'efficacité du codage en termes de débit et de qualité (mesurée en PSNR).

La méthode proposée dans la deuxième catégorie concerne l'optimisation algorithmique, où un gain en temps d'encodage est achevé au détriment de l'efficacité de compression. La dégradation est exprimée par une légère augmentation en débit et

une faible diminution en PSNR.

La troisième catégorie comprend une contribution à l'implémentation parallèle de l'encodeur HEVC en configuration All-Intra (AI). Deux plateformes multi-cœurs sont utilisées dans cette partie, à savoir une plateforme hétérogène embarquée et une plateforme homogène dédiée aux calculs à haute performance HPC.

Organisation de la thèse

Les chapitres de la thèse sont organisés comme suit :

Le chapitre 1 donne un aperçu sur la norme de codage vidéo HEVC. Un bref historique est présenté en premier lieu. Ensuite, les différents blocs de la chaîne de compression sont décrits en détail.

Le chapitre 2 reprend l'optimisation et l'implémentation de l'encodeur HEVC sur le DSP TMS320C6678. L'optimisation consiste à profiter de l'architecture SIMD (Single Instruction Multiple Data) du DSP. Ce type d'architecture inclue plusieurs unités de calcul, pouvant exécuter une instruction sur plusieurs données d'une façon simultanée.

Le chapitre 3 présente une deuxième contribution à l'optimisation de l'encodeur HEVC par proposition d'une optimisation algorithmique. La complexité du mécanisme de partitionnement est réduite en adoptant un algorithme rapide, permettant d'anticiper l'arrêt du processus de division des unités de codage CU.

Le chapitre 4 présente l'implémentation parallèle de l'encodeur HEVC en configuration AI. La stratégie adoptée dans cette partie conduit à une réduction en temps d'encodage sans pertes en performances de compression. Deux plateformes sont considérées dans les expériences : la première est une carte multi-cœurs hétérogène dédiée aux applications embarquées ; et la deuxième consiste en une station de calcul intensif (HPC) homogène.

Une conclusion générale présente un résumé des techniques et méthodes proposées dans l'optimisation de l'encodeur HEVC, en soulignant les différentes contributions originales de ce travail. Les perspectives de futurs travaux et idées d'investigation plus poussées dans le domaine d'optimisation de la norme HEVC, sont également présentées.

Chapitre 1

La norme de compression vidéo H.265/HEVC

Sommaire

1.1 Introduction **5**
1.2 Historique des standards/normes de codage vidéo **6**
1.2.1 Organismes de standardisation 7
1.2.2 Groupes de travail 8
1.3 Vue d'ensemble de la norme H.265/HEVC **11**
1.3.1 Représentation numérique de la vidéo 12
1.3.2 Mécanisme de partitionnement d'une séquence vidéo . . . 13
1.3.3 Partitionnement de l'image en unités de traitement 15
1.3.4 La prédiction spatiale : *Intra-prédiction* 19
1.3.5 La prédiction temporelle : *Inter-prédiction* 26
1.3.6 La transformée et la quantification en HEVC 34
1.3.7 Le codage entropique 36
1.3.8 Les Filtres en boucle 37
1.3.9 Options et configurations d'encodage en HEVC 39
1.4 L'optimisation Débit-Distorsion (RDO : *Rate-Distorsion Optimization*) **42**
1.4.1 Mesure de distorsion 44
1.4.2 Mesure de la qualité 45
1.5 Conclusion **45**

1.1 Introduction

Le terme multimédia peut être défini comme étant la combinaison de plusieurs sources vidéo, audio, image et texte. Les besoins en communications multimédia ont augmenté rapidement au cours des deux dernières décennies dans divers domaines tels que la télévision, le divertissement, les services interactifs, les télécommunications, les conférences, l'internet, les appareils électroniques grand public (EGP), la médecine, la sécurité, le trafic, la défense,... Habituellement, les données vidéo et audio doivent être compressées avant le stockage ou la transmission, car le volume de données d'origine est trop volumineux et par ailleurs, les données compressées doivent être décompressées avant affichage ou traitement ultérieur.

La compression est également appelée encodage ou codage et la décompression, décodage. Par conséquent, le logiciel ou le dispositif permettant de compresser et de décompresser les données vidéo/audio est appelé respectivement encodeur (codeur) et décodeur vidéo/audio. Le codeur et le décodeur sont abrégés en codec pour plus de commodité. Bien que de nombreux algorithmes de codage audio et vidéo aient été développés, ce sont les normes de codage audio et vidéo, qui garantissent l'interopérabilité entre logiciel et matériel émanant de plusieurs fournisseurs et qui rendent les communications multimédia pratiques.

Le processus de base des outils de codage multimédia consiste à supprimer la redondance des données multimédia [1]. Il s'agit principalement des redondances spatiales et temporelles que nous devons éliminer dans la tâche d'un codage vidéo [2]. Il existe également d'autres redondances, telles que la redondance perceptuelle de la vision humaine. Un bon système de codage vidéo doit prendre en compte tous ces types de redondances, afin de concevoir un ensemble d'outils qui devrait être une solution globale, optimale, et non restreinte à certaines redondances spécifiques.

Sans compression, le volume important de données vidéo dépassera facilement la capacité de tout périphérique de stockage disponible [3]. Par exemple, le format 1920×1080 (1920 pixels horizontales par 1080 lignes verticales) à 30 images par seconde, dont les chiffres sont à multiplier encore par 8 bits pour chacune des trois composantes (03 couleurs ou luminance et 02 chrominance). Le volume total des

données d'un film est environ de 930 Go, ce qui dépasse de loin la capacité du périphérique de stockage sur disque conventionnel (CD ROM 700Mo ou DVD 4 Go).

Même avec la nouvelle technologie, un disque Blu-ray à double couche ne peut stocker qu'environ 5 minutes de données vidéo brutes. De plus, le débit total requis serait d'environ 1,5 Gb/s. Cependant, selon les normes de télédiffusion (PAL, SECAM ou NTSC), seule une largeur de bande de canal de 4 à 6 MHz peut être allouée, ce qui est loin du débit de données brutes. Par conséquent, ces données brutes doivent être compressées selon un chiffre d'environ 100 :1 $\sim$ 200 :1, avec une qualité relativement élevée et le moins d'artefacts visibles possible. Parallèlement à l'augmentation foudroyante des données vidéo, en particulier les vidéos haute définition (HD), voire ultra-haute définition (UHD), la technique de compression est de plus en plus requise pour stocker et transmettre efficacement ces données.

1.2 Historique des standards/normes de codage vidéo

Les normes sont essentielles pour la communication, elles assurent un langage commun compris par l'émetteur et le récepteur. Dans les systèmes de communication multimédia, le langage est souvent défini comme une syntaxe de flux binaire normalisée. L'adoption de normes par les fabricants d'équipements et les fournisseurs de services, augmente le volume des ventes et réduit les coûts, tout en permettant aux consommateurs plus de liberté de choix parmi les fabricants.

Pour la transmission du contenu vidéo ou multimédia, les normes jouent un rôle encore plus important. Non seulement l'émetteur et le récepteur doivent utiliser le même langage, mais le langage doit également être efficace (assurer un taux de compression élevé), en raison du nombre relativement élevé de bits requis pour la transmission. Notez cependant que les normes ne spécifient pas tout le processus de communication, la seule spécification concerne la syntaxe du flux binaire et donc le processus de décodage. Par ailleurs, la norme laisse généralement le traitement du codage ouvert aux fournisseurs [4]. C'est la philosophie de normalisation minimale adoptée par la plupart des normes vidéo et multimédias, dont l'objectif est de laisser de la place à la concurrence entre les différents fournisseurs sur les technologies de

codage et permettre l'intégration des technologies futures dans les normes, au fur et à mesure de leur maturité. La conséquence est qu'une norme ne garantit pas la qualité d'un codeur vidéo, mais garantit que tout décodeur conforme à la norme peut recevoir et décoder correctement le flux binaire produit par n'importe quel codeur.

1.2.1 Organismes de standardisation

1.2.1.1 ITU-T

L'union internationale des télécommunications (ITU) est l'institution spécialisée dans le domaine des télécommunications. Le secteur de la normalisation des télécommunications de l'ITU (ITU-T) est un organe permanent du ITU. L'ITU-T est chargé d'étudier les questions techniques, d'exploitation et tarifaires et de publier des recommandations à leur sujet en vue d'une normalisation des télécommunications à l'échelle mondiale [5].

La mission de l'ITU-T est aussi de fournir un lieu unique au monde permettant à l'industrie et aux gouvernements de collaborer pour favoriser l'élaboration et l'utilisation de normes internationales interopérables, non discriminatoires et dictées par la demande. Il prend en compte les besoins des utilisateurs, afin de créer un environnement permettant aux utilisateurs d'accéder à des services abordables dans le monde entier, quelle que soit la technologie sous-jacente, en particulier dans les pays en développement.

1.2.1.2 ISO

L'ISO est l'organisation internationale de standardisation regroupant plusieurs organismes nationaux de normalisation issus de toutes les régions du monde. Plus de 18600 normes de l'ISO sont offertes aux entreprises et aux gouvernements pour les trois dimensions du développement durable : économique, environnementale et sociale [6]. Les normes ISO facilitent les échanges et diffusent les connaissances, et les avancées technologiques innovantes en mettant en commun, de bonnes pratiques de gestion et d'évaluation de la conformité. Elles concernent presque tous les secteurs d'activité, allant des technologies de l'information et de la communication à l'évaluation de la conformité et des services.

L'ISO élabore uniquement des normes pour lesquelles il existe une exigence claire du marché. Les travaux sont effectués par des experts en la matière, directement issus des secteurs industriel, technique et commercial qui ont identifié le besoin de la norme et l'ont utilisé par la suite. Ces experts peuvent être rejoints par d'autres personnes possédant les connaissances requises, telles que des représentants d'agences gouvernementales, de laboratoires d'essais, d'associations de consommateurs et d'universités.

1.2.1.3 IEC (International Electrotechnical Commission)

La commission électrotechnique internationale (IEC) a été officiellement créée en 1906 [7] dans le but de garantir la coopération internationale en matière de normalisation et de certification des technologies électriques et électroniques. Cette association est formée par les comités nationaux de plus de 40 pays du monde entier. Sa mission englobe aussi l'élaboration des normes internationales relatives à toutes les technologies électriques, électroniques et connexes. Cette collection de normes traite du développement des produits, des performances, de la compatibilité et des sujets connexes afin de garantir la compatibilité des produits et la sécurité environnementale.

L'ISO et l'IEC forment le système spécialisé de la normalisation mondiale. Les organismes nationaux membres de l'ISO ou de l'IEC participent au développement de normes internationales, par l'intermédiaire des comités techniques créés par l'organisation concernée, afin de s'occuper des domaines particuliers de l'activité technique. Les comités techniques de l'ISO et de l'IEC collaborent dans des domaines d'intérêt commun. D'autres organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'ISO et l'IEC participent également aux travaux [8].

1.2.2 Groupes de travail

1.2.2.1 ITU-T/VCEG

ITU-T VCEG est un groupe de travail sur les normes de codage vidéo, en particulier pour les communications image et vidéo, relevant de l'ITU-T [3]. VCEG (Video Communication Expert Group) est le nom informel de la question 6 (Codage visuel)

du Groupe de travail 3 (Codage des Médias) de la Commission d'études 16 (Codage, systèmes et applications multimédias). Son titre abrégé est ITU-T Q.6/SG 16.

La première norme de visioconférence établie par l'UIT-T était la norme H.120. Les codecs UIT-T H.120 utilisant la transmission par groupe numérique primaire ont été la première norme internationale pour la compression vidéo numérique. Outre la H.120, il existe deux autres normes de codage vidéo développées par VCEG, H.261 (1990) et H.263 (1995) [3]. Le Codec vidéo H.261 pour les services de téléconférence, publié en 1990 et révisé en 1993, a été la première norme de codage vidéo numérique à succès commercial, et a introduit l'architecture moderne de la technologie de codage vidéo par blocs hybrides. Le codage vidéo H.263 pour les communications à faible débit, publié en 1996 et révisé en 1998 (sous le nom de H.263 +) et amélioré en 2000 (sous le nom de H.263 ++), a permis des améliorations substantielles pour le codage vidéo en temps réel et a été déployé dans des centaines de millions d'appareils mobiles ainsi que dans des systèmes de vidéoconférence.

1.2.2.2 ISO/IEC MPEG

Le MPEG (Moving Pictures Expert Group) a lancé un effort parallèle en 1988. Ce groupe a exploité de nombreux concepts tirés de la norme H.261 et a publié la norme MPEG-1 (ISO / IEC 11172) en 1993 [9]. La norme visait notamment la distribution multimédia. Comme pour le H.261, MPEG-1 est un codage hybride. De nouvelles techniques ont été développées pour améliorer la qualité de la reconstruction des séquences vidéo. MPEG est le nom informel d'ISO/IEC. Il est chargé de l'élaboration de normes internationales pour la compression, la décompression, le traitement et la représentation codée d'images animées, de son et de leur combinaison. MPEG a créé la ligne standard MPEG-x, telle que MPEG-1, MPEG-4, MPEG-7, MPEG-21, etc.

1.2.2.3 Activités communes entre MPEG et VCEG

MPEG et VCEG s'associent de temps à autre, créant ainsi une norme utilisée par les deux organisations. Trois standards de codage vidéo ont été développés en collaboration, Figure 1.1.

1. H.262/MPEG-2 : Élaborée en 1994 [2]. La vidéo MPEG-2 est similaire à MPEG-1, mais fournit également une prise en charge de la vidéo entrelacée

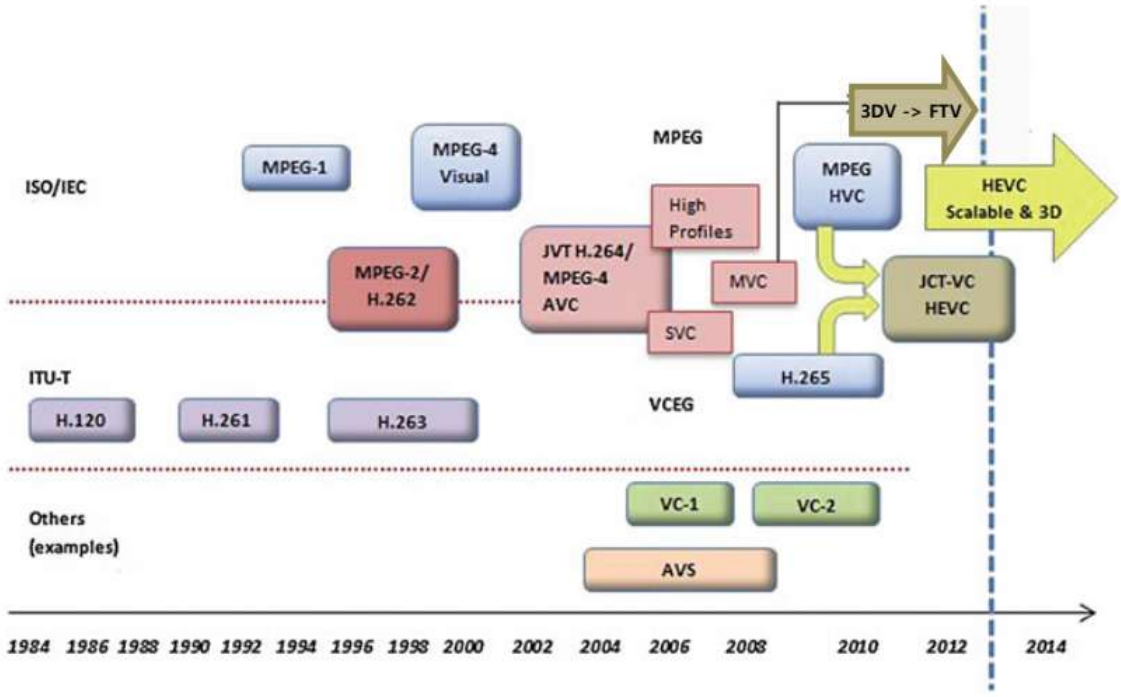


Figure 1.1. Évolution des standards de codage vidéo

(format utilisé par les systèmes de télévision par diffusion analogique). La vidéo MPEG-2 n'est pas optimisée pour les débits binaires faibles (moins de 1 Mbps), mais dépasse celle de MPEG-1 à 3 Mbps et plus. Pour la cohérence des normes, MPEG-2 est également compatible avec MPEG-1, ce qui signifie qu'un lecteur MPEG-2 peut lire des vidéos MPEG-1 sans aucune modification.

2. ISO/IEC 14496-10—ITU-T H.264 (en 2003) : Au début de l'an 2000, l'ITU et le MPEG ont commencé, à nouveau, à travailler ensemble sur une nouvelle norme de compression vidéo. Ils ont formé un groupe appelé Joint Video Team (JVT) chargé d'examiner les nouveaux problèmes de compression vidéo [10]. Les désignations officielles de l'UIT et MPEG sont respectivement H.264 et MPEG-4, Partie 10. Dans cette thèse, la désignation ITU H.264 sera utilisée pour faire référence à la norme. Cette norme a été conçue pour cibler une grande variété d'applications, y compris les réseaux sans fil, IP et le cinéma numérique. La norme a défini deux couches de codage principales distinctes : la couche de codage vidéo (VCL : Video Coding Layer) et la couche d'abstraction de réseau (NAL : Network Abstraction Layer). Cette norme se démarque des normes précédentes en introduisant de nombreuses nouvelles fonctionnalités

techniques pour augmenter encore l'efficacité de la compression, telles que le dimensionnement flexible des macroblocs, l'interpolation 1/4 pixel, la possibilité d'utiliser des images de référence multiples et le filtre en boucle (In-loop Filter).

3. ISO/IEC 23008-2 — UIT-T H.265 (2013) : Sur la base de l'expérience acquise lors de la précédente opération de normalisation, les deux organismes de normalisation, l'ITU-T et l'ISO/IEC, ont décidé de constituer une nouvelle équipe collaborative chargée de la mise au point de la nouvelle norme, dénommée Équipe collaborative sur le codage vidéo (JCT-VC : Joint Collaborative Team on Video Coding) [11]. Lors de la première réunion du JCT-VC à Dresde en Avril 2010 [12], les réponses à l'appel à propositions ont été évaluées et le nom du projet *codage vidéo à haute efficacité* (HEVC : High Efficiency Video Coding) a été établi. Les performances de compression ont été évaluées dans le cadre d'une évaluation visuelle, subjective et formelle, au cours de laquelle la qualité de la reconstruction a été comparée à celle du H.264/AVC. Sur la base des résultats, le premier projet de travail WD1 (working draft) et le modèle de test HM1 ont été publiés en octobre 2010 [13]. Le projet a ensuite été développé et affiné à chaque réunion du JCT-VC, environ tous les trois mois. La version finale de la version 1 de HEVC (WD10 et HM10) a été approuvée en janvier 2013 [14].

1.3 Vue d'ensemble de la norme H.265/HEVC

La norme HEVC a été conçue pour répondre à la demande croissante et la popularité de la vidéo haute définition (HD) , ainsi que l'émergence de formats autres que HD, avec la possibilité d'encoder plusieurs vues vidéo [15]. Actuellement, la norme de codage vidéo à haute efficacité (HEVC) est progressivement introduite sur le marché et devrait remplacer H.264 / AVC au cours des prochaines années. Cette nouvelle norme permet de réduire les débits binaires d'environ 40 à 50% par rapport à H.264/AVC grâce à l'utilisation de plusieurs nouveaux outils et modes de fonctionnement. De toute évidence, comme plusieurs modes de fonctionnement sont disponibles, la complexité des calculs impliqués dans le processus de codage HEVC devient supérieure à celle des normes antérieures.

1.3.1 Représentation numérique de la vidéo

La vidéo est utilisée pour enregistrer et/ou montrer des objets en mouvement, elle est composée d'une séquence d'images prises à des intervalles de temps réguliers. Le nombre d'images (frames) par seconde est appelé débit d'images. Des fréquences d'images inférieures à 10 images par seconde (fps) sont parfois utilisées pour les communications vidéo à très faible débit binaire (inférieur à 64 kbps). Entre 10 et 20 images par seconde est plus typique pour les communications vidéo à faible débit. L'échantillonnage à 25 ou 30 images par seconde est standard pour les images de télévision. Alors que 50 ou 60 images par seconde produisent un mouvement apparent lisse.

Chaque image d'une séquence vidéo comprend M par N éléments d'image (pixels), où M est le nombre de lignes et N le nombre de colonnes. Pour les images en couleur, chaque pixel est généralement composé de trois composantes de couleur, à savoir le rouge (R : red), le vert (G : green) et le bleu (B : blue) (abrégé en R, G, B). Toute couleur peut être créée en combinant R, G et B avec des proportions variables. Avec toutes les valeurs possibles de R, G et B, les vecteurs de (R, G, B) forment un espace appelé espace colorimétrique RGB. Chaque composante de couleur est représentée par un entier de K bits. Pour une utilisation ordinaire, 8 est une valeur suffisante de K. Une plus grande valeur de K, est nécessaire pour des utilisations plus précises telles que l'édition médicale, la diffusion, la surveillance, l'édition en studio, etc.

Bien que l'espace colorimétrique RGB soit bien adapté à la capture et à l'affichage des images couleur, l'espace YC_bC_r, formé de vecteurs de (Y, C_b, C_r), est plus efficace pour la compression, où Y représente la luminance (luminosité) d'un pixel, C_b et C_r sont les composantes de chrominance du pixel proportionnelles aux différences de couleur de B–Y et R–Y respectivement. Selon [16], les transformations de l'espace RGB à l'espace YC_bC_r et inversement peuvent être réalisés comme suit :

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ C_b = -0.169R - 0.331G + 0.500B \\ C_r = 0.500R - 0.419G - 0.081B \end{cases} \tag{1.1}$$

et

$$\begin{cases} R = Y + 1.42C_r \\ G = Y - 0.344C_b - 0.714C_r \\ B = Y + 1.771C_b \end{cases} \tag{1.2}$$

Il existe un autre modèle similaire au modèle YC_bC_r, qui est le modèle de couleur YUV. Il est généralement utilisé à la place du modèle RGB pour la réduction de bande passante, où Y est la composante luminance et UV sont les composantes de chrominance, qui peuvent être sous-échantillonnées pour la réduction des données. La technologie YUV est historiquement proposée pour une transmission compatible noir et blanc/couleur [3]. Y est le signal de luminance existant pour la radiodiffusion télévisuelle en noir et blanc, et un signal UV est ajouté pour les informations de couleur. Les transformations de l'espace RGB à l'espace YUV et inversement peuvent être réalisés comme suit :

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ U = -0.147R - 0.289G + 0.436B \\ V = 0.615R - 0.515G - 0.100B \end{cases} \tag{1.3}$$

et

$$\begin{cases} R = Y + 1.140V \\ G = Y - 0.395U - 0.581V \\ B = Y + 2.032U \end{cases} \tag{1.4}$$

1.3.2 Mécanisme de partitionnement d'une séquence vidéo

En HEVC, chaque séquence vidéo est divisée en groupes d'images (GOP), qui sont limités par deux images qui constituent des points d'accès aléatoires (RAP) à partir desquels le décodeur peut commencer à décoder sans avoir besoin d'images précédentes. Un exemple de division d'une séquence vidéo en plusieurs GOPs est illustré sur la Figure 1.2. Un GOP est composé d'un ensemble d'images pouvant être divisés en un ensemble de Slices. Une Slice est composée d'une séquence d'unités (CTU) (Figure 1.3), qui sont ensuite divisées en unités de codage (CU). Ces structures sont expliquées en détail dans la suite. Toutes les CUs de chaque Slice sont codées en fonction du type, qui peut être I, P ou B. Dans les Slices I, toutes les CU sont codées à l'aide de la

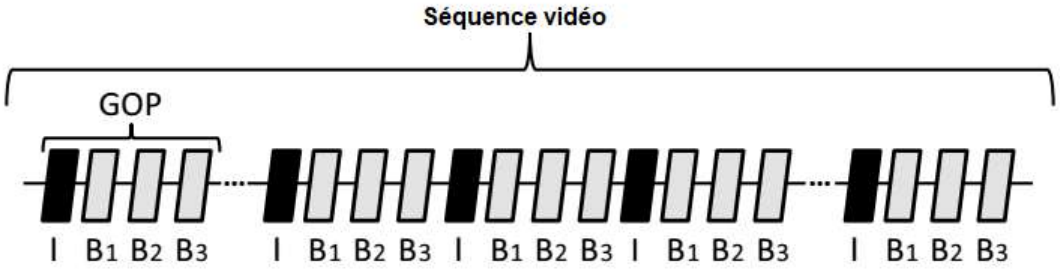


Figure 1.2. Exemple de division d'une séquence vidéo en plusieurs GOPs

prédiction intra uniquement. Dans une Slice P, outre la prédiction intra utilisée dans les Slices I, les CUs peuvent également être codées à l'aide de la prédiction inter avec une référence pour la compensation du mouvement (c'est la prédiction unidirectionnelle). Dans les Slices B, outre la prédiction intra et la prédiction inter unidirectionnelle, les CUs peuvent également être codées à l'aide de la prédiction inter avec deux références pour la compensation du mouvement (c'est la prédiction bidirectionnelle).

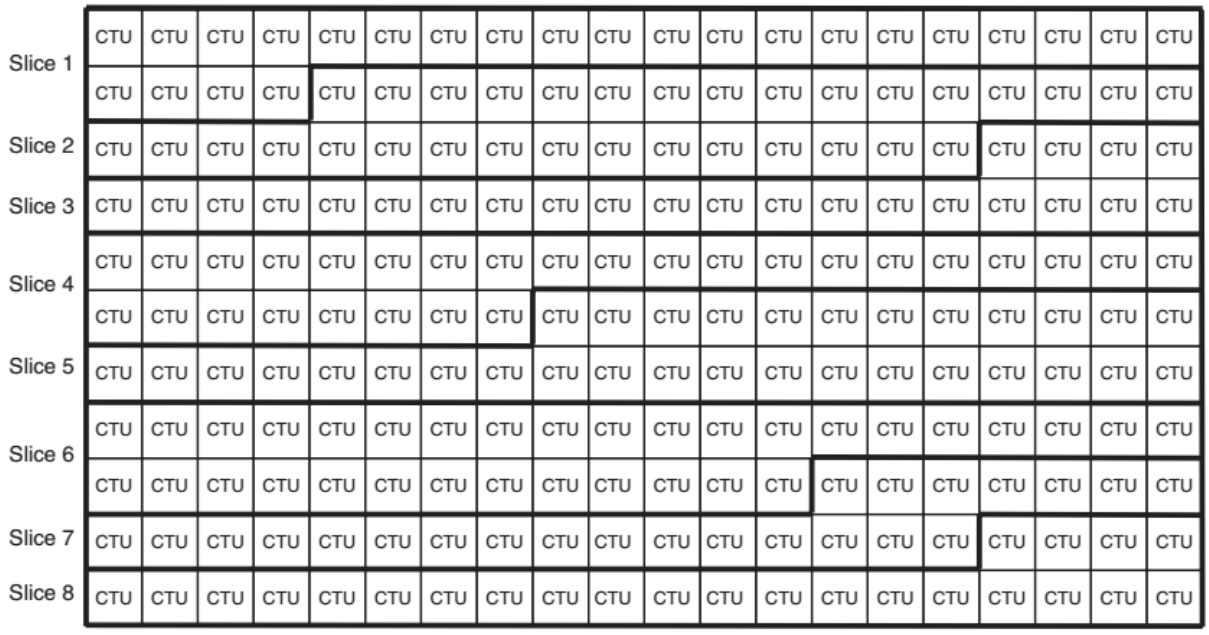


Figure 1.3. Exemple de partitionnement d'une image en 8 Slices

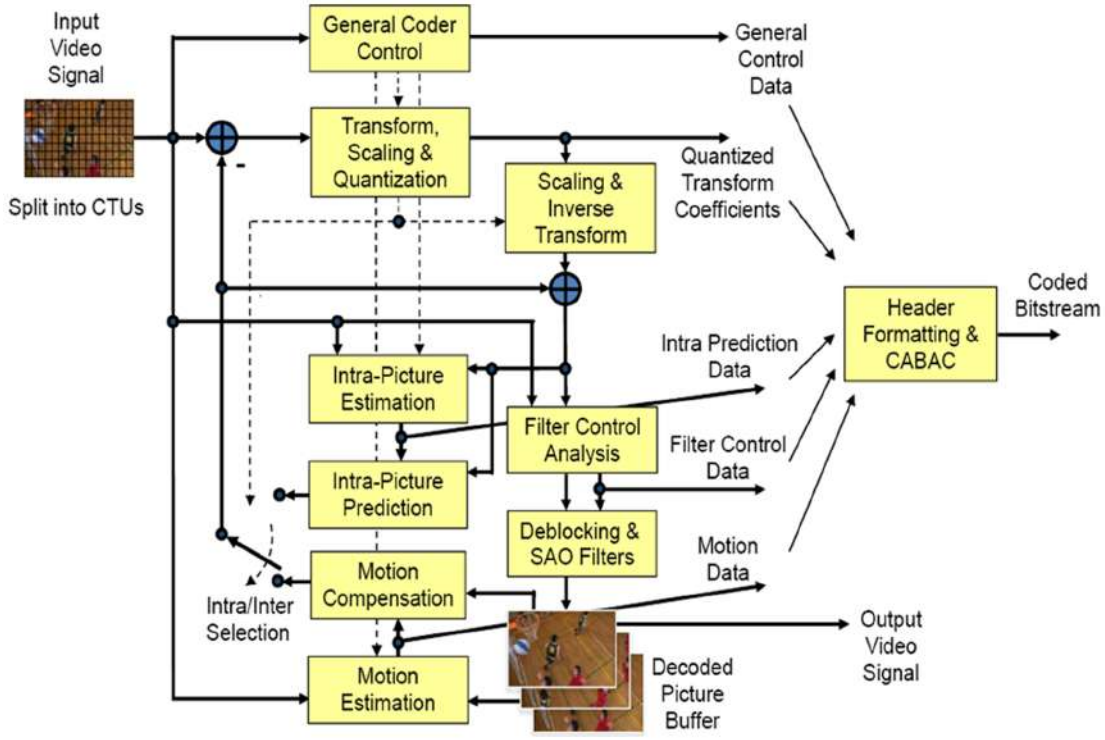


Figure 1.4. Schéma de principe de l'encodeur HEVC [15]

1.3.3 Partitionnement de l'image en unités de traitement

Depuis le H.261, le schéma de codage vidéo hybride, Figure 1.4, est la structure de base de toutes les normes de codage vidéo et des recommandations de l'UIT-T et de l'ISO/IEC MPEG. Le schéma de codage est appelé hybride, car il combine une prédiction temporelle entre les images de la séquence vidéo et des techniques de codage par transformation de l'erreur de prédiction. Dans toutes les normes de codage vidéo antérieures de l'ITU-T et de l'ISO/IEC, chaque image d'une séquence vidéo est partitionnée en ce que l'on appelle les macroblocs. Un macrobloc consiste en un bloc d'échantillons de luminance de 16×16 et de deux blocs de chroma associés sous-échantillonnés. Dans le format d'échantillonnage de chrominance 4 :2 :0, Figure 1.5, les deux blocs d'échantillons de chroma sont de taille de 8×8.

1.3.3.1 Arbre de codage : CTU (*Coding Tree Unit*) et CTB (*Coding Tree Block*)

Les macroblocs peuvent être considérés comme les unités de traitement de base de ces normes. En HEVC, chaque image est partitionnée en arbre de blocs de

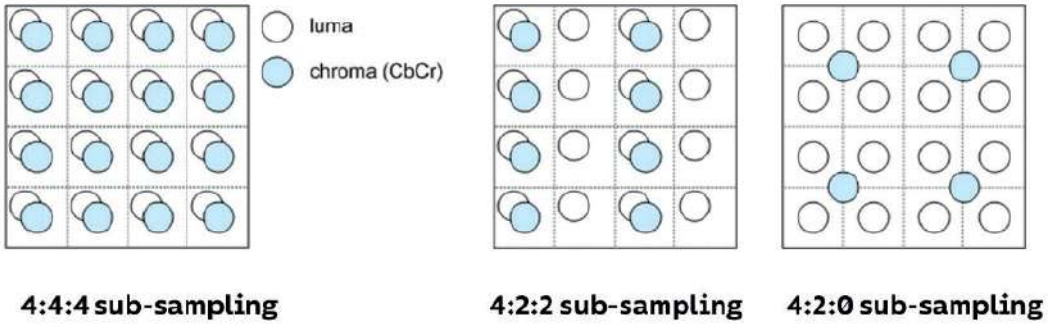


Figure 1.5. Positions des échantillons chroma en codage vidéo

codage (CTB), de forme carrée, tels que le nombre de CTB résultant soit identique pour les composantes d'image luma et chroma (en supposant un format vidéo non monochrome). En conséquence, chaque CTB d'échantillons de luminance et ses deux CTB de chrominance correspondants, ainsi que la syntaxe associée à ces blocs sont résumés dans une unité dite "arbre de codage" (CTU). Une CTU représente l'unité de traitement de base en HEVC et, à cet égard, est similaire au concept de macrobloc dans les normes de codage vidéo antérieures.

Un CTB luma couvre une zone d'image carrée d'échantillons de $2^N \times 2^N$ luma. Dans le format d'échantillonnage de chrominance 4 :2 :0, chacun des deux CTB de chrominance couvre la zone correspondante d'échantillons de chrominance $2^{N-1} \times 2^{N-1}$ de l'un des deux composants de chrominance. Le paramètre N est transmis dans l'ensemble de paramètres de séquence et peut être choisi par le codeur parmi les valeurs $N = 4$, 5 et 6, correspondant aux tailles de CTU de 16×16, 32×32 et 64×64 échantillons de luma. Les tailles plus grandes de CTU offrent généralement une meilleure efficacité de codage, mais peuvent également augmenter la complexité de calcul dans l'encodeur. Le codeur a la liberté de choisir la taille de la CTU qui offre le meilleur compromis pour l'application ciblée.

1.3.3.2 Unité de codage (CU) et bloc de codage (CB)

La CTU est partitionnée en plusieurs unités de codage CU pour s'adapter à diverses caractéristiques locales de l'image. Un processus de partitionnement (quadtree) appelé arbre de codage est utilisé pour partitionner la CTU en plusieurs CUs. Par conséquent, un bloc de codage CB correspond à une partition luma ou chroma.

La syntaxe de l'arbre CTU spécifie la taille et la position de ses blocs CB de luma

et de chroma [17]. La racine du quadtree est associée à la CTU, Figure 1.6. Par conséquent, la taille de la luma CTB est la plus grande taille prise en charge pour un luma CB. Le partitionnement d'une CTU en partitions luma et chroma est signalée conjointement. L'unité de codage CU est formée par l'association d'un bloc CB luma, deux CBs chroma et la syntaxe associée. Un CTB peut contenir une seule CU ou peut être scindée pour former plusieurs CUs. Chaque CU a un partitionnement associé en unités de prédiction (PU) et une arborescence d'unités de transformation (TU).

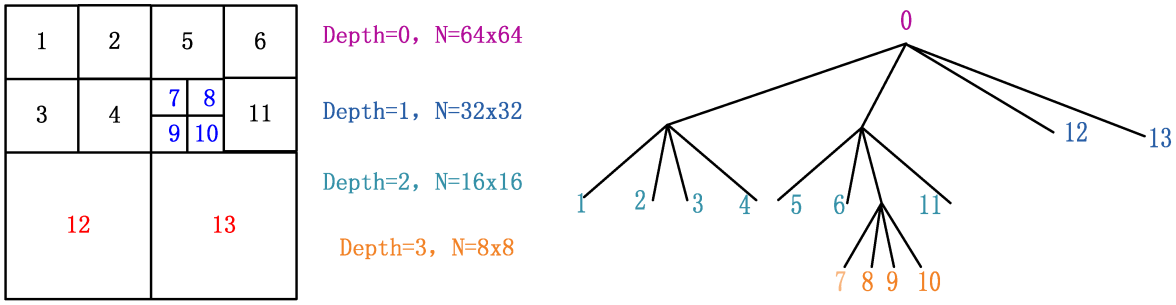


Figure 1.6. Exemple de partitionnement d'une CTU en CUs

La numérotation indiquée sur la Figure 1.6 représente l'ordre de traitement des CUs au sein de la CTU considérée le long du z-scan (scan en zig-zag).

Au niveau de la CTU, un drapeau nommé *split_cu_flag* est inclus dans le flux binaire (bitstream), indiquant si la CTU complète forme une CU ou si elle est subdivisée en quatre blocs de taille égale correspondant à des blocs d'échantillons de luma carrés. Si la CTU est divisée, pour chacun des blocs résultants, un autre *split_cu_flag* est transmis, spécifiant si le bloc représente une unité CU ou s'il est encore divisé en quatre blocs de taille égale. Cette subdivision hiérarchique est poursuivie jusqu'à ce qu'aucun des blocs résultants ne soit davantage subdivisé[18].

La taille potentiellement importante des CUs de base peut entraîner le problème suivant : la taille de l'image à coder ne peut pas être partitionnée en un nombre entier de CTBs. Le partitionnement peut rendre les CTBs de la dernière colonne ou de la dernière ligne de CTBs incomplets. Pour résoudre ce problème, ces CTBs sont implicitement partitionnés de manière à ce que la zone d'image soit couverte par des CBs complètes[11].

1.3.3.3 Unité de prédiction (PU) et bloc de prédiction (PB)

Pour la prédiction inter et la prédiction intra (voir plus loin), le bloc CB est partitionné en un ou plusieurs blocs de prédiction (PB). La décision sur le type de prédiction à choisir pour coder une zone d'image à l'aide d'une prédiction inter ou intra est prise au niveau de la CU [17].

Combinée avec la CU, la PU fonctionne comme l'unité représentative de base pour le partage des informations de la prédiction. À l'intérieur d'une CU, le même processus de prédiction est appliqué et les informations pertinentes sont transmises au décodeur sur la base de la CU.

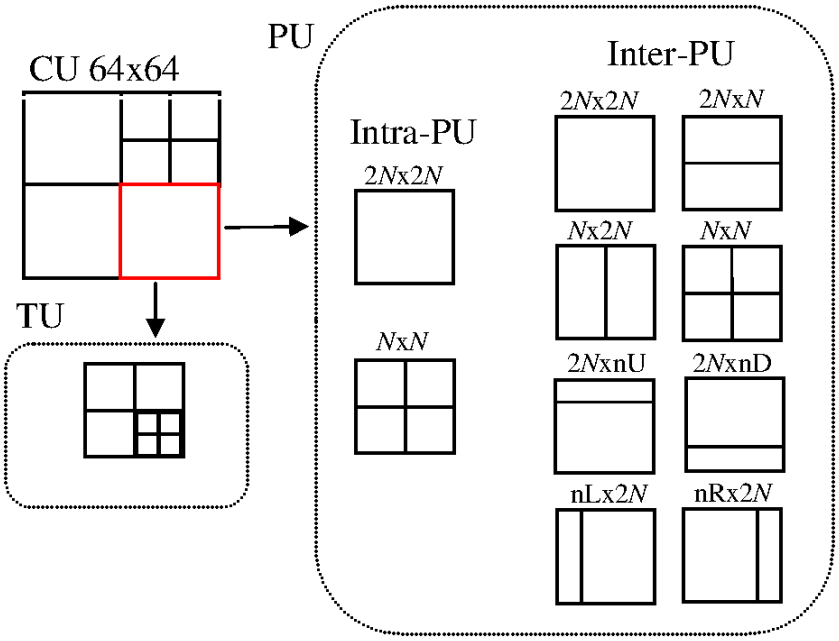


Figure 1.7. Les tailles possibles pour les blocs de prédiction PBs en HEVC

Pour la prédiction intra, la taille du bloc PB est généralement identique à la taille du bloc CB. Seulement pour la taille de CB minimale possible de 8×8, une étape de partitionnement en quatre blocs de prédiction PB 4×4 peut être requise. Pour la prédiction inter d'un PB, huit schémas de partitionnement différents en blocs de prédiction sont disponibles. Les blocs PBs inter et intra sont montrées à la Figure 1.7.

1.3.3.4 Unité de transformation (TU) et bloc de transformation (TB)

En HEVC, un bloc CB peut être partitionné en plusieurs blocs de transformation (TB). Un bloc de transformation représente un bloc carré d'échantillons

d'une composante de couleur ($Y, C_b ou C_r$) sur laquelle la même transformation bidimensionnelle est appliquée pour coder le signal résiduel. Le partitionnement d'un CB luma en TB luma est réalisé de manière récursive selon l'approche quadtree.

La structure correspondante, appelée quadtree résiduel (RQT), détermine pour chaque CB de luma une collection de TBs de luma au niveau des nœuds d'extrémité de l'arbre RQT de manière à ce que l'union des TBs luma disjointes, correspondantes, couvre toute le CB luma associé. La Figure 1.8 montre un exemple d'un CTB luma 64×64 divisé récursivement en CBs luma et TBs luma le long des structures à quadtree imbriquées correspondantes de CTB et RQT. En général, le partitionnement des CBs chroma en TBs chroma est décrit par le même quadtree résiduel [18].

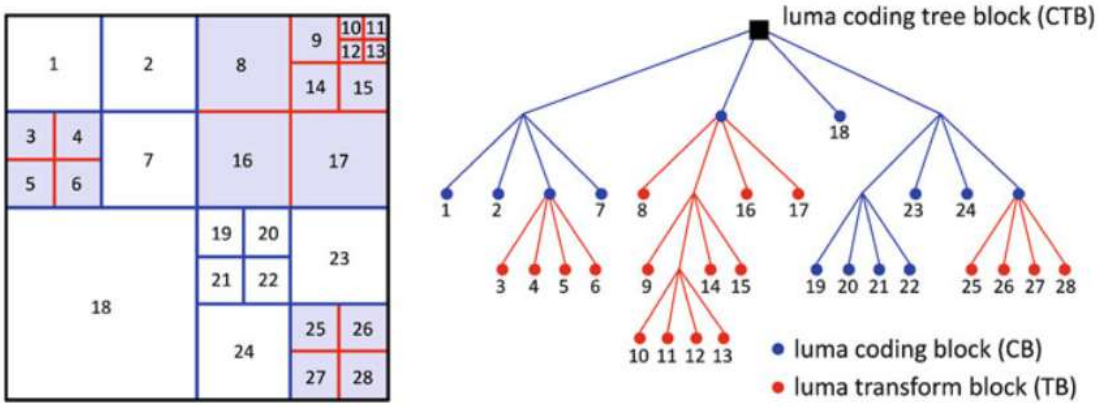


Figure 1.8. Exemple de partitionnement d'un CTB de 64×64 [18]

Dans l'exemple de la Figure 1.8, le bloc CTB luma (noir),de taille 64×64, est divisé en blocs CBs (bleu) et en blocs TBs (rouge). Le schéma en arbre est donné à droite, où le carré noire correspond à la racine qui est le CTB. Les branches bleues et rouges de l'arbre représentent les processus de partitionnement du CTB en CBs (cercles bleus) et des CBs en TBs (cercles rouges), respectivement. Pour les blocs TBs choisis de même taille que les blocs CBs correspondants, ils ne sont pas explicitement indiqués sur cette figure. L'ordre de codage des blocs TBs suit toujours l'ordre en z-scan, et il est indiqué par les nombres mentionnés sur la Figure 1.8.

1.3.4 La prédiction spatiale : *Intra-prédiction*

La prédiction intra est basée sur la redondance (la corrélation) dans les régions locales d'une image. L'hypothèse de base pour la prédiction intra est que la texture

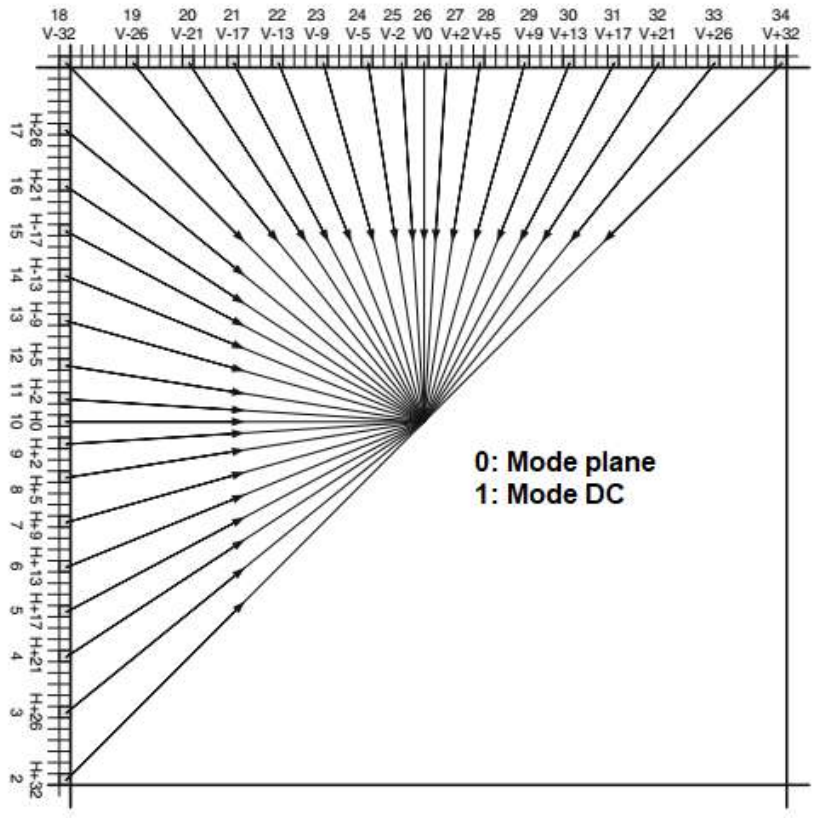


Figure 1.9. Directions et modes de prédiction intra adopté par la norme HEVC

d'une région d'image est similaire à la texture dans le voisinage local et peut donc être prédite à partir de là [11].

Pour prédire les valeurs de luma dans une unité PU, le standard HEVC adopte 35 modes de prédiction intra, alors que H.264/AVC utilise seulement 9 modes, et par conséquent, HEVC peut prédire avec plus de précision des structures complexes. Les 35 modes de prédiction intra dans HEVC sont étiquetés comme modes 0, 1,...,34 sur la Figure 1.9, et sont définis pour cinq tailles différentes des unités de prédiction PU [19]. La taille d'une PU est identique à celle de la CU si cette dernière possède l'une des tailles suivantes : 64×64, 32×32, 16×16. Pour une CU de taille 8×8, l'unité de prédiction peut être 8×8 ou 4×4, par conséquent, quatre PUs 4×4 sont nécessaires pour une CU 8×8.

Sur la Figure 1.9, les directions sont définies par un déplacement de fractions de 1/32 sur une grille de pixels. Les déplacements sont calculés par rapport aux directions principales horizontale (H0) et verticale (V0). La correspondance entre les modes de prédiction et les directions est donnée sur le Tableau 1.1. Pour les déplacements négatifs

Tableau 1.1. Correspondances entre modes de prédictions et paramètres d'angle

Les modes horizontaux	Le mode	2	3	4	5	6	7	8	9
	A	32	26	21	17	13	9	5	2
	Le mode	10	11	12	13	14	15	16	17
	A	0	-2	-5	-9	-13	-17	-21	-26
Les modes verticaux	Le mode	18	19	20	21	22	23	24	25
	A	-32	-26	-21	-17	-13	-9	-5	-2
	Le mode	26	27	28	29	30	31	32	33
	A	0	2	5	9	13	17	21	26

(modes 11,12,...,25), les angles inverses sont définis, Tableau 1.2, pour une utilisation ultérieure dans le calcul des pixels de référence.

Pour les modes DC, horizontal et vertical, un post-traitement supplémentaire est introduit dans l'HEVC, dans lequel une ligne et/ou une colonne est filtrée de manière à maintenir la continuité à travers les limites des blocs. Cet ajout ne devrait pas avoir d'incidence sur la complexité du pire des cas, car ces trois modes sont les plus simples et les plus utilisés [20, 21].

Tous les modes de prédiction intra HEVC utilisent des échantillons de référence provenant des blocs reconstruits adjacents, comme illustré à la Figure 1.10, où $p_{-1,-1}$, $p_{-1,y}$ et $p_{x,-1}$ pour $x, y = 0, 1, ..., 2N-1$ représentent les pixels de référence, $p_{x,y}$ pour $x, y = 0, 1, ..., N-1$, représentent les pixels du bloc à prédire. Pour un bloc N×N, la prédiction intra totalise donc $4N+1$ échantillons de référence. Par rapport au H.264/AVC, HEVC introduit un processus de substitution d'échantillons de référence qui permet à HEVC d'utiliser l'ensemble entier des modes de prédiction intra, quelle que soit la disponibilité des échantillons de référence voisins. De plus, il existe un processus de filtrage adaptatif qui peut pré-filtrer les échantillons de référence en fonction du mode de prédiction intra et de la taille du bloc pour augmenter la diversité des prédicteurs disponibles [18].

Dans le cas extrême où aucun des pixels de référence n'est disponible, tous les pixels de référence sont remplacés par une valeur moyenne nominale pour un nombre de bits donné par pixel (par exemple 128 pour des pixels à 8 bits). S'il existe au moins

Tableau 1.2. L'angle inverse en fonction du paramètre d'angle (B=(256 * 32)/A)

A	-32	-26	-21	-17	-13	-9	-5	-2
B	-256	-315	-390	-482	-630	-910	-1638	-4096

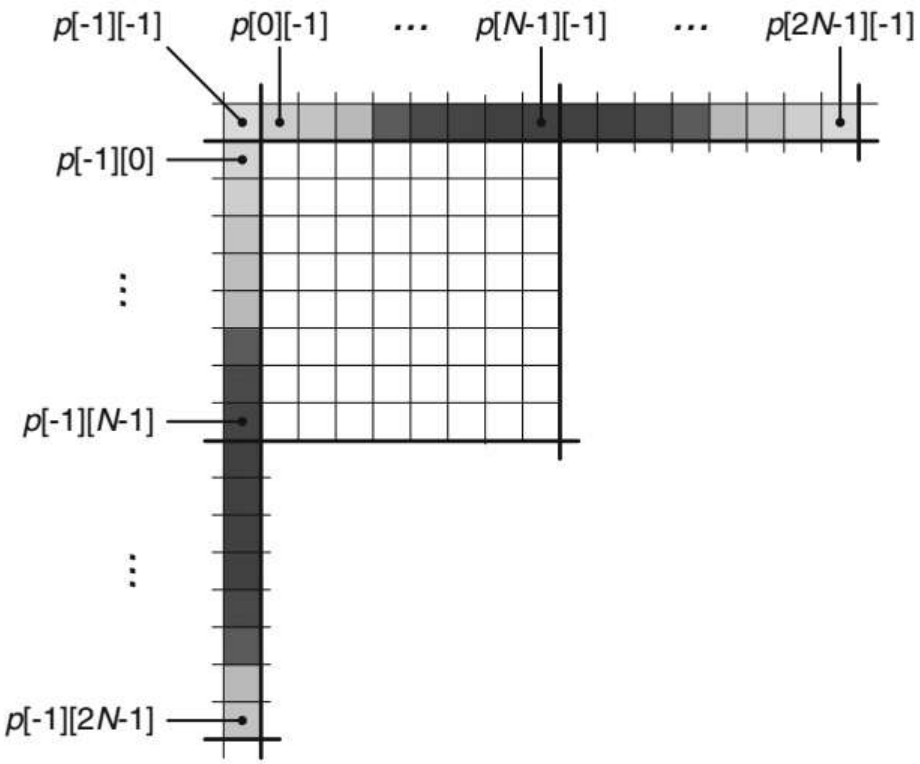


Figure 1.10. Pixels de référence utilisés par la norme HEVC en intra-prédiction

un pixel de référence marqué comme disponible pour la prédiction intra, les pixels de référence non disponibles sont remplacés par ce pixel de référence disponible. Les échantillons de référence non disponibles sont remplacés en balayant les échantillons de référence dans le sens des aiguilles d'une montre et en utilisant la dernière valeur d'échantillon disponible pour les échantillons non disponibles, Figure 1.11.

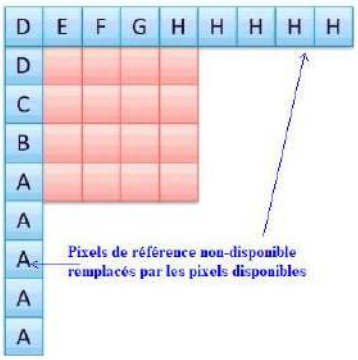


Figure 1.11. Pixels de référence utilisés par la norme HEVC en intra-prédiction

1.3.4.1 Le mode de prédiction DC

Le mode DC est souvent sélectionné lorsqu'il existe une faible variation des valeurs de luma dans l'unité de prédiction. Les valeurs d'échantillons prédites sont remplies avec une valeur constante représentant la moyenne des échantillons de référence immédiatement à gauche et au-dessus du bloc à prédire, selon l'équation. (1.5).

$$V_{DC} = \frac{1}{2N}\left(\sum_{x=0}^{N-1} p[x][-1] + \sum_{y=0}^{N-1} p[-1][y]\right) \tag{1.5}$$

Comme mentionné auparavant, Les blocs DC prédits de luminance de taille inférieure ou égale à 16×16 passent par un processus de filtrage de la luminance pour adoucir les bords gauche et supérieur du bloc.

1.3.4.2 Les modes de prédiction angulaire

L'intra-prédiction angulaire dans HEVC est conçue pour pouvoir modéliser efficacement différentes structures directionnelles, fréquemment présentes dans les contenus vidéo et image. Le nombre et l'angle des directions de prédiction sont choisis de manière à offrir un bon compromis entre la complexité et l'efficacité du codage. Le processus de prédiction d'échantillon lui-même est conçu pour avoir de faibles exigences en calcul, et pour être cohérent avec la taille des blocs et les directions de prédiction. Cela est particulièrement important, car le nombre de tailles de blocs et de directions de prédiction pris en charge par le codage intra HEVC dépasse de loin celui des codecs vidéo précédents, tels que H.264/AVC. En HEVC, il existe quatre tailles de blocs intra-prédiction allant de 4×4 à 32×32 échantillons, chacun prenant en charge 33 directions de prédiction distinctes, Figure 1.9. Un décodeur doit donc prendre en charge 132 combinaisons de taille de bloc et de direction de prédiction [18, 22].

1.3.4.3 Génération des pixels de référence et prédiction angulaire

Le processus de prédiction intra des modes angulaires peut impliquer l'extrapolation des pixels à partir de l'emplacement du pixel de référence, projeté selon une direction donnée. Pour éviter la permutation pixel par pixel entre les vecteurs de ligne et de colonne de référence, pour intra-angulaire [k] avec k compris entre 2 et 17,

les pixels situés dans la ligne de-dessus sont projetés en tant que pixels supplémentaires situés dans la colonne gauche. Pour k compris entre 18 et 34, les pixels situés dans la colonne gauche sont projetés sous forme de pixels situés dans la ligne de-dessus. L'interpolation bilinéaire est utilisée pour obtenir la valeur du pixel de référence, projeté à l'aide des deux pixels de référence les plus proches [15].

Avant d'établir la prédiction, les pixels de référence doivent se mettre en un seul tableau (vecteur). Pour les modes avec paramètre angulaire positif A (selon le Tableau 1.1 : les modes verticaux 26 à 33 et les modes horizontaux 2 à 10), le tableau de référence, à une dimension, est simplement créé en copiant des pixels de référence à partir de la direction de prédiction :

$$ref[x] = p[x-1][-1], (x \geq 0) \text{ pour les modes verticaux} \tag{1.6}$$

$$ref[y] = p[-1][y-1], (x \geq 0) \text{ pour les modes horizontaux} \tag{1.7}$$

Si le paramètre angulaire A est négatif pour le mode sélectionné, les pixels provenant à la fois de la colonne gauche et de la ligne supérieure du bloc sont requis comme référence. Dans ces cas, les échantillons du tableau $ref[x]$ avec des index non négatifs sont remplis comme décrit par les expressions Eq. (1.6) et Eq. (1.7). Les valeurs des pixels pour les index négatifs sont obtenues en projetant la colonne de référence gauche pour étendre la ligne de référence supérieure vers la gauche, ou en projetant la ligne de référence supérieure pour étendre la colonne de référence gauche vers le haut dans le cas de prédictions verticales et horizontales, respectivement. Cette projection est définie comme suit [18, 22] :

$$ref[x] = p[-1][-1 + ((x * B + 128) \gg 8)], (x < 0) \text{ pour les modes verticaux} \tag{1.8}$$

$$ref[y] = p[-1 + ((y * B + 128) \gg 8)][-1], (y \geq 0) \text{ pour les modes horizontaux} \tag{1.9}$$

où B représente l'angle inverse du paramètre angulaire A, Tableau 1.2, et "$\gg n$" représente l'opération de décalage vers la droite de "n" bits (l'équivalent d'une division par 2^n).

La valeur d'un pixel prédit $p_{x,y}$ pour l'un des modes horizontaux, est donnée par :

$$p[x][y] = ((32 - f) * ref[y + i + 1] + f * ref[y + i + 2] + 16) \gg 5 \tag{1.10}$$

Ainsi, la valeur d'un pixel prédit $p_{x,y}$ pour l'un des modes verticaux, est donnée par :

$$p[x][y] = ((32 - f) * ref[x + i + 1] + f * ref[x + i + 2] + 16) \gg 5 \tag{1.11}$$

où i est le déplacement entier prévu sur la ligne y (pour les modes verticaux) ou la colonne x (pour les modes horizontaux), et calculé en fonction du paramètre angulaire A comme suit :

$$i = ((x + 1) * A) \gg 5 \text{ pour les modes horizontaux} \tag{1.12}$$

$$i = ((y + 1) * A) \gg 5 \text{ pour les modes verticaux} \tag{1.13}$$

f représente la partie fractionnaire du déplacement, projeté sur la même ligne ou la même colonne et est calculé comme suit :

$$f = ((x + 1) * A)\&31 \text{ pour les modes horizontaux} \tag{1.14}$$

$$f = ((y + 1) * A)\&31 \text{ pour les modes verticaux} \tag{1.15}$$

où & représente le **ET** logique.

1.3.4.4 Le mode de prédiction plane (ou planaire)

Bien que la prédiction angulaire offre de bonnes approximations pour les structures avec des contours significatifs, elle peut créer des contours visibles dans les zones d'images lisses. De la même manière, on peut généralement observer certains effets de bloc dans les zones d'image lisses lorsque la prédiction DC est appliquée à des débits binaires faibles ou moyens. La prédiction plane de l'HEVC est conçue pour surmonter ces problèmes en permettant de générer une surface de prédiction sans discontinuités sur les limites des blocs. Ceci est réalisé en faisant la moyenne d'une prédiction linéaire

horizontale et verticale, comme suit :

$$p[x][y] = (p_h[x][y] + p_v[x][y] + N) \gg (log_2(N) + 1) \tag{1.16}$$

où la prédiction horizontale $p_h[x][y]$ et la prédiction verticale $p_v[x][y]$ pour l'emplacement $[x][y]$, sont calculées par :

$$p_h[x][y] = (N - 1 - x) * p[-1][y] + (x + 1)\, p[N][-1] \tag{1.17}$$

$$p_v[x][y] = (N - 1 - y) * p[x][-1] + (y + 1)\, p[-1][N] \tag{1.18}$$

1.3.4.5 Filtres de lissage

HEVC applique le lissage des pixels dans deux situations distinctes pour améliorer les résultats de la prédiction intra, à savoir le filtrage des pixels de référence et le filtrage des pixels prédits [23]. Dans la norme H.264/AVC, seuls les pixels de référence sont filtrés. En HEVC, ce filtrage est effectué de manière adaptative, en fonction de la taille du bloc et du mode de prédiction intra. Le Tableau 1.3 présente les cas dans lesquels le lissage des pixels de référence est appliqué. L'HEVC utilise le même filtre à trois coefficients [1 2 1]/4, tel que défini dans la norme H.264/AVC.

L'autre filtrage proposé dans HEVC est utilisé pour lisser les discontinuités générées le long des limites de blocs dues à la prédiction intra. Ce filtre est appliqué sur les pixels prédits aux limites pour les modes DC, angulaire 10 et angulaire 26. En mode DC, lorsque le bloc est inférieur à 32×32, les pixels de la première ligne et de la première colonne du bloc intra prédit sont filtrés. Un filtre à deux coefficients [3 1]/4 dont l'entrée est la valeur d'origine et le pixel de référence adjacent est utilisé. Dans le cas du mode angulaire 10, seuls les échantillons de la première colonne sont modifiés en ajoutant la moitié de la différence entre le pixel de référence adjacent et le pixel de référence du haut à gauche. Un filtrage similaire est effectué pour le mode angulaire 26, mais uniquement pour la première rangée des pixels prédits[15, 18, 23].

1.3.5 La prédiction temporelle : *Inter-prédiction*

Bien que la prédiction intra exploite la corrélation entre les échantillons voisins dans l'espace, la prédiction inter de l'HEVC exploite la corrélation temporelle entre les

Tableau 1.3. Les tailles des blocs PU et modes de prédiction qui sont concernés par le filtre de lissage des pixels de référence en HEVC

Taille du bloc	Mode de prédiction
4×4	Aucun
8×8	Planaire, angulaire 2, 18 et 34
16×16	Planaire et tous les modes angulaires sauf :9, 10,11, 25, 26 et 27
32×32	Planaire et tous les modes angulaires sauf :10 et 26

images et est basée sur la compensation du mouvement (MC : *Motion Compensation*). Ce qui permet de prédire un bloc PB en utilisant des zones de taille égale à partir d'images codées précédemment, utilisées comme références, Figure 1.12. Le processus MC forme un bloc de pixels translatés à partir de l'image référencée à l'aide d'un vecteur de mouvement (MV :*Motion Vector*), qui décrit le déplacement du PB de l'image de référence à l'image courante. Le bloc prédit est ensuite soustrait du bloc courant pour calculer les résidus de prédiction entrés dans les modules de transformation (T) et de quantification (Q) [18, 24].

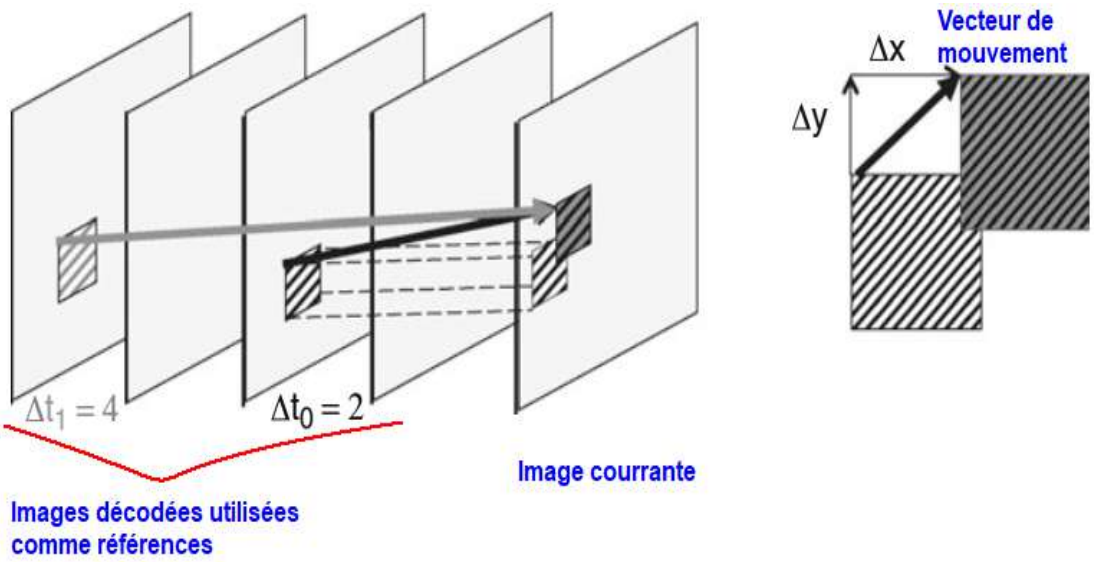


Figure 1.12. L'inter-prédiction d'un bloc à partir de plusieurs images de référence. Le vecteur de mouvement est défini par les déplacements Δx et Δy.

En HEVC, un prédicteur ou une combinaison de deux prédicteurs peut être utilisé pour la prédiction à compensation de mouvement, similaire aux normes précédentes [9, 11]. Dans la conception finale des spécifications, la bi-prédiction a été choisie comme le meilleur compromis entre performances de compression et complexité de mise en œuvre. La quantité d'accès mémoire requise pour extraire les blocs de prédiction à

partir de plusieurs images de référence, est considérée comme un problème majeur de complexité dans ce contexte, en particulier pour les implémentations matérielles [11, 25].

Le processus de détermination du MV pour un bloc PB est l'estimation de mouvement (ME : *Motion Estimation*), qui est l'opération la plus exigeante du codeur HEVC en termes de complexité de calcul. Le rôle du ME consiste à rechercher dans les images de référence, la région la plus proche du PB courant. Lorsque la meilleure région d'appariement est trouvée, l'algorithme ME calcule le vecteur MV correspondant avec une composante verticale et une composante horizontale, indiquant l'emplacement relatif de cette région par rapport à la position du PB courant [24].

L'estimation de mouvement basée sur la correspondance de blocs (BM : *Block Matching*) est l'une des procédures les plus complexes en calcul du codeur HEVC. La solution optimale pour l'estimation de mouvement est fournie par l'algorithme de recherche complète (Full Search), qui teste tous les blocs candidats possibles de la zone de recherche et sélectionne le meilleur. Malgré son optimalité, l'algorithme de recherche complète requiert la plus grande complexité de calcul. Cependant, l'estimation du mouvement étant une procédure non figée, les codeurs peuvent mettre en œuvre d'autres algorithmes de recherche fournissant des solutions sous-optimales avec une complexité de calcul réduite. Des algorithmes de recherche rapides, qui évitent la plupart des points de recherche, ont été proposés dans la littérature pour atténuer le problème de la complexité de calcul. Un exemple d'algorithme de recherche rapide, utilisé dans le software HM de référence, est *Test Zone Search* (TZS) [23].

1.3.5.1 Génération des pixels aux positions fractionnaires

Similaire à H.264/AVC, l'HEVC prend en charge l'estimation de mouvement avec une précision de 1/4 pixel pour les échantillons de luminance et une précision de 1/8 pixel pour les échantillons de chrominance, dans le cas du format 4 : 2 : 0. Pour obtenir les échantillons de luminance aux positions fractionnaires, deux filtres unidimensionnels séparables (un filtre HF de 8 coefficients et un filtre QF de 7 coefficients) sont appliqués horizontalement et verticalement pour générer les échantillons de demi-pixels et de quart de pixel, respectivement [11, 15, 18, 24]. Dans la Figure 1.13, les pixels de luminance disponibles dans des positions entières sont

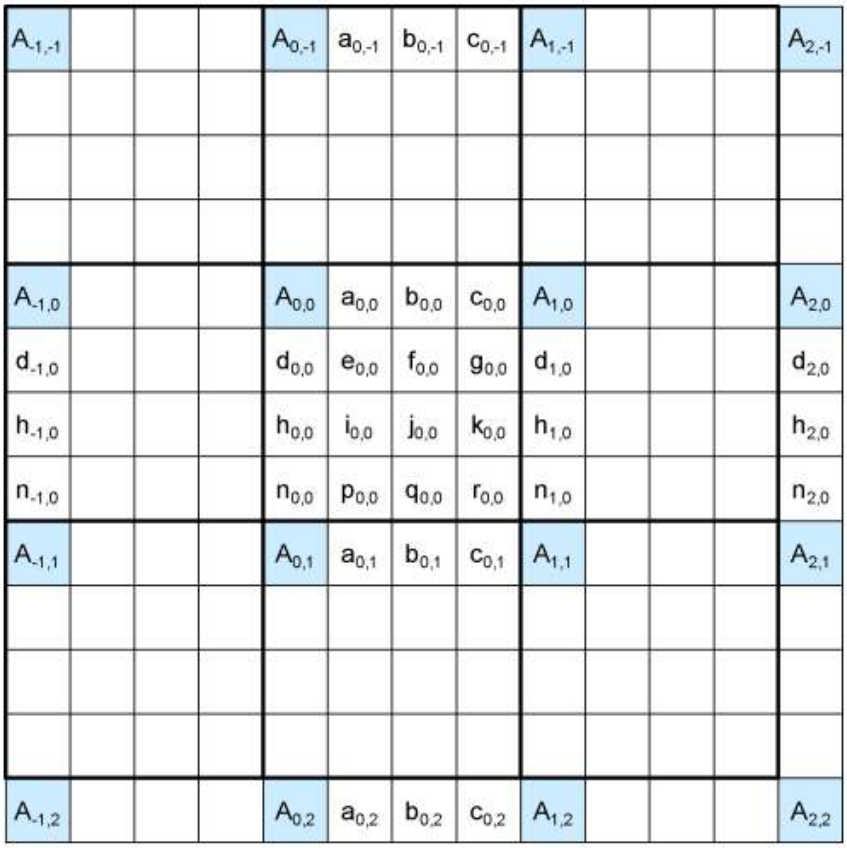


Figure 1.13. Positions entières et fractionnaires des pixels de Luminance

étiquetés avec des lettres majuscules, $A_{i,j}$, tandis que les autres positions étiquetées avec des lettres minuscules représentent des pixels avec des emplacements non entiers, et qui doivent être générés par interpolation.

Premièrement, les échantillons étiquetés $a_{0,j}$, $b_{0,j}$, $c_{0,j}$, $d_{0,0}$, $h_{0,0}$ et $n_{0,0}$ sont dérivés des échantillons $A_{i,j}$ en appliquant les filtres du Tableau 1.4, selon les expressions 1.19 :

Tableau 1.4. Les tailles des blocs PU et modes de prédiction concernés par le filtre de lissage des pixels de référence en HEVC

Indice i	-3	-2	-1	0	1	2	3	4
HF(i)	-1	4	-11	40	40	-11	4	1
QF(i)	-1	4	-10	58	17	-5	1	

$$
\begin{aligned}
a_{0,j} &= \left(\sum_{j=-3}^{3} A_{i,j}\ QF[i] \right) \gg (B-8) \\
b_{0,j} &= \left(\sum_{j=-3}^{4} A_{i,j}\ HF[i] \right) \gg (B-8) \\
c_{0,j} &= \left(\sum_{j=-2}^{4} A_{i,j}\ QF[1-i] \right) \gg (B-8) \\
d_{0,0} &= \left(\sum_{j=-3}^{3} A_{i,j}\ QF[j] \right) \gg (B-8) \\
h_{0,0} &= \left(\sum_{j=-3}^{4} A_{i,j}\ HF[j] \right) \gg (B-8) \\
n_{0,0} &= \left(\sum_{j=-2}^{4} A_{i,j}\ QF[1-j] \right) \gg (B-8)
\end{aligned}
\tag{1.19}
$$

où B représente le nombre de bits utilisés pour coder un pixel (typiquement $B = 8$).

Les pixels étiquetés $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$, et $r_{0,0}$, dans la Figure 1.13, peuvent être dérivés en appliquant les filtres correspondants aux échantillons

situés aux positions verticalement adjacentes à $a_{0,j}$, $b_{0,j}$ et $c_{0,j}$ comme suit :

$$\begin{aligned}
e_{0,0} &= \left(\sum_{v=-3}^{3} a_{0,v} QF[v] \right) \gg 6 \\
f_{0,0} &= \left(\sum_{v=-3}^{3} b_{0,v} QF[v] \right) \gg 6 \\
g_{0,0} &= \left(\sum_{v=-3}^{3} c_{0,v} QF[v] \right) \gg 6 \\
i_{0,0} &= \left(\sum_{v=-3}^{4} a_{0,v} HF[v] \right) \gg 6 \\
j_{0,0} &= \left(\sum_{v=-3}^{4} b_{0,v} HF[v] \right) \gg 6 \\
k_{0,0} &= \left(\sum_{v=-3}^{4} c_{0,v} HF[v] \right) \gg 6 \\
p_{0,0} &= \left(\sum_{v=-2}^{4} a_{0,v} QF[1-v] \right) \gg 6 \\
q_{0,0} &= \left(\sum_{v=-2}^{4} b_{0,v} QF[1-v] \right) \gg 6 \\
r_{0,0} &= \left(\sum_{v=-2}^{4} c_{0,v} QF[1-v] \right) \gg 6
\end{aligned} \tag{1.20}$$

1.3.5.2 Le mode de fusion en HEVC : *Merge Mode*

Les informations de mouvement comprennent généralement :

- les valeurs de déplacement de vecteur de mouvement horizontal et vertical,
- un ou deux indices d'image de référence,
- et, dans le cas de régions de prédiction dans des Slices B, une identification de la liste d'images de référence associée à chaque indice.

HEVC comprend un mode de fusion (*Merge Mode*), similaire aux modes direct et SKIP du H.264/AVC, permettant de déduire les informations de mouvement des blocs voisins dans l'espace ou dans le temps. Il est appelé mode de fusion, car il forme une région fusionnée partageant toutes les informations de mouvement.

En mode de fusion, les informations de mouvement applicables à la partition PU actuelle sont dérivées d'un ensemble de candidats de taille configurable. Les

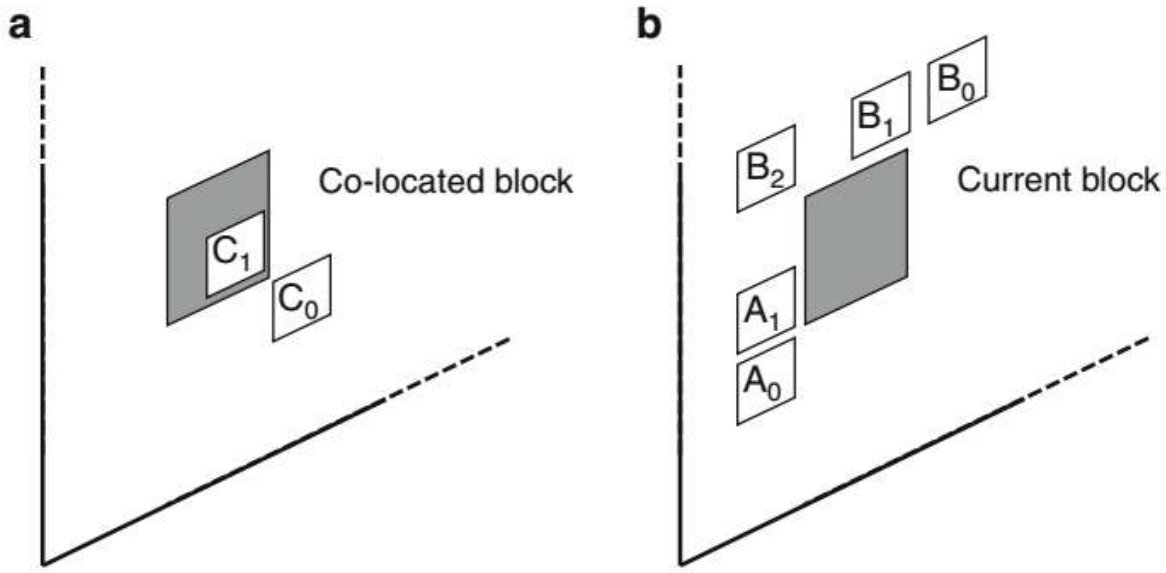


Figure 1.14. Positions des candidats pour le mode de fusion en HEVC : a) candidat temporel, b) candidats spatiaux

informations de mouvement du candidat à utiliser sont indiquées dans la syntaxe PU par un indice de fusion dans la liste des candidats. Le nombre maximal de candidats à la fusion à utiliser est spécifié dans l'en-tête de la Slice, il peut être compris entre 0 et 5. Sur la base de la disponibilité des voisins, une liste de candidats (à la fusion) est construite, y compris de multiples candidats spatiaux et potentiellement temporels [11]. Dans l'ensemble, la liste des candidats à la fusion est construite à partir des candidats suivants :

- jusqu'à quatre candidats spatiaux dérivés de cinq blocs voisins.
- un candidat temporel dérivé de deux blocs temporels co-localisés
- des candidats à la fusion supplémentaires, comprenant des candidats bi-prédictifs combinés et des candidats à vecteur de mouvement nul.

La Figure 1.14b montre les positions de cinq candidats spatiaux. Pour chaque position de candidat, la disponibilité est vérifiée selon l'ordre A_1, B_1, B_0, A_0, B_2. Le bloc est considéré comme indisponible s'il est situé à une position où la prédiction est intra, ou si la position est en dehors de la Slice ou de la Tile actuelle.

Pour le candidat temporel, la position en bas à droite juste en dehors du PU co-localisé de l'image de référence est utilisée si elle est disponible (le candidat C_0 de la Figure 1.14a). Sinon, la position centrale est utilisée. La manière de choisir le

PU co-localisé est similaire à celle des normes précédentes, mais HEVC offre plus de flexibilité en transmettant un indice pour spécifier la liste d'images de référence offrant l'image de référence [15].

Les candidats supplémentaires sont choisis selon le type de Slice. Jusqu'à deux types de candidats sont utilisés pour remplir complètement la liste :

– candidats bi-prédictifs combinés

– Candidats à vecteur de mouvement nul

Dans les Slices B (bi-prédictives), des candidats supplémentaires peuvent être générés en fonction des candidats existants, en combinant les données de mouvement d'un candidat de la liste d'images de référence *list0* avec les données de mouvement d'un autre de la liste *list1*.

Lorsque la liste n'est pas encore complète après l'ajout des candidats bi-prédictifs combinés, ou pour les Slices uni-prédictives, les candidats à vecteur de mouvement nul sont calculés pour compléter la liste. Les indices de référence sont mis à zéro et sont incrémentés de un pour chaque candidat supplémentaire jusqu'à ce que le nombre maximal d'indices de référence soit atteint. Si tel est le cas et qu'il manque encore des candidats, un indice de référence égal à zéro est utilisé pour les créer. Pour tous les candidats supplémentaires, aucune vérification de redondance n'est effectuée, car il s'est avéré qu'omettre ces vérifications n'introduirait pas de perte d'efficacité du codage [18].

1.3.5.3 Prédiction avancée de vecteur de mouvement (AMVP : *Advanced Motion Vector Prediction*)

Pour le codage de vecteur de mouvement prédictif, un processus de dérivation de candidat similaire à celui du mode de fusion est appliqué. Étant donné que le processus de dérivation est beaucoup plus élaboré que dans les spécifications précédentes, cette méthode de prédiction de vecteur de mouvement est désignée par "prédiction avancée de vecteur de mouvement", AMVP. Contrairement au mode de fusion dans lequel l'utilisation de cinq candidats au maximum peut être configurée, seuls deux candidats sont dérivés.

En fonction de leur disponibilité, deux candidats de vecteurs de mouvement du

voisinage spatial sont dérivés. Comme en mode fusion, un candidat temporel peut être inclus. Si la liste des candidats n'est pas remplie, des vecteurs à mouvement nul sont ajoutés, de sorte que la présence de deux candidats prédicteurs de vecteur de mouvement soit affirmée à des fins de résilience aux erreurs. Le prédicteur applicable est indiqué par un indice. Contrairement au mode de fusion dans lequel l'indice de référence applicable est déduit, l'indice de la liste d'images de référence est explicitement codé dans la syntaxe de l'unité PU dans le cas du AMVP [11].

1.3.6 La transformée et la quantification en HEVC

1.3.6.1 La transformée

Le module de transformation reçoit le résidu de prédiction, résultant de la prédiction inter ou intra, et le transforme dans le domaine fréquentiel (représenté par un ensemble de coefficients de transformée). Les transformées permettent de réduire les corrélations de signaux existantes dans le domaine spatial, ce qui conduit à un signal plus décorrélé dans le domaine de la transformation et assure une représentation plus compacte. Cela revêt une grande importance pour les étapes à venir de la numérisation et du codage entropique [1, 24].

À l'instar des normes précédentes, HEVC utilise des méthodes de transformée et de quantification 2D pour coder le signal résiduel. La transformée 2D est calculée en appliquant la transformée 1D dans les directions horizontale et verticale. Les matrices de transformation sont basées sur les fonctions de base de la transformée en cosinus discrète (DCT : *Discrete Cosine Transform*) et les tailles utilisées comprennent les formats 4×4, 8×8, 16×16 et 32×32, en fonction de la taille du bloc TB. Pour simplifier, seule la matrice de transformation 32×32 est définie. Les versions sous-échantillonnées de cette matrice sont utilisées pour calculer les tailles des matrices de transformée restantes [23].

Typiquement, les matrices de transformée directe et transformée inverse sont transposées l'une par rapport à l'autre et sont conçues pour permettre une reconstruction presque sans perte du bloc résiduel d'entrée, lorsqu'elles sont concaténées sans les étapes intermédiaires de quantification et de dé-quantification.

Pour les résidus intra de 4×4 luma, une approximation entière de la transformée en

sinus discrète (DST : *Discret Sinus Transform*)peut être utilisée comme alternative. La motivation de la DST est liée aux statistiques du signal résiduel des blocs intra prédits, qui tendent à présenter des amplitudes résiduelles plus élevées lorsque la distance aux limites du bloc gauche et supérieur, augmente. En effet, les pixels proches de ceux utilisés pour l'intra-prédiction (c'est-à-dire près des limites supérieure ou gauche) sont généralement prédits avec plus de précision que les échantillons plus éloignés. Ainsi, les résidus ont tendance à être plus importants pour les pixels situés plus loin des limites. La DST est recommandé pour le codage de ces types de résidus, car les fonctions de base démarrent lentement et puis augmentent à la difference des fonctions de base de la DCT qui démarrent hautes et baissent par la suite. La transformation DST à 4 points n'est pas beaucoup plus exigeante en calcul que la transformation DCT à 4 points mais elle ne procure qu'une réduction d'environ 1% du débit binaire dans le codage intra. L'utilisation de la transformation DST est limitée à seulement 4×4 blocs de transformée luma car, pour les blocs de taille supérieure, l'amélioration supplémentaire de l'efficacité du codage est généralement négligeable [26].

1.3.6.2 La quantification

Afin de supprimer les données non significatives dans les coefficients de transformation, une étape de quantification est utilisée après la transformée direct. Tous les coefficients de transformée sont quantifiés par un paramètre de quantification (QP), qui peut prendre des valeurs entières dans l'intervalle [0, 51] (pour 8bits/pixel). Elle est implémentée sous la forme d'une multiplication par une constante, de l'addition d'un facteur d'arrondi et d'un décalage à droite contrôlé par le paramètre de quantification (QP).

La relation résultante entre QP et la taille du pas de quantification peut être donnée par :

$$Q_{step}(QP) = (2^{1/6})^{QP-4} \tag{1.21}$$

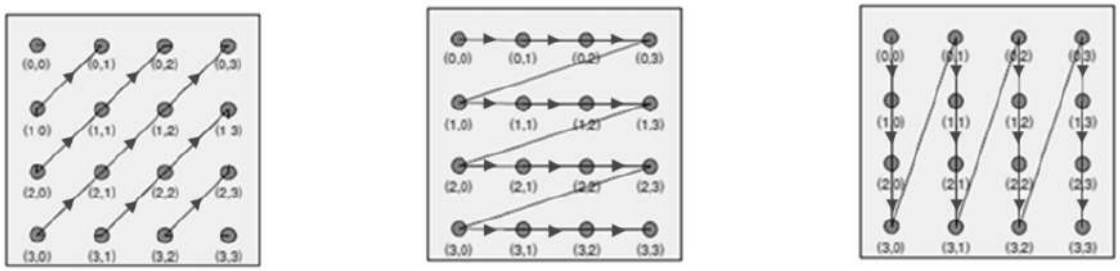
Une augmentation de 1 de QP signifie une augmentation d'environ 12% du pas de quantification Q_{step} (c'est-à-dire $2^{1/6}$). Une augmentation de 6 entraîne une augmentation de la taille du pas de quantification par un facteur 2. En plus de spécifier la différence relative entre les tailles de pas de deux valeurs QP consécutives, il est également nécessaire de définir la taille du pas absolue associée à la plage des valeurs

QP. Cela a été fait en sélectionnant $Q_{step} = 1$ pour $QP = 4$.

1.3.7 Le codage entropique

Le processus de codage entropique de l'HEVC utilise un seul outil, également disponible en H.264/AVC : le codage arithmétique binaire adaptatif au contexte (CABAC : *context-adaptive binary arithmetic coding*) [27, 28]. Le CABAC :

- reçoit les coefficients quantifiés,
- les réorganise en fonction de l'un des trois ordres de balayage disponibles (diagonal, horizontal et vertical, Figure 1.15),
- sélectionne un modèle de probabilité pour chaque élément syntaxique en fonction de son contexte,
- met à jour les modèles de probabilité et code les coefficients.



(a) Balayage diagonal **(b)** Balayage horizental **(c)** Balayage vertical

Figure 1.15. Les modes de balayage d'une partition TU

Le codage arithmétique et la modélisation du contexte utilisés dans CABAC permettent généralement d'obtenir un meilleur rendement de codage que CAVLC (codage à longueur variable adaptable au contexte). Cependant, le CABAC est plus complexe. La complexité est amplifiée à des débits binaires codés plus élevés (par exemple, en utilisant de petites valeurs QP). Pour améliorer le temps de codage entropique CABAC, le codage des coefficients transformés dans H.265 est conçu pour un débit supérieur à celui de H.264 en réduisant les dépendances de données, facilitant ainsi les opérations en parallèle [26].

CABAC est hautement séquentiel et présente une grande dépendance des données, ce qui rend difficile l'exploitation du parallélisme et du traitement en pipeline dans les implémentations de codec vidéo. La nature en série de CABAC découle de ses boucles de retour aux étapes de codage arithmétique et de modélisation du contexte,

en particulier du côté du décodeur. La sélection de contexte pour les bits binaires réguliers a deux dépendances ; la sélection du contexte et la valeur du contexte (état CABAC) peuvent dépendre des bits binaires précédents [29].

En raison du pourcentage élevé de bits consacrés au codage des résidus, il est particulièrement important que la conception du codage par transformée limite ces dépendances pour permettre des implémentations à haut débit [27, 30].

1.3.8 Les Filtres en boucle

Le post-filtrage et le filtrage en boucle sont deux approches permettant de réduire les artefacts provenant du partitionnement en blocs. Le post-filtrage n'est pas spécifié par la norme de codage vidéo et peut être effectué, par exemple, au moment d'affichage. Par conséquent, ce lui qui implémente le filtre (dans le codeur ou dans le décodeur) est libre de concevoir un algorithme basé sur les exigences propres à l'application. Les filtres en boucle (IF : *In-loop Filters*) fonctionnent dans les boucles d'encodage et de décodage. Pour cette raison, ils doivent être normalisés pour éviter toute dérive entre le codeur et le décodeur [31].

En HEVC, deux filtres en boucle consécutifs sont spécifiés et appliqués aux échantillons reconstruits avant de les écrire dans le tampon d'image décodé (DPB) de la boucle de décodeur. Le filtre DBF(*Debloking Filter*) est destiné à réduire les artefacts des blocs dus au codage par bloc, et il est similaire au DBF du standard H.264/AVC, tandis que le SAO(*Sample-Adaptive Offset*) est nouvellement introduit dans HEVC [11, 24, 26]. Alors que le DBF n'est appliqué qu'aux échantillons situés aux limites de blocs, le filtre SAO est appliqué de manière adaptative à tous les échantillons satisfaisant certaines conditions, par exemple celle d'un gradient. Au cours du développement du HEVC, il avait également été envisagé d'opérer une troisième étape de traitement appelée filtre de boucle adaptative (ALF :*Adaptive Loop Filter*) après le filtre SAO [32] ; Cependant, la fonctionnalité ALF n'est pas incluse dans la conception finale du codeur HEVC [15].

1.3.8.1 Le filtre DBF

Le DBF est un filtre passe-bas appliqué de manière adaptative au décodeur sur les bords, afin de lisser tous les artefacts des blocs résultant du codage par bloc. Le

DBF en HEVC est conçu pour être plus simple que dans H.264, et pour permettre la prise en charge du traitement en parallèle. Il est appliqué sur une taille de bloc plus grande et est effectué séparément aux autres tâches d'encodage. Cela réduit le nombre des calculs et permet leur exécution en parallèle. Le DBF est appliqué à tous les échantillons adjacents à une limite d'une partition PU ou TU, sauf lorsque la limite est également une limite d'image ou lorsque le codeur désactive le filtrage à travers les limites de Slice ou de Tile. Les limites de PU et de TU sont prises en compte, car les limites de PU ne sont pas toujours alignées sur les limites de TU pour certains blocs CB inter-prédits. Les éléments de syntaxe SPS (*Sequence Parameter Set*) et les en-têtes de Slice déterminent si le DBF est appliqué au-delà des limites des Slices et des Tiles [26, 33].

Alors que le plus petit bloc de prédiction spécifié et la plus petite taille de bloc de transformation sont 4×4, le filtre de déblocage est limité à la grille de blocs 8×8. Ainsi, les bords d'un bloc 4×4 dans un bloc 8×8 ne sont pas considérés pour les blocs luma et chroma. Cette restriction réduit considérablement la complexité du filtre DBF. Étant donné que les bords de bloc 8×8 adjacentes horizontales ou verticales ne se chevauchent pas, le traitement parallèle de tous les bords dans une orientation est possible. Avec une organisation correcte de l'ordre de traitement, un calcul parallèle sur des blocs 8×8 est également faisable [11, 34].

1.3.8.2 Le filtre SAO

SAO (Sample Adaptive Offset) est une opération de filtrage au niveau des pixels dans une partition CTU. Elle est appliquée juste après le filtre DBF s'il est activé. Deux modes SAO sont spécifiés : *edge offset* et *band offset*. Le premier est guidé par les structures directionnelles locales dans l'image à filtrer, tandis que le dernier modifie les valeurs d'intensité des pixels sans dépendance vis-à-vis du voisinage. De plus, SAO peut être désactivé au niveau des CTUs [23]. Étant donné que SAO suit directement le filtre DBF dans la chaîne de traitement du signal, les deux filtres peuvent être implémentés sous forme d'opérations successives sans stockage intermédiaire. Ainsi, l'accès à la mémoire requis peut être réduit. L'entrée du SAO est indépendante de la sortie du DBF, c'est-à-dire qu'elle n'est pas affectée par le DBF. Par conséquent, le processus SAO peut être mis en parallèle à grande échelle. Les limites des Slices ou

des Tiles sont traitées comme s'il s'agissait de limites d'image avec filtrage à travers les bords de Slices ou de Tiles non activé [11, 18].

Le SAO tente de réduire les artefacts visuels indésirables, qui pourraient devenir plus graves avec des transformées importantes et des interpolations plus longues. Le SAO tente également de réduire la distorsion moyenne entre les échantillons originaux et les échantillons reconstitués en classant d'abord les pixels reconstruits en différentes catégories, en obtenant une offset pour chaque catégorie, puis en ajoutant l'offset à chaque pixel de la catégorie. l'indice du classifieur utilisé et l'offset de la région sont codés dans le bitstream [35].

Le mode SAO applicable (*edge offset* ou *band offset*) est indiqué pour chaque CTU. Les paramètres peuvent être explicitement signalés ou dérivés du voisinage. Pour réduire le coût de signalisation, le mode et les paramètres applicables peuvent être hérités de la CTU voisine supérieure ou gauche. De plus, SAO peut être appliqué à la luma ainsi qu'aux composantes chroma, où les deux composantes chroma partagent le même mode SAO. Les paramètres d'offset sont néanmoins configurés individuellement pour chaque composante. Pour hériter une configuration SAO, les paramètres de la ligne CTU doivent être stockés pour une utilisation ultérieure dans la ligne suivante. Étant donné que les informations doivent uniquement être stockées pour un partitionnement au niveau des CTUs, les besoins en mémoire tampon de ligne pour SAO sont faibles [11].

1.3.9 Options et configurations d'encodage en HEVC

Le Software HM fournit deux configurations d'outils de codeur, désignées *Main* et *Main10* dans le document de condition de test JVC-VC. Les outils de codage pour la configuration *Main* correspondent à ceux pris en charge dans le *profil Main* de la spécification HEVC. Les outils de codage pour la configuration *Main10* correspondent au *profil Main10* de la spécification HEVC avec une longueur de bits pour la luminance et la chrominance toutes deux définies sur 10 bits [36].

Le software HM fonctionne avec trois types de structures de prédiction temporelle en fonction des conditions expérimentales : All-Intra, Low-Delay and Random Access. La gestion de la liste d'images de référence dépend de la structure de prédiction temporelle utilisée.

1.3.9.1 La configuration *Intra-only* ou *AI :All Intra*

Dans le cas du codage All-Intra, chaque image d'une séquence vidéo est codée en tant qu'image IDR (*Instantaneous Decoding Refresh*). Aucune image de référence temporelle n'est utilisée. La valeur du paramètre QPI (*QP Intra*) est identique pour toutes les images de la séquence vidéo. La Figure 1.16 donne une représentation graphique d'une configuration AI, où le numéro associé à chaque image représente l'ordre du codage (POC : *Picture Order Coding*).

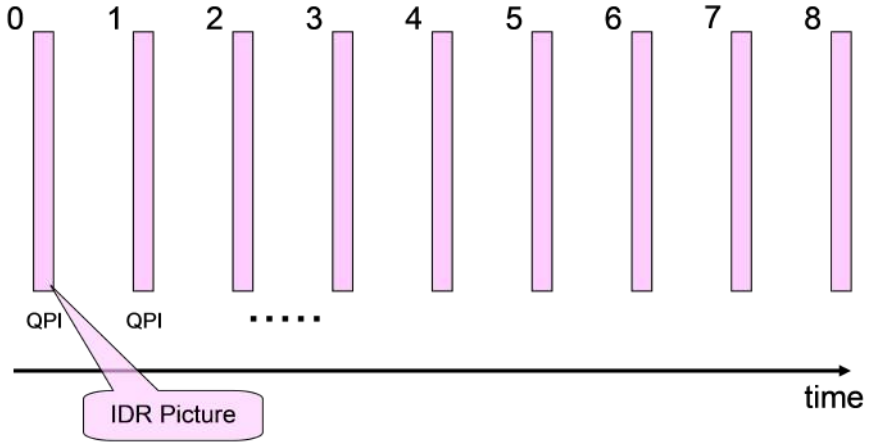


Figure 1.16. Représentation graphique de la configuration AI

1.3.9.2 La configuration *LD : Low-Delay*

Deux configurations de codage ont été définies pour tester les performances de codage LD. Pour ces deux configurations, seule la première image d'une séquence vidéo est codée en tant qu'image IDR. Les autres images sont codées sous forme d'images P et B (GPB) généralisées. La GPB ne peut utiliser que des images de référence dont le POC est plus petit que l'image actuelle (c'est-à-dire que toutes les images de référence dans les listes RefPicList0 et RefPicList1 sont temporairement antérieures dans l'ordre d'affichage par rapport à l'image actuelle). Le contenu de RefPicList0 et RefPicList1 est, par conséquent, identique. La combinaison de liste d'images de référence est utilisée pour la gestion et le codage entropique de l'indice d'images de référence. La Figure 1.17 montre une représentation graphique de cette configuration LD. Le numéro associé à chaque image représente l'ordre de codage. Le QP de chaque image codée inter est obtenu en ajoutant une offset au QP de l'image codée intra, en

fonction de la couche temporelle. Dans la configuration alternative du LD, toutes les images inter sont codées en tant qu'images P, où seul le contenu de la liste RefPicList0 est utilisé pour la prédiction inter [24].

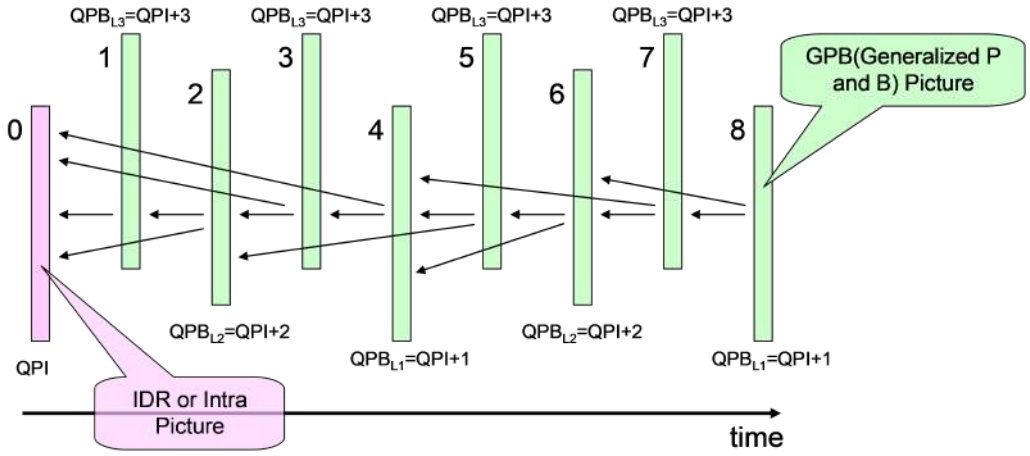


Figure 1.17. Exemple de codage utilisant la configuration LD

1.3.9.3 La configuration *RA : Random Access*

Pour la configuration RA, une structure B hiérarchique est utilisée pour le codage. La Figure 1.18 montre une représentation graphique d'une configuration RA, où le numéro associé à chaque image représente l'ordre de codage. Une image intra est codée à environ une seconde d'intervalle. La première image intra d'une séquence vidéo est codée en tant qu'image IDR et les autres images intra sont codées en tant qu'images intra non IDR. Les images situées entre les images intra successives dans l'ordre d'affichage sont codées en tant qu'images B. L'image GPB est utilisée comme couche temporelle la plus basse pouvant faire référence à des images I ou GPB pour la prédiction inter. Les deuxième et troisième couches temporelles sont constituées d'images B référencées, tandis que la couche temporelle la plus élevée contient uniquement des images B non référencées. Le QP de chaque image inter-codée est obtenu en ajoutant une offset au QP de l'image codée intra en fonction de la couche temporelle [18].

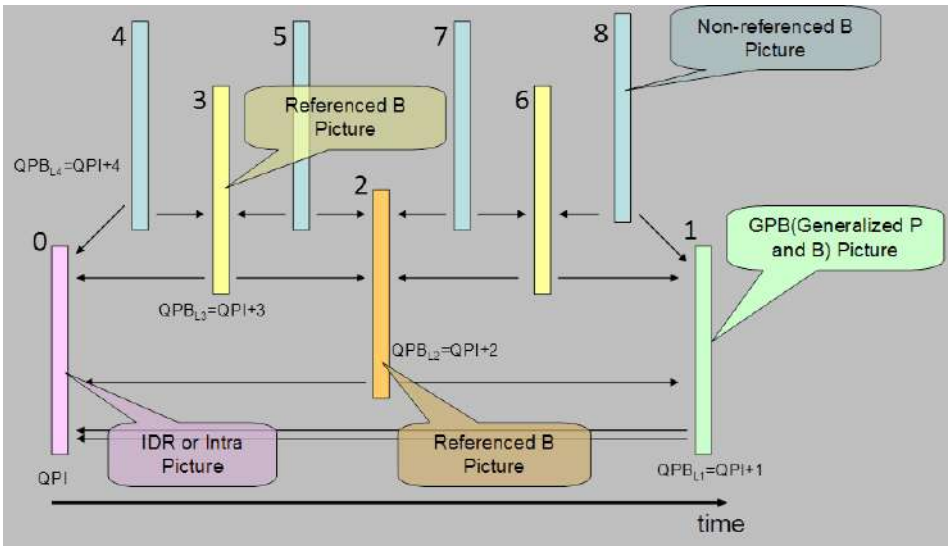


Figure 1.18. Représentation graphique de la configuration RA

1.4 L'optimisation Débit-Distorsion (RDO : *Rate-Distorsion Optimization*)

Le débit binaire d'une séquence vidéo codée peut être réduit en admettant une distorsion acceptable dans la vidéo. Ce processus est connu sous le nom d'optimisation Rate-Distorsion (RDO). La RDO permet à un fournisseur de différencier ses produits de codeurs. Les algorithmes RDO choisissent essentiellement les modes de codage pour obtenir le meilleur compromis entre faible distorsion et débit réduit. La Figure 1.19 illustre un compromis typique entre le débit binaire et la distorsion. Le débit peut être limité à un taux maximal au détriment de certaines distorsions. Alternativement, un niveau maximum de distorsion peut être défini et le débit peut être minimisé. Ces problèmes d'optimisation avec contraintes peuvent être complexes [26].

Supposons que l'image complète soit partitionnée en un ensemble de N blocs CTBs, $P = CTB_n, n = 0, ..., N-1$, et que le mode de codage du bloc CTB soit désigné par un indice de mode de codage M_n. L'ensemble des modes de codage appliqués à l'image est désigné par $M = M_n, n = 0, ..., N-1$. La tâche d'optimisation pour le codeur consiste à trouver le meilleur mode de codage M_{opt}, réglé pour l'image en cours soumise à une contrainte de débit donné R_c, de sorte que la distorsion de l'image soit minimisée.

Soit :

$$\min_{M} \left(D\left(P, M\right) \right), \text{ avec } R\left(P, M\right) < R_c \tag{1.22}$$

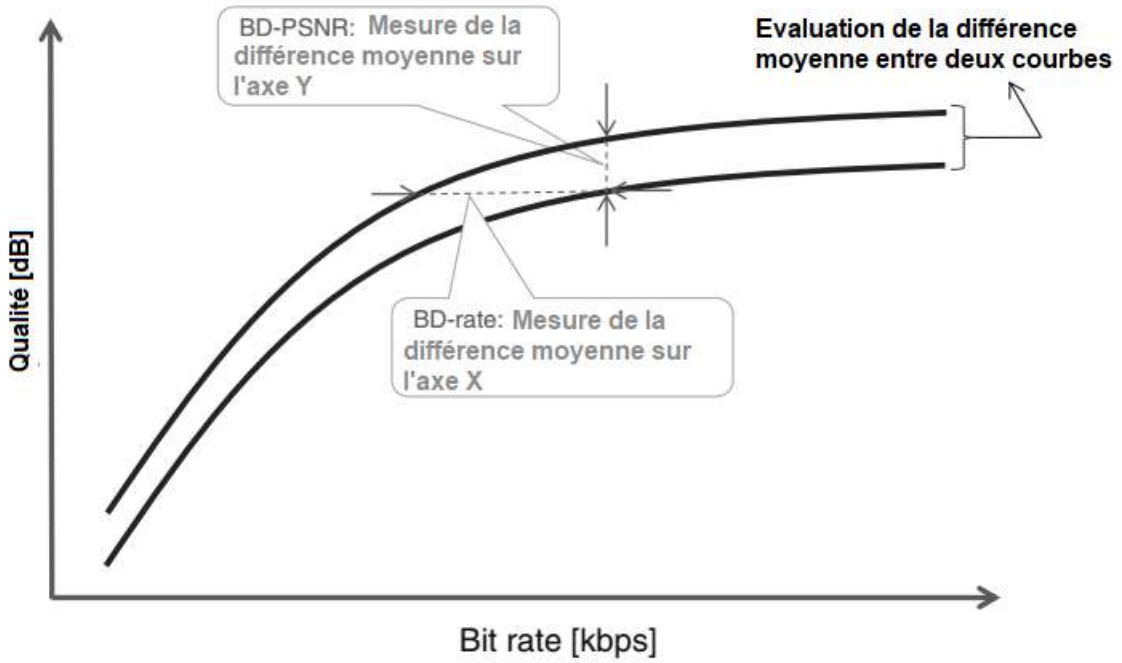


Figure 1.19. Exemple de courbes Distorsion-Débit

où R est le débit binaire, et D est la distorsion.

En utilisant le facteur de pondération lagrangien λ, ce problème se transforme en un problème sans contrainte,

$$M_{opt} = \min_{M} \left(J\left(P, M|\lambda\right)\right) \tag{1.23}$$

avec :

$$J\left(P, M|\lambda\right) = D\left(P, M\right) + \lambda R\left(P, M\right) \tag{1.24}$$

On suppose que le coût des blocs CTB est additif. Sous cette hypothèse, le coût total commun pour le codage de l'image peut être déterminé en sélectionnant le coût minimum pour chaque CTU :

$$\min_{M} \sum_{n=0}^{N-1} J\left(CTB_n, M|\lambda\right) = \sum_{n=0}^{N-1} \min_{M_n} J\left(CTB_n, M_n|\lambda\right) \tag{1.25}$$

Le facteur de pondération λ applicable dépend de la mesure de distorsion utilisée ainsi que de la configuration du codage pour la séquence vidéo donnée.

1.4.1 Mesure de distorsion

Si $B_{i,j}$ et $P_{i,j}$ sont les échantillons respectifs des blocs originel et prédit de $M \times N$ pixels, la mesure de distorsion D peut être définie comme suit :

$$D = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |B_{i,j} - P_{i,j}|^b \tag{1.26}$$

pour $b = 1$, D représente la somme des différences absolues (SAD :*Sum of Absolute Differences*), et lorsque $b = 2$, la distorsion correspond à la somme des différences (ou erreurs)au carré (SSD :*Sum of Squared Differences* ou SSE :*Sum of Squared Error*).

SSD est la métrique de distorsion la plus commune alors que SAD est utilisé pour l'estimation du mouvement. Par conséquent, tous les codeurs sont principalement optimisés par rapport au SSE en utilisant l'erreur quadratique moyenne (MSE) définie par :

$$MSE = \frac{SSE}{N \times M} \tag{1.27}$$

La somme des différences transformées absolues (SATD :*Sum of Absolute Transformed Differences*) est une autre métrique de qualité vidéo utilisée pour la correspondance de blocs dans l'estimation de mouvement. SATD utilise une transformée de fréquence (par exemple, la transformée de Hadamard, qui est une version généralisée de la transformée de Fourier) des différences entre les pixels dans le bloc d'origine et les pixels correspondants dans le bloc codé. Un petit sous-bloc (par exemple, 4×4 pixels) est généralement utilisé pour la transformée plutôt que le bloc de taille complète (par exemple, 16×16). Bien que SATD est plus lent que SAD en raison de la complexité élevée, SATD peut prédire la qualité vidéo objective et subjective avec plus de précision.

$$SATD = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |HT(i,j)| \tag{1.28}$$

où $HT(i,j)$ est le coefficient (i,j) d'un bloc obtenu en appliquant une transformée de Hadamard à la différence entre les blocs P et B.

1.4.2 Mesure de la qualité

Le pic du rapport signal/bruit (PSNR) est la mesure de la qualité objective la plus utilisée en traitement d'image et vidéo. Le PSNR (équation. (1.29)) est mesuré sur une échelle logarithmique et dépend de l'erreur quadratique moyenne (EQM) entre une image originale et une image altérée , par rapport à $(2^n - 1)^2$ qui représente le carré de la plus grande valeur possible, avec n le nombre de bits par pixel.

$$PSNR_{dB} = 10 \log_{10} \left((2^n - 1)^2 / MSE \right) \tag{1.29}$$

Le PSNR peut être calculé facilement et rapidement, et constitue donc une mesure de qualité très répandue, largement utilisée pour comparer la «qualité» des images vidéo compressées et décompressées.

Le PSNR est calculé pour chaque composant YC_bC_r. Dans le domaine YC_bC_r, le système visuel humain est plus sensible à la luminance (Y) qu'à la chrominance (C_b ou C_r). Par conséquent et dans la pratique, le PSNR pour la luminance (PSNR Y) est la plus importante mesure de la qualité objective.

Afin de comparer l'efficacité de codage d'un codec évalué à celui de référence, la différence moyenne des deux courbes R–D (*Rate-Distosion*) est calculée (courbes généralement construites à partir de quatre valeurs de QP). La différence de débit binaire moyen (différence dans la direction horizontale) est appelée BD-R (*Bjøntegaard's Delta rate* [37]) et la différence moyenne de PSNR (différence dans la direction verticale) est appelée BD PSNR.

1.5 Conclusion

Ce chapitre est dédié à la motivation pour la compression vidéo, ainsi que quelques concepts généraux concernant la norme de codage vidéo H.265/HEVC.

Les redondances existant dans le signal vidéo sont exploitées par les codeurs vidéo actuels via des prédictions. Deux types de prédictions peuvent êtres envisagées à savoir la prédiction spatiale ou la prédiction temporelle, selon que la source de prédiction est la même image ou une autre image.

Le résultat de prédiction est combiné avec la source pour former

l'erreur de prédiction appelée résidu, qui ensuite passe par une étape de transformée/quantification afin de concentrer le résidu transformé en un ensemble de coefficients le plus limité possible.

Une grande flexibilité est introduite par la norme HEVC dans le choix de la taille de bloc appropriée pour la prédiction PU et les tailles de transformation TU. La diversité des tailles possibles pour les différentes unités de traitement conduit à des performances de compression remarquables. Cependant, la complexité de calcul engendrée est beaucoup plus élevée par rapport aux standards de codage vidéo existants.

La suite de notre étude portera sur l'implémentation et l'optimisation du codeur HEVC en vue d'une réduction de la complexité de calcul. Le chapitre suivant présente l'une des techniques d'optimisation sans dégradation de l'efficacité de codage, basée sur l'utilisation des instructions SIMDs (*Single-Instruction-Multiple-Data*) d'un DSP.

Chapitre 2

Optimisation et implémentation de l'encodeur HEVC sur DSP TMS320C6678

Sommaire

2.1 Introduction **47**
2.2 Description de la plateforme à DSP TMS320C6678 . . . . . **48**
2.2.1 Caractéristiques du DSP TMS320C6678 [38] 48
2.2.2 Structure interne du cœur C66x 50
2.3 Environnement de développement **52**
2.4 Considérations pratiques pour l'implémentation du code HM sur DSP **54**
2.5 Optimisation de l'encodeur HEVC pour l'implémentation sur DSP **56**
2.5.1 Optimisations au niveau compilateur 57
2.5.2 Optimisation des modules SADs 63
2.5.3 Optimisation du module SSE 64
2.5.4 Optimisation du Module HAD 65
2.5.5 Résultats expérimentaux et discussion 69
2.6 Conclusion **73**

2.1 Introduction

La complexité et le temps de calcul dans la norme HEVC sont beaucoup plus élevés, ce qui rend difficile la réalisation d'une solution embarquée assurant l'exécution en temps réel avec la technologie des processeurs classiques. Néanmoins, les nouvelles technologies de processeurs programmables, telles que les processeurs de signal numérique multicœur (DSP), offrent une solution très prometteuse

pour surmonter ces contraintes. Dans ce chapitre, une implémentation optimisée de l'encodeur vidéo HEVC sur un seul cœur parmi les huit cœurs du DSP TMS320C6678 est présentée avant de procéder à une implémentation multicœur. Sur la base des caractéristiques architecturales des DSPs, différentes optimisations SIMD (Single-Instruction-Multiple-Data) sont adoptées pour minimiser le temps d'exécution des fonctions à grande latence.

2.2 Description de la plateforme à DSP TMS320C6678

Avec l'émergence de nouveaux produits comprenant la communication numérique sans fil, la reconnaissance vocale, le multimédia et la téléphonie par Internet, les concepteurs s'appuient sur les DSP pour fournir la puissance de calcul nécessaire à des performances cruciales en temps réel .

Afin d'aider les développeurs dans l'implémentation et la conception de systèmes embarqués, Texas Instrument (TI) fournit une nouvelle solution intégrant des fonctionnalités à virgule fixe et à virgule flottante dans le même noyau DSP. La conversion en point fixe n'est plus nécessaire avec les capacités de calcul des nouveaux DSP C66× de TI.

2.2.1 Caractéristiques du DSP TMS320C6678 [38]

Le processeur de signal numérique C66× rehausse les performances des DSP C64×+ et C674× grâce à des améliorations et de nouvelles fonctionnalités, dont certaines visent des performances accrues pour le traitement vectoriel. Dans les DSPs C66×, la capacité de traitement vectoriel est améliorée en étendant la largeur des instructions SIMD. Ils peuvent exécuter des instructions qui fonctionnent sur des vecteurs de données à 128 bits. Par exemple, l'instruction QMPY32 est capable d'exécuter la multiplication d'élément par élément entre deux vecteurs de quatre données à 32 bits chacune. Les DSPs C66× prennent également en charge des instructions SIMD pour les opérations en virgule flottante. L'amélioration de la capacité de traitement vectoriel (où chaque instruction peut traiter plusieurs données en parallèle), associée au parallélisme naturel de l'architecture C6000 au niveau des

instructions (exécution maximale de 8 instructions par cycle) permet d'obtenir un très haut niveau de parallélisme exploitable à travers le compilateur C/C++ optimisé de TI.

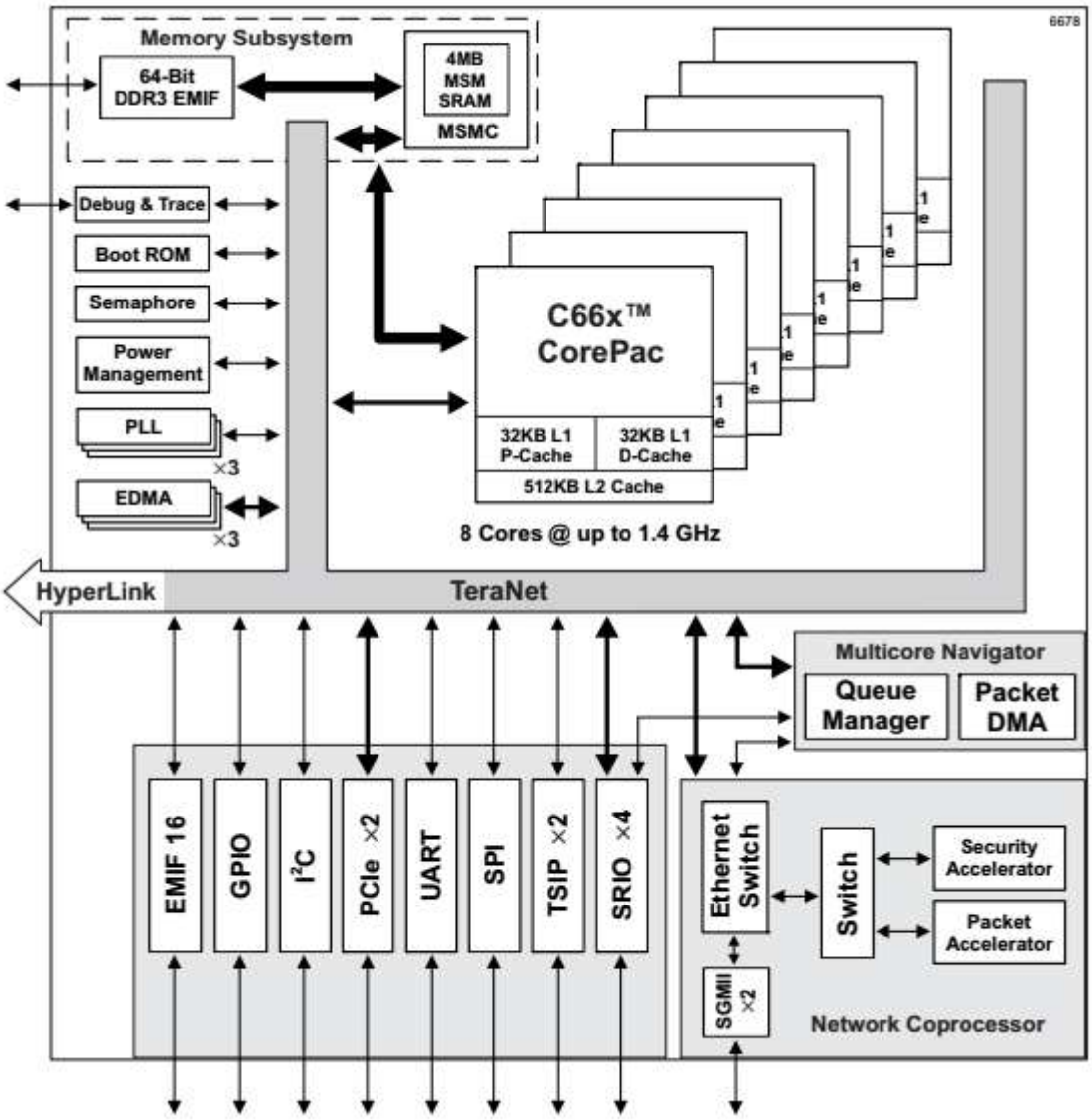


Figure 2.1. Vue d'ensemble du DSP TMS320C6678

La Figure 2.1 montre le schéma synoptique d'un DSP multicœur TMS320C6678. Il repose sur l'architecture multicœur KeyStone de TI, et comprend huit cœurs DSP d'une puissance de traitement pouvant atteindre 10 GHz. Les performances de traitement peuvent atteindre 320 GMAC ou 160 GFLOPS à 1,25 GHz par cœur. En plus de 32 Ko de mémoire cache de programme et de données L1, chaque cœur comprend 512 Ko de mémoire L2 locale et 4 Mo de mémoire cache niveau L2 partagée. Le DSP C6678 peut également accéder à 8 Go de mémoire DDR3 externe partagée, à

une vitesse de transfert de 12,8 Go/s. Les fonctionnalités de l'architecture KeyStone de TI comportent plusieurs éléments novateurs permettant des taux de traitement élevés. Parmi ces fonctionnalités, citons le contrôleur MSMC (Multicore Shared Memory Controller), qui permet aux différents cœurs de partager efficacement la mémoire et l'accès à la mémoire externe.

2.2.2 Structure interne du cœur C66x

Le cœur DSP C66× se compose de huit unités fonctionnelles, de deux groupes de registres et de deux chemins de données, Figure 2.2. Les deux fichiers de registres à usage général (A et B) contiennent chacun trente-deux registres de 32 bits pour un total de 64 registres. Les registres à usage général peuvent être utilisés pour les données et/ou comme pointeurs d'adresses. Les types de données pris en charge comprennent

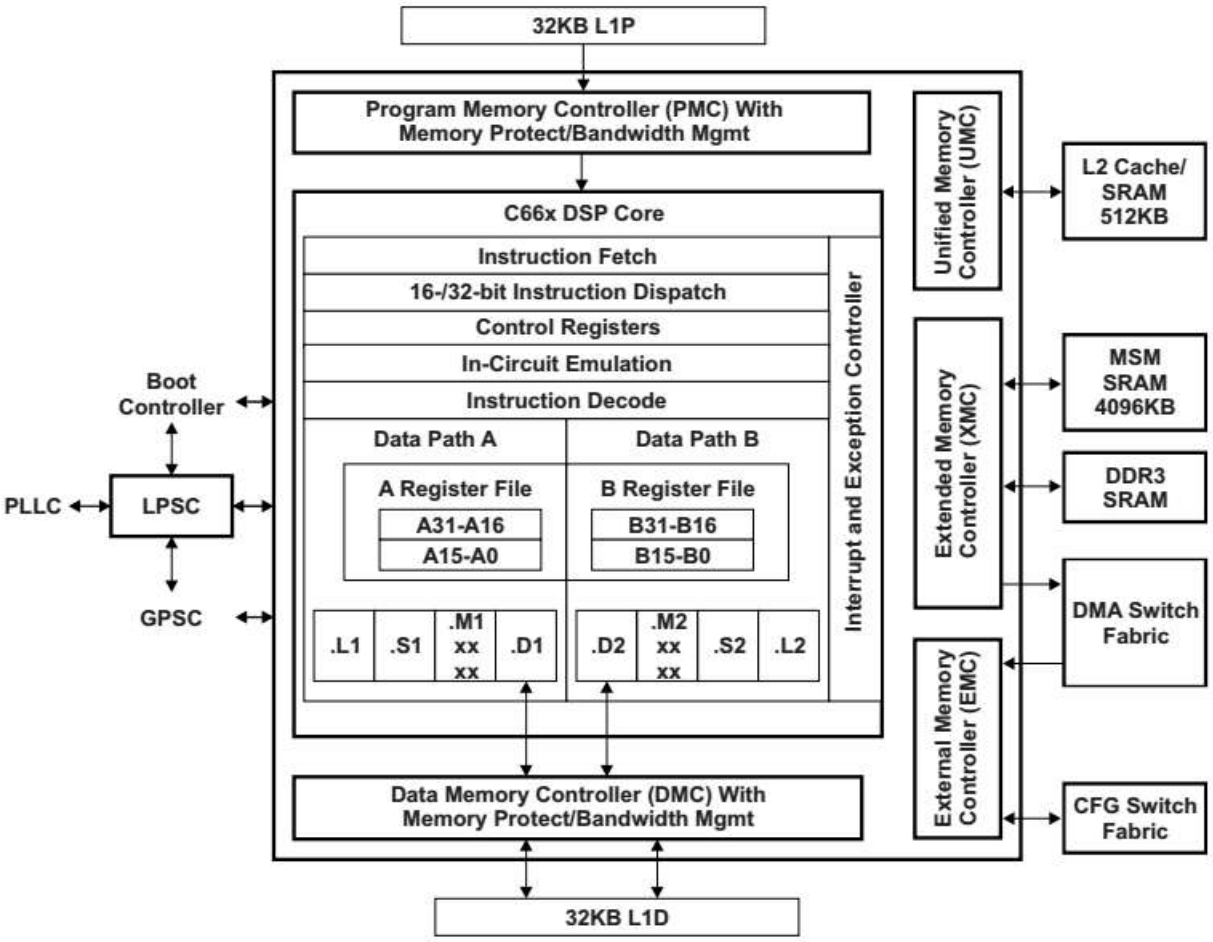


Figure 2.2. Structure interne du C66x

les données en groupes de 8 bits, les données en groupes de 16 bits, les données à 32 bits, les données à 40 bits et les données à 64 bits. Les multiplications prennent également en charge les données 128 bits. Les valeurs de 40 bits ou de 64 bits sont stockées dans des paires de registres, les 32 LSB de données placés dans un registre pair et les 8 ou 32 MSB restants dans le registre supérieur suivant qui est impair. Les valeurs de données de 128 bits sont stockées dans des quadruplés de registre, les 32 bits de poids faible de données étant placés dans un registre multiple de 4 et les 96 MSB restants dans les 3 registres supérieurs suivants.

Les huit unités fonctionnelles (.M1, .L1, .D1, .S1, .M2, .L2, .D2 et .S2) sont capables chacune d'exécuter une instruction par cycle d'horloge. Les unités fonctionnelles .M effectuent toutes les opérations de multiplication. Les unités .S et .L effectuent un ensemble général d'opérations arithmétiques, logiques et de branchement. Les unités .D sont utilisées principalement pour le chargement/stockage de données.

Chaque unité .M du C66× peut effectuer l'une des opérations à virgule fixe suivantes, à chaque cycle :

- quatre multiplications 32×32 bits,
- seize multiplications 16×16 bits,
- quatre multiplications 16×32 bits,
- quatre multiplications 8×8 bits,
- quatre multiplications 8×8 bits avec accumulation
- et quatre multiplications 16×16 avec possibilités d'addition/soustraction.

La multiplication de champ de Galois est également prise en charge pour les données 8 bits et 32 bits. De nombreux algorithmes de communication tels que les FFT et les modulateurs/démodulateurs nécessitent une multiplication complexe. Chaque unité .M du C66× peut ainsi effectuer :

- une multiplication complexe de 16×16 bits avec ou sans arrondi,
- deux multiplications de complexe 16×16 bits avec arrondi
- et une multiplication d'un complexe de 32×32 bits avec arrondi.

Le C66× peut également exécuter deux instructions de multiplication complexes de 16×16 bits et une de 32×32 bits, entre un nombre complexe avec le conjugué d'un autre nombre complexe doté de la capacité d'arrondissement.

Le traitement du signal dans les communications nécessite également une utilisation

intensive des opérations matricielles. Chaque unité .M du C66× est capable de multiplier un vecteur complexe [1×2] par une matrice complexe [2×2] par cycle, avec ou sans possibilité d'arrondissement. Une autre version existe également permettant de multiplier le conjugué d'un vecteur [1×2] avec une matrice complexe [2×2].

Chaque unité .M du C66× supporte également des opérations de multiplication à virgule flottante IEEE à partir du DSP C674×. Elle comprend un multiplicateur simple précision chaque cycle et un multiplicateur double précision tous les 4 cycles. Il existe également une multiplication à précision mixte, qui permet de multiplier une valeur à simple précision par une valeur à double précision, et une opération permettant de multiplier deux nombres à simple précision conduisant à un nombre à double précision. Le DSP C66× améliore les performances par rapport aux multiplicateurs double précision C674×, en ajoutant une instruction permettant la multiplication double précision par cycle, et réduit également le nombre de cycles de retard de 10 à 4. Chaque unité .M du C66× peut également effectuer les opérations suivantes en virgule flottante à chaque cycle d'horloge : une, deux ou quatre multiplications à simple précision ou une multiplication à simple précision complexe.

Les unités .L et .S peuvent désormais prendre en charge des opérandes allant jusqu'à 64 bits. Cela permet de nouvelles versions d'un grand nombre d'instructions arithmétiques, logiques, voire de compression de données, afin d'avoir davantage d'opérations parallèles par cycle. Des instructions supplémentaires ont été rajoutées pour améliorer les performances des instructions d'addition et de soustraction en virgule flottante, notamment la possibilité d'effectuer une addition ou une soustraction en double précision par cycle. La conversion de type entre valeurs entières et à simple précision peut se faire sur les unités .L ou .S du C66×. L'unité .L dispose également d'instructions supplémentaires pour les opérations logiques AND et OR, ainsi que de la rotation à 90 degrés ou 270 degrés des nombres complexes (jusqu'à deux opérations par cycle).

2.3 Environnement de développement

Un software de gestion efficace est nécessaire pour exploiter pleinement la puissance des DSP. En fait, le software est sans doute le composant le plus critique dans la

conception par DSP. Selon les estimations de l'industrie [39], environ 80% de l'effort réside dans le développement d'un système embarqué et 80% de la complexité du système réside dans le software associé.

En dépit de ces résultats, les concepteurs négligent souvent la disponibilité d'outils sophistiqués de développement et faciles d'utilisation, lors du choix d'un processeur au sein d'un système fonctionnant en temps réel. Un système de développement qui offre à ses utilisateurs une gamme étendue d'outils fonctionnant de manière efficace permet au développeur de se concentrer sur l'innovation.

Conçu pour les plates-formes C6000 à hautes performances et C5000 à consommation réduite d'énergie de firme Texas Instruments, Code Composer Studio est un environnement de développement qui intègre étroitement les fonctionnalités des composants suivants (dont certains sont illustrés à la Figure 2.3), le tout dans une architecture extensible [39] :

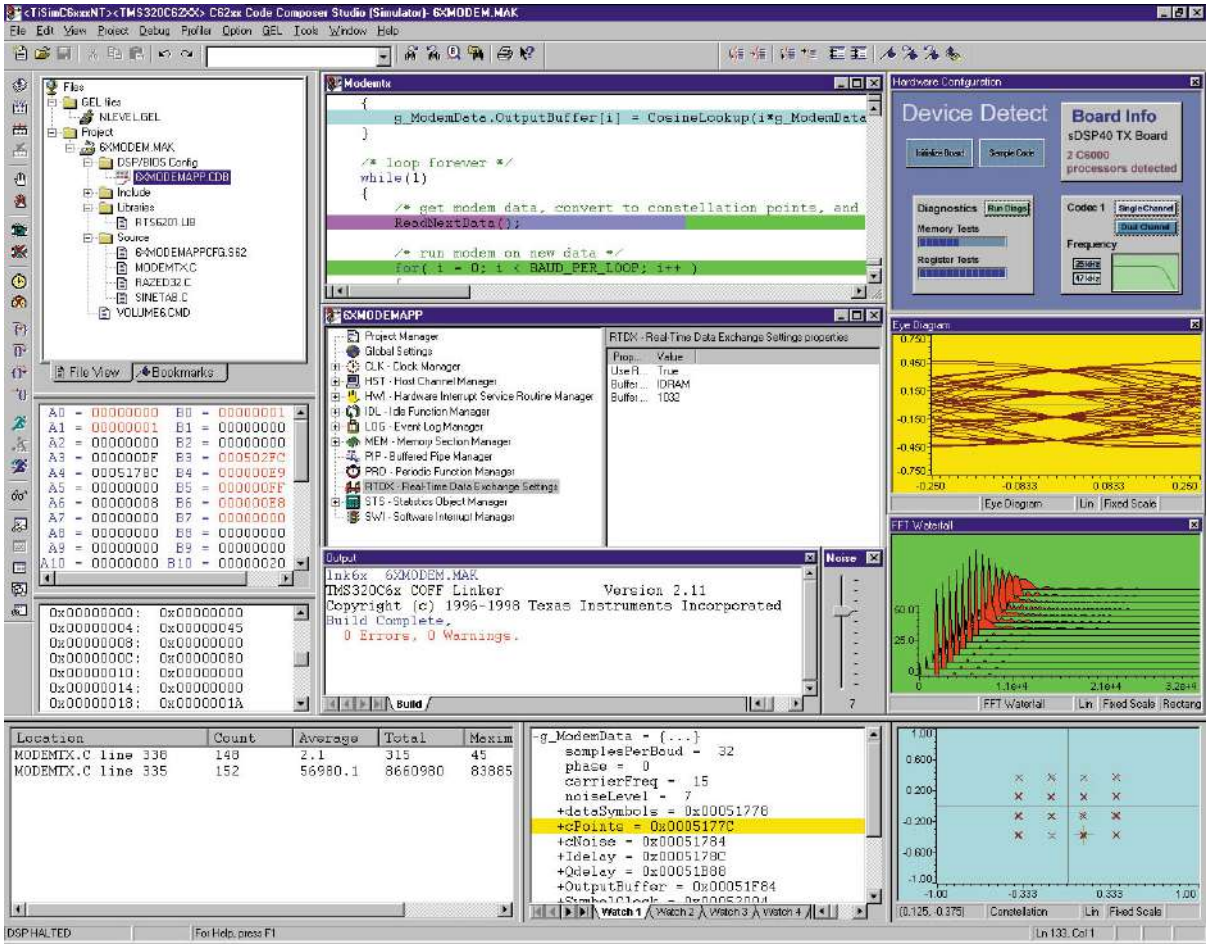


Figure 2.3. L'environnement Code Composer Studio de Texas Instruments

- Environnement de développement intégré (Code Composer) avec éditeur, débogueur, gestionnaire de projet, profileur, points de sonde, etc,
- Compilateur C, Optimiseur d'Assembleur et Éditeur de liens (Outils de génération de code),
- Simulateur de jeu d'instructions,
- Logiciel de base en temps réel (DSP/BIOS, SYS/BIOS),
- Échange de données en temps réel entre hôte et cible (RTDX),
- Analyse en temps réel et visualisation de données.

Code Composer Studio intègre tous les outils hôtes et cibles dans un environnement unifié. Il simplifie également la configuration du système DSP et la conception des applications pour aider les concepteurs à démarrer plus rapidement.

L'organigramme de la Figure 2.4 résume les étapes clefs à suivre pour obtenir les meilleures performances d'un code.

Dans la phase 1 : le code peut être écrit en C/C ++ sans aucune connaissance du C6000. On peut utiliser la compilation avec l'option **opt_level = 3**, pour identifier les zones inefficaces dans le code C/C++.

La Phase 2 consiste en l'utilisation des options intrinsèques et du CCS pour améliorer le code C/C ++. Si le code n'est toujours pas aussi efficace que l'on souhaite, la phase 3 est adoptée.

Au cours de la phase 3, on doit extraire les zones critiques du code C/C ++ et les réécrire en assembleur linéaire avec l'utilisation de l'outil "optimiseur de l'assembleur". L'amélioration des performances dans cette étape prend plus de temps que dans la phase 2.

2.4 Considérations pratiques pour l'implémentation du code HM sur DSP

Le software de référence pour HEVC s'appelle HM (**H**EVC Test **M**odel). Ce modèle de test suggéré est destiné à fournir une efficacité de codage proche des propositions les plus performantes dans le test subjectif des soumissions de CfP (*Call for Proposal*), et à fournir un point de complexité proche de celui des soumissions les plus complexes, ce qui permet une amélioration substantielle de l'efficacité de codage. L'un des principaux

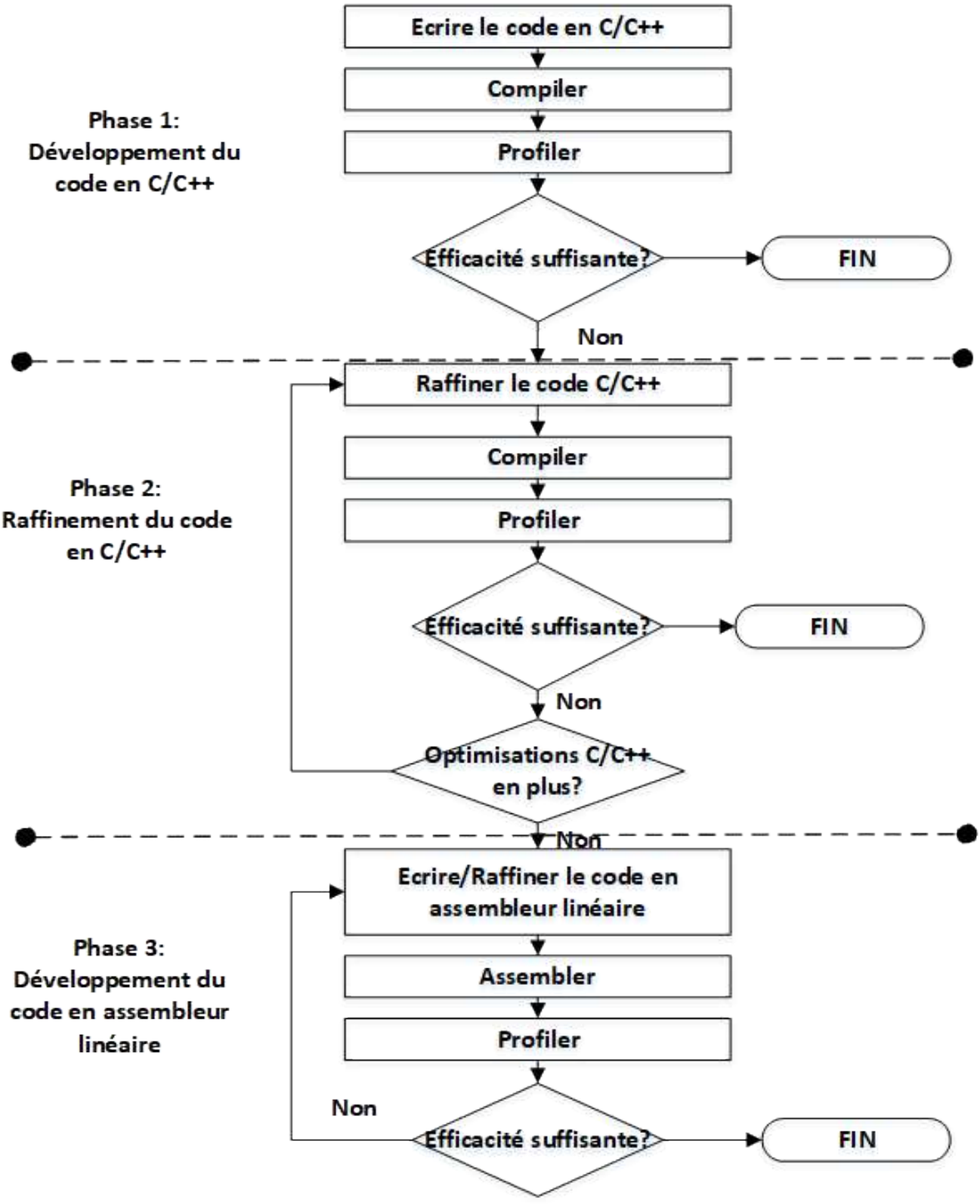


Figure 2.4. Les étapes à suivre lors du développement d'un code.

objectifs du software de référence est de fournir une base, sur laquelle des expériences sont effectuées afin de déterminer quels outils de codage fournissent les performances

de codage souhaitées.

Le software peut être récupéré (téléchargé) à partir de l'un des serveurs SVN suivants :

- https ://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/
- svn ://hevc.kw.bbc.co.uk/svn/jctvc-hm/

Le software comprend le code source de l'encodeur, du décodeur et des parties communes sous forme de plusieurs projets sous VS (Visuel Studio). Étant intéressé uniquement par le codeur à optimiser, notre première étape consistait à isoler les fichiers source de l'encodeur afin de construire un projet unique admettant en entrée une séquence vidéo à encoder et produisant le code binaire (bitstream) en sortie. Néanmoins, certaines modifications doivent être apportées sur le code, pour le rendre fonctionnel sur DSP. Ceci est dû aux différences entre le compilateur C/C++ du DSP et le compilateur C/C++ (plus complet) du VS. Il s'agit principalement de :

1. Intégration de la bibliothèque *String*,
2. Redéfinition de la fonction *find*,
3. Redéfinition de la fonction *strdup*,
4. Redéfinition du type *bool*,
5. Suppression des fonctions permettant de mesurer les performances à l'aide des Timers du PC,
6. Inclusion des Timers internes au DSP pour la mesure des performances temporelles.

2.5 Optimisation de l'encodeur HEVC pour l'implémentation sur DSP

En général, l'optimisation d'une implémentation à base d'un DSP peut être assurée par le biais d'un ensemble de techniques, notamment :

- L'inclusion d'informations supplémentaires au compilateur pour aider l'optimiseur d'assembleur à générer un code plus efficace (utilisation des #pragma),

- Le déroulement des boucles peut améliorer parfois les performances. Si le compteur de boucles est impair et qu'il y a des unités fonctionnelles non utilisées, il est préférable de dérouler les boucles pour activer les unités vacantes. Cela permet un parallélisme poussé et réduit considérablement le temps d'exécution,
- L'utilisation d'instructions intrinsèques,
- L'utilisation des instructions SIMD.

Toutes ces techniques ont été essayées, et en particulier les instructions SIMDs pour l'optimisation des modules suivants :

1. La somme des différences absolues : SAD.
2. La somme des différences au carré : SSE.
3. Le calcul de la transformée Hadamard : HAD.

Ces modules se trouvent dans la classe TComRdCost du software de référence. Nous avons utilisé le langage assembleur linéaire pour atteindre l'optimum de performances. L'assembleur linéaire est similaire au code assembleur pur où le même jeu d'instructions est utilisé dans l'écriture du code assembleur linéaire. Dans ce dernier code, il n'est pas nécessaire de spécifier toutes les informations nécessaires comme dans le code de l'assembleur pur. L'optimiseur d'assembleur est appelé à fournir de telles informations, qui concernent :

- L'indication des instructions à exécuter en parallèle
- La latence du pipeline liée aux instructions multi-cycles.
- L'usage des registres internes
- L'indication de l'unité fonctionnelle à utiliser
- L'indication du chemin croisé entre unités

Il se trouve éventuellement qu'il est plus intéressant d'inclure dans notre cas, certaines informations spécifiques orientant l'optimiseur d'assemblage dans le but d'atteindre une performance plus élevée.

2.5.1 Optimisations au niveau compilateur

Le compilateur C/C ++ peut effectuer un ensemble diversifié d'optimisations. Des optimisations de haut niveau sont effectuées dans l'optimiseur, tandis que des optimisations de bas niveau se produisent dans le générateur de code. L'utilisation

des niveaux d'optimisation de haut niveau, tels que *--opt_level = 2* et *--opt_level = 3*, permet l'obtention d'un code optimal.

Le moyen le plus simple d'invoquer l'optimisation consiste à spécifier l'option *--opt_level = n* sur la ligne de commande du compilateur. Nous pouvons utiliser **-On** pour aliaser l'option *--opt_level*. Le **n** indique le niveau d'optimisation (0, 1, 2 et 3), qui contrôle le type et le degré d'optimisation.

* *--opt_level = 0* ou **-O0**
  - Effectue une simplification du graphe de contrôle
  - Alloue des registres aux variables
  - Effectue une rotation de boucle
  - Élimine le code inutilisé
  - Simplifie les expressions et les déclarations
  - Développe les appels aux fonctions déclarées **inline**
* *--opt_level = 1* ou **-O1**
  - • Effectue toutes les optimisations --opt_level = 0 (-O0), plus :
  - Supprime les assignations non utilisées
  - Élimine les expressions communes locales
* *--opt_level = 2* ou **-O2**
  - • Effectue toutes les optimisations --opt_level = 1 (-O1), plus :
  - Effectue un pipeline software
  - Effectue des optimisations de boucles
  - Élimine les sous-expressions communes globales
  - Élimine les affectations globales non utilisées
  - Convertit les références de tableau en forme de pointeur incrémenté
  - Effectue le déroulement des boucles

  NB : L'optimiseur utilise *-/opt_level = 2* (-O2) par défaut si vous utilisez --opt_level (-O) sans niveau d'optimisation.
* *--opt_level = 3* ou **-O3**
  - • Effectue toutes les optimisations --opt_level = 2 (-O2), plus :
  - Supprime toutes les fonctions qui ne sont jamais appelées
  - Simplifie les fonctions avec des valeurs de retour qui ne sont jamais utilisées
  - Mettre les petites fonctions en *Inline*

- Réordonne les déclarations de fonction ; les attributs de fonctions appelées sont connus lorsque l'appelant est optimisé
- Propage les arguments dans le corps des fonctions lorsque tous les appels transmettent la même valeur dans la même position d'argument
- Identifie les caractéristiques des variables au niveau fichier

La mise en pipeline planifie les instructions d'une boucle de sorte que plusieurs itérations de la boucle s'exécutent en parallèle. Aux niveaux d'optimisation *--opt_level = 2* (ou **-O2**) et *--opt_level = 3* (ou **-O3**), le compilateur essaie généralement de pipliner les instructions d'une boucle. L'option *--opt_for_space* affecte également la décision du compilateur sur la mise en pipeline des boucles. En général, la taille du code et les performances sont meilleures lorsqu'on utilise les options *--opt_level = 2* ou *--opt_level = 3*.

Tableau 2.1. L'impact des options d'optimisation sur les performances de calcul de SAD

La valeur du SAD	Niveau d'optimisation	Nombre de cycles requis	
		En langage C/C++	En langage Assembleur linéaire
1726	Sans Optimisation	1174	133
	-O0	640	133
	-O1	637	130
	-O2	297	60
	-O3	87	60

Dans nos expériences, nous avons considéré le niveau *--opt_level = 3* (optimisation maximale possible du compilateur). Pour souligner l'importance et l'impact de chaque option, nous avons effectué des simulations sur un exemple de calcul d'une SAD dont les résultats sont montrés dans le Tableau 2.1. Ce tableau réunit les deux versions de code, en C/C++ et assembleur linéaire. Les résultats de simulation montrent que les performances de l'assembleur linéaire sont toujours meilleures par rapport au langage C/C++ et ceci quel que soit le niveau d'optimisation choisi dans le compilateur. L'insertion de la directive **.trip 8,8,8**, devant une boucle, dans le code assembleur (l'équivalent du *#pragma MUST_ITERATE(8,8,8)* pour le langage C/C++), informe le compilateur que le nombre d'itérations de la boucle est au moins 8, au maximum 8 et

il est aussi un multiple de 8. Cette information permet de réduire le temps d'exécution, dans l'exemple précédent, à 50 cycles seulement.

Les codes C/C++ et d'assembleur linéaire de l'exemple du Tableau 2.1, sont listés ci-dessous :

Programme de test en C/C++

```
#include <stdio.h>
#include <c6x.h>
#include <math.h>
#include <stdint.h>

unsigned short refBuff[8*8];
unsigned short srcBuff[8*8];

unsigned int sad8C(unsigned short  *piOrg, unsigned short iStrideOrg,
unsigned short  *piCur, unsigned short iStrideCur, int iRows, int iCols);
void main()
{
unsigned int sumOfAbsDiff=0;
unsigned int x;
unsigned long long int timer_start=0, timer_stop=0, total;
// Initialise the reference buffer to a known value
for(x=0; x < 8*8; x++)
{
refBuff[x] = x;
}
//also initialise the  source buffer to a known value
for(x=0; x < 8*8; x++)
{
srcBuff[x] = 5;
}
TSCH=0;TSCL=0; // Registers for time measurement

timer_start = _itoll(TSCH,TSCL);
// linear assembly version
// sumOfAbsDiff = sad8(srcBuff,8, refBuff, 8, 8, 8);
// C/C++ version
sumOfAbsDiff = sad8C(srcBuff,8, refBuff, 8, 8, 8);
timer_stop = _itoll(TSCH,TSCL);

total = timer_stop-timer_start;
printf("The sum of the absolute difference is %d", sumOfAbsDiff);
printf("Calculatation time is %lld Cycles ",  total);
}
unsigned int sad8C(unsigned short  *piOrg, unsigned short iStrideOrg,
unsigned short  *piCur, unsigned short iStrideCur, int iRows, int iCols)
{
unsigned int uiSum=0;int n;
for(; iRows != 0; iRows --)
{       for( n = 0; n <iCols; n ++)
uiSum += abs(piOrg[n]-piCur[n]);
piOrg += iStrideOrg;
piCur += iStrideCur;
}

return uiSum;
}
```

Sous-Programme de la fonction SAD en assembleur linéaire

```
.global sad8      ;
sad8: .cproc A_src, B_srcPitch, A_ref, B_refPitch, A_height, B_width
.reg B_src, B_ref, A_srcPitch, A_refPitch
;A side registers
.reg A_srcPix32:A_srcPix10, A_refPix32:A_refPix10
.reg A_absDiff10,A_absDiff32
.reg A_sumOfAbs, A_srcOffset, A_refOffset
.reg A_packSumOfAbsDiff10, A_packSumOfAbsDiff32
.reg A_const1111
;B side registers
.reg B_srcPix32:B_srcPix10, B_refPix32:B_refPix10
.reg B_absDiff10,B_absDiff32
.reg B_sumOfAbs, B_srcOffset, B_refOffset
.reg B_packSumOfAbsDiff10, B_packSumOfAbsDiff32
.reg B_const1111
.reg loopCntX, loopCntY

ADD.2x    A_src,            8,            B_src
ADD.2x    A_ref,            8,            B_ref

SHR.1x   B_srcPitch, 2,     A_srcPitch
SHR.2    B_srcPitch, 2,     B_srcPitch
SHR.1x   B_refPitch, 2,     A_refPitch
SHR.2    B_refPitch, 2,     B_refPitch
ZERO.1              A_sumOfAbs
ZERO.2    B_sumOfAbs

MVKL.1    0x0101,           A_const1111
MVKH.1    0x01010101,       A_const1111
MV.2x     A_const1111,      B_const1111
SUB         A_height,       2,     loopCntY
;loop label used to branch
loopY:     .trip 8
;A side processing
;load the pixel at the current position then increment
```

```
;the pointer by A_srcPitch bytes to setup for the next interation
LDDW.1     *A_src++[A_srcPitch], A_srcPix32:A_srcPix10
LDNDW.1      *A_ref++[A_refPitch], A_refPix32:A_refPix10
;subtract the srcPix loaded with the reference pixel
SUBABS4.1      A_srcPix10,     A_refPix10, A_absDiff10
SUBABS4.1      A_srcPix32,     A_refPix32, A_absDiff32
DOTPU4.1       A_absDiff10,    A_const1111,  A_packSumOfAbsDiff10
DOTPU4.1       A_absDiff32,    A_const1111,  A_packSumOfAbsDiff32
;add the absolute value of the difference to the accumulation
;register on side A
ADD.1      A_sumOfAbs,  A_packSumOfAbsDiff10,    A_sumOfAbs
ADD.1      A_sumOfAbs,  A_packSumOfAbsDiff32,    A_sumOfAbs
;B side processing
LDDW.2     *B_src++[B_srcPitch], B_srcPix32:B_srcPix10
LDNDW.2      *B_ref++[B_refPitch], B_refPix32:B_refPix10
;subtract the srcPix loaded with the reference pixel
SUBABS4.2      B_srcPix10,     B_refPix10, B_absDiff10
SUBABS4.2      B_srcPix32,     B_refPix32, B_absDiff32
DOTPU4.2       B_absDiff10,    B_const1111,  B_packSumOfAbsDiff10
DOTPU4.2       B_absDiff32,    B_const1111,  B_packSumOfAbsDiff32
;add the absolute value of the difference to the accumulation
;register on side A
ADD.2      B_sumOfAbs,  B_packSumOfAbsDiff10,    B_sumOfAbs
ADD.2      B_sumOfAbs,  B_packSumOfAbsDiff32,    B_sumOfAbs
BDEC       loopY,                              loopCntY
;here the A side and B side counters are added to produce
;the correct result
ADD.1x     A_sumOfAbs,          B_sumOfAbs,     A_sumOfAbs
.return    A_sumOfAbs

.endproc
```

L'analyse du listing assembleur généré par le compilateur pour le code C/C++, montre que le chargement (l'opération la plus pénible nécessitant 5 cycles) de données est maintenu pixel par pixel via l'instruction **LDH** (*Load Half-word*). Par contre, pour le code assembleur, nous avons déroulé la boucle interne puisque nous avons utilisé deux chargements (sur les deux chemins de données A et B) de mots doubles grâce

aux instructions **LDDW** et **LDNDW** (c-à-d deux fois 64 bits qui représentes 8 pixels de 16 bits, à la fois).

2.5.2 Optimisation des modules SADs

La métrique SAD est utilisée pour exprimer la distorsion entre le bloc d'origine et le bloc actuel (prédit), afin de trouver le meilleur vecteur de mouvement d'inter-prédiction.

En C/C ++, le calcul SAD est donné par :

```
for(; iRows != 0; iRows --)
{       for(Int n = 0; n <iCols; n ++)
```

Tableau 2.2. Les instructions SIMD utilisées pour le calcul du SAD

Instruction	Description
LDW/LDNW *src, dst	Effectue un chargement de données alignées/non alignées sur 32 bits, ce qui permet de charger deux pixels à la fois de 16 bits chacun.
LDDW/LDNDW	Effectue un chargement de données alignées/non alignées sur 64 bits, ce qui permet le chargement de quatre pixels simultanément.
SUB2 src1, src2, dst	Les moitiés supérieure et inférieure de **src2** sont soustraites des moitiés supérieure et inférieure de **src1** et le résultat est placé dans **dst**. De cette manière, il est possible de soustraire simultanément deux pixels de deux autres pixels.
ABS2 src, dst	Les valeurs absolues des moitiés supérieure et inférieure de l'opérande **src** sont placées dans les moitiés supérieure et inférieure du **dst**. Cette instruction est placée après **SUB2** pour évaluer la différence absolue de pixels entre une image originale et une image courante.
SUBABS4 src1, src2, dst	Calcule la valeur absolue des différences entre les données 8 bits compactées non signées et contenues dans les registres source. Le résultat est écrit dans dst dans un format 8 bits non signé et compressé. Cette instruction est très utile pour les vidéos avec une taille de pixels de 8 bits.
DOTP2 src1, src2, dst	Renvoie le produit scalaire entre deux paires de valeurs 16 bits signées et compactées. Les valeurs dans **src1** et **src2** sont traitées comme des quantités 16 bits compactées et signées. Avec **0x10001** comme second opérande, cette instruction effectue l'accumulation de différences absolues.
DOTPU4 src1, src2, dst	Pour chaque paire de quantités de 8 bits dans **src1** et **src2**, la valeur de 8 bits non signée de **src1** est multipliée par la valeur de 8 bits non signée de **src2**. Les quatre produits sont additionnés et le produit scalaire résultant est écrit sous forme de résultat 32 bits en **dst**.

```
                uiSum + = abs (piOrg [n] - piCur [n]);
        piOrg + = iStrideOrg;
        piCur + = iStrideCur;
}
```

où **iRows** et **iCols** sont respectivement la hauteur et la largeur du bloc, **iStrideOrg** et **iStrideCur** sont les pas d'incrémentation dans les images d'origine et actuelle.

Dans le software de référence, il y a onze sous-versions de SAD en fonction de la taille des partitions CUs considérées. Cette diversité est maintenue dans l'optimisation proposée. En général, la taille des pixels est de 16 bits, ainsi l'instruction de chargement dûment utilisée est **LDH** , néanmoins, la solution proposée effectue plusieurs chargements (pixels) à la fois. Le jeu d'instruction du C66× comprend les instructions **LDW** et **LDDW** pour charger 32 et 64 bits (deux ou quatre pixels alignés pour l'image originale) ou leurs versions non alignées **LDNW** et **LDNDW** (pour l'image actuelle). De cette manière, il est possible d'avoir une réduction de la redondance et du nombre d'itérations dans les boucles.

Après avoir chargé les pixels, la soustraction et la valeur absolue peuvent être évaluées via **SUB2** et **ABS2**, ou **SUBABS4** lorsque la taille des pixels est fixée à 8. L'accumulation des différences absolues est calculée ultérieurement à l'aide de **DOTP2** ou **DOTPU4**.

Le Tableau 2.2 présente les différentes instructions SIMD utilisées dans le processus de calcul SAD. Il est a noter que dans certains cas, le déroulement des boucles est la meilleure solution pour réduire le temps d'exécution.

Un exemple de calcul est illustré sur la Figure 2.5, où la fonction *xGetSad8* est remplacée par un code assembleur équivalent.

2.5.3 Optimisation du module SSE

Le calcul du SSE est réalisé dans une même manière que le calcul SAD. La seule différence est le remplacement de la fonction valeur absolue par le calcul du carré. En général, le calcul SSE peut être effectué à l'aide du pseudo-code suivant :

```
for( ; iRows != 0; iRows-- )
{ for (Int n = 0; n <iCols; n++ )
        { iTemp = piOrg[n ] - piCur[n ];
                uiSum += iTemp * iTemp ;
        }
        piOrg += iStrideOrg;
        piCur += iStrideCur;
}
```

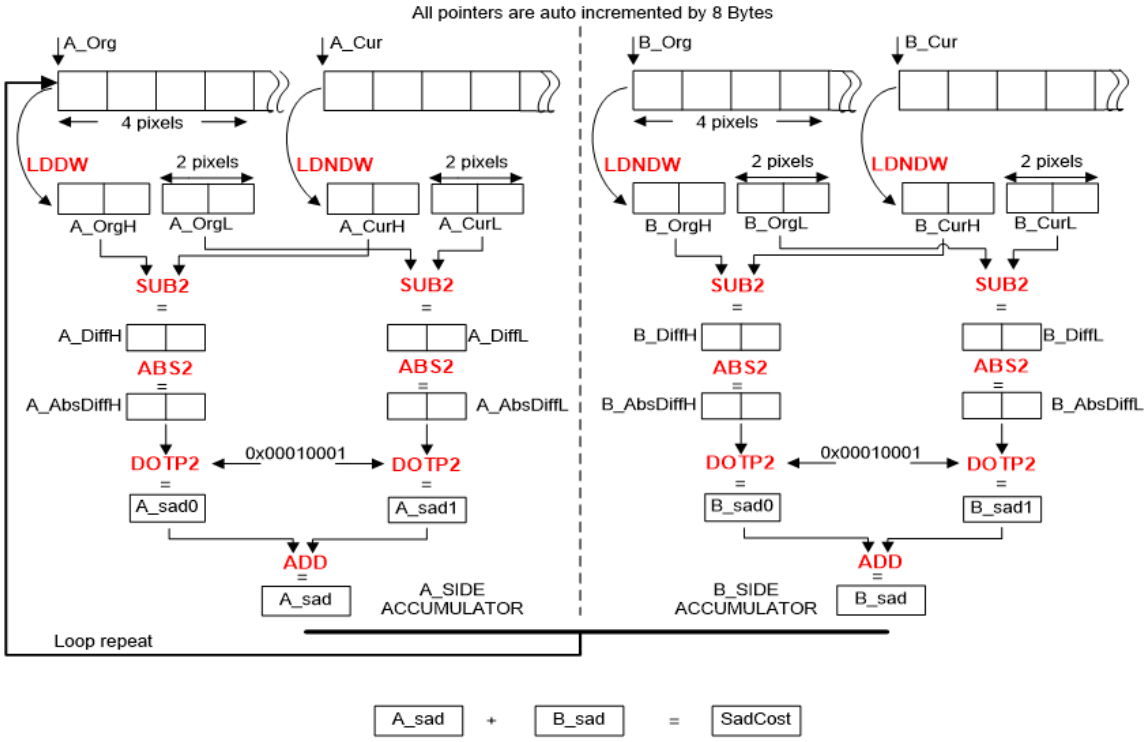


Figure 2.5. Exemple de calcul dans un module SAD (xGetSad8).

où **iRows** et **iCols** sont respectivement la hauteur et la largeur du bloc, **iStrideOrg** et **iStrideCur** sont les pas d'incrémentation des images d'origine et courante. Outre les instructions **LDDW/LDNDW** et **SUB2** précédemment utilisées, les instructions SIMD **DSUB2, DADD, MPY2** et **DMPY2** sont introduites dans le calcul du SSE. Ces instructions sont présentées en détail au Tableau 2.3. La Figure 2.6 illustre un exemple de calcul SSE remplaçant la fonction xGetSSE dans la classe TComRdCost.

2.5.4 Optimisation du Module HAD

La transformée de Walsh-Hadamard est définie par[40] :

$$\begin{cases} X = \frac{1}{N} H_h \cdot x \text{ Transformée directe} \\ x = H_h \cdot X \text{ Transformée inverse} \end{cases} \tag{2.1}$$

Ce sont les paires de transformée Walsh-Hadamard WHT_h directe et inverse, et qui sont identiques.

Soit $\mathbf{x} = [x[0], x[1], \cdots, x[N-1]]^T$ et $\mathbf{X} = [X[0], X[1], \cdots, X[N-1]]^T$ sont les vecteurs signal et spectre, respectivement. L'élément k de la transformée WH peut

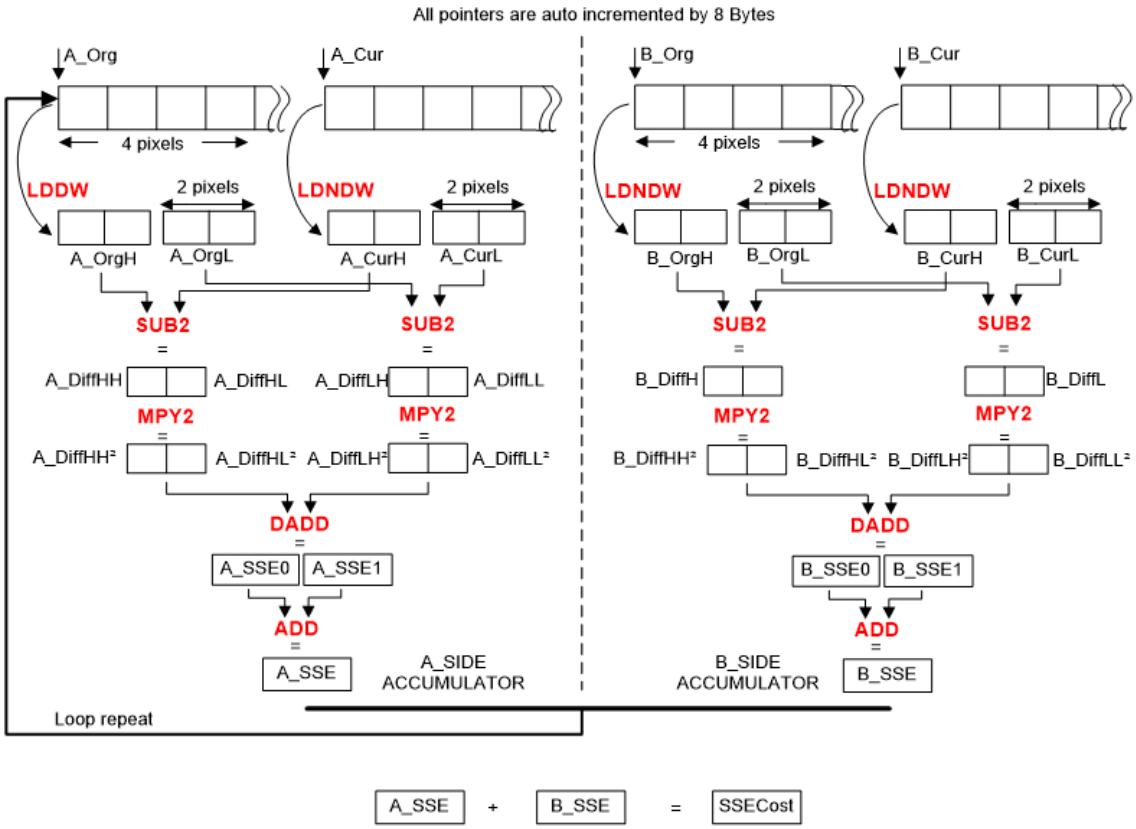


Figure 2.6. Exemple de calcul dans le module SSE (xGetSSE).

également être écrit sous la forme :

$$X[k] = \sum_{m=0}^{N-1} h[k,m]x[m] = \sum_{m=0}^{N-1} x[m] \prod_{i=0}^{n-1} (-1)^{m_i k_i} \tag{2.2}$$

où

$$k = \sum_{i=0}^{n-1} k_i 2^i = (k_{n-1}k_{n-2}\cdots k_1 k_0)_2 \quad (k_i = 0, 1)$$

$$m = \sum_{i=0}^{n-1} m_i 2^i = (m_{n-1}m_{n-2}\cdots m_1 m_0)_2 \quad (m_i = 0, 1)$$

i.e., $(k_{n-1}k_{n-2}\cdots k_1 k_0)_2$ and $(m_{n-1}m_{n-2}\cdots m_1 m_0)_2$ sont les représentations binaires de k et m, respectivement. Évidemment, $n = log_2 N$.

La complexité de WHT est $O(N^2)$. Semblable à l'algorithme FFT, nous pouvons dériver un algorithme WHT rapide avec une complexité de $O(Nlog_2 N)$ suivant le schéma de la Figure 2.7.

La transformée de Hadamard est utilisée pour améliorer la qualité de prédiction.

Tableau 2.3. Les instructions SIMD supplémentaires utilisées pour le calcul du SSE

Instruction	Description
MPY2 src1, src2, dst_o :dst_e	Le produit des demi-mots inférieurs de **src1** et **src2** est écrit dans le registre de destination pair, **dst_e**. Le produit des demi-mots supérieurs de **src1** et **src2** est écrit dans le registre de destination impaire, **dst_o**
DMPY2 src1, src2, dst	L'instruction **DMPY2** effectue quatre multiplications de 16 bits parmi des quantités de 16 bits emballées signées. Les valeurs dans **src1** et **src2** sont traitées comme des quantités 16 bits emballées signées. Les résultats 32 bits sont placés dans un registre quadruple de 128 bits
DSUB2 src1, src2, dst	L'instruction **DSUB2** effectue quatre soustractions de 16 bits des nombres de 16 bits contenus dans les deux registres sources de 64 bits. Les résultats de la soustraction sont renvoyés sous la forme de quatre résultats 16 bits contenus dans un registre **dst** de 64 bits.
DADD src1, src2, dst	L'instruction **DADD** effectue deux additions sur 32 bits des nombres à 32 bits contenus dans les deux paires de registres sources. Les résultats de l'addition sont renvoyés sous forme de deux résultats 32 bits réunis dans un mot double (de 64 bits) dans **dst**

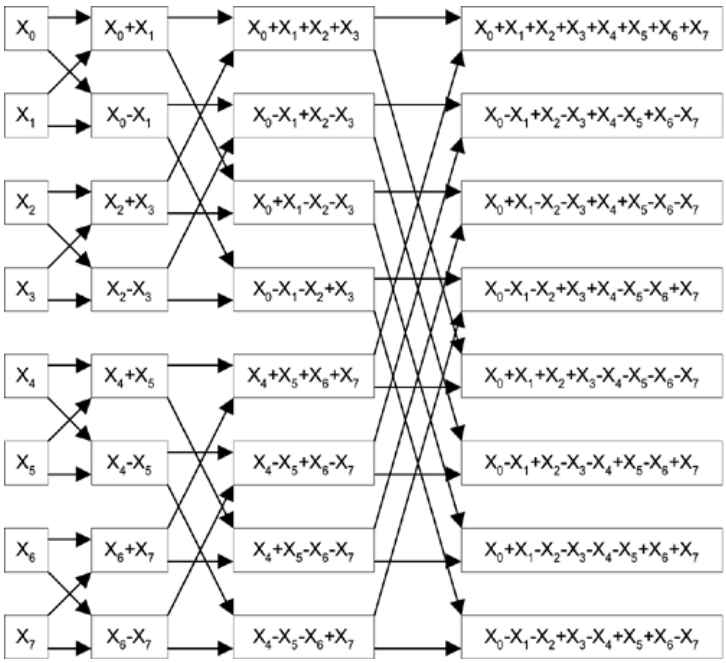


Figure 2.7. L'algorithme de calcul dans la transformée Hadamard à huit points.

Dans le software de référence, la version optimisée de la Transformée Hadamard basée sur l'algorithme de la Figure 2.7, est utilisée. L'optimisation proposée dans cette thèse consiste à remplacer plusieurs instructions du module Hadamard par une seule SIMD. Les instructions SIMDs clés utilisées dans le module HAD sont les suivantes :

Tableau 2.4. Les instructions SIMD utilisées pour le calcul du HAD

Instruction	Description
ADDSUB src1, src2, dst_o :dst_e	**src2** est ajouté à **src1**. Le résultat est placé dans **dst_o**. En parallèle, **src2** est soustrait de **src1**. Le résultat est placé dans **dst_e**.
ADDSUB2 src1, src2,dst_o :dst_e	Pour l'opération **ADD2**, les moitiés supérieure et inférieure de l'opérande **src2** sont ajoutées aux moitiés supérieure et inférieure de l'opérande **src1** et les résultats sont écrits au format 16 bits signé et compacté dans **dst_o**. Pour l'opération **SUB2**, les moitiés supérieure et inférieure de l'opérande **src2** sont soustraites des moitiés supérieure et inférieure de l'opérande **src1** et les résultats sont écrits au format 16 bits signé et compressé dans **dst_e**.
DADD2 src1, src2, dst	L'instruction **DADD2** effectue quatre additions à 16 bits des nombres à 16 bits contenus dans les deux registres source de 64 bits. Les résultats de l'addition sont renvoyés sous la forme de quatre résultats 16 bits regroupés dans **dst**.
PACK2 src1, src2, dst	Déplace les demi-mots inférieurs de **src1** et **src2** et les place tous les deux dans dst. Le demi-mot inférieur de **src1** est placé dans le demi-mot supérieur de **dst**.
DPACK2 src1, src2, dst_o :dst_e	Cette instruction prend le demi-mot inférieur de **src1** et le demi-mot inférieur de **src2**, et les regroupe tous les deux dans **dst_e**. Le demi-mot inférieur de **src1** est placé dans le demi-mot supérieur de **dst_e**. Le demi-mot supérieur de **src1** et le demi-mot supérieur de **src2** sont regroupés dans **dst_o**. Le demi-mot supérieur de **src1** est placé dans le demi-mot supérieur de **dst_o**.

- DADD2/DSUB2
- ADDSUB/ADDSUB2
- PACK2/DPACK2.

Le Tableau 2.4 présente l'utilisation de ces instructions pour réduire le nombre d'instructions et, par conséquent, le temps d'exécution.

La Figure 2.7 montre que seules des additions et des soustractions sont nécessaires pour effectuer la transformation de Hadamard. Les entrées $(x_0, x_1, \ldots, x_7)$ de l'algorithme sont les différences entre les blocs prédits et les blocs d'origine. L'instruction **ADDSUB2** est appliquée aux paires d'entrée pour effectuer des opérations **ADD2** et **SUB2** en parallèle sur les entrées communes.

2.5.5 Résultats expérimentaux et discussion

Pour l'ensemble des tests présentés dans cette partie, nous avons retenu un niveau d'optimisation fixé à **-O3** pour la plupart des fichiers source, sauf pour certains fichiers où l'optimisation **-O3** génère des erreurs d'exécution. Pour la segmentation de la mémoire, nous avons configurés une plateforme avec les spécifications résumées dans le Tableau 2.5.

Tableau 2.5. Segmentation et usage de la mémoire

Type de mémoire		Adresse de début	Capacité	Usage
Mémoire interne	MSMCSRAM[a]	0x0C000000	0x00400000	Code
	L1DSRAM[b]	0x00F00000	0x00000000	-
	L1PSRAM[c]	0x00E00000	0x00000000	-
	L2SRAM	0x00800000	0x00080000	Pile
Mémoire externe	DDR3	0x80000000	0x20000000	Données

[a]**MSMCSRAM** : *Multicore Shared Memory Controller Static RAM*,
[b]**L1DSRAM** : *Level-1 Data SRAM*, [c]**L1PSRAM** : *Level-1 Program SRAM*

L'idéal est de placer les différentes variables et structures de données dans la mémoire interne dont le temps d'accès est minimal, mais ceci est impossible dans notre cas vue les exigences élevées en ressources mémoire. Par conséquent, nous avons utilisé la mémoire externe **DDR3** de taille 512 Mo.

Avant d'entamer l'optimisation de l'encodeur HEVC, il est vivement recommandé de localiser les zones critiques, les plus exigeantes en temps de calcul dans le software de référence. Le profil principal du HM 12.1 a été utilisé pour l'analyse de sa complexité, en considérant la configuration à accès aléatoire (RA). Un ensemble de séquences de test de classe D (416x240), de classe C (832x480) et de classe E (1280x720) est utilisé.

La Figure 2.8 illustre le temps moyen d'exécution des modules les plus importants dans HM 12.1. A partir de cette figure, la pertinence de la classe TComRdCost qui inclut les calculs SAD , SSE et HAD, est mise en évidence.

Afin de souligner les avantages apportés par les optimisations proposées, nous présentons à la fois les résultats initiaux de la version de base C/C++ avec activation des options d'optimisation dans CCS, et la version optimisée manuellement. Le gain en temps d'exécution pour le calcul des fonctions SAD, HAD et SSE est présenté dans le Tableau 2.6.

La version proposée et optimisée manuellement à l'aide des instructions SIMD, permet d'économiser 61.5% en temps moyen de calcul du module SAD et peut atteindre

Tableau 2.6. Le gain en temps d'exécution pour le calcul du SAD, SSE et HAD

Taille du bloc	SAD			SSE			HAD		
	l'implémentation de Base (Cycles)	l'implémentation proposée (Cycles)	Gain (%)	l'implémentation de Base (Cycles)	l'implémentation proposée (Cycles)	Gain (%)	l'implémentation de Base (Cycles)	l'implémentation proposée (Cycles)	Gain (%)
4×4	179	77	56.98	209	143	31.58	276	252	08.70
8×4	319	117	63.32	373	166	55.50	460	405	11.96
4×8	227	85	62.56	241	123	48.96	401	345	13.97
8×8	415	127	69.40	437	195	55.38	918	646	29.63
16×8	791	211	73.32	829	339	59.11	1613	1143	2914
8×16	651	206	68.36	565	245	56.64	1505	1003	33.36
16×16	1259	382	69.66	1085	413	61.94	2785	1853	33.46
32×16	2475	734	70.34	2125	749	64.75	5343	3549	33.58
16×32	2027	1034	48.99	1597	560	64.93	5317	3273	38.44
32×32	4011	1978	50.69	3149	1008	67.99	10407	6389	38.61
64×32	7979	3866	51.55	6253	1904	69.55	20587	12621	38.69
32×64	7403	3198	56.80	5552	1898	65.81	19779	12549	36.55
64×64	14763	6238	57.75	11045	3658	66.88	39307	24941	36.55
Moyenne	-	-	61.52	-	-	59.16	-	-	29.43

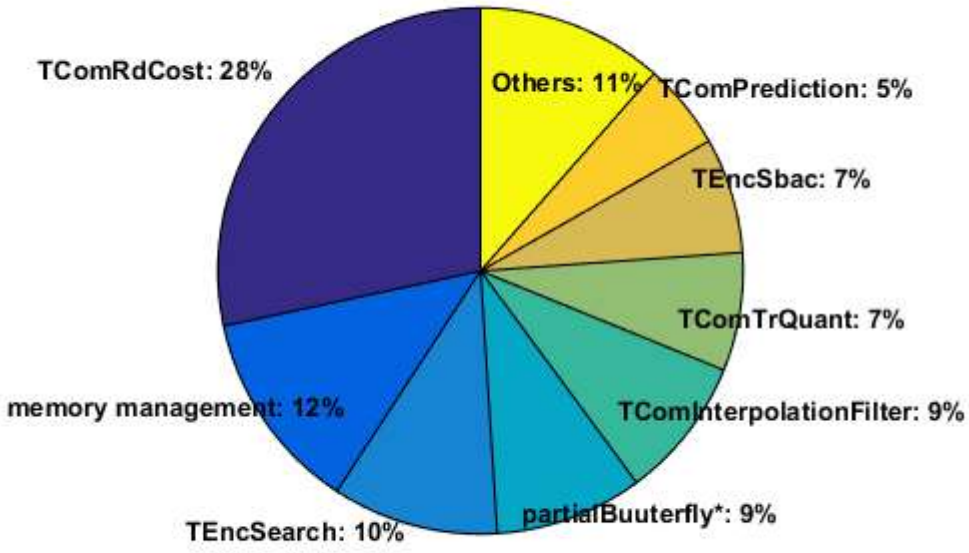


Figure 2.8. Temps d'encodage HEVC pour la configuration RA.

73.3% pour certaines tailles de bloc. L'optimisation SSE proposée permet quand à elle, d'économiser environ 59% du temps moyen de calcul par rapport à la version C/C++ du SSE avec un maximum atteignant 69.5%.

En ce qui concerne la fonction HAD, la version proposée du SIMD permet d'économiser environ 29% du temps moyen de calcul par rapport à la version C/C++ avec un maximum de 38.7%. Par conséquent, les versions SIMD proposées de SAD, SSE et HAD sont intégrées dans l'encodeur HEVC complet pour évaluer leur performance globale. Les résultats sont présentés dans le Tableau 2.7 pour différentes séquences vidéo et sous différentes résolutions, en utilisant les paramètres de codage détaillés dans le Tableau 2.8. Une configuration de codage rapide est également disponible en activant les options ASR (*Adaptive motion Search Range algorithm*) [41], ECU (*Early CU termination algorithm*), CFM (*Coded bloc flag Fast Mode algorithm*) et ESD (*Early Skip Detection algorithm*) dans la configuration de l'encodeur HEVC. Les expériences suivantes ont été effectuées :

– Test 1 : En tant que test de base, l'implémentation de HM 12.1 avec pipeline et optimisation au niveau 3 sous CCS, sont activés.

Tableau 2.7. Mesure des performances des algorithmes proposés

Séquences		Frame rate	Test 2 (Sans pertes)	Test 3		
			ATS (%)	BR (%)	PSNR (dB)	ATS (%)
Classe C 832×480	BasketballDrill	50	08.70	3.8932	-0.1234	48.00
	RaceHorses	30	10.98	4.0077	-0.1592	36.86
	PartyScene	50	10.06	2.1984	-0.0918	41.98
	BQMall	60	11.67	2.9769	-0.1311	54.28
Moyenne C		-	10.35	3.2691	-0.1264	45.28
Classe D 416×240	BasketballPass	50	13.20	3.6582	-0.1678	63.90
	RaceHorses	30	10.53	4.1906	-0.1701	35.72
	BQSquare	60	11.65	1.6734	-0.0766	58.50
	BlowingBubbles	50	10.95	1.3706	-0.0435	46.12
Moyenne D		-	11.58	2.7007	-0.1145	51.06
Classe E 1280×720	Vidyo1	60	14.28	1.2696	-0.0425	78.69
	Vidyo3	60	15.27	1.6654	-0.0534	72.73
	Vidyo4	60	14.66	0.9768	-0.0259	74.94
Moyenne E		-	14.74	1.3039	-0.0406	75.45
Classe B 1920×1080	ParkScene	24	06.34	0.9857	-0.0298	35.67
	BQTerrace	60	03.97	-1.0223	0.0346	28.71
	Cactus	50	05.09	1.2447	-0.0305	31.09
	Kimono1	24	05.57	1.6076	-0.0489	2446
Moyenne B		-	05.24	0.7039	-0.0186	29.98
Classe A 2560×1600	PeopleOnStreet	30	05.19	1.8973	-0.0865	21.14
	Traffic	30	05.59	0.3412	-0.0084	37.95
Moyenne A		-	05.39	1.1193	-0.0475	29.55

– Test 2 : Dans ce test, un ensemble de techniques d'optimisation est impliqué, notamment l'utilisation de #pragma, d'instructions intrinsèques (substitution des fonctions abs, max et min) et le remplacement des fonctions de calcul de distorsion dans la classe TComRdCost par leurs versions d'assemblage correspondantes.
– Test 3 : Outre les implications du test 2, la configuration du codage rapide est considérée dans ce test.

Afin d'évaluer les performances des méthodes proposées, nous avons utilisé les quatre valeurs de QP, Tableau 2.8, pour la mesure de distorsion de Bjøntegaard Delta BD-PSNR (en dB) et du débit BD-Rate(en %). Pour le gain en temps moyen, ATS

Tableau 2.8. Paramètres d'encodage considérés dans les tests

Paramètre	Valeur
Profile	Random Access
Video resolution	Class : A,B, C, D and E
Number of encoded frames	100
QP (Quantification parameter)	22, 27, 32, 37
Number of reference frames	4
Motion Estimation Algorithm	TZ search
GOP size	8
Maximum coding unit depth	4
Search range	64
Internal Bit Depth	8

(en %), nous avons utilisé l'expression :

$$ATS(\%) = \frac{T_{HM} - T_{Optimal}}{T_{HM}} \tag{2.3}$$

où T_{HM} est le temps de codage initial (implémentation de base du HM), et $T_{Optimal}$ est le temps d'encodage pour l'optimisation proposée.

Les résultats expérimentaux présentés dans le Tableau 2.7, montrent que la méthode proposée permet un gain de temps allant de 8,70% à 15,27% sans perte de performances (en terme de qualité et débit), et un gain de temps de 35,72% à 78,69% avec seulement 4% d'incrément en bitrate total et 0,17dB de perte dans le PSNR. Les meilleurs résultats sont obtenus pour des séquences de la classe E (1280×720).

2.6 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation et l'optimisation de l'encodeur HEVC sur une plateforme à base d'un DSP de la famille TMS320C6678. La configuration *Random Access* a été utilisée pour évaluer les performances des différentes optimisations proposées, sur un ensemble de séquences vidéo homologué par la norme HEVC.

Nous avons commencé par la description de la plateforme cible et ses caractéristiques principales. Nous sommes passés par la suite, à l'adaptation du software de référence en vue d'une implémentation sur DSP. Certains changements devraient être apportés sur le software de référence HM, pour aboutir à un code

fonctionnel. Le profilage du software de référence nous a permis de localiser les parties lourdes en temps de calcul, qui devraient être optimisées.

Notre contribution a consisté en la réécriture de certaines parties du code en langage assembleur linéaire, ce qui a permis de tirer avantage des instructions SIMD. Ce type d'instructions offre la possibilité de manipuler plusieurs données (pixels) à l'aide d'une seule instruction, et par conséquent, ceci a conduit à une réduction du temps d'encodage sans altération des performances d'encodage (que ce soit en termes de PSNR ou de débit). Nos expériences ont inclus aussi, certaines optimisations algorithmiques disponibles sous forme d'options à activer dans le fichier de configuration associé à l'encodage.

Les résultats numériques dénoncent un gain maximal de 78% en combinant l'optimisation proposée avec l'optimisation algorithmique. Néanmoins, ces résultats restent insuffisants et loin du fonctionnement en temps réel. Ce fait nous a incité à réfléchir à d'autres solutions éventuelles de type algorithmique ou exploitant le parallélisme offert par le processeur multi-cœurs.

Dans le chapitre suivant, nous présentons une optimisation algorithmique pour la réduction du temps d'encodage HEVC sous la configuration *All-Intra*. Cette optimisation concerne le mécanisme de partitionnement dans la norme HEVC.

Chapitre 3

Optimisation du processus de partitionnement des unités de codage

Sommaire

3.1 Introduction **75**
3.2 Travaux connexes **76**
3.3 Optimisation du processus de partitionnement **78**
3.3.1 Processus de partitionnement hiérarchique des unités de codage (CUs) dans la norme HEVC 78
3.3.2 Description de l'algorithme rapide proposé pour le partitionnement des CUs 81
3.4 Résultats et discussion **86**
3.4.1 Conditions expérimentales 86
3.4.2 Critères d'évaluation 86
3.4.3 Résultats des algorithmes proposés 87
3.5 Conclusion **92**

3.1 Introduction

L'un des principaux contributeurs à des performances plus élevées de compression dans la norme HEVC, est l'introduction de plus grandes unités de codage (CU) avec des mécanismes de partitionnement récursifs. L'objectif principal du partitionnement récursif est l'adaptation des tailles des unités de traitement au contenu de l'image à encoder, ce qui permet de réduire le flux binaire du signal encodé sans altération significative de la qualité. En contre partie, ces performances de codage s'accompagnent

d'une complexité de calcul très élevée rendant très difficile, la réalisation d'un codage en temps réel. Notamment, si l'implémentation envisagée vise une plateforme embarquée.

Dans ce contexte, est proposé dans ce chapitre, un algorithme de partitionnement rapide des unités CUs pour la configuration tous-intra (All-Intra). La proposition consiste à anticiper l'arrêt de partitionnement d'une CU pour les régions homogènes, ou à ignorer la prédiction à certaines profondeurs, pour les régions riches en texture. La décision de Split/Non-split est basée sur un algorithme de classification d'homogénéité, ce qui nous permet d'éviter le test de toutes les profondeurs, ou toutes les tailles possibles pour une CU), afin d'en retenir la meilleure taille en cours de traitement. L'algorithme de classification est choisi le plus simple possible, afin de ne pas introduire un temps supplémentaire d'analyse.

3.2 Travaux connexes

Récemment, plusieurs travaux ont été consacrés aux optimisations du partitionnement des CUs dans les encodeurs HEVC. La structure de partitionnement dans la norme HEVC a un impact direct sur les performances d'encodage. En effet, grâce aux différentes dimensions de CU, PU et TU, la norme HEVC réussit un meilleur rapport qualité/débit. Cependant, le temps de codage augmente en fonction du nombre de tailles possibles de CU, raison pour laquelle les chercheurs et les développeurs s'attardent sur ce point en vue d'optimiser l'encodeur HEVC.

Divers algorithmes sont proposés pour la décision précoce de la taille d'une partition CU. Certains auteurs [42] ont proposé une méthodologie intéressante pour accélérer le processus de partitionnement d'une CU dans le codage intra HEVC, à l'aide du data mining pour l'apprentissage d'un classifieur hors ligne. Leur algorithme utilise la moyenne et la variance des CUs et des sous-CUs du voisinage, en tant qu'attributs pour la décision Split/Non-split. Cet algorithme peut réduire le temps d'encodage jusqu'à 52% par rapport au software de référence HM16.6, tandis que l'augmentation en débit binaire reste inférieure à 2%.

Une autre solution décrite dans [43], vise à anticiper l'arrêt du processus de partitionnement des unités CUs à l'aide de la règle de décision Bayésienne, en utilisant un apprentissage commun en ligne et hors ligne. Un problème à deux classes a été

envisagé pour classer les CUs, en classes de partitionnement et non-partitionnement. Cet algorithme prend en charge toutes les configurations de codage, y compris l'accès aléatoire (RA), le faible délai (LD) et tous-intra (AI).

Les auteurs de la référence [44] ont proposé un algorithme rapide pour la prédiction intra, utilisant la corrélation entre les unités de codage et les unités de prédiction voisines. Dans ce cas, la décision Split/Non-split est conforme à la texture de la CU en cours de traitement et à la taille de la CU voisine. Un choix de mode de prédiction rapide est conclu, en utilisant la corrélation entre le mode à une profondeur supplémentaire et les modes de PUs voisins. Dans un autre travail [45], une analyse statistique basée sur l'opérateur de variance a été adoptée, afin de constituer une base de données à seuil hors ligne, utilisée pour prendre la décision Split/Non-split. Ce qui permet d'ignorer les modes les moins probables. L'algorithme proposé était dédié aux vidéos Ultra Haute Définition.

Dans un article récent [46], la complexité de l'encodeur HEVC pour la configuration intra, est réduite grâce à une décision rapide sur le choix du mode intra et la taille d'unité de prédiction. Pour éviter de la chercher parmi les 35 modes de prédiction, l'algorithme proposé [46] est basé sur l'utilisation des gradients moyens dans les directions, horizontale et verticale (AGH, AGV), afin de limiter les modes candidats. En outre, la décision Split/Non-split anticipée des CUs repose sur certaines caractéristiques, telles que la différence de profondeur et le rapport de coût HADAMARD, entre la CU actuelle et les CUs adjacentes.

Les auteurs dans [47] ont proposé un algorithme de réduction de la complexité de la prédiction intra HEVC, en utilisant une décision rapide sur le mode de prédiction, basée sur la distribution de l'assentiment du bord dominant. La décision sur le bord dominant est prise conformément à l'évaluation du contour directionnel (DEA) minimal, compte tenu des quatre directions de 0, 45, 90 et 135 degrés.

L'algorithme proposé dans [48] vise à réduire la complexité du codage intra en fonction des informations de contour. Une étape de pré-traitement est introduite pour estimer l'amplitude et la direction des contours, en vue d'une utilisation ultérieure dans la décision de partitionnement des unités centrales. Les informations sur les contours sont évaluées à l'aide de deux noyaux de convolution Sobel, d'une taille de 3x3 pixels. Dans une région homogène, la taille de l'unité de codage est éventuellement

sélectionnée comme étant la plus grande (64x64). En outre, les modes de prédiction directionnelle sont exclus et seuls le mode de prédiction DC et le mode de prédiction Planar sont conservés pour déduire le mode de prédiction optimal. Inversement, une région complexe est prédite avec une taille de PU inférieure.

Les auteurs dans [49] ont également proposé un schéma pour le partitionnement rapide des CUs, utilisant l'opérateur Sobel en codage intra. Tout d'abord, l'amplitude du vecteur de gradient est calculée pour tous les pixels de la CU actuelle, ensuite un choix basé sur les valeurs seuillées de cette amplitude est effectué afin de décider si la CU comprend des contours ou non. S'il n'y a pas de contour dans l'unité CU, le bloc est étiqueté comme une zone lisse et une fin anticipée de partitionnement de la CU traitée est lancée.

Dans [50], la corrélation entre le contour local et le mode de prédiction intra est exploitée pour réduire la complexité de l'intra encodage. L'opérateur de Sobel est utilisé pour détecter les quatre directions du contour principal (vertical, horizontal, 45 degrés et 135 degrés). Cette information est utilisée pour présélectionner le mode de prédiction parmi les 33 modes directionnels. En outre, les auteurs ont proposé de fusionner des sous-partitions avec la même direction de contour, afin d'éviter les processus redondants de partitionnement.

Plusieurs auteurs [51–53] ont proposé divers algorithmes visant à accélérer l'inter-prédiction HEVC. Leurs méthodes bénéficient de la propriété de corrélation spatiale et/ou temporelle, entre les images successives dans une séquence vidéo.

3.3 Optimisation du processus de partitionnement

3.3.1 Processus de partitionnement hiérarchique des unités de codage (CUs) dans la norme HEVC

Les images vidéo sont généralement échantillonnées progressivement avec des tailles d'image rectangulaires en échantillons de luminance $W \times H$, où W est la largeur et H la hauteur de l'image. Chaque matrice de composantes de chrominance avec un échantillonnage $4:2:0$, est alors de taille $\frac{W}{2} \times \frac{H}{2}$. Avec un tel signal vidéo, l'encodeur HEVC partitionne chaque image en un ensemble complet de blocs non chevauchés.

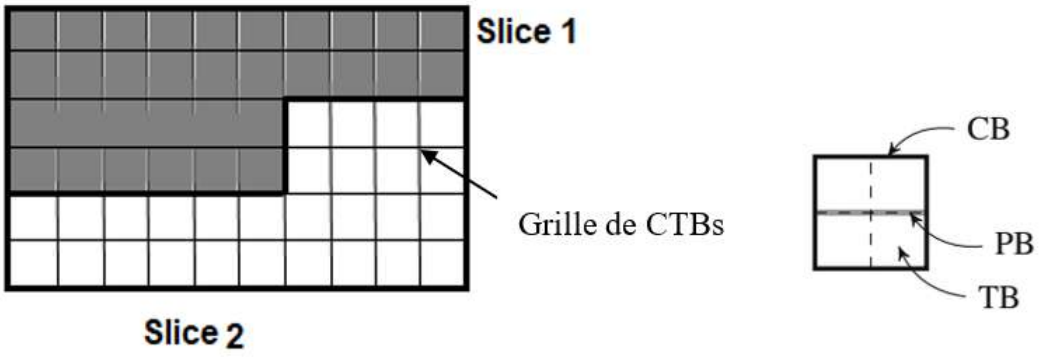


Figure 3.1. Exemple de partitionnement d'une image en Slices et CTBs, et d'un bloc CB en PB et TB

La Figure 3.1 illustre un exemple de partitionnement de base d'une image. Dans les spécifications des normes de codage vidéo précédentes, la structure de bloc de base était le macro-bloc de 16×16 de luminance et les échantillons de chrominance

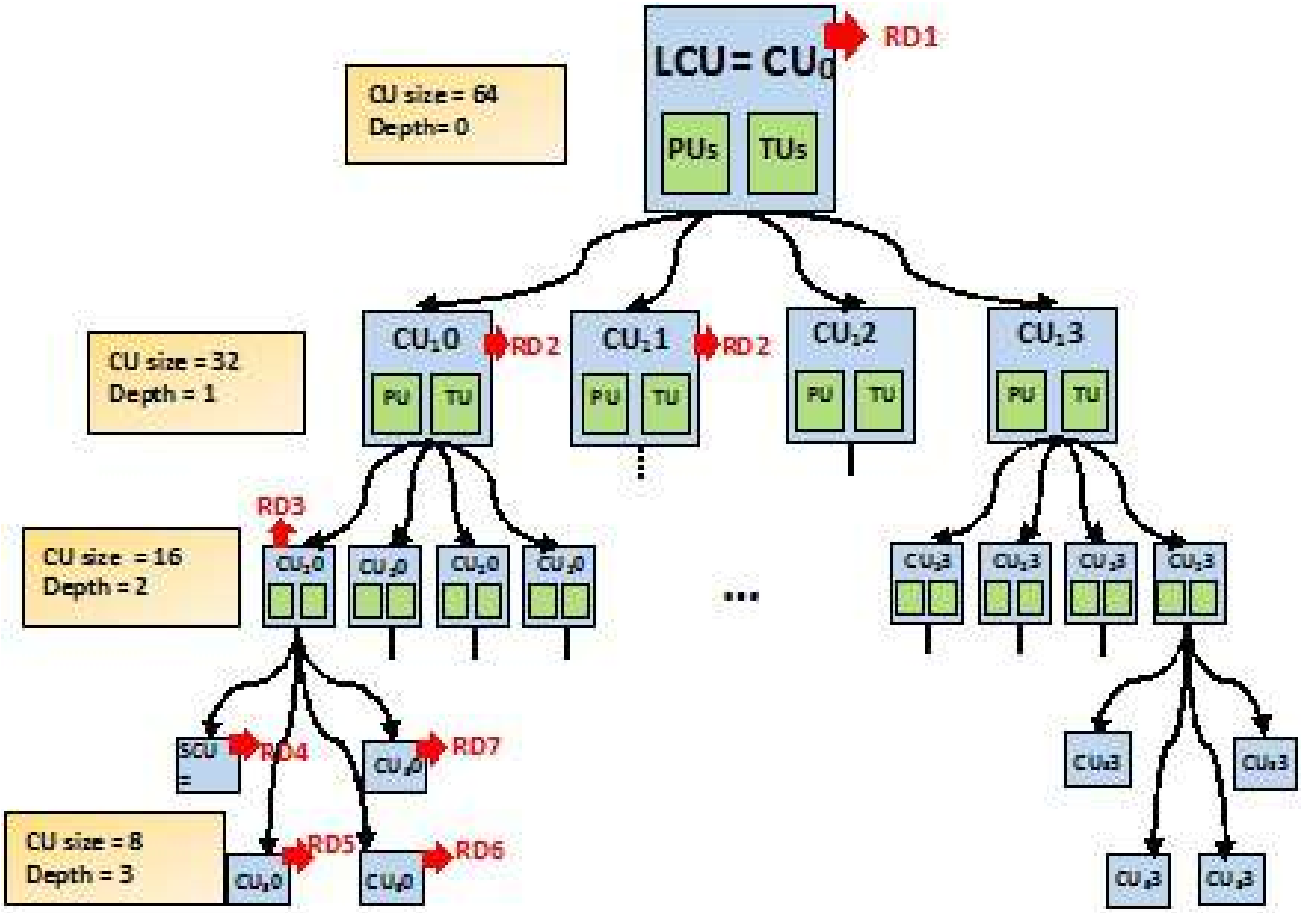


Figure 3.2. Exemple structurel du concept quad-tree en HEVC avec CU = 64×64 et profondeur = 3.

correspondants. En tant que généralisation du concept de macro-bloc, le bloc d'arbre de codage (CTB) est introduit dans HEVC. Il peut être divisé en plusieurs blocs de codage (CB). Pour la prédiction, chaque bloc de codage est partitionné en un ensemble d'un ou plusieurs blocs de prédiction (PB). En parallèle, le bloc de codage peut être partitionné en blocs de transformation (TB). L'image peut en outre être partitionnée en une ou plusieurs Slices, composées d'un nombre entier de CTB et pouvant être décodées indépendamment [11].

L'unité de codage est caractérisée par sa taille et sa profondeur. Le choix de la profondeur dans chaque CTU passe par un processus de décision basé sur le calcul du coût, noté RDcost, de chaque partition CU à l'intérieur de la CTU. La structure récursive est représentée au moyen d'une séquence d'indicateurs appelée drapeau de partitionnement (*Split flag*), qui détermine la décision de partitionnement de l'unité CU.

Dans la plus grande CTU de taille 64×64, appelée également LCU (la plus grande unité de codage), le RDcost $RD1$ est calculé avec l'indicateur de partitionnement et initialisé à 0, Figure 3.2. Ensuite, l'indicateur passe à 1 et la LCU est partitionnée en quatre sous-unités CU de taille 32×32. Le premier $CU_1 0$, tel que présenté à la Figure 3.2, a un RDcost égal à $RD2$. A la profondeur suivante, la CU est partitionnée en quatre CUs de 16×16. $RD3$ est le RDcost de la première CU de taille 16×16 ($CU_2 0$). Si son indicateur est 1, la dernière profondeur (profondeur = 3) sera atteinte et la CU est donc partitionnée en quatre sous-unités de codage (SCU) de taille 8×8. Le RDcost pour chaque SCU sera noté $RD4$, $RD5$, $RD6$ et $RD7$, respectivement. La première décision sera prise de bas en haut, déterminant si la première CU de taille 16×16 est choisie ou non. Nous avons besoin de comparer la somme des quatre RDcost de la SCU 8×8 avec le $RD3$ du CU 16×16 pour prendre une décision. Si $RD3$ est supérieur à la somme de $RD4$, $RD5$, $RD6$ et $RD7$, la décision de partitionnement de $CU_2 0$ sera prise, sinon la $CU_2 0$ ne sera pas divisée. Par conséquent, si l'inégalité 3.1 est vérifiée, aucune division n'est effectuée.

$$RDcost_{CU} < \sum_{k=0}^{3} RDcost_{sub_CUs}(k) \tag{3.1}$$

Comme pour les autres CUs, la décision est toujours basée sur cette équation.

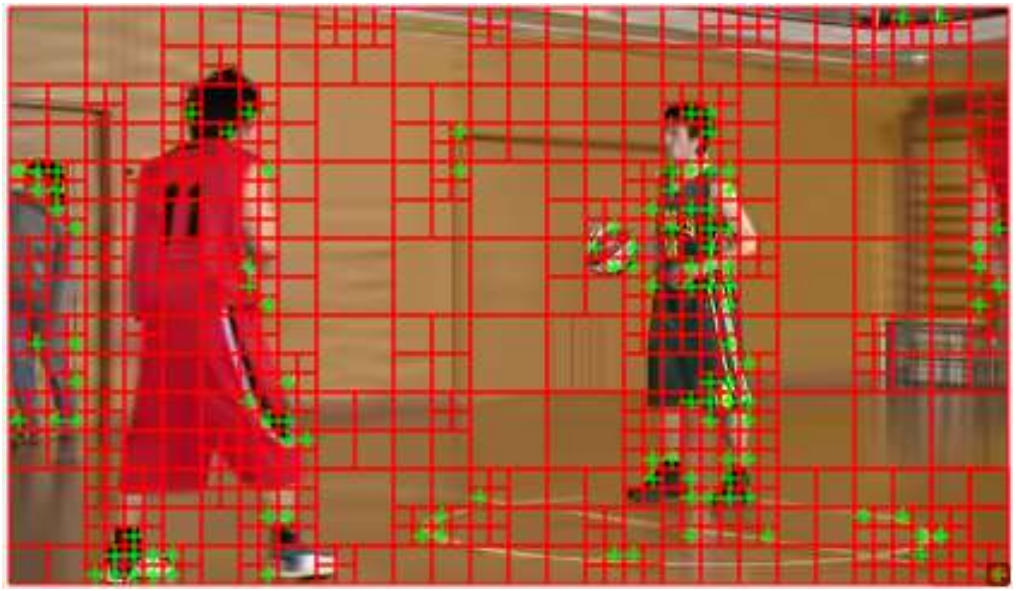


Figure 3.3. Résultat de partitionnement de la première image de la séquence BasketballPass (taille : 416x240, QP=37) par le software de référence HM16.6

3.3.2 Description de l'algorithme rapide proposé pour le partitionnement des CUs

L'adoption du partitionnement flexible dans la norme HEVC vise à couvrir la diversité du contenu vidéo. La Figure 3.3 illustre un exemple de partitionnement d'une image en CUs. Sur cette figure, nous pouvons noter que les blocs larges s'adaptent aux zones lisses, alors que les blocs de petites tailles sont beaucoup plus utilisés avec les zones texturées [32].

Nous proposons un algorithme jugé efficace, sans opérations complexes, pour l'analyse de la complexité de la texture. Dans une première approche, nous proposons d'effectuer l'analyse au niveau des CTUs. Selon la complexité de la CTU, nous pouvons estimer les profondeurs, minimale et maximale, pour son partitionnement, Figure 3.4. Comme le partitionnement suit un schéma récursif, il commence toujours depuis une profondeur 0 (la taille d'une partition CU à la profondeur 0 correspond à la taille de la CTU). Par conséquent, si la profondeur minimale choisie par notre algorithme est différente de zéro, nous maintenons le partitionnement à partir du niveau 0 sans calculer la prédiction. Dans ce cas, le coût RDcost est initialisé à la valeur maximale du format de données utilisé (***MAX_DOUBLE=1.7e+308***), pour ne pas affecter le choix final de la partition CU satisfaisant l'expression 3.1.

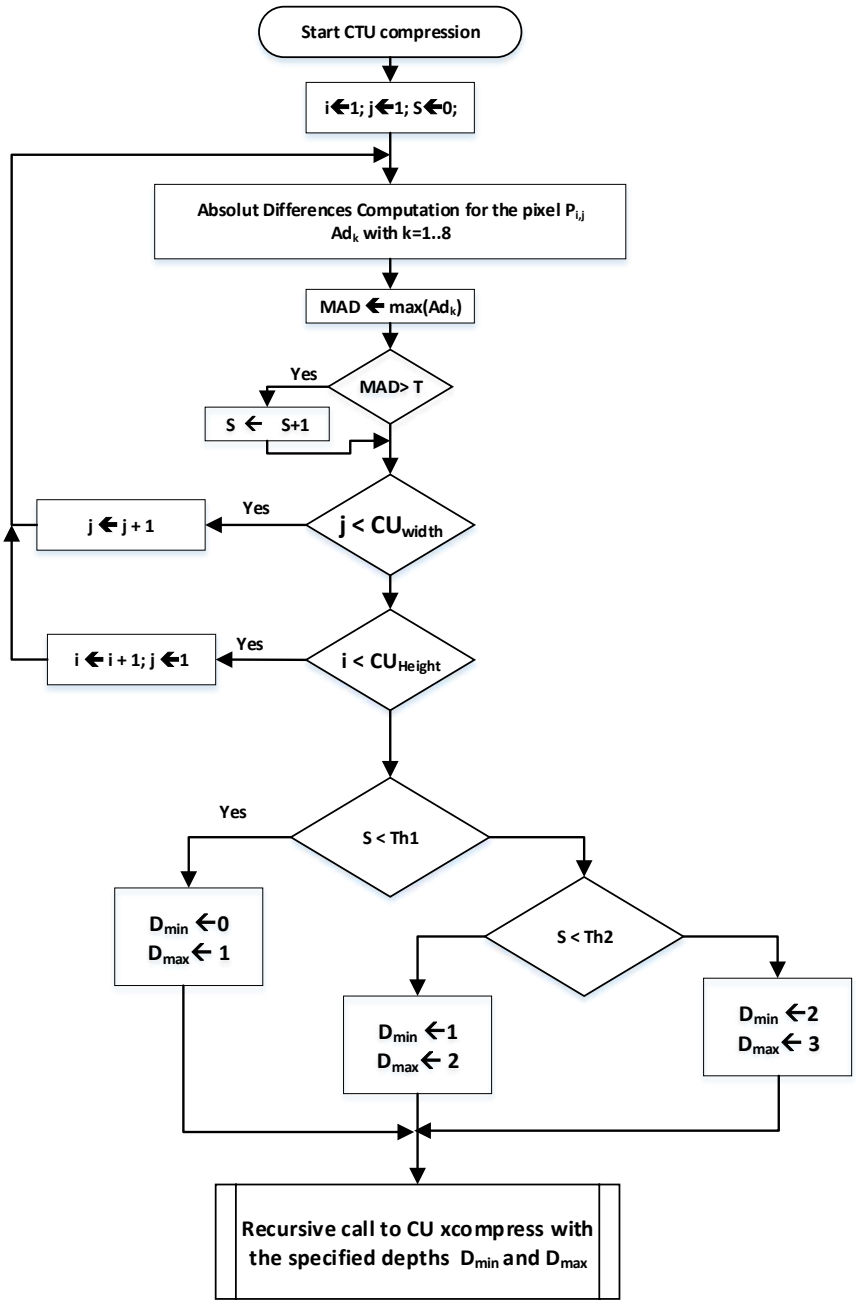


Figure 3.4. L'algorithme proposé pour le partitionnement rapide des CUs : version initiale

Dans une deuxième approche, nous proposons une analyse de la texture au niveau des partitions CUs. A la différence de la première approche, l'analyse de texture suit le même mécanisme récursif du partitionnement adopté pour les partitions CUs. Par conséquent, la taille du bloc dont on vérifie sa texture est variable en fonction de la

profondeur en cours de traitement. Si un bloc CU est noté comme homogène, une fin précoce du partitionnement de la CU est effectuée. Sinon,le bloc est considéré texturé et le partitionnement est poursuivi de manière approfondie. Les étapes principales de l'algorithme proposé pour cette approche, sont illustrées à la Figure 3.5.

Pour compresser une CTU, le software de référence utilise un appel récursif à la fonction *xCompressCU*. Le premier appel est lancé avec une profondeur initialisée à 0, qui veut dire que les tailles de la CU et de la CTU sont égales. Après le partitionnement en quatre sous-partitions CU, *xCompressCU* est appelée de nouveau pour chaque sous-partition avec une profondeur incrémentée. Ce processus de partitionnement et d'appel est répété jusqu'à achèvement de la profondeur maximale autorisée (où la taille minimale d'une CU est atteinte).

Dans nos algorithmes, l'appel récursif est maintenu seulement si le partitionnement est nécessaire (cas d'un bloc texturé). Pour une CTU ou CU donnée, nous commençons par le calcul du maximum des différences absolues (noté **MAD**) entre un pixel $P_{i,j}$ (i et j sont les coordonnées du pixel P) et ses huit voisins, Figure 3.6.

La valeur **MAD** est calculée comme suit :

$$MAD = \max_{k} (Ad_k) \ \text{, k=1,2,...,8} \tag{3.2}$$

avec :

$$\begin{aligned} Ad_1 &= |P_{i,j} - P_{i-1,j-1}|; & Ad_2 &= |P_{i,j} - P_{i-1,j}| \\ Ad_3 &= |P_{i,j} - P_{i-1,j+1}|; & Ad_4 &= |P_{i,j} - P_{i,j-1}| \\ Ad_5 &= |P_{i,j} - P_{i,j+1}|; & Ad_6 &= |P_{i,j} - P_{i+1,j-1}| \\ Ad_7 &= |P_{i,j} - P_{i+1,j}|; & Ad_8 &= |P_{i,j} - P_{i+1,j+1}| \end{aligned}$$

Par la suite, nous calculons la somme (**S**) de tous les **MAD** pour prendre une décision sur l'homogénéité de la CTU ou la CU, à la profondeur actuelle. Dans la

Tableau 3.1. Choix du seuil pour chaque approche

Profondeur	Valeurs du seuil pour la première approche	Valeurs du seuil pour la deuxième approche
0 (64x64) **1 (32x32)** **2 (16x16)**	• **Th1 = 100** • **Th2 = 800** • **Th3 = 1700**	• **Th1 = 9000** • **Th2 = 4500** • **Th3 = 2200**

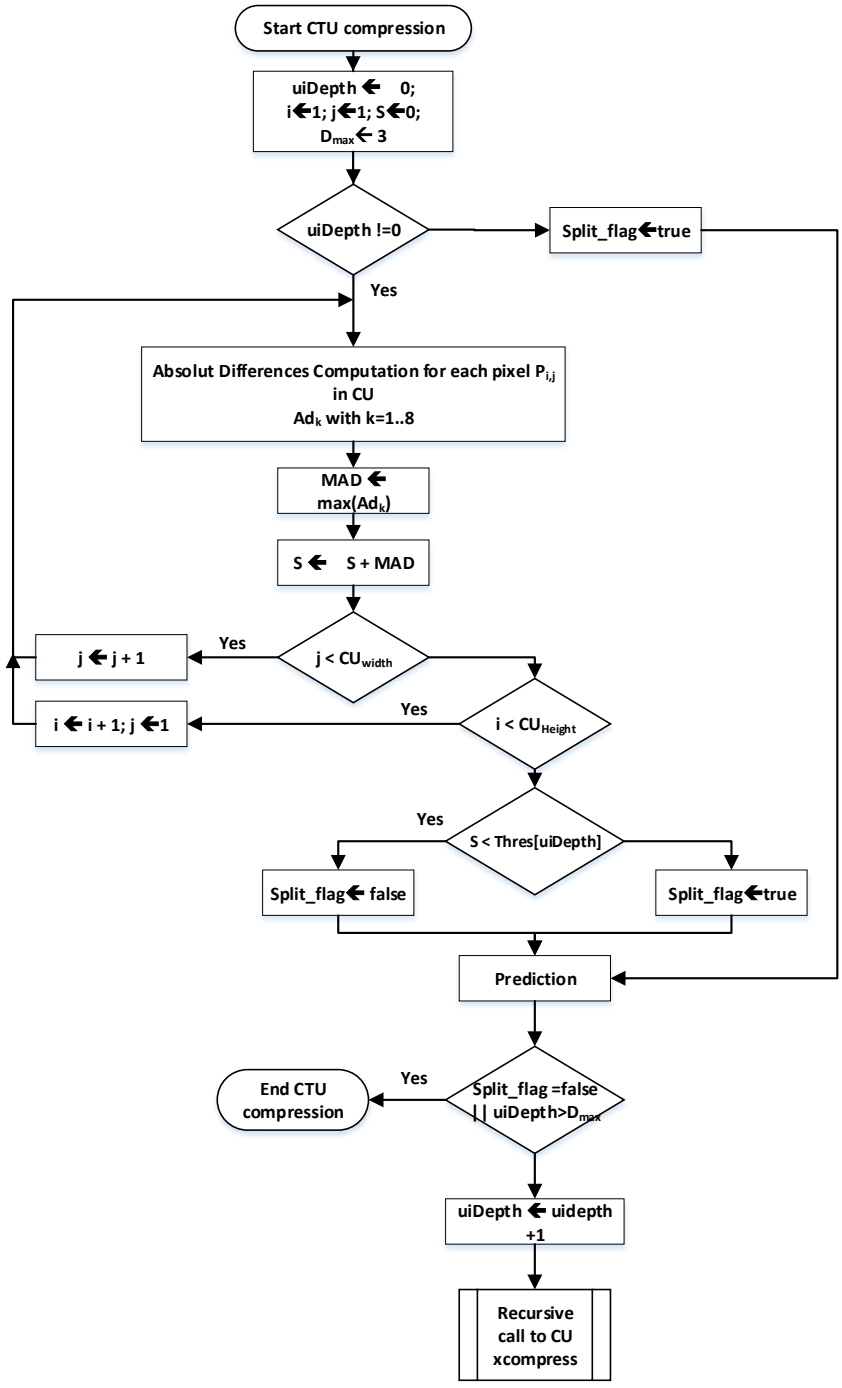


Figure 3.5. L'algorithme proposé pour le partitionnement rapide des CUs, 2^{eme} version

première approche, nous distinguons trois cas possibles selon la valeur de la somme **S**, Tableau 3.1. Les cas pour lesquels $\mathbf{D_max} < 3$, un arrêt anticipé du processus de partitionnement est envisagé, tandis que si $\mathbf{D_min} > 0$, le partitionnement commence

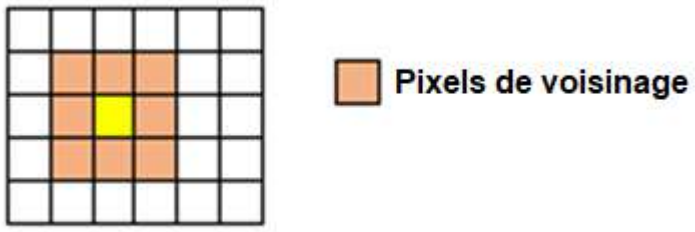


Figure 3.6. Voisinage d'un pixel

à partir de la profondeur zéro sans calcul de la prédiction.

Pour la deuxième approche, si la somme **S** est supérieure à un seuil **Thres[uiDepth]**, le bloc CU correspondant est porté comme texturé, et par conséquent, il est possible de procéder au partitionnement jusqu'à la profondeur suivante. Au contraire, si la somme **S** est inférieure au seuil, le partitionnement doit être terminé. Les blocs CTUs incomplets situés aux limites des images, sont exclues de l'analyse susmentionnée car ils ne suivent pas le même schéma de partitionnement récursif.

La détermination du seuil est très importante pour réaliser de telles améliorations. Dans notre algorithme (version 2), la valeur de seuil dépend de la taille du bloc considéré (profondeur). Une série d'expériences a été menée afin de choisir une valeur de seuil qui permet de résoudre le compromis entre l'efficacité de la compression et le temps de codage. Les résultats présentés dans la section suivante sont obtenus pour les valeurs de seuil suivantes :

Profondeur = 3 (8x8) est exclu du contrôle d'homogénéité, car il correspond à la plus petite taille de la CU. Nous avons envisagé un rapport de $\frac{1}{4}$ entre deux seuils aux profondeurs **D** et **D+1**, mais nos tests expérimentaux ont montré qu'un rapport de $\frac{1}{2}$ peut aboutir à de meilleurs résultats.

Pour montrer l'impact de l'approche proposée, nous avons illustré les résultats d'application de l'algorithme d'homogénéité sur une image, en comparaison par rapport à d'autres algorithmes de détection de contour, notamment les algorithmes de Sobel, Canny et Prewitt [54]. La Figure 3.7 montre que l'algorithme proposé pour l'analyse de la texture (avec des valeurs **MAD** seuillées) conduit à des résultats similaires avec des bords plus épais par rapport aux résultats des filtres de Sobel et de

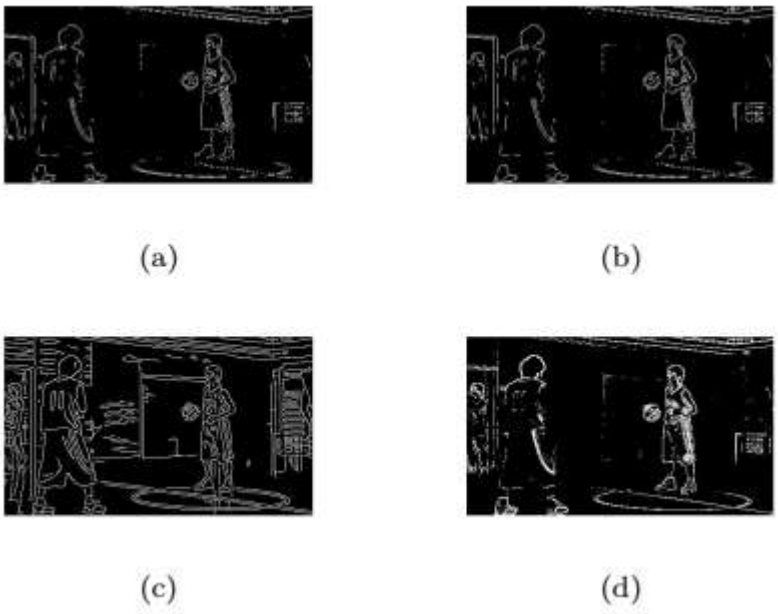


Figure 3.7. Résultats des algorithmes de détection de contour appliqués à la première image de la séquence BasketballPass (416x240). (a) filtre Sobel, (b) filtre de Prewitt, (c) filtre de Canny et (d) notre algorithme

Prewitt.

3.4 Résultats et discussion

3.4.1 Conditions expérimentales

Dans cette section, les différents résultats d'implémentation sont présentés. Nous commençons par l'évaluation des performances de notre algorithme de partitionnement rapide des unités CUs, qui constitue une optimisation algorithmique à un seul thread. Pour toutes nos expériences, nous avons utilisé le software de référence HM16.6, le Profil principal (*Main Profil*), les configurations All-Intra, et un sous-ensemble de séquences recommandées par JCT-VC comprenant les classes A, B, C, D et E.

3.4.2 Critères d'évaluation

L'évaluation de l'efficacité de la compression est basée dans nos expériences, sur les métriques de taux de Bjøntegaard Delta, BD-BR (%) et BD-PSNR (dB) [37], ainsi que sur :

– ΔBR relative à l'incrément en bitrate

– $\Delta PSNR$ pour mesurer la dégradation de la qualité en PSNR

où

$$\Delta BR(\%) = \left(\frac{BR_{proposed}}{BR_{HM16.6}} - 1\right) \times 100 \tag{3.3}$$

$$\Delta PSNR(dB) = PSNR_{proposed} - PSNR_{HM16.6} \tag{3.4}$$

Avec QP couvrant les valeurs $\{22, 27, 32, 37\}$. Le gain en temps d'encodage ΔTS pour le partitionnement rapide des CUs, est défini par :

$$\Delta TS(\%) = \left(\frac{EncTime_{proposed}}{EncTime_{HM16.6}} - 1\right) \times 100 \tag{3.5}$$

3.4.3 Résultats des algorithmes proposés

Nous avons commencé par l'implémentation du HM16.6 en tant que référence, dans l'intention d'évaluer comparativement les performances des méthodes proposées dans ce travail. Les résultats de la première proposition sont donnés dans le Tableau 3.2. D'après ces résultats, il est très difficile de maintenir un gain en temps d'encodage important sans dégradation significative en PSNR ou augmentation excessive en débit. Pour notre première approche, la dégradation en performances est due aux mauvaises décisions prises sur le partitionnement ou non des unités CUs. L'analyse de la texture au niveau CTU présente les inconvénients suivants :

– Cas d'une CTU à texture très variée concentrée en un coin isolé : dans ce cas l'algorithme proposé peut considérer la CTU riche en texture et par conséquent un partitionnement rigoureux jusqu'à un niveau avancé (niveau 3), est envisagé.
– Cas d'une CTU quasi-homogène : l'algorithme proposé classifie la CTU comme homogène, ce qui conduit à un arrêt précoce du partitionnement.

Pour les deux cas précédents, une décision erronée peut figurer, ce qui résulte soit en une dégradation en performance de compression, soit en une très faible amélioration en temps d'encodage. Pour rattraper les inconvénients du premier algorithme, nous avons suggéré une analyse de la texture au niveau des unités CUs. Cette solution a fait l'objet du deuxième algorithme décrit dans ce chapitre.

Comme on peut le constater sur la Figure 3.8, la deuxième proposition conserve presque le même schéma de partitionnement que HM16.6 de la Figure 3.3, à l'exception

Tableau 3.2. Performances de l'algorithme N° 1 de partitionnement rapide proposé par rapport à l'HM16.6 de référence

Classes	Sequences	ΔBR(%)	ΔPSNR (dB)	ΔT (%)
Class A	Traffic	1.36	-0.02	-32.61
2560x1600	PeopleOnStreet	0.91	-0.02	-31.56
Average A		***1.13***	***-0.02***	***-32.09***
	Kimono1	3.90	-0.06	-41.36
Class B	BQTerrace	0.65	-0.05	-25.38
1920x1080	Cactus	1.15	-0.01	-30.43
	ParkScene	1.30	-0,00	-26.75
	BasketballDrive	2.38	-0.07	-36.74
Average B		***1.88***	***-0.04***	***-32.13***
Class E	Vidyo1	3.90	-0.11	-41.77
1280x720	Vidyo3	4.15	-0.09	-35.58
	Vidyo4	3.53	-0.09	-40.11
Average E		***3.86***	***-0.10***	***-39.15***
	BasketBallDrill	0.37	-0.03	-28.40
Class C	BQMall	1.34	-0.01	-27.19
832x480	RaceHorses	0.97	-0.02	-29.10
	PartyScene	0.05	-0,00	-23.86
Average C		***0.68***	***-0.02***	***-27.14***
	BasketballPass	1.01	-0.03	-26.87
Class D	RaceHorses	0.58	-0.02	-23.62
416x240	BlowingBubbles	0.16	-0.00	-20.84
	BQSquare	0.03	-0.01	-21.80
Average D		***0.44***	***-0.01***	***-23.28***
Average		**1.52**	**-0.03**	**-30.98**

de certaines CUs qui sont classées comme homogènes par notre algorithme.

Le Tableau 3.3 résume les résultats expérimentaux de notre deuxième approche pour la décision rapide de taille des CUs, en comparaison avec l'HM16.6 de référence. Ces résultats prouvent que l'algorithme proposé permet d'économiser en moyenne 41% du temps de codage avec une légère augmentation du débit de 0,69% en ΔBR, et une dégradation non significative de la qualité de 0,05 dB en terme de PSNR. L'évaluation des performances de l'algorithme proposé dépend essentiellement du contenu intrinsèque de la séquence vidéo, et varie d'une séquence à une autre et d'une classe à une autre.

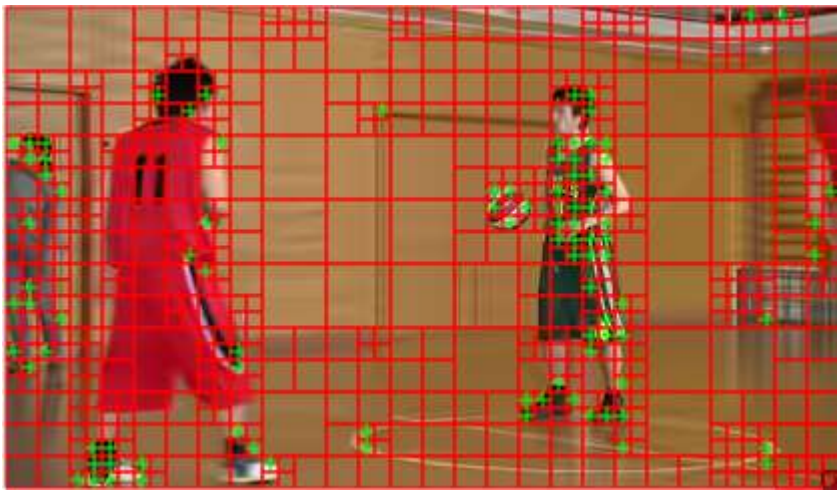


Figure 3.8. Résultats de partitionnement de la première image de BasketballPass (taille : 416x240, QP = 37) en utilisant l'algorithme de partitionnement rapide proposé.

Les meilleurs résultats en termes de gain en temps d'encodage sont obtenus pour la classe E (1280×720), avec un gain ΔTS allant jusqu'à -63,28%, mais l'augmentation de débit est la plus élevée de toutes les autres classes. Dans nos expériences, la classe E contient les séquences FourPeople, KristenAndSara et Johnny qui sont filmées par une caméra fixe (fond fixe avec de nombreux blocs homogènes). En plus, le mouvement des personnes dans la scène est relativement lent, ce qui induit une forte corrélation temporelle. En conséquence, notre algorithme classe la plupart des CTU comme étant homogènes et les code à une faible profondeur (c'est-à-dire, une grande taille de la CU). D'autre part, les CTU appartenant à des séquences vidéo de classe D (basse résolution : 416×240) ont généralement une texture élevée, et notre algorithme les code à une profondeur élevée (CU de taille 8×8 et 4×4 de taille PU, d'où l'augmentation de la complexité de calcul), ce qui diminue le gain de temps.

En général, notre proposition est plus adaptée aux séquences de haute résolution, dans lesquelles un bloc CTU couvre une petite région de l'image et donc un contenu plus homogène. La comparaison des performances de l'approche proposée avec celle de certains algorithmes d'actualité, est présentée dans le Tableau 3.4. Il est clair que notre méthode devance celle de l'algorithme proposé dans [49] en termes d'efficacité de compression et de réduction du temps d'encodage.

Concernant le temps d'encodage, notre algorithme assure des résultats supérieurs à ceux de [44], [55], [56] et [57] mais leurs débits binaire et pénalité PSNR sont meilleurs. Cependant, les résultats dans [58] montrent un gain de temps d'environ 53%, meilleur

Tableau 3.3. Performances de l'algorithme de partitionnement rapide proposé par rapport à l'HM16.6 de référence

Classes	Sequences	BD-BR(%)	BD-PSNR(dB)	ΔBR(%)	ΔPSNR (dB)	ΔT (%)
Class A	Traffic	1,85	-0,10	0,65	-0,06	-47,50
2560x1600	PeopleOnStreet	1,67	-0,10	0,87	-0,05	-43,91
Average A		***1,76***	***-0,10***	***0,76***	***-0,05***	***-45,70***
	Kimono1	1,17	-0,04	0,46	-0,02	-72,95
Class B	BQTerrace	0,59	-0,03	0,04	-0,03	-31,15
1920x1080	Cactus	1,10	-0,04	0,03	-0,03	-26,81
	ParkScene	1,07	-0,04	0,04	-0,04	-27,52
	BasketballDrive	1,37	-0,04	-0,15	-0,03	-53,85
Average B		***1,06***	***-0,04***	***0,08***	***-0,03***	***-42,46***
Class E	FourPeople	3,09	-0,18	1,62	-0,08	-51,66
1280x720	KristenAndSara	3,25	-0,16	1,78	-0,07	-63,28
	Johnny	3,47	-0,14	2,01	-0,06	-60,73
Average E		***3,27***	***-0,16***	***1,80***	***-0,07***	***-58,56***
	BasketBallDrill	2,80	-0,20	0,90	-0,07	-44,36
Class C	BQMall	1,40	-0,09	0,49	-0,04	-30,98
832x480	RaceHorses	1,20	-0,08	0,45	-0,04	-37,64
	PartyScene	0,31	-0,03	0,09	-0,01	-15,74
Average C		***1,43***	***-0,10***	***0,48***	***-0,04***	***-32,18***
	BasketballPass	1,54	-0,09	0,58	-0,04	-42,97
Class D	RaceHorses	1,46	-0,13	0,48	-0,05	-26,49
416x240	BlowingBubbles	0,76	-0,06	0,11	-0,03	-18,52
	BQSquare	1,14	-0,15	0,10	-0,05	-22,90
Average D		***1,22***	***-0,11***	***0,32***	***-0,04***	***-27,72***
Average		**1,75**	**-0,10**	**0,69**	**-0,05**	**-41,32**

que notre réduction de temps. Mais l'augmentation du débit binaire ΔBR est de 1,98% en moyenne, environ trois fois plus élevée que notre résultat ($\Delta BR = 0{,}69\%$). Pour les séquences à haute définition, telles que celles des classes A et B, l'algorithme proposé diminue efficacement le temps de codage avec une augmentation tolérable du débit.

Tableau 3.4. Comparaison des performances du partitionnement rapide proposé avec certains travaux connexes

	[49]			[55]			[44]							
	ΔBR	ΔPSNR	ΔT	BD-BR	BD-PSNR	ΔT	BD-BR	BD-PSNR	ΔT					
Class A	0.70	-0.08	-27.85	1.33	-	-24.50	2.28	-0.11	-21.95					
Class B	0.74	-0.07	-38.85	0.80	-	-27.40	2.16	-0.08	-28.58					
Class C	0.53	-0.11	-25.80	0.62	-	19.00	1.50	-0.08	-17.80					
Class D	0.70	-0.12	-25.03	-	-	-	1.18	-0.07	-14.50					
Class E	1.12	-0.08	-36.72	-	-	-	-	-	-					
Average	0.76	-0.09	-31.30	0.83	-	24.00	1.74	-0.08	-21.10					
	[56]			[58]			[57]			Our proposed				
	BD-BR	BD-PSNR	ΔT	ΔBR	ΔPSNR	ΔT	BD-BR	BD-PSNR	ΔT	BD-BR	BD-PSNR	ΔBR	ΔPSNR	ΔT
Class A	0.05	-	-09.05	1.17	-0.06	-53.41	0.37	-	-29.30	0.76	-0.10	0.76	-0.05	-45.70
Class B	0.00	-	-10.75	0.88	-0.06	-52.15	0.72	-	-30.10	1.06	-0.04	0.08	-0.03	-42.46
Class C	0.20	-	-16.95	2.34	-0.18	-55.07	0.49	-	-20.50	1.43	-0.10	0.48	-0.04	-32.18
Class D	0.20	-	-21.20	2.72	-0.23	-52.84	0.30	-	-19.70	1.22	-0.11	0.32	-0.04	-27.72
Class E	0.01	-	-14.10	3.44	-0.12	-54.78	0.67	-	-41.00	3.27	-0.16	1.80	-0.07	-58.56
Average	0.01	-	-14.40	1.98	-0.13	-53.52	0.51	-	-28.12	1.75	-0.10	0.69	-0.05	-41.32

3.5 Conclusion

Dans ce chapitre, nous avons présenté un algorithme jugé efficace, basé sur l'homogénéité mesurée pour la réduction de la complexité du codage intra HEVC. Il a été constaté que les blocs de grande taille conviennent mieux au codage de régions homogènes, tandis que les zones texturées sont souvent codées par de petits blocs.

Pour anticiper le partitionnement des blocs CTU, notre algorithme commence par l'analyse des blocs CU à la profondeur actuelle, afin de vérifier la complexité de la texture. Si le bloc est étiqueté homogène, le processus de partitionnement est arrêté. Sinon, le partitionnement se poursuit de la même manière que pour le software de référence. Selon les résultats trouvés, nous pouvons conclure que l'algorithme proposé pour le partitionnement rapide des CUs, permet d'économiser en moyenne 41% du temps d'encodage avec des incréments moyens de 1,75% et 0,69% respectivement dans BD-BR et ΔBR. La perte du PSNR reste négligeable. Comparé à certains algorithmes récents, notre algorithme offre un meilleur gain de temps de codage avec pertes acceptables en débit.

Dans le chapitre suivant, nous exposons une méthodologie efficace pour l'implémentation de l'encodeur HEVC sur une plateforme multi-processeurs.

Chapitre 4

Implémentation parallèle de l'encodeur HEVC en configuration AI

Sommaire

4.1 Introduction **94**
4.2 Paradigmes de programmation parallèle **95**
4.2.1 Architectures à mémoire partagée 95
4.2.2 Architectures à mémoire distribuée 97
4.3 Niveaux de parallélisme pour l'encodeur HEVC **98**
4.3.1 Parallélisme au niveau images 99
4.3.2 Parallélisme au niveau Slices 99
4.3.3 Parallélisme au niveau des blocs 100
4.3.4 Parallélisme au niveau des Tiles 101
4.3.5 Traitement parallèle en front d'onde, **WPP** : (*Wavefront Parallel Processing*) 102
4.4 Travaux connexes **102**
4.5 Description de l'algorithme proposé pour l'implémentation parallèle **104**
4.6 Résultats d'implémentation sur une plateforme hétérogène de type Cubieboard4 . . . . . **107**
4.6.1 Description de la plateforme 107
4.6.2 Résultats et discussions 107
4.7 Résultats d'implémentation sur une plateforme homogène de type HPC **113**
4.7.1 Description de la plateforme 113
4.7.2 Résultats et discussions 113
4.8 Conclusion **114**

4.1 Introduction

Au cours des 50 dernières années, la loi de **Moore** a prédit avec précision que le nombre de transistors sur un circuit intégré doublerait tous les deux ans. Pour transformer ces transistors en niveaux de performances système équivalents, les concepteurs de puces ont augmenté :

- Les fréquences d'horloge (sollicitant des pipelines d'instruction plus complexes).
- Le parallélisme au niveau des instructions (nécessitant la prédiction des threads simultanées et de branchements).

Augmenter la fréquence d'horloge pour améliorer les performances, est une technique suivie pendant de nombreuses années, et qui ne pouvait plus offrir d'avantages, en raison de plusieurs problèmes tels que la consommation d'énergie, les fuites de courant, les interférences électriques, les problèmes technologiques de conception des puces, etc. Afin d'exploiter les avantages d'une mise d'échelle technologique plus réduite, le processeur multicœur a vu le jour. Il est devenu évident que l'ajout de plusieurs cœurs processeurs à une même puce, offre plus d'avantages et de performances que la surcharge d'un seul processeur. L'utilisation de plusieurs cœurs sur une seule puce permet aux concepteurs d'atteindre leurs objectifs de performances, sans utiliser la fréquence de fonctionnement maximale. Ils peuvent sélectionner une fréquence de travail dans la zone idéale d'une technologie, permettant de réduire la consommation d'énergie. Les performances globales sont atteintes avec des cœurs dotés d'architectures de pipeline simplifiées, par rapport à une solution équivalente à un seul cœur.

Dans ce chapitre, une évaluation des performances de la réalisation parallèle du codeur HEVC en configuration All-Intra est proposée. Une plate-forme Octa-core est utilisée, c'est la CubieBoard4 hétérogène, comprenant deux processeurs quad-core ARM A7 et ARM A15. Nous avons utilisé le paradigme OpenMP pour l'implémentation de parallélisme, dans laquelle chaque thread est affecté à un cœur processeur pour coder une image distincte. La configuration All-Intra sert à rompre les dépendances de codage entre les trames successives, ce qui permet un traitement simultané de plusieurs images.

4.2 Paradigmes de programmation parallèle

Il est reconnu que les applications parallèles peuvent être classées dans des paradigmes de programmation bien définis. Quelques paradigmes de programmation sont utilisés à plusieurs reprises, pour développer de nombreux programmes parallèles. Chaque paradigme est une classe d'algorithmes ayant la même structure de contrôle.

L'expérience acquise à ce jour, suggère qu'un nombre relativement restreint de paradigmes couvre la plupart des programmes parallèles. Le choix du paradigme est déterminé par les ressources du calcul parallèle disponibles, et par le type de parallélisme inhérent au problème. Les ressources peuvent définir le niveau de granularité pouvant être efficacement utilisé par le système. Le type de parallélisme reflète la structure de l'application ou des données, et les deux types peuvent exister dans différentes parties d'une même application. Le parallélisme émanant de la structure de l'application est appelé parallélisme fonctionnel. Dans ce cas, différentes parties du programme peuvent effectuer différentes tâches de manière simultanée et coopérative. Le parallélisme est aussi appliqué à la structure des données. Ce type de parallélisme permet l'exécution de processus parallèles de fonctionnement identique sur différentes parties de données [59].

4.2.1 Architectures à mémoire partagée

Le terme SMP (Shared-Memory Parallel computers) a été créé à l'origine pour désigner un système multiprocesseur symétrique, un ordinateur parallèle à mémoire partagée dont les processeurs individuels partagent la mémoire et les entrées-sorties de manière à ce que chacun puisse accéder à n'importe quel emplacement à la même vitesse. Ils ont donc un temps d'accès uniforme à la mémoire (UMA). De nombreuses petites machines à mémoire partagée sont symétriques dans ce sens. Cependant, les machines à mémoire partagée plus volumineuses, ne répondent généralement pas à cette définition. Une partie de la mémoire peut être « plus proche » d'un ou de plusieurs processeurs et ainsi, être utilisée plus rapidement. Ces machines ont un accès mémoire cohérent et non uniforme (cc-NUMA) [60].

Aujourd'hui, les principaux fournisseurs de matériel informatique proposent tous, un modèle d'ordinateur parallèle à mémoire partagée, allant de deux à des centaines

de processeurs.

La plupart des architectures multicœurs modernes à multitraitement symétrique [8, 9], ont une mémoire vive (RAM) généralement partagée entre tous les cores de calcul. Ainsi, tous les cores du système ont la même vue de la mémoire, Figure 4.1. Techniquement, le terme de mémoire partagée est aussi utilisé pour parler de threads softwares, partageant une région de mémoire entre eux. La mémoire partagée présente l'avantage d'une intercommunication rapide entre plusieurs threads, s'exécutant éventuellement sur différents cores de calcul. De plus, la programmation est rès simple car tous les cores/threads utilisent le même espace d'adresses mémoire.

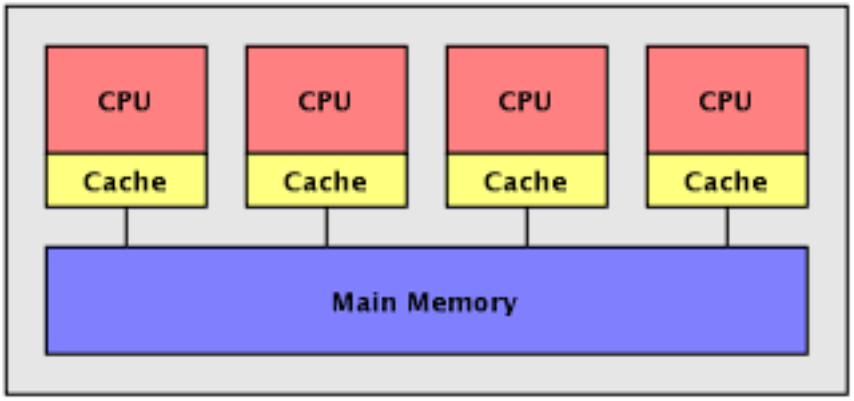


Figure 4.1. Système à mémoire partagée

Cependant, les systèmes de mémoire partagée ne sont pas utilisables pour une parallélisation massive, car ils ne peuvent pas évoluer au-delà d'une douzaine de cœurs de calcul. Ceci est dû au fait que la fréquence d'horloge du microprocesseur est plus élevée que la fréquence d'horloge de la mémoire et que par conséquent, un goulot d'étranglement apparait dans le transfert des données entre mémoire et cores de calcul. Une façon de minimiser ce problème, consiste à créer de petites mémoires caches à proximité des cœurs de calcul. Les performances de ces caches sont supérieures à celles de la RAM et sont donc utilisées pour stocker les données fréquemment consultées, à proximité d'un noyau donné. De plus, les cores processeurs peuvent partager le cache en le divisant en plusieurs niveaux, où un seul cœur peut accéder au cache de niveau 1, et un cache de niveau 2 partagé entre deux cœurs, et un de niveau 3 entre quatre cœurs, etc. Malgré cette conception, les caches ajoutent également des problèmes lorsqu'un cœur met à jour une valeur dans un cache, qui doit la communiquer à tous les autres caches, pour garantir une cohérence. Par conséquent, la mémoire partagée dans sa

conception actuelle, n'est pas adaptée aux architectures homogènes comportant plus de dix à vingt cœurs[61].

Lors de la conception d'un algorithme, il est parfois nécessaire de prendre en compte le temps nécessaire à une unité de calcul donnée pour accéder à un emplacement de mémoire donné. C'est particulièrement le cas, lorsque certaines unités de calcul peuvent accéder à une partie de la mémoire plus rapidement que d'autres. Par exemple, si la première unité de calcul peut accéder dix fois plus rapidement aux X premiers octets de la mémoire que le reste de la mémoire, et que le second noyau de calcul peut accéder dix fois plus rapidement aux X octets suivants, etc. Le terme UMA est utilisé pour signifier que le temps d'accès à un emplacement unique de mémoire, est le même pour tous les cœurs. Cependant, les caches privés des unités de calcul peuvent rendre certains accès à la mémoire plus rapides que d'autres, s'ils contiennent une copie valide de leur contenu. Par conséquent, l'effet de la plupart des caches doit également être pris en compte lors de la conception d'un algorithme.

4.2.2 Architectures à mémoire distribuée

A la différence d'une architecture à mémoire partagée, Figure 4.1, chaque processeur possède sa propre mémoire dans une architecture à mémoire distribuée, Figure 4.2. Pour partager des informations, un réseau d'interconnexion entre les différents processeurs est nécessaire. Ce réseau d'interconnexion, appelé souvent MPI « Message Passing Interface », assure la communication inter-processeurs à travers l'envoi et la réception de messages [62].

Le MPI a été définie au début des années 90, il fournit un ensemble complet de

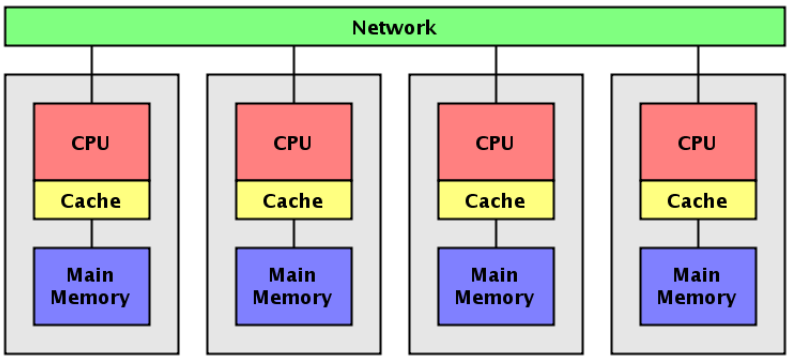


Figure 4.2. Système à mémoire distribuée

routines de bibliothèque pour la gestion des processus et d'échange de messages. Il est relativement facile à mettre en œuvre sur une grande variété de plates-formes, et offre donc une excellente portabilité. Cependant, la portabilité a un coût. La création d'un programme parallèle basé sur cette API (application programming interface) nécessite généralement une réorganisation majeure du code séquentiel d'origine. L'effort de développement peut être volumineux et complexe par rapport à une approche supportée par un compilateur, telle que l'approche proposée par OpenMP pour des systèmes à mémoire partagée.

On peut également combiner certaines API de programmation. MPI et OpenMP peuvent notamment être utilisés ensemble dans un programme, ce qui peut être utile si un programme doit être exécuté sur des ordinateurs massivement parallèles (MPP : massively parallel computers), constitués de plusieurs nœuds à mémoire partagée entre plusieurs coeurs éventuellement, Figure 4.3. Les raisons pour cela incluent l'exploitation d'une granularité de parallélisme plus fine que possible avec MPI, la réduction de l'utilisation de la mémoire, ou la réduction de la communication réseau [63].

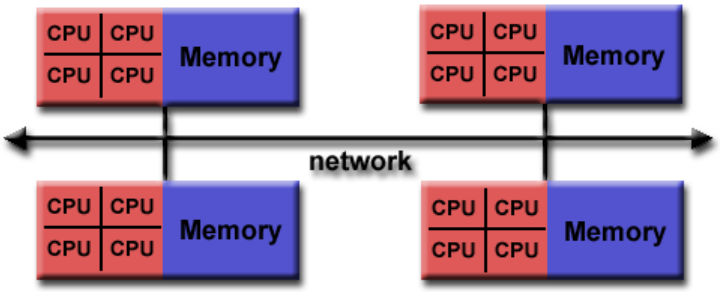


Figure 4.3. Système hybrides : mémoire partagée et mémoire distribuée

4.3 Niveaux de parallélisme pour l'encodeur HEVC

Diverses techniques de parallélisation ont été conçues pour tirer profit des ressources de calcul, utilisées dans l'implémentation des normes de codage vidéo. Une réalisation parallèle fiable ne peut être accomplie que par une stratégie de parallélisme efficace. Les niveaux de parallélisme décrits ci-dessous, sauf indication

explicite contraire, ne sont pas limités à l'utilisation avec la norme HEVC, mais sont largement utilisés aussi par H.264/MPEG-4 AVC.

4.3.1 Parallélisme au niveau images

Un ensemble d'images peut être traité simultanément, à condition que les dépendances de prédiction temporelle soient satisfaites. Ce type de parallélisme est plus adapté aux systèmes multicœurs limités en nombre de cœurs [18]. Ses principaux avantages résident dans sa simplicité de mise en œuvre et sa capabilité à maintenir l'efficacité du codage (pas de pertes d'efficacité au codage). Il est devenu l'état de l'art en matière de mise en œuvre logicielle de la norme H.264—MPEG-4 AVC. Néanmoins, les limitations suivantes peuvent se présenter :

- L'évolutivité de la parallélisation est conditionnée par les longueurs des listes d'images de référence, utilisées pour l'interprédiction et/ou la taille du groupe d'images (GOP).
- Les temps de codage/décodage de l'image dépendent du contenu de l'image qui peut varier considérablement d'une image à l'autre, ce qui entraîne un déséquilibre de travail entre les cœurs processeurs.
- L'utilisation du parallélisme au niveau de l'image permet d'augmenter le débit du codeur mais ne réduit pas le temps de codage de l'image.

4.3.2 Parallélisme au niveau Slices

Une image peut être partitionnée en une ou plusieurs Slices. Une Slice, pas nécessairement à forme rectangulaire, renferme un nombre entier d'unités CTU consécutives (dans l'ordre de balayage du raster) et est constituée d'un ou de plusieurs segments de Slice sans chevauchement [11]. Les segments de Slice d'une Slice partagent les informations de l'en-tête du segment de Slice associé au premier segment, indiqué sous le nom de segment de Slice indépendant. Il est dit indépendant, car il peut être décodé indépendamment de tout autre segment y compris un segment de Slice de la même image. Les segments de Slice dépendants, le cas échéant, sont un autre type de segment qui suit le segment de Slice indépendant. Ils contiennent un ensemble réduit de paramètres d'en-tête de Slice, en raison de la réutilisation des informations de l'en-tête du segment de Slice indépendant.

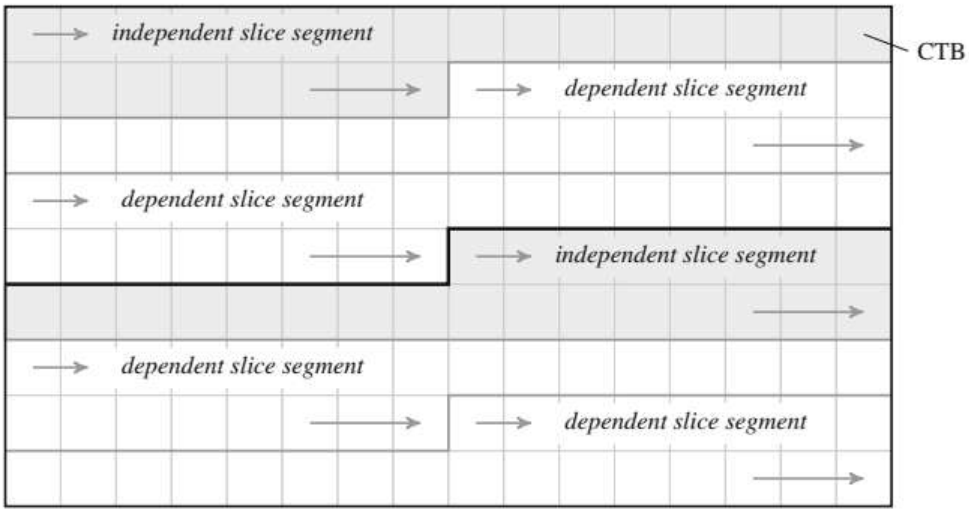


Figure 4.4. Exemple de partitionnement d'image en segments de Slice, segments indépendants et leurs segments dépendants associés.

La Figure 4.4 donne un exemple de partitionnement d'image en deux segments de Slice (séparés par un trait gras), chaque segment de Slice ayant un segment de Slice indépendant avec deux segments de Slice dépendants associés. Étant donné que les Slices d'une image sont indépendantes les unes des autres, la parallélisation au niveau des Slices devient une technique intéressante pour le traitement en parallèle, en particulier lorsque le nombre de Slices par image est réduit. Cependant, l'augmentation du nombre de Slices par image entraîne la création de plusieurs en-têtes de segment de Slice supplémentaires, qui réduisent considérablement l'efficacité du codage.

4.3.3 Parallélisme au niveau des blocs

Ce type de parallélisme est couramment utilisé dans les implémentations matérielles H.264/AVC en tant que pipeline au niveau macrobloc [18], où les macroblocs constituent les unités de traitement de base. Dans l'architecture en pipeline, la charge de travail peut être déséquilibrée pour chaque cœur processeur, ce qui convient mieux aux plates-formes multicœurs hétérogènes. Par exemple, un cœur est chargé de la prédiction, le second pour la transformation/quantification, le troisième pour le codage entropique, le quatrième pour le filtrage en boucle, etc. En outre ce type d'architecture en pipeline, nécessite un ordonnancement spécifique pour assurer l'ordre de traitement des blocs dans les différents cœurs, sous la contrainte de dépendance entre blocs lors de la prédiction, du filtrage, et du codage entropique CABAC.

4.3.4 Parallélisme au niveau des Tiles

Les Tiles et le traitement parallèle du front d'onde (WPP), sont les principales caractéristiques clés des nouveaux outils fournis par HEVC dans le contexte de l'amélioration du traitement parallèle. Les Tiles sont introduites en tant que région d'intérêt (ROI) [64] dans une grande image, toujours rectangulaire et contenant un nombre entier de CTU d'une ou de plusieurs Slices. Par conséquent et dans la même image, il peut y avoir des Slices contenant plusieurs Tiles et des Tiles contenant plusieurs Slices. En d'autres termes, une ou les deux conditions suivantes doivent être remplies pour chaque Slice et Tile [65] :

- Les Slices doivent inclure des Tiles complètes.
- Les Tiles doivent inclure des Slices complètes.

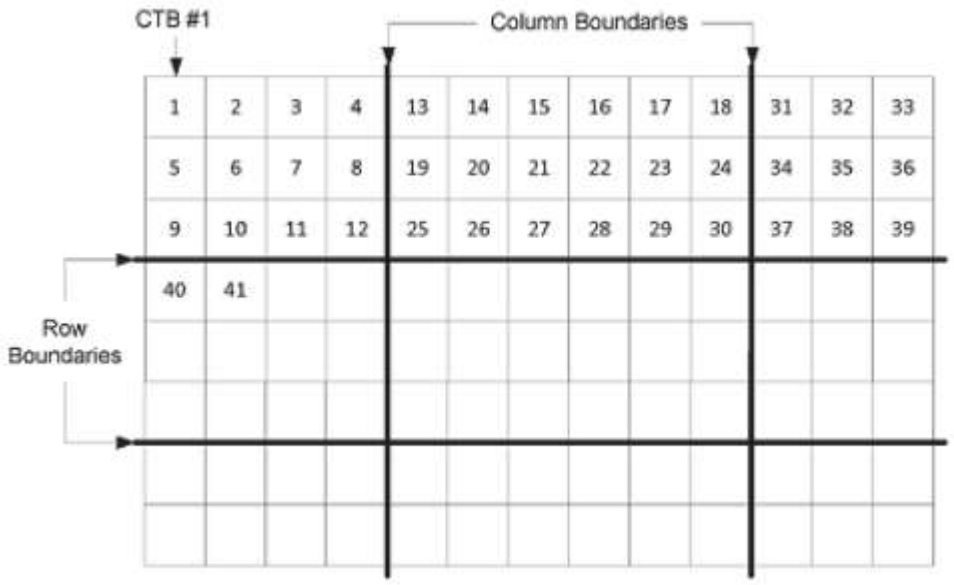


Figure 4.5. Partitionnement d'image en Tiles.

L'ordre de balayage en raster est utilisé pour traiter les CTU dans les Tiles et les Tiles dans une image. La première CTU à traiter est la CTU en haut à gauche dans la Tile du haut à gauche (première tile). La Figure 4.5 montre un exemple de division d'une image en neuf Tiles (trois lignes et trois colonnes). Les CTU de la Figure 4.5 sont numérotées dans l'ordre de traitement. Les Slices sont délimitées des lignes et des colonnes, qui cassent les dépendances de codage. En conséquence, les Tiles permettent de subdiviser une image en plusieurs partitions pouvant être traitées en parallèle.

4.3.5 Traitement parallèle en front d'onde, WPP : (*Wavefront Parallel Processing*)

Le traitement parallèle en front d'onde WPP est une forme particulière de parallélisme au niveau bloc, où les blocs (CTB) sont regroupés en lignes pouvant être traitées en parallèle, Figure 4.6. Au contraire des Tiles, il n'y a pas de discontinuité obligatoire pour le codage entropique ou la prédiction. Par conséquent, les pertes de performance de codage sont très limitées. Pour maintenir les dépendances de codage, un délai de deux CTB est requis entre les lignes de CTB successives. Par conséquent, une augmentation de la latence due au mécanisme de synchronisation, est observée [66]. Contrairement aux Slices, le traitement en front d'onde WPP n'introduit pas d'artefacts aux limites des partitions.

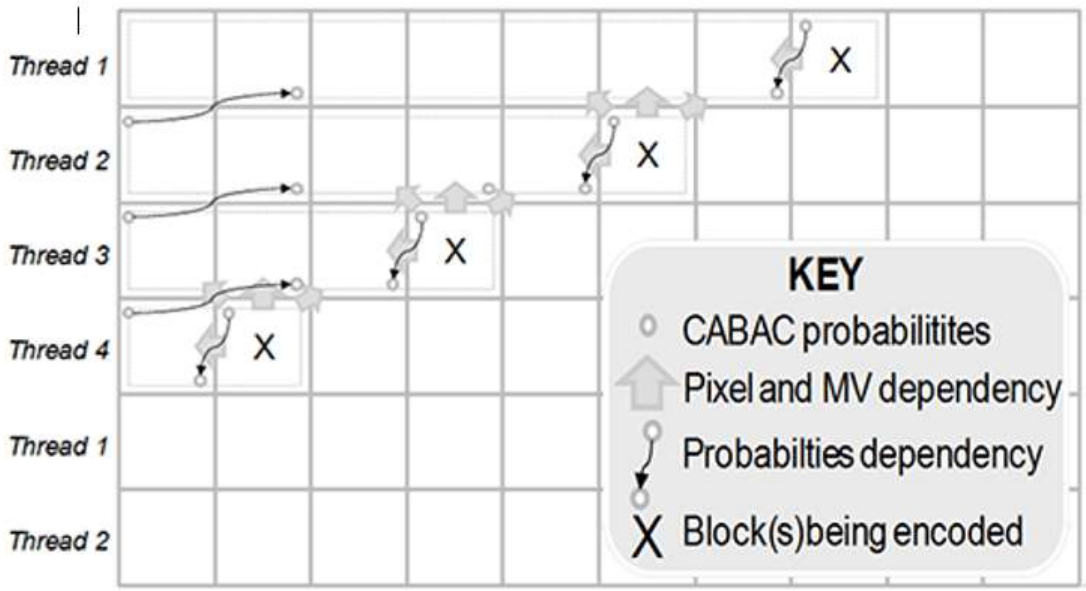


Figure 4.6. Dépendances de codage et parallélisation au niveau WPP.

4.4 Travaux connexes

La littérature sur le codage/décodage HEVC montre une variété d'approches. Cependant et à notre connaissance, très peu de publications traitant la question de la réalisation en parallèle du codeur HEVC, sont rencontrées Les auteurs de [67, 68] ont suggéré une plateforme puissante à 64 cœurs, afin d'accélérer le processus de codage.

Ils ont utilisé un graphe acyclique dirigé (DAG) pour paralléliser le traitement des CTUs et l'estimation de mouvement (ME), sur la base de la méthode parallèle locale (LPM). Une autre solution est décrite dans [69, 70] utilisant deux processeurs Intel XEON X5660 hexacore, jusqu'à 2,8 GHz et 12 Mo de cache par processeur pour 48 Go de RAM, sans hyper-threading activé. Leur proposition utilise le paradigme de programmation OpenMP pour réaliser différents niveaux de parallélisme, notamment Slice, I-frame et GOP. La parallélisation des Slices a également été expérimentée par Ahn et al. [71] sur un PC équipé d'un processeur Intel® Core ™ i7-3930 K (six cœurs, 12 threads avec hyper-threading) et 16 Go de mémoire.

Outre le software de référence (HM), il existe des codeurs HEVC à code source libre tels que x256 [72] et Kvazaar [73] qui constituent un point d'intérêt de plusieurs chercheurs [74–78]. Zhao et al. [76] ont discuté sur le parallélisme de granularité fine pour le codage intra HEVC, dans lequel une station de travail HP contenant 8 cœurs : Intel (R) Xeon (R) CPU E5-2670@2,6 GHz, avec une mémoire de 24 Go, a été utilisée pour implémenter et évaluer le codeur x265. L'encodeur Kvazaar HEVC, est un code source libre académique [79] sous licence GNU GPLv2.1. Il est implémentée en langage C avec des optimisations en assembleur, et utilise le schéma de codage HM comme référence. L'utilisation du codeur Kvazaar a pour principal avantage de prendre en charge le multithreading et divers types de granularité de parallélisme, notamment les niveaux Tile, Slice, frame et WPP. Dans notre précédent travail [80], nous avons présenté différentes implémentations parallèles du codeur Kvazaar HEVC sur une plate-forme Octa-core CubieBoard4, comprenant deux processeurs quad-core, ARM A7 et ARM A15 dans une seule puce. Cependant, la plupart des études précédentes ne traitent pas le parallélisme de HM, ou utilisent des plateformes sophistiquées et coûteuses.

Dans cette partie de thèse, nous explorons la possibilité d'accélérer le codage HEVC grâce à une implémentation parallèle. La réalisation en parallèle du codeur HEVC étant la clé pour réduire le temps de codage, nous adoptons le paradigme OpenMP pour la parallélisation à un niveau grossier de granularité (parallélisation au niveau images). Ce choix de granularité est plus approprié pour la configuration AI, où aucune synchronisation sophistiquée des threads n'est requise. Outre l'amélioration du temps d'encodage, notre approche est implémentée sur une plateforme moins chère nommée

Cubieboard4, qui facilite l'intégration de notre encodeur dans les systèmes embarqués. Dans le but d'évaluer les performances de la méthode proposée sur des plateformes multicœurs homogènes, des tests supplémentaires sur un serveur IBM® x3550 ont été effectués.

4.5 Description de l'algorithme proposé pour l'implémentation parallèle

L'une des principales caractéristiques du HEVC est la prise en charge du parallélisme au niveau de l'encodage ou du décodage. Plusieurs outils sont disponibles dans le fichier de configuration de l'encodage afin de permettre le traitement en parallèle, tels que : Slice, Tile et WPP. Cependant, le logiciel de référence ne met en œuvre aucune stratégie de parallélisme, et seule la mesure des performances est possible à travers les options Slice, Tile et WPP. Cet aspect est l'inconvénient majeur du HM et, pour cette raison, d'autres logiciels tels que x265, f265 et Kvazaar ont été développés.

Les solutions pour l'implémentation parallèle du codeur HEVC sont très limitées, et la plupart d'entre elles concernent les logiciels x265, f265 et Kvazaar. Par contre, notre proposition de parallélisation vise le software de référence HM, pour ses performances par rapport aux autres logiciels. Nous avons choisi un niveau de granularité du parallélisme relativement élevé, dans lequel un nombre donné d'images peut être encodé en parallèle sous la configuration AI.

La Figure 4.7 décrit les principales étapes utilisées pour réaliser une implémentation parallèle d'un codeur AI HEVC. L'étape d'initialisation est maintenue à partir du logiciel de référence HM. Ensuite, les paramètres collectés à partir du fichier de configuration (.cfg) sont partagés entre tous les threads de traitement (cœurs). Avant d'entamer le traitement parallèle, nous avons dupliqué les classes et les variables du codeur afin de permettre le codage en parallèle des images indépendantes. En conséquence, la lecture à partir du fichier d'entrée est effectuée pour un ensemble d'images, où le nombre d'images lues à chaque itération dépend du nombre d'unités de traitement. Le codage parallèle débute effectivement au moment de l'appel en parallèle de la fonction « encode » (fork node). Ainsi, chaque thread (un

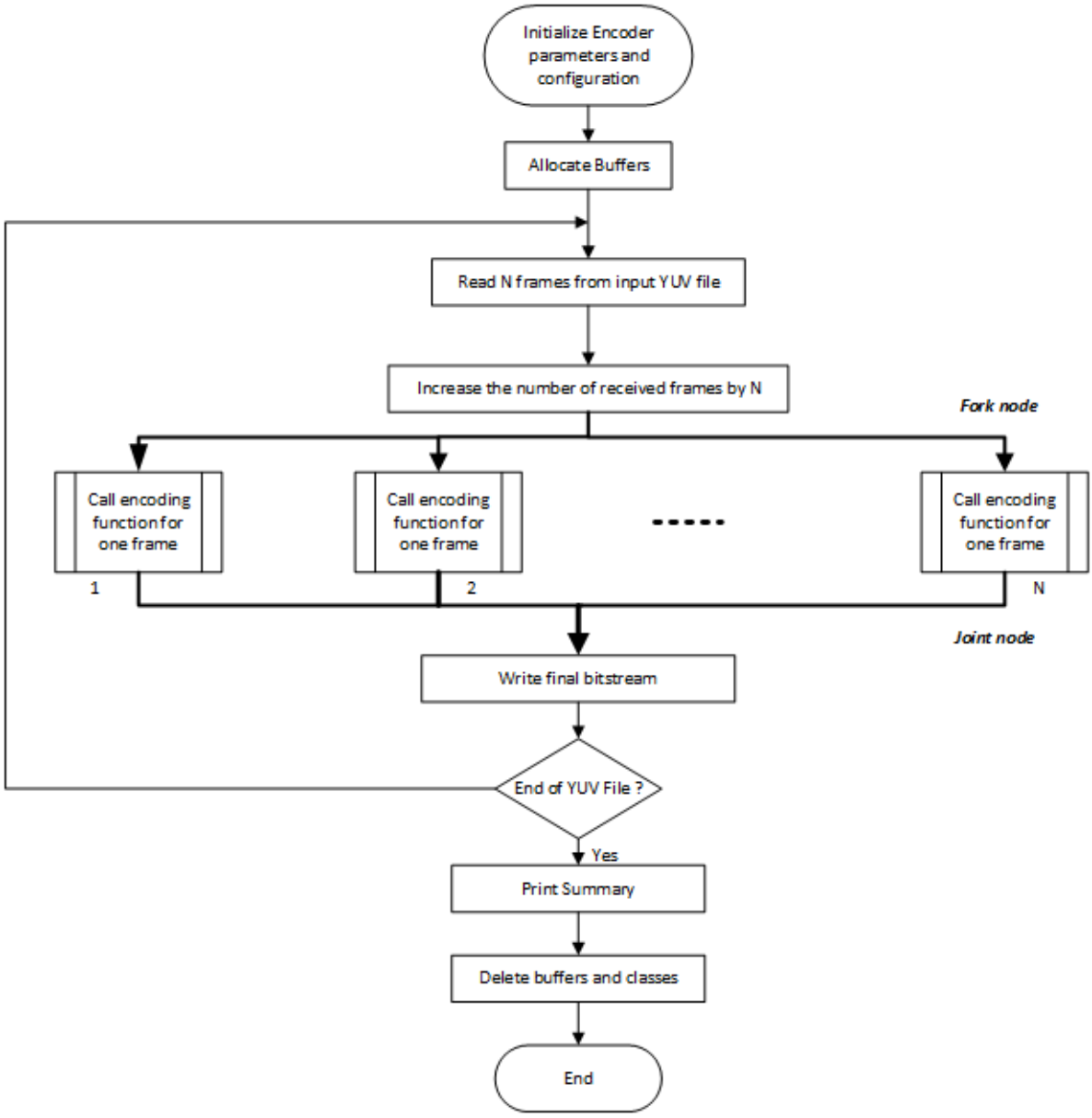


Figure 4.7. L'algorithme proposé pour le codage AI HEVC parallèle.

thread par processeur) encode une image d'entrée privée pour produire un bitstream de sortie privé. Une fois le codage parallèle établi, la construction du bitstream final est effectuée en ordonnant les bitstreams élémentaires résultants des différents threads, dans le même ordre chronologique d'images du fichier d'entrée. Cette synchronisation au niveau du « joint-node » est très importante pour assurer le décodage correct du bitstream final. L'affichage récapitulatif de performances est effectué lorsque toutes les images du fichier d'entrée sont codées. Toutes les performances de codage, y compris le type de Slice, le PSNR et le débit, sont donc affichés. Enfin, tous les tampons et classes utilisés dans l'encodeur sont supprimés, pour libérer la mémoire.

Analyse de dépendance des données :

Plusieurs problèmes doivent être pris en compte lors du passage d'une exécution séquentielle à une exécution parallèle. Ces problèmes proviennent de la dépendance des données dans des régions parallèles. Etant donné que les régions parallèles peuvent avoir plusieurs appels de fonction imbriqués, un alias de pointeur peut survenir au moment de l'exécution, et compliquer le contrôle de l'exécution en augmentant le risque d'erreur. Dans notre cas, le parallélisme approprié est choisi à travers les itérations de la boucle FOR. Une source primordiale d'erreurs, dans OpenMP et dans nombreux autres modèles de programmation parallèle, apparaît dans la phase où les programmeurs doivent affirmer si les itérations peuvent être parallélisées. Pour cela, une évaluation approfondie de la dépendance des variables à travers les itérations de boucles, est capitale. La présence de dépendance entre les données au cours des itérations d'une boucle, rend le parallélisme difficile, voire impossible des fois.

Une solution classique pour exécuter un programme séquentiel sur une machine dotée de plusieurs processeurs, consiste à insérer plusieurs points de synchronisation dans le code, afin de garantir le bon ordre d'exécution et d'éviter tout type de « Race conditions » ou « Data races ». Une situation de concurrence critique survient lorsque deux ou plusieurs threads tentent d'accéder ou de modifier des données partagées, en même temps. Cependant, l'exclusion mutuelle peut être un grave problème, entraînant une baisse de performances lors de l'exécution parallèle.

Notre solution proposée dans le but de rendre le HM multithread est basée sur la duplication de modules d'encodage pour chaque trame afin d'éviter les problèmes liés à la synchronisation. De cette manière, chaque thread de codage a ses propres données et outils d'encodage pour produire une sortie indépendante (bitstream). Par conséquent, différents threads peuvent être amorcés simultanément. Pour ce faire, une boucle est installée pour un ensemble de threads correspondant au nombre d'unités de traitement disponibles, où chaque thread est chargé d'encoder une image à la fois. Techniquement, cela se fait en spécifiant un ordonnancement statique « static schedule» avec une taille de bloc de 1, signifiant que chaque thread traite une et une seule itération à la fois. La seule synchronisation incontournable se situe à la fin de la boucle comme barrière de boucle implicite. Elle a pour objectif, de construire le flux binaire final tout en assurant

l'ordre séquentiel des différents bitstreams élémentaires. La planification statique peut réduire les performances par rapport à la planification dynamique, mais c'est le seul moyen pour produire un bitstream final correct et conforme.

4.6 Résultats d'implémentation sur une plateforme hétérogène de type Cubieboard4

4.6.1 Description de la plateforme

Toutes les expériences ont été réalisées sur une plateforme Cubieboard4 basée sur un processeur d'architecture big.LITTLE Allwinner A80 Octa-Core A15/A7. La technologie Big.LITTLE allie la haute performance et l'efficacité dans la même puce, pour offrir une performance de traitement en parallèle accrue avec une consommation minimale d'énergie. Le circuit A80 est un processeur Octa-core comprenant un quad-core ARM Cortex ® A7 et un quad-core ARM Cortex ® A15. Le noyau Cortex-A15 est conçu pour atteindre des performances élevées, en exécutant davantage d'instructions en parallèle sur un pipeline plus grand et plus complexe. Le pipeline du noyau Cortex-A7 est relativement simple et conçu pour être extrêmement économe en énergie. Étant donné que les deux processeurs prennent en charge la même architecture ISA (Instruction Set Architecture) ARMv7-A, les mêmes instructions ou le même programme peuvent être exécutés de manière cohérente sur les deux processeurs. En utilisant la technologie big.LITTLE, chaque tâche peut être allouée de manière dynamique à un noyau «big» ou «LITTLE» en fonction des exigences instantanées de performance de cette tâche. Le Tableau 4.1 résume les principales fonctionnalités de la plateforme Cubiebord4 et de l'environnement de développement.

4.6.2 Résultats et discussions

Nous avons utilisé le paradigme OpenMP pour la mise en œuvre parallèle du codeur HEVC. La directive « #pragma omp parallel for » permet l'exécution en parallèle des itérations d'une boucle « for », où l'affectation d'itérations aux threads de traitement peut être effectuée de plusieurs manières. Dans notre cas, nous avons conservé le mode par défaut qui est la planification statique. Pour ce type de planification, la boucle est

Tableau 4.1. Caractéristiques de la plateforme Cubieboard4 et de l'environnement de développement

	Cubieboard4
Number of cores	02 Quad-core (08 cores)
CPU frequency	Cortex-A7 MPCore @1.2GHz , Cortex-A15 MPCore @1.8GHz
Memory	2 Go RAM
Operating System	Linux (Lubuntu)
Compiler version	GCC and G++ v4.5

divisée en segments de taille égale, ou aussi égale que possible dans le cas d'un nombre d'itérations de boucle non divisible de manière égale par le nombre de threads. Ce choix d'ordonnancement a été fait pour préserver l'ordre du codage temporel (TOC) des images successives. La principale limitation de la planification statique est la barrière implicite à la fin de chaque itération, où la synchronisation entre les threads est requise pour entamer une nouvelle itération. Ce problème est particulièrement perceptible lorsque les différentes itérations prennent différentes durées d'exécution, donc un temps de latence plus long.

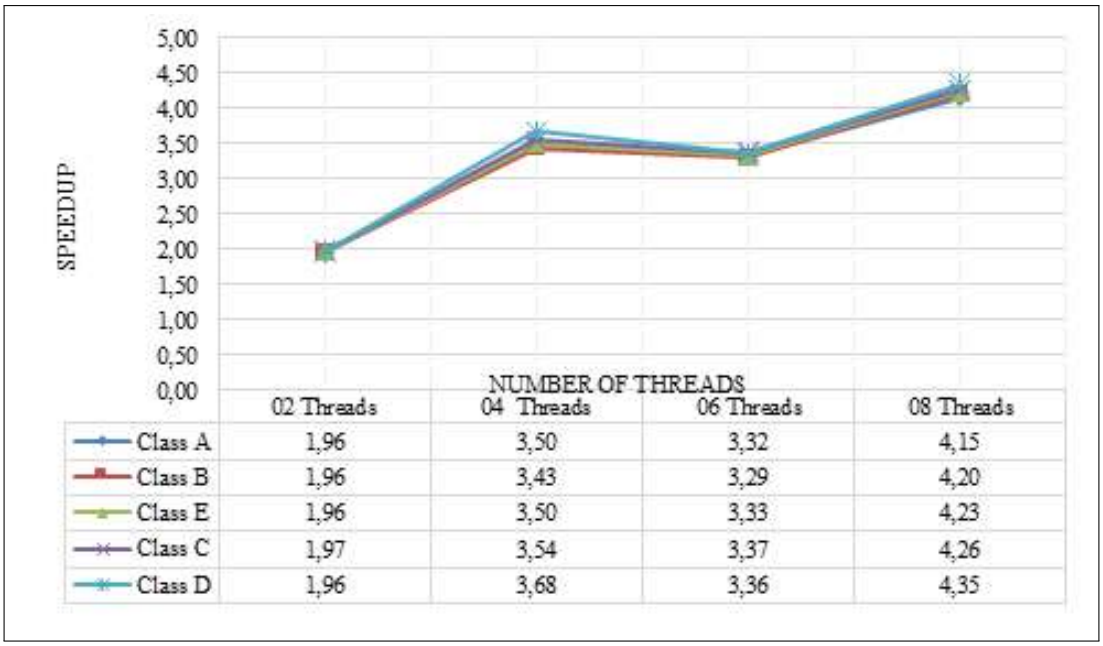
Pour décrire les performances de la réalisation en parallèle, nous avons effectué plusieurs expériences en utilisant un nombre de threads dans l'ensemble $\{1, 2, 4, 6, 8\}$. Il est à noter que l'efficacité de la compression n'est pas affectée par le traitement parallèle (le débit et le PSNR sont maintenus sans altération). Les résultats globaux pour le codage parallèle sous la configuration AI, sont résumés dans le Tableau 4.2. Afin d'évaluer l'efficacité de la compression, la réduction de temps s'exprime de la manière suivante :

$$Speedup = \frac{T_{Sequential}}{T_{Parallel}} \tag{4.1}$$

Pour une meilleure représentation, nous avons également illustré l'accélération en fonction du nombre de threads dans la Figure 4.8. En traitement parallèle, l'idéal est d'atteindre une accélération égale au nombre d'unités de traitement, c'est-à-dire pour un processeur à N cœurs, le gain maximal est égal à N. En regardant les résultats obtenus, le gain fourni avec deux threads est de 1,96 en moyenne, ce qui est presque égal à l'accélération idéale (pour deux threads). La légère différence est due au fait

Tableau 4.2. Performances de l'implémentation parallèle de l'encodeur HEVC en configuration AI

		Plateforme hétérogène (Cubieboard4)				Plateforme homogène (IBM® System x3550 M3)			
Classes	Sequences	02 Threads	04 Threads	06 Threads	08 Threads	02 Threads	04 Threads	06 Threads	08 Threads
Class A 2560×1600	Traffic	1,96	3,56	3,32	4,19	1,99	3,92	5,69	7,55
	PeopleOnStreet	1,96	3,43	3,31	4,11	1,99	3,93	5,71	7,54
Average A		1,96	3,50	3,32	4,15	1,99	3,92	5,70	7,54
Class B 1920×1080	Kimono1	1,94	3,40	3,32	4,28	1,99	3,93	5,68	7,60
	BQTerrace	1,96	3,41	3,33	4,24	1,98	3,95	5,72	7,58
	Cactus	1,96	3,42	3,32	4,12	1,99	3,94	5,70	7,61
	ParkScene	1,97	3,51	3,27	4,33	1,98	3,94	5,72	7,64
	BasketballDrive	1,96	3,42	3,22	4,05	1,98	3,97	5,74	7,59
Average B		1,96	3,43	3,29	4,20	1,98	3,95	5,71	7,60
Class E 1280×720	Vidyo1	1,96	3,59	3,37	4,34	1,99	3,88	5,72	7,60
	Vidyo3	1,96	3,45	3,31	4,17	1,99	3,96	5,73	7,59
	Vidyo4	1,97	3,47	3,30	4,19	1,99	3,91	5,72	7,60
Average E		1,96	3,50	3,33	4,23	1,99	3,92	5,73	7,60
Class C 832×480	BasketBallDrill	1,99	3,68	3,37	4,28	1,98	3,96	5,77	7,49
	BQMall	1,97	3,54	3,38	4,28	1,99	3,96	5,78	7,58
	RaceHorses	1,98	3,46	3,37	4,20	1,99	3,95	5,69	7,59
	PartyScene	1,95	3,50	3,35	4,30	1,99	3,97	5,81	7,55
Average C		1,97	3,54	3,37	4,26	1,99	3,96	5,76	7,55
Class D 416×240	BasketballPass	1,96	3,72	3,45	4,46	2,00	3,94	6,10	7,63
	RaceHorses	1,96	3,76	3,35	4,39	1,98	3,96	5,94	7,92
	BlowingBubbles	1,98	3,64	3,31	4,30	2,00	4,09	6,00	7,50
	BQSquare	1,96	3,58	3,35	4,25	1,98	3,96	5,71	8,08
Average D		1,96	3,68	3,36	4,35	1,99	3,99	5,94	7,78
Average		1,96	**3,53**	**3,33**	**4,23**	**1,99**	**3,95**	**5,77**	**7,62**



	02 Threads	04 Threads	06 Threads	08 Threads
Class A	1,96	3,50	3,32	4,15
Class B	1,96	3,43	3,29	4,20
Class E	1,96	3,50	3,33	4,23
Class C	1,97	3,54	3,37	4,26
Class D	1,96	3,68	3,36	4,35

Figure 4.8. Accélération de la réalisation parallèle avec Cubieboard4.

que certaines parties de code doivent être exécutées séquentiellement (regroupement de sous-bitstreams), en plus de la barrière de synchronisation implicite entre les différents threads à la fin de chaque itération de boucle «omp for». L'utilisation de quatre threads donne une accélération moyenne de $3,53$. La différence entre cette valeur et la valeur idéale (de quatre) est un peu plus grande par rapport au cas de deux threads, cela concerne essentiellement la partie séquentielle qui prend en charge le regroupement de quatre sous-bitstreams.

Pour assurer la crédibilité de la comparaison, il est très important de signaler que les tests précédents (avec un, deux et quatre threads) sont exécutés sur le même type de processeurs, qui est le Cortex-A15 MPCore dans notre cas. Le passage à plus de quatre threads nécessite l'utilisation du deuxième processeur MPCore Cortex-A7. L'accélération pour six threads était inattendue, après avoir constaté une dégradation des performances par rapport à l'utilisation de quatre threads. Le gain trouvé dans notre cas , est 3,33 en moyenne.

La même conclusion est affirmée avec huit threads, dont le gain est de 4,23 en moyenne. Pour expliquer ces deux derniers relativement aux premiers cas, nous nous inspirons du schéma de la Figure 4.9. Nous remarquons que les cœurs Cortex-A15 MPCore (cadencés à 1,8 GHz) parachèvent l'encodage et restent en attente pour les quatre cœurs Cortex-A7 (cadencés à 1,2 GHz). Par conséquent, tous les threads de

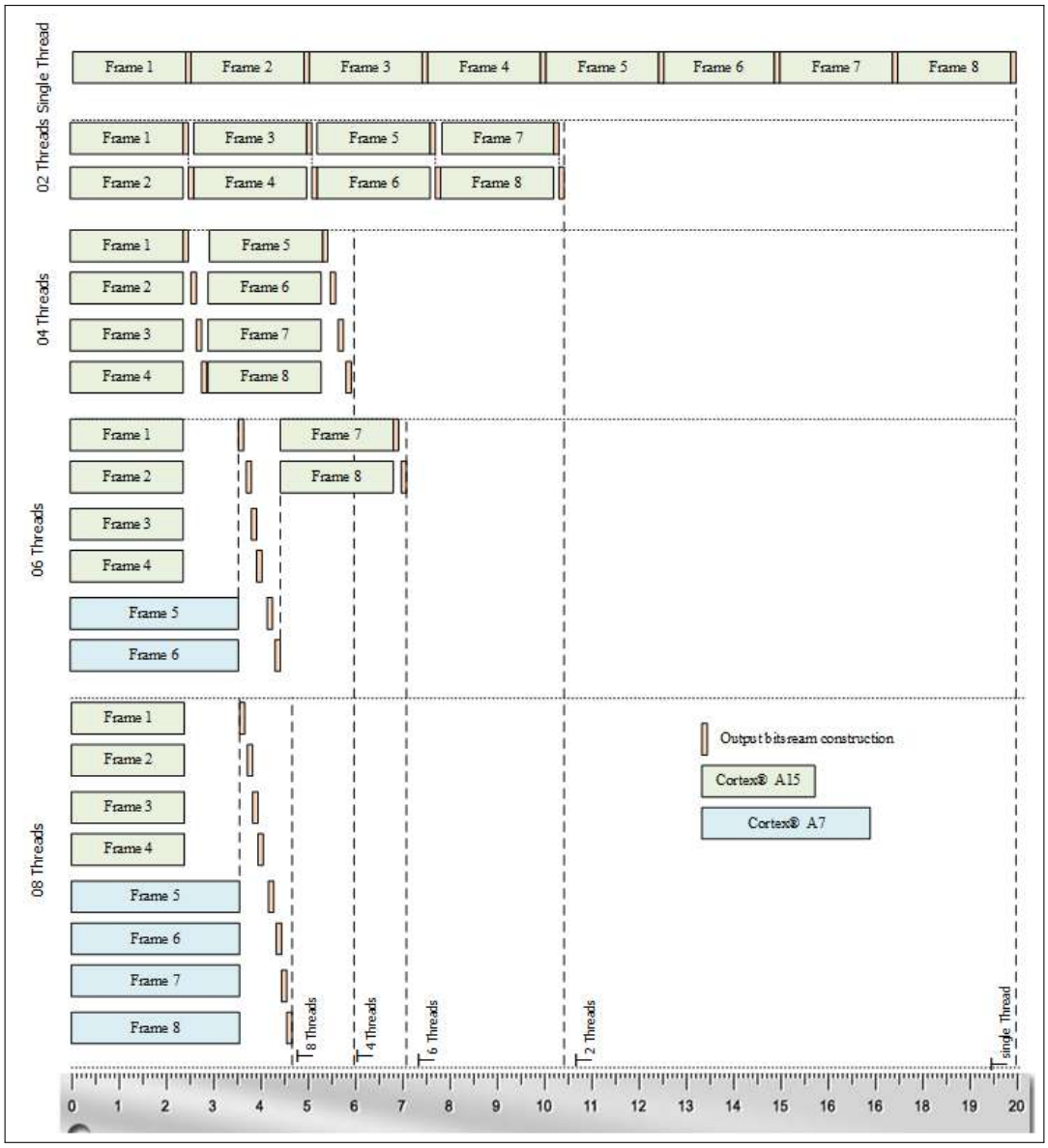


Figure 4.9. Représentation des threads d'exécution de la réalisation parallèle avec Cubieboard4.

traitement seront pénalisés par le temps d'encodage du thread le plus lent (s'exécutant sur l'un des cœurs du MPCore Cortex-A7). Dans l'exemple de la Figure 4.9, seulement huit images sont considérées, mais cela reste valable pour un plus grand nombre d'images. L'échelle de l'axe du temps sont données à titre indicatif. Avec cette illustration de la Figure 4.9, le résultat de six threads est devenu justifié ainsi que pour toutes les valeurs du Tableau 4.2. La charge de travail de chaque cœur est illustrée à la Figure 4.10, où figurent les noyaux numérotés de 1 à 4 du Cortex A7 (cadencé à 1,2 GHz) et les cœurs 5 à 8 du Cortex A15 (cadencé à 1,8 GHz). Nous constatons que l'utilisation de cœurs de hautes performances est privilégiée quel que soit le nombre

de threads.

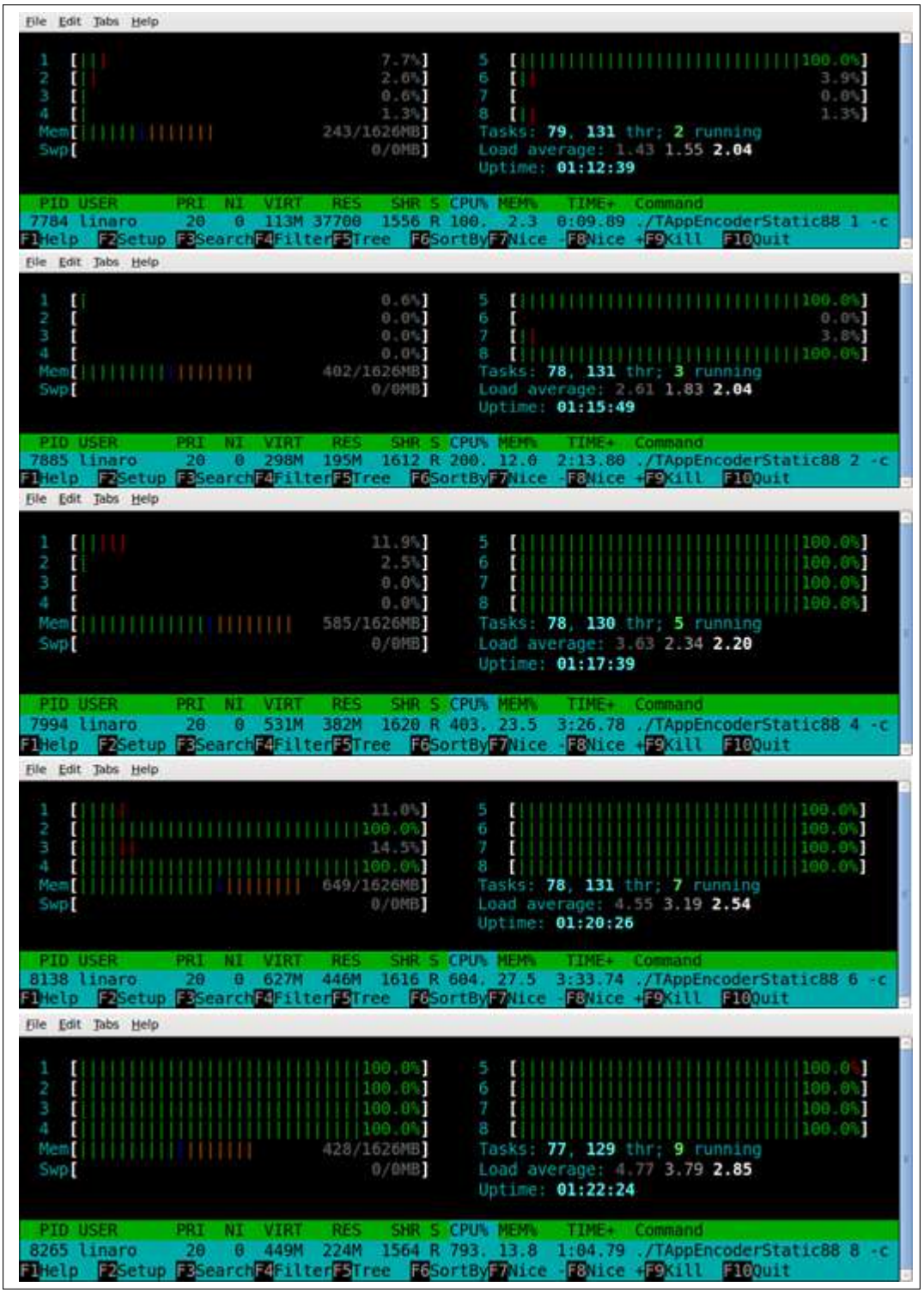


Figure 4.10. Charge de travail des processeurs utilisant un, deux, quatre, six et huit threads (du haut vers le bas).

4.7 Résultats d'implémentation sur une plateforme homogène de type HPC

4.7.1 Description de la plateforme

Pour montrer l'efficacité de l'algorithme proposé, nous avons enrichi notre étude avec d'autres expériences sur une plateforme homogène (serveur IBM® × 3550 M3). Ce serveur est composé de deux processeurs Intel Xeon X5675, chaque processeur contient six cœurs, Tableau 4.3 .

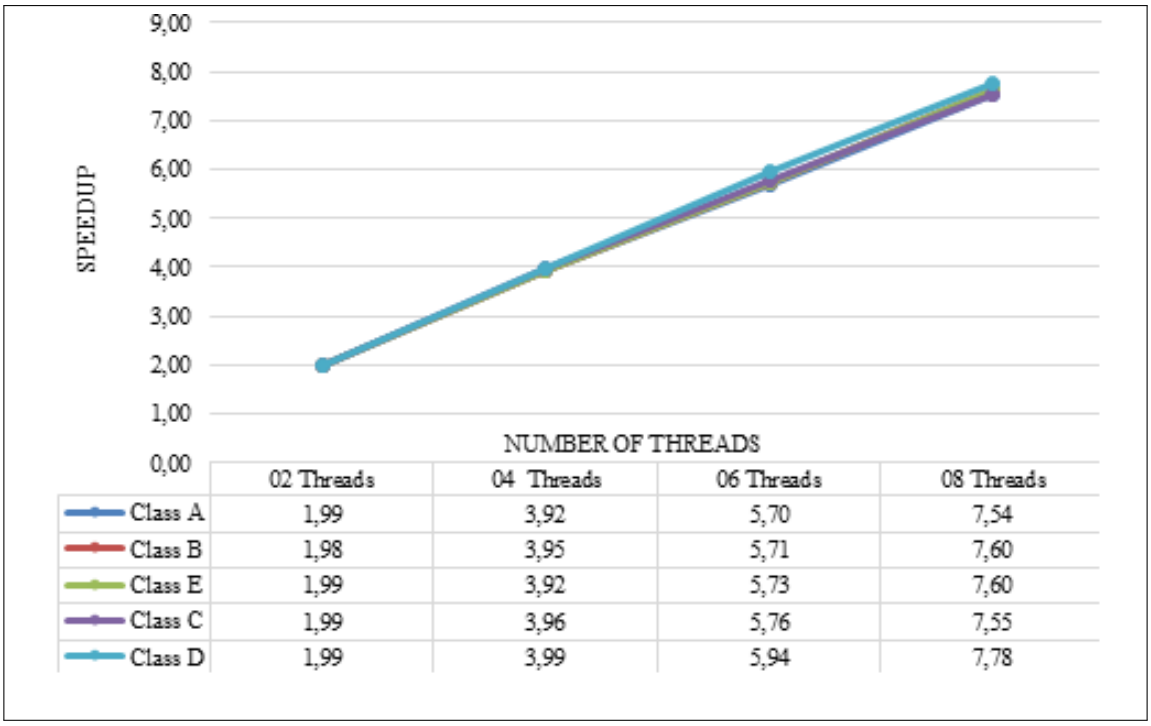
Tableau 4.3. Caractéristiques de la plateforme IBM® System x3550 M3 et de son environnement de développement

	IBM® System x3550 M3
Number of cores	02 Intel Xeon X5675 (12 cores)
CPU frequency	3.07 GHz
Memory	24 Go RAM
Operating System	Redhat 6.0 Enterprise Linux for HPC
Compiler version	GCC and G++ v4.4.7

4.7.2 Résultats et discussions

Nous avons répété les mêmes tests établis sur la plateforme Cubieboard4, limitée à huit cœurs seulement. Les résultats des expériences sur le serveur IBM® × 3550 sont également présentés dans le Tableau 4.2. Ces résultats montrent un gain de performance quasi idéal pour les différents nombres de threads. Contrairement aux résultats obtenus avec la plateforme Cubieboard4, Figure 4.8, la courbe de performance illustrée à la Figure 4.11 conserve sa monotonie de croissance et présente un gain proportionnel au nombre de threads.

Cebrián-Márquez et al. [81] ont proposé deux architectures pour le codage HEVC parallèle, utilisant le parallélisme au niveau des Tiles et des Slices. Leurs meilleurs résultats en terme de réduction du temps d'encodage, utilisant huit threads, ont été obtenus pour un parallélisme au niveau des Slices, où une accélération de 6,63x est enregistrée en moyenne pour les classes A, B, C et D, et 7,52x pour les classes A et



	02 Threads	04 Threads	06 Threads	08 Threads
Class A	1,99	3,92	5,70	7,54
Class B	1,98	3,95	5,71	7,60
Class E	1,99	3,92	5,73	7,60
Class C	1,99	3,96	5,76	7,55
Class D	1,99	3,99	5,94	7,78

Figure 4.11. Accélération de la réalisation parallèle avec IBM® × 3550 M3.

B. Un travail similaire également proposé par Cebrián-Márquez et al. [82] concerne la réalisation en parallèle du codeur HEVC à différents niveaux de parallélisme, y compris les niveaux WPP et GOP. Pour ce dernier travail, les meilleurs résultats d'accélération sont obtenus au niveau GOP. Une accélération moyenne de 4,45x est obtenue avec huit threads où quatre threads sont activés avec un multithreading simultané (SMT). De plus, il convient de noter que, pour les deux travaux cités dans [81, 82], la plateforme comprend un processeur graphique (GPU) couplé à un processeur multicœur. Malgré ces différences de plateformes utilisées, la comparaison des résultats avec les nôtres reste possible et montre que notre proposition dépasse en qualité obtenue, celle des algorithmes proposés dans [81, 82]. L'avantage principal de notre méthode est que le parallélisme n'introduit ni pénalité de débit (BR) ni perte de qualité (PSNR). De plus, notre algorithme peut être adaptatif à n'importe quelle plateforme multicœur, et il est exempt de toute spécification matérielle.

4.8 Conclusion

Dans ce chapitre, une réalisation parallèle et efficace du codeur intra HEVC est proposée. Nos expériences ont été menées sur une carte embarquée hétérogène,

comprenant deux ARM Cortex-A15 à quatre cœurs et un ARM Cortex-A7 à quatre cœurs également. En utilisant le paradigme OpenMP, une approche au niveau des images est proposée pour coder simultanément un ensemble d'images sans perte d'efficacité en codage. L'évaluation des performances pour la réalisation parallèle du codage intra-HEVC, conduit à une accélération de 4,23 en utilisant huit cœurs. Cette accélération est jugée insuffisante pour huit cœurs, ce qui montre l'inconvénient des plateformes hétérogènes dans l'exécution parallèle de tâches à charge équilibrée. Néanmoins, l'approche proposée peut être utilisée sur n'importe quelle plateforme multicœurs prenant en charge le paradigme multithreading et OpenMP. Cela a été prouvé par des tests supplémentaires sur un serveur IBM® × 3550, où les résultats obtenus étaient presque parfaits, atteignant une vitesse d'exécution de 7,62 × en utilisant huit threads. Nos résultats sont jugés meilleurs que ceux d'autres travaux actuels.

En perspective, la réalisation parallèle peut être améliorée en utilisant une granularité plus fine et lancer les tests sur une nouvelle plateforme, représentée par le HPC à 32 nœuds de notre université.

Conclusions et perspectives

Conclusions

Au cours des vingt dernières années, les applications multimédias ont connu un développement fulgurant. En parallèle, les normes de codage/décodage vidéo ont beaucoup évolué, pour prendre en charge le traitement du multimédia avec les technologies avancées disponibles. Les recherches actuelles en matière de compression vidéo portent essentiellement sur le codage des données multimédias, circulant sur Internet, ordinateurs de bureau et portables, tablettes et télévision. La résolution vidéo a tendance à être de 1280×720 pixels ou supérieure, avec une bande passante disponible de taille de plusieurs mégaoctets.

La norme de codage vidéo H.264/AVC a montré une nette amélioration de l'efficacité du codage vidéo. En 2013, les organismes ITU-T VCEG et ISO/IEC MPEG ont préparé l'arrivée de la nouvelle norme H.265/HEVC, qui devrait remplacer la norme H.264. Cependant, la nouvelle norme HEVC souffre d'un handicap majeur lié à la consommation des ressources mémoires et temps d'encodage.

L'objectif principal de cette thèse est l'optimisation de la norme HEVC avec de nouvelles techniques, qui peuvent servir dans les variantes à venir. Le rapport débit/distorsion a fait l'objet du critère de décision pour toute optimisation suggérée dans cette thèse.

Dans le chapitre 2 consacré à l'optimisation et implémentation sur DSP, nous avons utilisé le DSP TMS320C6678 comme cible pour implémenter l'encodeur HEVC. C'est l'un des DSPs les plus performants de Texas Instruments. Une implémentation de base a été menée constituant dans la littérature, une première

implémentation de l'encodeur HEVC dans un environnement DSP. Dans une seconde contribution, nous avons proposé une accélération de l'encodeur HEVC en utilisant les instructions SIMD. Cette optimisation concernait les modules de calcul de distorsion notamment SAD, SSE, HAD, repris en langage assembleur. L'avantage majeur de cette technique d'optimisation, réside dans la réduction du temps d'encodage sans altération des performances de compression. Les résultats expérimentaux montrent un gain atteignant 69% pour le module SAD, et un gain global de 15% en temps entier d'encodage. La combinaison de cette optimisation avec les options de codage rapide au niveau du fichier de configuration, permet d'aboutir à un gain de 78% au prix d'une augmentation inférieure à 4% en débit avec perte négligeable en PSNR.

La complexité de calcul dans la norme HEVC, s'explique principalement par la flexibilité et la diversité des tailles de blocs et d'unités de codage. Pour optimiser le mécanisme de partitionnement de l'encodeur HEVC, nous avons proposé un algorithme rapide permettant d'anticiper l'arrêt du processus de partitionnement et/ou éviter la prédiction inutile pour certaines profondeurs. La décision pour qu'un bloc soit exploré à une profondeur donnée, repose sur un prétraitement concernant la complexité de la texture dudit bloc. Dans notre algorithme, un bloc fortement texturé nécessite un partitionnement approfondi jusqu'à la taille minimale d'une unité CU (8×8). De plus, la prédiction pour les niveaux primaires (0 et 1) est inutile. En contre partie, pour un bloc homogène et moins texturé, le processus de partitionnement est interrompu à une légère profondeur. Les résultats de l'algorithme proposé pour le partitionnement rapide, ont permis de réduire en moyenne 41% du temps d'encodage avec des incréments moyens de 1,75% et 0,69% dans BD-BR et BR,respectivement. Une perte négligeable du PSNR est également ressentie.

Le chapitre 4 présente la réalisation parallèle de l'encodeur HEVC comme une troisième optimisation proposée le long de cette thèse. Le parallélisme consiste à mettre en œuvre des algorithmes permettant de traiter des informations de manière simultanée. Cette technique a pour but de réaliser un plus grand nombre d'opérations, en un temps minime. Néanmoins, pas tous les algorithmes peuvent être parallélisables. La dépendance des données entre-elles est un facteur primordial, lors de la préparation des tâches à exécuter en parallèle. La structure du software de référence HM est donnée sous forme d'un ensemble de modules imbriqués, dont les variables (données)

sont souvent partagées et modifiées en plusieurs paliers par différents modules. Ce type de structures ne supporte pas le parallélisme ou nécessite une réorganisation et/ou relocation des variables, afin de supprimer toute dépendance ou alias possible entres les données rapportées aux différents threads. Dans notre cas, nous avons utilisé le paradigme OpenMP pour le parallélisme au niveau image. Ce choix de niveau du parallélisme sollicite la configuration AI, pour permettre l'encodage d'une image indépendamment des autres. Malgré cela, une duplication de certaines variables et structures de données reste nécessaire pour maintenir le fonctionnement ordinaire de l'encodeur. En outre, une synchronisation est aussi nécessaire afin de garantir l'ordre de transcription des sous-bitsreams issus des différents threads, pour former un bitstream final conforme. Les résultats expérimentaux de cette proposition, ont abouti à une accélération de 4,23 en utilisant huit cœurs. Cette accélération est jugée insuffisante par rapport au nombre de cœurs utilisés, ce qui montre l'inconvénient des plateformes hétérogènes dans l'exécution parallèle de tâches à charge équilibrée. Toutefois, l'approche proposée est transportable à n'importe quelle plateforme multicœurs, prenant en charge le paradigme multithreading et OpenMP. Cela a été prouvé par des tests supplémentaires sur un serveur IBM® × 3550, où les résultats obtenus sont presque parfaits, atteignant une vitesse d'exécution de 7,62 × avec huit threads.

Perspectives pour les travaux futurs

Pour poursuivre le travail effectué dans cette thèse, plusieurs perspectives intéressantes peuvent être suggérées. L'optimisation proposée utilisant un DSP en assembleur avec emploi d'instructions SIMD, peut être étendue à d'autres modules et fonctions que ceux de SAD, SSE et HAD. Bien que l'effort dépensé dans la programmation en assembleur est trop élevé, l'accélération résultante n'induit pas de pertes en efficacité de compression. Par ailleurs, l'extension de l'implémentation sur DSP pour une exécution sur plusieurs cœurs est un autre domaine d'investissement, pouvant conduire à des gains considérables en temps d'encodage.

L'optimisation algorithmique considérée dans cette thèse, concerne uniquement le processus de partitionnement des unités CU. Il est toutefois possible de combiner notre proposition avec d'autres algorithmes, traitants les modes de prédiction et/ou

l'estimation du mouvement. De cette manière, un gain cumulatif peut être obtenu.

Finalement, l'implémentation du parallélisme peut être amendé en utilisant une granularité plus fine avec un paradigme approprié pour le parallélisme, plutôt qu'avec OpenMP. Un ajout intéressant serait une combinaison du parallélisme avec d'autres optimisations algorithmiques.

Production Scientifique

A- Publications dans des revues

1- **Maazouz Mohamed**, Batel Noureddine, Bahri Nejmeddine, Masmoudi Nouri. A DSP-Based Implementation of HEVC Encoder. International Review on Computers and Software (IRECOS), Praise Worthy Prize, 2016, 11 (10), (doi :10.15866/irecos.v11i10.10172).

2- **Maazouz, Mohamed**, Batel, Noureddine, Bahri, Nejmeddine, et al. Homogeneity-based fast CU partitioning algorithm for HEVC intra coding. Engineering Science and Technology, an International Journal, 2019. (doi :10.1016/j.jestch.2018.12.016)

B- Communication à une Conférence internationale avec acte

1- **Maazouz Mohamed**, Bahri Nejmeddine, Batel Noureddine, Toubal Abdelmoughni and Masmoudi Nouri, "Parallel implementation of Kvazaar HEVC on multicore ARM processor," 2016 8th International Conference on Modelling, Identification and Control (ICMIC), Algiers, 2016, pp. 1086-1091.(doi : 10.1109/ICMIC.2016.7804275)

2- Bahri Nejmeddine, El Ansari Abdessamad, **Maazouz Mohamed**, Ahaitouf Ali and Masmoudi Nouri, "HEVC video encoder implementation on Texas Instruments platforms," 2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), Sousse, 2016, pp. 165-170. (doi : 10.1109/STA.2016.7952073)

3- **Maazouz Mohamed**, Bahri Nejmeddine, Batel Noureddine, Toubal Abdelmoughni and Masmoudi Nouri, "Performance Evaluation of Frame-level Parallelization in HEVC Intra Coding Using Heterogeneous Multicore Platforms," 2018 International Conference on Applied Smart Systems (ICASS), Medea, Algeria, 2018, pp. 1-6. (doi : 10.1109/ICASS.2018.8652076)

Bibliographie

[1] A. A. Batalla, "Multiple transforms for video coding," Ph.D. dissertation, INSA de Rennes, dec 2015.

[2] K. Rao, D. N. Kim, J. J. Hwang *et al.*, *Video coding standards*. Springer, 2014.

[3] W. Gao and S. Ma, *Advanced Video Coding Systems*. Springer International Publishing, 2014.

[4] J. L. Ling Guan, Sun-Yuan Kung, *Multimedia image and video processing*. CRC press, 2001.

[5] "Telecommunication Standardization Sector (ITU-T)," ITU-T, Tech. Rep.

[6] "liaisons ISO and the UN Working together for international standardization," International Organization for Standardization, Tech. Rep., 2018.

[7] *Electrical installation handbook Protection, control and electrical devices*. ABB SACE, Bergamo (Italy), 2010.

[8] "Welcome to the IEC International Electrotechnical Commission," International Electrotechnical Commission /IEC, Geneva, Switzerland., Tech. Rep., 2014.

[9] R. Garcia, "Hevc Optimization in Mobile Environments," Ph.D. dissertation, Florida Atlantic University, May 2014.

[10] M. Igarta, "a Study of Mpeg-2 and H.264 Video Coding," Ph.D. dissertation, Purdue University, Dec 2004.

[11] M. Wien, *High Efficiency Video Coding*, ser. Signals and Communication Technology. Berlin, Heidelberg : Springer, 2015.

[12] B. Vittorio, O. Jens-Rainer, and S. Gary, "Report of subjective test results of responses to the joint call for proposals (cfp) on video coding technology for high efficiency video coding (hevc)," in *Document :JCTVC-A204, JCT-VC of ITU-T and ISO/IEC*, Dresden, DE, 15-23 April, 2010.

[13] W. Thomas, H. Woo-Jin, B. Benjamin, O. Jens-Rainer, and G. J. Sullivan, "Wd1 : Working draft 1 of high-efficiency video coding," in *Document :JCTVC-C403, JCT-VC of ITU-T and ISO/IEC*, Guangzhou, CN, 7-15 October, 2010.

[14] F. Bossen, "Common test conditions and software reference configurations," in *Document : JCTVC-L1100, JCT-VC of ITU-T and ISO/IEC*, Geneva, CH, 14 – 23 January, 2013.

[15] G. J. Sullivan, J.-r. Ohm, W.-j. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[16] W. Gao and S. Ma, *Advanced Video Coding Systems*. Springer International Publishing, 2014, ch. 2 : Video Coding Basic Principle.

[17] I. K. Kim, J. Min, T. Lee, W. J. Han, and J. H. Park, "Block partitioning structure in the HEVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697–1706, 2012.

[18] V. Sze, M. Budagavi, and S. Gary J, *High Efficiency Video Coding (HEVC) : Algorithms and Architectures*, ser. Integrated Circuits and Systems. Springer International Publishing, 2014.

[19] Y. H. Chiu, K. L. Chung, W. N. Yang, C. H. Lin, and Y. H. Huang, "Universal intra coding for arbitrary RGB color filter arrays in HEVC," *Journal of Visual Communication and Image Representation*, vol. 24, no. 7, pp. 867–884, 2013.

[20] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, dec 2012.

[21] Y. Shan, "Fast Intra-frame Coding Algorithm for HEVC Based on TCM and Machine Learning," Master Dissertation, University of Waterloo, Ontario, Canada, 2016.

[22] J. Lainema, F. Bossen, W. J. Han, J. Min, and K. Ugur, "Intra coding of the HEVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1792–1801, 2012.

[23] L. F. Rosário Lucas, E. A. Barros da Silva, S. M. Maciel de Faria, N. M. Morais Rodrigues, and C. Liberal Pagliari, *Efficient Predictive Algorithms for Image Compression*. Springer International Publishing, 2017.

[24] G. Corrêa, P. Assunção, L. Agostini, and L. A. d. S. Cruz, *Complexity-Aware High Efficiency Video Coding*. Springer International Publishing, 2016.

[25] K. K. Barik, S. Sengupta, and M. R. Jena, "Quarter-Pixel Accuracy Motion Estimation (ME) - A Novel ME Technique in HEVC," *International Transaction of Electrical and Computer Engineers System*, vol. 2, no. 3, pp. 107–113, 2014.

[26] B. Bing, *Next-Generation Video Coding and Streaming.* Hoboken, NJ, USA : John Wiley & Sons, Inc, sep 2015.

[27] J. Sole, R. Joshi, N. Nguyen, T. Ji, M. Karczewicz, G. Clare, F. Henry, and A. Duenas, "Transform Coefficient Coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1765–1777, dec 2012.

[28] M. Gao, X. Fan, D. Zhao, and W. Gao, "An enhanced entropy coding scheme for HEVC," *Signal Processing : Image Communication*, vol. 44, pp. 108–123, may 2016.

[29] J. D. Ser, *Recent Advances on Video Coding.* InTech, Croatia, Jul 2011.

[30] J.-A. Choi and Y.-S. Ho, "High Throughput Entropy Coding in the HEVC Standard," *Journal of Signal Processing Systems*, vol. 81, no. 1, pp. 59–69, oct 2015.

[31] A. Norkin, G. Bjøntegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van Der Auwera, "HEVC deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746–1754, 2012.

[32] M. T. Pourazad, C. Doutre, M. Azimi, and P. Nasiopoulos, "HEVC : The New Gold Standard for Video Compression : How Does HEVC Compare with H.264/AVC ?" *IEEE Consumer Electronics Magazine*, vol. 1, no. 3, pp. 36–46, 2012.

[33] T. Watanabe, N. Wada, G. Yasuda, A. Tanizawa, T. Chujoh, and T. Yamakage, "In-loop filter using block-based filter control for video coding," *Proceedings - International Conference on Image Processing, ICIP*, pp. 1013–1016, 2009.

[34] D. F. De Souza, A. Ilic, N. Roma, and L. Sousa, "HEVC in-loop filters GPU parallelization in embedded systems," *Proceedings - 2015 International Conference on Embedded Computer Systems : Architectures, Modeling and Simulation, SAMOS 2015*, no. Samos Xv, pp. 123–130, 2015.

[35] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample Adaptive Offset in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, dec 2012.

[36] K. Il-Koo, M. Ken, S. Kazuo, B. Benjamin, and H. Woo-Jin, "High efficiency video coding (hevc) test model 10 (hm10) encoder description," in *Document : JCTVC-L1002_v3, JCT-VC of ITU-T and ISO/IEC*, Geneva, CH, 14 – 23 January, 2013.

[37] G. Bjontegaard, "Calculation of average psnr differences between rd-curves," in *Document : VCEG-M33, Video Coding Experts Group (VCEG)*, Austin, Texas, USA,, 2-4 April, 2001.

[38] "TMS320C6678 - Multicore Fixed and Floating-Point Digital Signal Processor," Texas Instruments, Tech. Rep. March, 2014. [Online]. Available : www.ti.com/lit/ds/symlink/tms320c6678.pdf

[39] "Code Composer Studio White Paper," Texas Instruments, Tech. Rep. May, 1999.

[40] K. Rao and N. Ahmed, *Orthogonal transforms for digital signal processing*, vol. 1. [Online]. Available : http://ieeexplore.ieee.org/document/1170121/

[41] S. Kim, D.-K. Lee, C.-B. Sohn, and S.-J. Oh, "Fast motion estimation for HEVC with adaptive search range decision on CPU and GPU," in *2014 IEEE China Summit & International Conference on Signal and Information Processing (ChinaSIP)*. IEEE, jul 2014, pp. 349–353. [Online]. Available : http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6889262

[42] D. Ruiz, G. Fernández-Escribano, V. Adzic, H. Kalva, J. L. Martínez, and P. Cuenca, "Fast CU partitioning algorithm for HEVC intra coding using data mining," *Multimedia Tools and Applications*, vol. 76, no. 1, pp. 861–894, 2017. [Online]. Available : http://dx.doi.org/10.1007/s11042-015-3014-6

[43] H.-S. Kim and R.-H. Park, "Fast CU Partitioning Algorithm for HEVC Using an Online-Learning-Based Bayesian Decision Rule," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 130–138, jan 2016. [Online]. Available : http://ieeexplore.ieee.org/document/7122899/

[44] X. Shang, G. Wang, T. Fan, and Y. Li, "Fast CU size decision and PU mode decision algorithm in HEVC intra coding," in *2015 IEEE International Conference on Image Processing (ICIP)*, vol. 2015-Decem. IEEE, sep 2015, pp. 1593–1597. [Online]. Available : http://ieeexplore.ieee.org/document/7351069/

[45] N. Dhollande, X. Ducloux, O. Le Meur, and C. Guillemot, "Fast block partitioning method in HEVC intra coding for UHD video," *5th IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin 2015*, pp. 10–11, 2016.

[46] T. Zhang, M. T. Sun, D. Zhao, and W. Gao, "Fast Intra-Mode and CU Size Decision for HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 8, pp. 1714–1726, 2017.

[47] Y. Yao, X. Li, and Y. Lu, "Fast intra mode decision algorithm for HEVC based on dominant edge assent distribution," *Multimedia Tools and Applications*, vol. 75, no. 4, pp. 1963–1981, feb 2016. [Online]. Available : http://link.springer.com/10.1007/s11042-014-2382-7

[48] W. Shi, X. Jiang, T. Song, and T. Shimamoto, "Edge Information Based Fast Selection Algorithm for Intra Prediction of HEVC," in *Circuits and Systems (APCCAS), 2014 IEEE Asia Pacific Conference on*. IEEE, nov 2014, pp. 17–20. [Online]. Available : http://ieeexplore.ieee.org/document/7032708/

[49] F. Belghith, H. Kibeya, M. A. Ben Ayed, and N. Masmoudi, "Fast coding unit partitioning method based on edge detection for HEVC intra-coding," *Signal, Image and Video Processing*, vol. 10, no. 5, pp. 811–818, 2016.

[50] S. Na, W. Lee, and K. Yoo, "Edge-based fast mode decision algorithm for intra prediction in HEVC," in *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*. IEEE, jan 2014, pp. 11–14. [Online]. Available : http://ieeexplore.ieee.org/document/6775887/

[51] K. Goswami, B.-G. Kim, D. Jun, S.-H. Jung, and J. S. Choi, "Early Coding Unit-Splitting Termination Algorithm for High Efficiency Video Coding (HEVC)," *ETRI Journal*, vol. 36, no. 3, pp. 407–417, jun 2014. [Online]. Available : http://doi.wiley.com/10.4218/etrij.14.0113.0458

[52] M. Asfandyar, M. Nawaz, and M. Hussain, "Accelerated CU decision based on enlarged CU sizes for HEVC UHD videos," in *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*. IEEE, aug 2016, pp. 374–378. [Online]. Available : http://ieeexplore.ieee.org/document/7888287/

[53] S. Ahn, B. Lee, and M. Kim, "A novel fast CU encoding scheme based on spatiotemporal encoding parameters for HEVC inter coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 3, pp. 422–435, 2015.

[54] P. P. Acharjya, R. Das, and D. Ghoshal, "Study and Comparison of Different Edge Detectors for Image Segmentation," *Global Journal of Computer Science and Technology Graphics and Vision*, no. 13, pp. 29–32, 2012. [Online]. Available : https://globaljournals.org/GJCST{_}Volume12/5-Study-and-Comparison-of-Different-Edge.pdf

[55] Jongho Kim, Yoonsik Choe, and Yong-Goo Kim, "Fast Coding Unit size decision algorithm for intra coding in HEVC," in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, vol. E99.D, no. 7. IEEE, jan 2013, pp. 637–638. [Online]. Available : https://www.jstage.jst.go.jp/article/transinf/E99.D/7/E99.D{_}2015EDL8231/{_}articlehttp://ieeexplore.ieee.org/document/6487050/

[56] M. Ramezanpour Fini and F. Zargari, "Two stage fast mode decision algorithm for intra prediction in HEVC," *Multimedia Tools and Applications*, vol. 75, no. 13, pp. 7541–7558, 2016. [Online]. Available : http://dx.doi.org/10.1007/s11042-015-2675-5

[57] D. G. Fernandez, A. A. Del Barrio, G. Botella, and C. Garcia, "Fast CU size decision based on temporal homogeneity detection," *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, vol. 77, no. 5, pp. 1–6, nov 2016. [Online]. Available : http://ieeexplore.ieee.org/document/7845379/http://link.springer.com/10.1007/s11042-017-4503-6

[58] Kyungmin Lim, Jaeho Lee, Seongwan Kim, Sangyoun Lee, K. Lim, J. Lee, S. Kim, and S. Lee, "Fast PU Skip and Split Termination Algorithm for HEVC Intra Prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 8, pp. 1335–1346, aug 2015. [Online]. Available : http://ieeexplore.ieee.org/document/6983560/

[59] L. M. Silva and R. Buyya, "Parallel programming models and paradigms," *High Performance Cluster Computing : Architectures ...*, pp. 4–27, 1999. [Online]. Available : http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Parallel+Programming+Models+and+Paradigms{#}0

[60] R. v. d. P. Barbara Chapman, Gabriele Jost, *Barbara Chapman - Using OpenMP*. London, England : The MIT Press Cambridge, Massachusetts, 2008.

[61] M. N. Larsen, "Parallel Libraries to support High-Level Programming," Ph.D. dissertation, UNIVERSITY OF COPENHAGEN, 2013.

[62] E. L. William Gropp, *Using MPI : Portable parallel programming with the message-passing interface*, E. L. William Gropp, Ed. London, England : The MIT Press Cambridge, Massachusetts London, England, 2003, vol. 30, no. 9.

[63] U. Openmp, *Using OpenMP : portable shared memory parallel programming*, 2013, vol. 46, no. 02.

[64] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in HEVC," *IEEE Journal on Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 969–977, 2013.

[65] A. Fuldseth, "Tiles, joint collaborative team on video coding (jct-vc)," JCTVC-E408_r1, 5th Meeting : Geneva, CH, 16-23 March, 2011, Tech. Rep., 2011.

[66] H. Félix, "Wavefront parallel processing, joint collaborative team on video coding (jct-vc)," JCTVC-E196, 5th Meeting : Geneva, CH, 16-23 March, 2011, Tech. Rep., 2011.

[67] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, and F. Wu, "A highly parallel framework for HEVC Coding unit partitioning tree decision on many-core processors," *IEEE Signal Processing Letters*, vol. 21, no. 5, pp. 573–576, 2014.

[68] Chenggang Yan, Yongdong Zhang, Feng Dai, and Liang Li, "Highly Parallel Framework for HEVC Motion Estimation on Many-Core Platform," in *2013 Data Compression Conference*, no. figure 1. IEEE, mar 2013, pp. 63–72.

[69] P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres, "Slice-based parallel approach for HEVC encoder," *Journal of Supercomputing*, vol. 71, no. 5, pp. 1882–1892, 2015.

[70] P. Piñol, H. Migallón, O. López-Granado, and M. Malumbres, "Parallel strategies analysis over the HEVC encoder," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 671–683, nov 2014.

[71] Y. J. Ahn, T. J. Hwang, D. G. Sim, and W. J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," *Eurasip Journal on Image and Video Processing*, vol. 2014, pp. 1–19, 2014.

[72] (2018) x265 HEVC Encoder / H.265 Video Codec.

[73] "GitHub - ultravideo_kvazaar_ An open-source HEVC encoder." [Online]. Available : https://github.com/ultravideo/kvazaar

[74] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2-optimized Kvazaar HEVC intra encoder," *Proceedings - International Conference on Image Processing, ICIP*, vol. 2016-Augus, pp. 549–553, 2016.

[75] A. Koivula, M. Viitanen, J. Vanne, T. D. Hamalainen, and L. Fasnacht, "Parallelization of Kvazaar HEVC intra encoder for multi-core processors," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, vol. 2015-Decem. IEEE, oct 2015, pp. 1–6.

[76] Y. Zhao, S. Li, X. Wang, M. Chen, and J. Wang, "Efficient realization of parallel HEVC intra encoding," *Electronic Proceedings of the 2013 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2013*, pp. 4–9, 2013.

[77] A. Koivula, M. Viitanen, A. Lemmetti, J. Vanne, and T. D. Hamalainen, "Performance evaluation of Kvazaar HEVC intra encoder on Xeon Phi many-core processor," *2015 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2015*, pp. 1250–1254, 2016.

[78] M. Viitanen, A. Koivula, J. Vanne, and T. D. Hamalainen, "Live demonstration : Run-time visualization of Kvazaar HEVC intra encoder," *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2016-July, p. 454, 2016.

[79] M. Viitanen, A. Koivula, A. Lemmetti, J. Vanne, and T. D. Hamalainen, "Kvazaar HEVC encoder for efficient intra coding," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2015-July. IEEE, may 2015, pp. 1662–1665.

[80] M. Maazouz, N. Bahri, N. Batel, A. Toubal, and N. Masmoudi, "Parallel implementation of Kvazaar HEVC on multicore ARM processor," in *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*. IEEE, nov 2016, pp. 1086–1091. [Online]. Available : http://ieeexplore.ieee.org/document/7804275/

[81] G. Cebrián-Márquez, V. Galiano, H. Migallón, J. Martínez, P. Cuenca, and O. López-Granado, "Heterogeneous CPU plus GPU approaches for HEVC," *Journal of Supercomputing*, 2018. [Online]. Available : https://doi.org/10.1007/s11227-018-2353-2

[82] G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, M. Tang, and J. Wen, "Accelerating HEVC using heterogeneous platforms," *Journal of Supercomputing*, vol. 71, no. 2, pp. 613–628, 2014.